Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería Informática

TFG
Módulo de recopilación de información sobre Cloud Computing

Autor: Aitor Trébol Martínez

Tutor: José Javier Martínez Herraiz

2021

UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TFG

Módulo de recopilación de información sobre Cloud Computing

Autor: Aitor Trébol Martínez

Director: José Javier Martínez Herraiz

Tribunal:

Presidente:

Vocal 1º:

Vocal 2º:

Calificación: .....

Fecha: .....

Dedicado a todos aquellos que me han apoyado a lo largo de este viaje, no han dejado que me rinda y han estado ahí cuando lo he necesitado.

"Arguing that you don't care about the right to privacy because you have nothing to hide is no different than saying you don't care about free speech because you have nothing to say."
Edward Snowden

Agradecimientos

Cuando la gratitud es tan absoluta las palabras sobran

Álvaro Mutis (1923-2013)

Este trabajo significa el final de un largo viaje en el que he sido acompañado por mucha gente maravillosa. En primer lugar me gustaría agradecer a mi familia y amigos por el apoyo y la motivación que siempre me han brindado, a José Javier Martínez Herraiz por la ayuda y paciencia que ha tenido conmigo, y a todos esos compañeros de la universidad con los que he compartido este viaje.

También quiero agradecer a Alberto por todo lo que me ha enseñado y a mis compañeros Enigmas, de los cuales he aprendido más que de cualquier profesor.

Por último, gracias a Sonia por haber estado siempre ahí y no solo haberme acompañado a lo largo de la carrera, sino también, por hacerme crecer como persona.

Gracias a todos vosotros y a otros muchos hoy estoy aquí.

Resumen

El aumento del uso de cloud computing en el ámbito empresarial ha llevado a la necesidad de encontrar nuevas formas de auditar estos entornos, los cuales son de gran importancia en los procesos de una organización.

Con este proyecto se pretende desarrollar una herramienta con la que poder obtener información de fuentes abiertas de manera automática sobre la nube de Microsoft Azure, facilitando la labor de los auditores en la fase de recopilación de información durante un proceso de hacking ético. Para ello se investigará que datos se pueden obtener sobre estos entornos y las distintas maneras de recopilarlos.

Palabras clave: cloud, Azure, hacking ético, fuentes abiertas, recopilación de información.

Abstract

The increased use of cloud computing in the business environment has led to the need to find new ways of auditing these environments, which are of great importance in the processes of an organization.

This project aims to develop a tool with which to obtain information from open sources automatically on the Microsoft Azure cloud, facilitating the work of auditors in the information gathering phase during a process of ethical hacking. To do this, we will investigate what data can be obtained about these environments and the different ways to collect them.

Keywords: cloud, Azure, ethical hacking, open sources, information gathering.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Presentación

Se conoce como hacking ético al proceso mediante el cual se pretende identificar posibles debilidades, así como determinar las desviaciones en las políticas de seguridad que pudieran materializar un incidente de seguridad.

A nivel general existen varias fases en la realización de este proceso, siendo de vital importancia la fase de recopilación de información, en la que se recogen todo tipo de datos sobre la organización y los sistemas que se van a auditar.

Debido al auge de la computación en la nube y a las ventajas que proporciona a las organizaciones, las empresas son cada vez más partidarias de utilizar estas tecnologías, dando paso a entornos en nubes públicas, privadas o híbridas. Esto genera la necesidad de descubrir y automatizar nuevas formas de obtener información sobre dichos entornos, los cuales deben ser de igual manera auditados.

Este trabajo pretende desarrollar una herramienta con la que poder obtener información de manera automática sobre la nube de Microsoft Azure, facilitando la labor de los auditores en la fase de recopilación de información durante un proceso de hacking ético o pentesting. Para ello se investigará que datos se pueden obtener sobre estos entornos y las distintas maneras de recopilarlos, además de estudiar las herramientas ya existentes desarrolladas con esta finalidad.

1.2. Definición del problema

Debido al aumento del uso de la tecnología en la nube [2], aún más estos últimos años debido a la pandemia, cada vez es más difícil la identificación de los activos y los servicios que una organización deja expuestos a internet de forma pública. Este problema puede plantearse tanto desde la perspectiva ofensiva como defensiva.

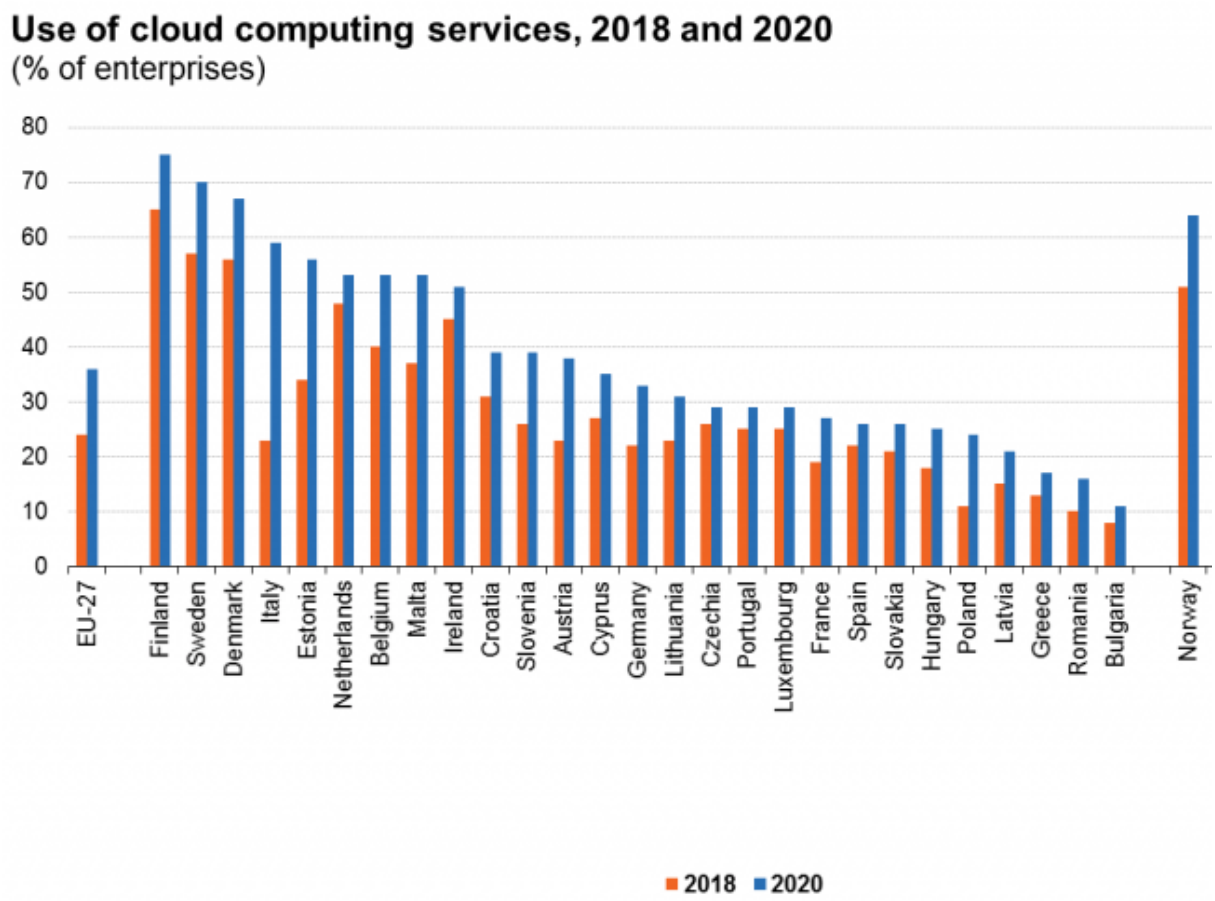

Figura 1.1: Uso de servicios cloud computing, 2018 y 2020

Por la parte ofensiva, referido a las auditorias de seguridad, debido a la gran cantidad de información que se puede encontrar sobre la nube en una organización, el esfuerzo que debe llevar a cabo un auditor para recopilar esta es cada vez mayor. Esto puede resultar en un ralentizamiento de su trabajo o en el descuido de información relevante.

Desde el punto de vista defensivo, tener monitorizados todos los servicios y activos de la organización cada vez es una tarea más compleja. Esto puede resultar en descuidos en las configuraciones y en la exposición de información vulnerable por parte de la organización, lo que puede utilizarse para formular vectores de ataque por actores maliciosos.

1.3. ¿Qué es Microsoft Azure?

Microsoft Azure, es la plataforma de computación en la nube de Microsoft. Esta permite construir, probar, desplegar y administrar una alta gama de servicios y aplicaciones mediante el uso de sus centros de datos.

La plataforma de Azure permite a las empresas alcanzar y gestionar sus objetivos organizativos, poniendo a su disposición herramientas que admiten todo tipo de industrias y que son compatibles con tecnologías de código abierto.

Azure proporciona diferentes formas de computación en la nube: software como servicio (SaaS), plataforma como servicio (PaaS) e infraestructura como servicio (IaaS). Todo esto basado en el sistema de

pago por uso, lo que implica que cada suscriptor paga cada mes por los recursos específicos que haya utilizado.

1.4. Objetivos

El objetivo principal del proyecto es desarrollar una herramienta independiente que permita obtener toda la información posible de fuentes abiertas sobre una empresa en la plataforma de Azure. Esto implica identificar sus activos, los servicios expuestos y toda la información posible de forma automatizada.

Para realizar esta tarea, se estudiarán las distintas herramientas existentes con la intención de mejorarlas o modificarlas para que trabajen en conjunto y obtengan la mejor información posible. Además, se pretende investigar qué información es admisible obtener que no esté contemplada en dichas herramientas, como esta puede complementar la que ya se obtiene, y, en el caso de ser útil, desarrollar un método para recopilarla de forma automatizada.

También se pretende estudiar que vulnerabilidades existen en la plataforma de Azure y conocer cuáles son los principales vectores de ataque actuales. Entender esto ayudará a que la herramienta se enfoque en obtener información de mayor calidad.

Aunque el desarrollo principalmente se enfoca en generar una herramienta de reconocimiento para labores ofensiva, está podrá ser utilizada de igual manera de forma defensiva por una organización, ayudando a identificar todos esos activos, servicios e información vulnerable que son necesarios de bastionar.

1.5. Alcance

La herramienta se enfocará principalmente en la investigación de las estructuras de almacenamiento proporcionadas por Microsoft Azure. También se tratará de obtener toda la información posible sobre el apartado de Office 365, software como servicio (SaaS) de Microsoft.

El objetivo de esta herramienta es puramente la recolección de información de fuentes públicas, por lo que toda la información obtenible se encontrará principalmente en internet. En ningún momento se obtendrá información de una fuente privada. Además, no será utilizado ningún método que requiera de monetización.

Es importante matizar que este proyecto trata exclusivamente sobre la recopilación de información, sin utilizar esta en ningún momento para explotar los sistemas de una organización.

1.6. Prerrequisitos

Para realizar este proyecto es necesario tener conocimientos básicos respecto a la computación en la nube, concretamente en Microsoft Azure. Además, se debe tener un nivel moderado de conocimientos sobre programación en lenguaje Python 3, ya que es el lenguaje principal en el que se desarrollará la herramienta.

Por otro lado, también es necesario entender el funcionamiento de PowerShell, la interfaz de consola de Microsoft para sistemas Windows.

1.7. Limitaciones

La principal limitación del proyecto se encuentra en el uso de APIs gratuitas, conjunto de subrutinas, funciones y procedimientos ofrecidos por cierta biblioteca para ser utilizada por otro software. Estas limitan el uso de ciertas funciones de la herramienta a un número de usos concretos.

Por otro lado, gran parte del funcionamiento requiere de la utilización de diccionarios de palabras, por lo que los resultados dependerán en cierta medida de la calidad de estos.

Debido a que la herramienta se basa en la recopilación de información de forma pasiva, no es posible utilizar métodos más agresivos que proporcionarían una información más precisa o de mayor calidad.

Capítulo 2

Estado del Arte

Y así, del mucho leer y del poco dormir, se le secó el cerebro de manera que vino a perder el juicio.

Miguel de Cervantes Saavedra

2.1. Introducción

En este apartado se tratará de establecer que otras investigaciones, herramientas o productos han surgido cuyo contexto esté relacionado con el del proyecto.

2.2. Proveedores de computación en la nube

Gracias al gran desarrollo de internet y las nuevas tecnologías, ha surgido la posibilidad de que las empresas puedan prescindir de infraestructuras informáticas físicas, optando por la contratación de servicios de cloud computing. Esto permite que las organizaciones puedan acceder a información, documentación y aplicaciones almacenadas en la nube, e incluso, trabajar en sobre ella mediante el uso de máquinas virtuales, escritorios remotos o unidades de almacenamiento.

Estos servicios de computación en la nube son ofrecidos por diferentes empresas a lo largo del mundo. Debido a la gran cantidad de proveedores existentes, se hablará únicamente de los cinco principales y cuya relevancia es mayor en el mercado [3].

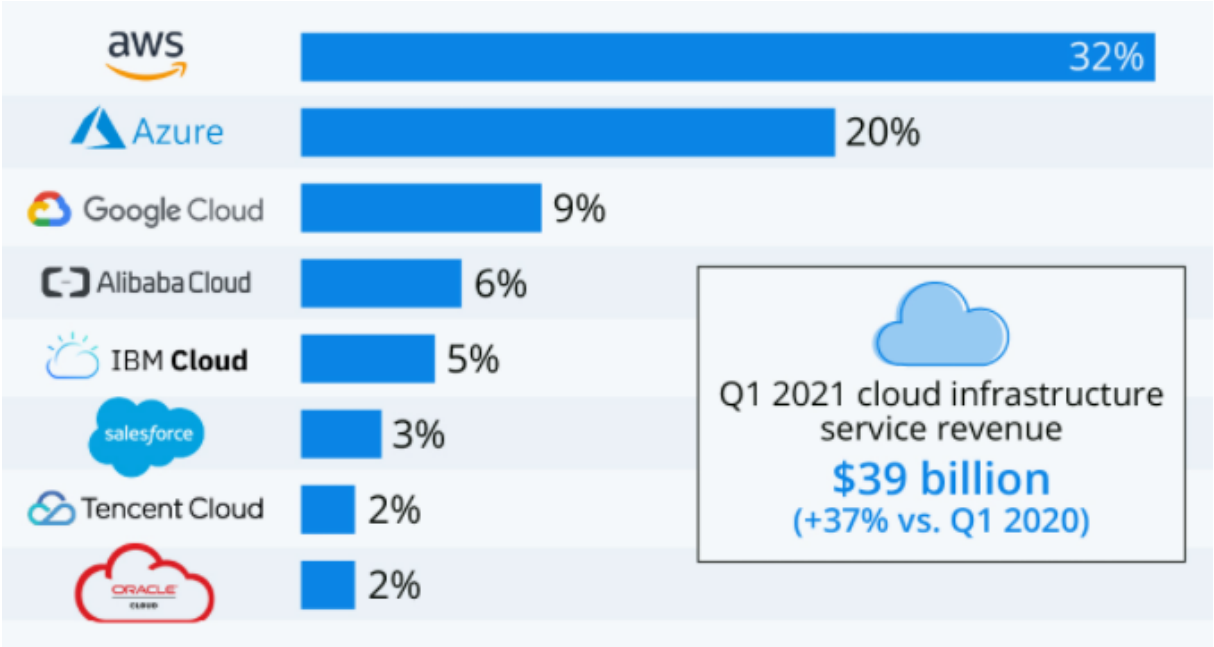

Figura 2.1: Proveedores de Cloud Computing más relevantes [1]

2.2.1. Amazon

Amazon Web Service o AWS fue uno de los pioneros en ofrecer servicios de computación en la nube. Aunque su oferta en Software as a Service (SaaS) es más limitada que la de sus competidores, ofrece una gran cantidad de soluciones en Infraestructure as a Service (IaaS) y Platform as a Service (PaaS). Esta cuenta con una gran cantidad de opciones, gran facilidad de uso y una gran escalabilidad y flexibilidad. Todo esto con precios competitivos y ajustados al mercado.

Por otra parte, Amazon EC2 (Amazon Elastic Compute Cloud), parte central de la plataforma de cómputo en la nube de Amazon, permite a los usuarios alquilar máquinas virtuales en las que ejecutar sus aplicaciones. Esto permite trabajar de forma eficiente y rápida, contando con tecnologías de CPU y GPU como Intel, AMD o NVIDIA. Respecto al almacenamiento, utiliza Amazon Elastic Block Store que dispone de varios tipos de capacidades disponibles.

2.2.2. Microsoft

Como ya se ha mencionado anteriormente, Microsoft Azure proporciona servicios en SaaS, PaaS e IaaS, convirtiéndose en uno de los proveedores que alojan sistemas completos. Además, este cuenta con un gran catálogo de aplicaciones y herramientas. Gracias a todo esto, en los últimos años ha ganado gran notoriedad y se ha expandido en gran medida.

Aunque su oferta principal son las máquinas virtuales con su sistema operativo, ofrece de igual manera máquinas basadas en otros sistemas como Linux. También destaca en su facilidad a la hora de utilizar código abierto y en la posibilidad de permitir entornos de nube híbrida.

2.2.3. Google

Google Cloud cuenta con una gran variedad de productos en el apartado de PaaS y SaaS, entre los cuales destacan: máquinas virtuales, almacenamiento, bases de datos, entornos sobre los que ejecutar

aplicaciones en contenedores, redes en las que publicar contenido web, gestión de APIs, inteligencia artificial (IA) y aprendizaje automático, además de entornos híbridos y multicloud.

2.2.4. IBM

IBM ha escalado posiciones estos últimos años gracias a su plataforma IBM Cloud, la cual ofrece servicios SaaS,PaaS e IaaS, con entornos de nube pública e híbrida. Su catálogo aun no es muy amplio, pero tiene algunos puntos fuertes como IA, Big Data, Bots y Analitycs. Estos servicios ofrecen diferentes planes de pago escalables.

2.2.5. Salesforce

Salesforce ofrece exclusivamente servicios SaaS, lo que limita su oferta en comparación a otros proveedores. Sin embargo, esta plataforma orientada a la mejora del rendimiento y la productividad de las ventas cuenta con uno de los catálogos de soluciones y productos CRM (Customer Relationship Manager) más completos, además de permitir la creación de aplicaciones personalizadas.

2.3. Herramientas existentes

Actualmente existen múltiples herramientas con el objetivo recopilar información sobre Microsoft Azure, y, aunque estas actúan sobre un mismo proveedor, su objetivo es muy distinto. A continuación, se contemplaran las herramientas más relevantes, de las cuales, algunas han sido utilizadas en el proyecto:

- **CloudBrute** [4]: Esta herramienta trata de enumerar las infraestructuras, archivos y aplicaciones que una organización utiliza en un entorno cloud. Está basada en la generación de palabras mediante mutaciones gracias a la utilización de diccionarios. Una vez generadas las palabras, realiza fuerza bruta probando las posibles direcciones y procesando el resultado. Algunas herramientas similares son **Cloud_Enum** [5] y **BlobHunter** [6].
- **AAD Internals** [7]: AAD Internals es un módulo de PowerShell con una gran cantidad de funciones relacionadas con los directorios activos de Azure (Azure AD) y Office 365. En este caso nos centraremos en los módulos **Invoke-AADIntReconAsOutsider**, el cual reúne información sobre el Azure AD de una organización a partir de un dominio, e **Invoke-AADIntUserEnumerationAsOutsider**, que permite comprobar si un usuario existe en ese Azure AD.
- **o365creeper** [8]: Este script permite validar cuentas de correo que pertenezcan a Office 365. Una alternativa más agresiva sería **MSOLSpray** [9], herramienta con la que es posible probar contraseñas y que ofrece una información más completa con respecto a la existencia de una cuenta.
- **MicroBurst** [10]: MicroBurst es una herramienta muy completa, la cual incluye funciones y scripts en PowerShell que admiten el descubrimiento de servicios de Azure, la comprobación de configuraciones débiles y acciones de post-explotación.
- **ROADtools** [11]: ROADtools es un framework que permite interactuar con Azure AD y obtener información. Es útil tanto de forma ofensiva como defensiva. Para poder utilizar la herramienta es necesario hacer uso de credenciales.

Capítulo 3

Estudio teórico

First to mind when asked what 'the cloud' is, a majority respond it's either an actual cloud, the sky, or something related to weather.

Citrix Cloud Survey Guide

3.1. Introducción

A continuación, se van a exponer todos los conceptos teóricos sobre los que se ha basado el proyecto y que son necesarios para comprender el funcionamiento de la herramienta.

3.2. Modelos de servicio en la nube

En la actualidad, existen tres modelos de servicio principales de computación en la nube [12], estos son se conocen como Software como Servicio(SaaS), Plataforma como Servicio (PaaS) e Infraestructura como Servicio (IaaS). Cada uno de estos se ofrece con la intención de resolver los diferentes requerimientos basados en las necesidades de cada empresa.

- **SaaS**: El Software como Servicio (SaaS) es el modelo de servicio en la nube encargado de proporcionar acceso a un producto software junto con toda su infraestructura IT y plataformas subyacentes. Este es administrado y gestionado por el proveedor de la aplicación, lo que permite a las organizaciones no tener que encargarse de mantenimiento, disponibilidad o seguridad. Un SaaS suele facturarse en función del número de usuarios, tiempo de uso o cantidad de datos almacenados.
- **PaaS**: La Plataforma como Servicio (PaaS) es el modelo de servicio en la nube en el que se accede a un ambiente con herramientas tanto de hardware como de software a través de un proveedor de servicios. Es comúnmente utilizado para el desarrollo de aplicaciones. Al tratarse de un servicio basado en la nube, no es necesario preocuparse por la configuración o el mantenimiento de servidores, actualizaciones o autenticaciones.
- **IaaS**: La Infraestructura como Servicio (IaaS) es el modelo de servicio en la nube que ofrece una forma estandarizada de adquirir recursos IT (incluyendo sistemas, redes y almacenamiento) por demanda durante un periodo de tiempo definido. Este puede ser escalado para satisfacer las necesidades de procesamiento y almacenamiento del cliente.

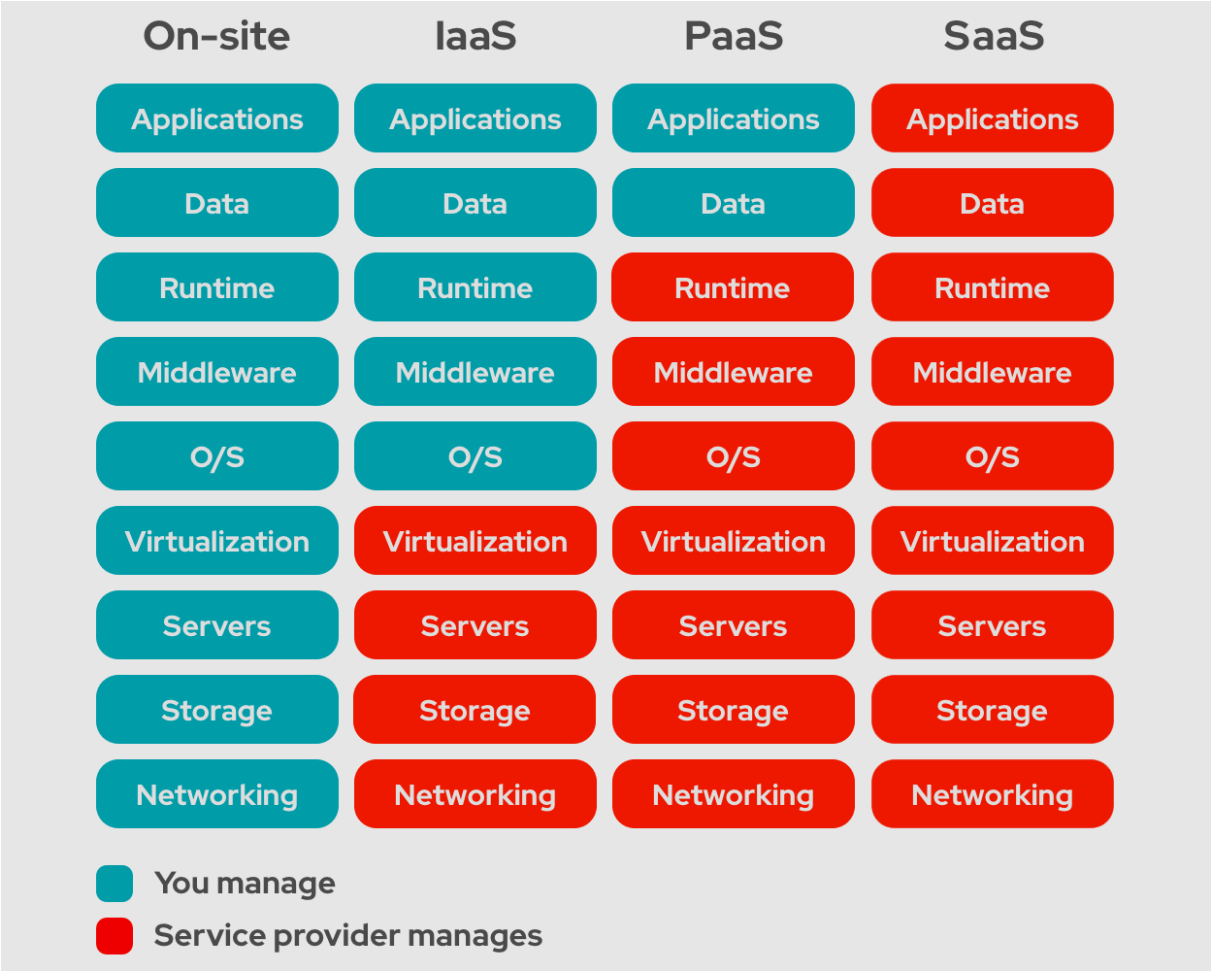

Figura 3.1: Diferencias entre IaaS, PaaS y SaaS

3.3. Tipos de nubes

A parte de los tres modelos de servicio en cloud computing, también podemos diferenciar el tipo de nube con respecto a su ubicación y la propiedad. Por esta razón, se procede a analizar los principales tipos de nube y a comentar sus diferentes aspectos[13].

- **Nubes públicas**: Es un servicio en la nube que es ofrecido por un proveedor a varios clientes. Normalmente, el servicio de nube pública se ejecuta en servidores en remoto gestionados por un proveedor y se accede a él a través de internet. Este tipo de nube incluye servicios SaaS, PaaS e IaaS.

- **Nubes privadas**: Es un servicio en la nube destinado exclusivamente a un usuario u organización. Una nube pasa a ser privada cuando la infraestructura IT se destina a un único cliente con acceso aislado.

- **Nubes híbridas**: Una nube híbrida combina la infraestructura del entorno local o nube privada, con la una nube pública. Este tipo de nubes permiten que los datos y las aplicaciones se muevan entre los dos entornos.

3.4. Microsoft Azure

Azure es la plataforma de computación en la nube y el portal online que permite acceder y administrar los servicios cloud y los recursos proporcionados por Microsoft. Esta plataforma permite construir, probar, desplegar y administrar una alta gama de servicios y aplicaciones mediante el uso de sus centros de datos. Proporciona modelos de servicio SaaS, PaaS e IaaS y es compatible con un gran número de lenguajes, herramientas y frameworks diferentes, entre estos, software y sistemas tanto de Microsoft como de terceros.

Azure provee más de 200 servicios divididos en 22 categorías [14], lo que permite a las empresas alcanzar y gestionar sus objetivos organizativos, poniendo a su disposición herramientas que admiten todo tipo de industrias y que son compatibles con tecnologías de código abierto.

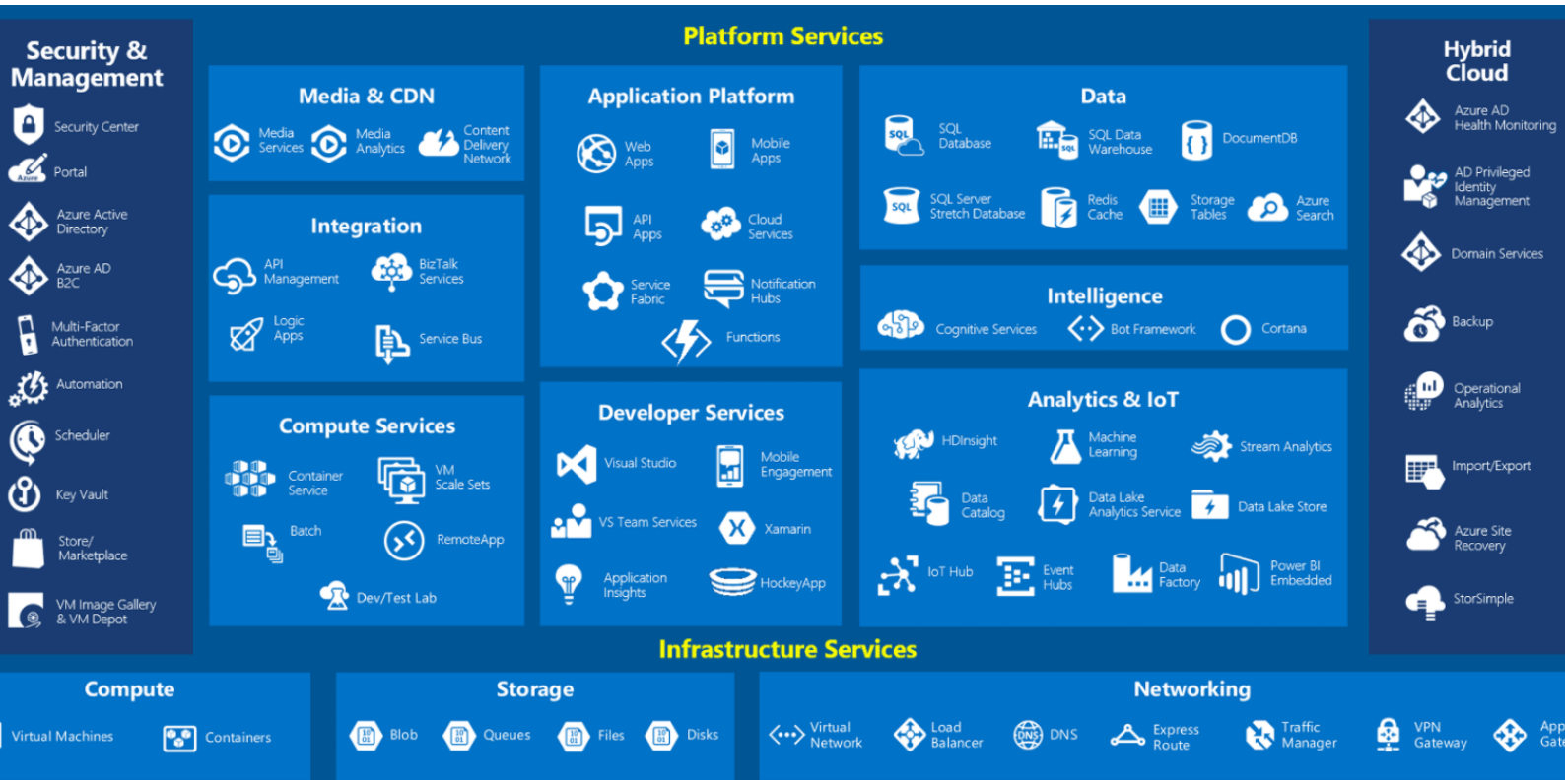

Figura 3.2: Servicios Azure IaaS y PaaS

A continuación, se hablará de los servicios y categorías más relevantes para el desarrollo del proyecto.

3.4.1. Azure Storage

Azure Storage [15] es el servicio de almacenamiento de datos en la nube de Microsoft. Estos ofrecen un almacén de objetos escalable de forma masiva para almacenar datos, máquinas virtuales, sistemas de archivos, mensajería y tablas NoSQL. Las características principales de sus servicios son:

- **Durabilidad y disponibilidad**: Mediante la redundancia se garantiza que los datos estén seguros en caso de errores de hardware. Además, existe la posibilidad de replicar los datos entre distintos data centers o regiones geográficas para protegerlos de desastres naturales o catástrofes locales. Esto permite que los datos permanezcan con una alta disponibilidad.
- **Seguridad**: Todos los datos introducidos en una cuenta de Azure Storage son cifrados mediante el servicio. También proporciona un control sobre los usuarios que pueden acceder a estos.

- **Escalabilidad**: Estos servicios pueden ser escalados de forma masiva y ofrecer el rendimiento y almacenamiento de datos necesario por las aplicaciones de hoy en día.
- **Administración**: Microsoft Azure se encarga de manera automática de las actualizaciones, mantenimiento y problemas del hardware.
- **Accesibilidad**: Microsoft proporciona acceso a los datos almacenados a través de HTTP o HTTPS desde cualquier parte del mundo, además de poner a disposición bibliotecas en múltiples lenguajes de programación y una API REST con la que interactuar con Azure Storage. Por otro lado, admite la escritura en Azure PowerShell o la CLI de Azure, mientras que mediante Azure Portal y el Explorador de Azure Storage ofrece soluciones visuales sencillas con las que trabajar con los datos.

Los principales servicios de almacenamiento en la plataforma de Azure Storage son Blobs, Tables, Queues y Files [16].

3.4.1.1. Azure Blob Storage

Blob Storage permite almacenar objetos binarios de gran tamaño o Blobs, que suelen estar compuestos por datos no estructurados como imágenes, vídeos o texto, junto con sus metadatos. Este servicio se estructura de la siguiente manera:

- **Blob**: Son los objetos de datos no estructurados de cualquier tipo.
- **Containers**: Almacenan varios Blobs en estructuras similares a directorios.
- **Azure Storage Account**: Contiene todos los objetos de datos de almacenamiento de Azure.

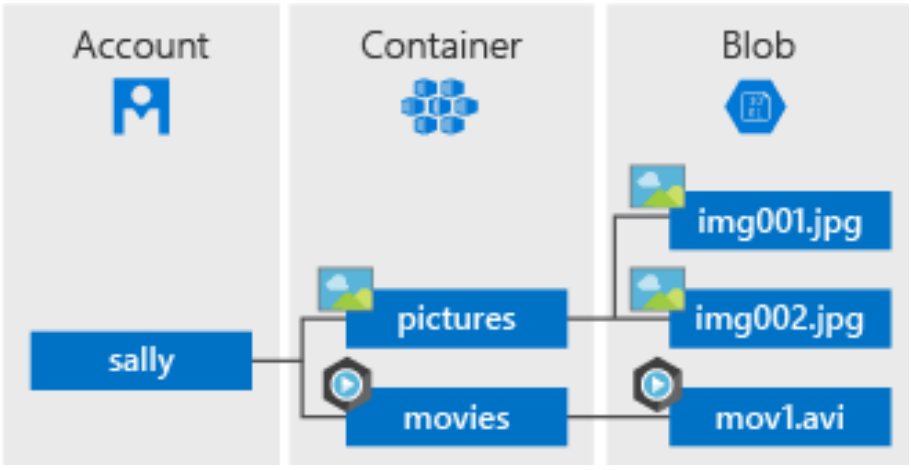

Figura 3.3: Estructura de Blob Storage

Por otra parte, los Blobs están optimizados para tres escenarios de almacenamiento diferentes:

- **Block Blobs**: Estos Blobs almacenan objetos discretos como archivos de registros o imágenes. Pueden llegar a almacenar hasta 5 terabytes o 50.000 bloques de 100 megabytes cada uno.
- **Page Blobs**: Están optimizados para operaciones de lectura y escritura aleatoria. Su tamaño puede llegar a los 8 terabytes. En esta categoría Azure distingue entre dos tipos de almacenamiento, estándar y premium, siendo el premium el más adecuado para el almacenamiento de máquinas virtuales.

- **Append Blobs**: Esta categoría está optimizada para el almacenamiento de registros. Los Append Blobs se componen por varios bloques de distinto tamaño de hasta 4 megabytes, pudiendo acumular un tamaño total de 200 gigabytes compuestos por hasta 50.000 bloques.

Para finalizar, existen tres tipos de clasificación de Blobs en función de la frecuencia con la que estos vayan a ser accedidos:

- **Hot Access**: La categoría de Hot Access es la más óptima para los datos a los cuales se va a acceder de forma frecuente. Esta ofrece el menor coste de acceso, pero el mayor coste de almacenamiento.

- **Cold Access**: Esta opción es usada en casos en los que los datos van a almacenarse al menos 30 días y no van a ser accedidos de forma frecuente. Ofrece menor coste de almacenamiento a cambio de mayor latencia al acceder a estos.

- **Archive Access**: Este tipo de almacenamiento se enfoca en datos que no necesitan ser accedidos de forma inmediata. Esto permite almacenar una gran cantidad a un bajo coste de almacenamiento. Está diseñado para casos en los que los datos van a estar almacenados más de 180 días y no se suele acceder a ellos.

3.4.1.2. Azure Table Storage

Este sistema de almacenamiento de datos escalable, NoSQL y de clave valor puede utilizarse para almacenar grandes cantidades de datos. Este tipo de almacenamiento tiene un diseño sin esquema, basado en tablas con filas que se componen de pares clave-valor. Esta solución permite almacenar datos estructurados y no relacionales, que abarca casos de usos desde el almacenamiento para aplicaciones web, hasta el almacenamiento de conjuntos de datos que no requieran uniones complejas o claves externas.

Es importante saber que las tablas de Azure no tienen noción de columnas, restricciones o relaciones 1:1 o 1:*, ni ninguna de sus variaciones. El servicio está compuesto por:

- **Azure Storage Account**: Como ya se ha comentado antes, contiene todos los objetos de datos de almacenamiento de Azure, incluidas las tablas.

- **Tablas**: Agrupan colecciones de *entities*.

- **Entities**: Conjuntos que pueden envolver hasta 252 propiedades para almacenar datos, además de tres propiedades del sistema que define su clave de partición, clave de fila y su marca de tiempo. Son similares a las filas en las bases de datos. Tienen un tamaño máximo de 1 megabyte.

- **Properties**: El elemento más granular de estos. Se componen de pares nombre-valor.

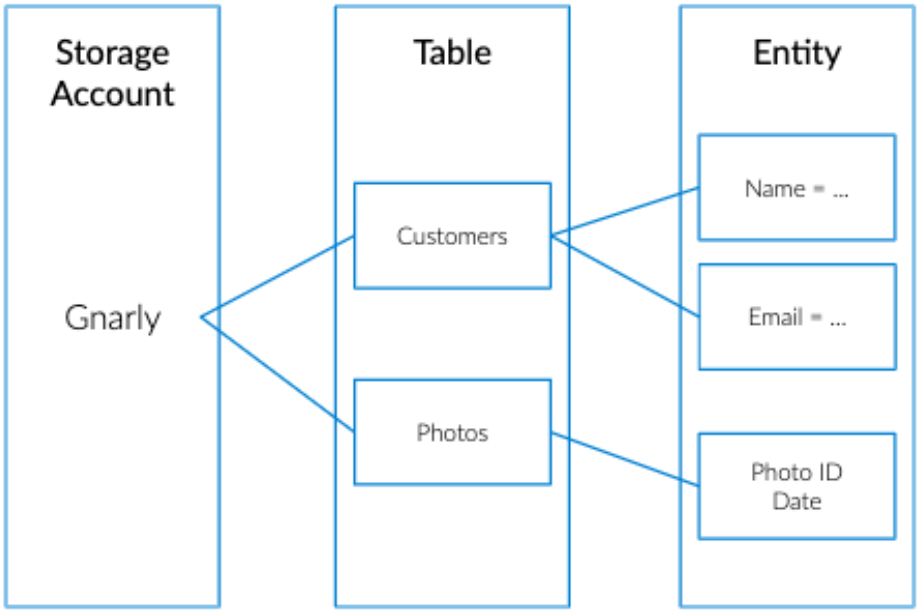

Figura 3.4: Estructura de Table Storage

3.4.1.3. Azure Queue Storage

Las colas son una estructura de almacenamiento sencillo de arquitectura FIFO (first in, first out) que se ofrece como solución versátil para almacenar mensajes que no necesitan estar en un orden determinado. Permite almacenar un gran volumen de mensajes, procesarlos de forma asíncrona y consumirlos cuando sea necesario, manteniendo un bajo coste.

Un mensaje incluye cualquier tipo de información, como puede ser un mensaje de texto necesario para activar un evento en una aplicación, datos sobre un evento en una página web, o incluso cadenas de bytes que contengan datos en formatos como XML o CSV. Independientemente del formato, el tamaño máximo de un mensaje es de 64 kilobytes y no puede permanecer en una cola más de 7 días.

3.4.1.4. Azure File Storage

Azure Files es una red compartida de almacenamiento de archivos. Esta permite a los administradores acceder de forma nativa a recursos compartidos SMB en la nube. Estos recursos pueden configurarse como parte de la Azure Storage Account, proporcionando una forma de que las aplicaciones ejecutadas en máquinas virtuales en la nube compartan archivos entre ellas mediante protocolos estándar.

3.4.2. Azure Virtual Machines

Azure Virtual Machine [17] es uno de los servicios proporcionados por Microsoft Azure. Este ofrece una gran flexibilidad de la virtualización sin tener que preocuparse por la adquisición de hardware ni su mantenimiento. Sin embargo, sí que deja a disposición del cliente las tareas relacionadas con la máquina virtual, como su mantenimiento, actualizaciones, o software.

Las máquinas virtuales pueden ser utilizadas de múltiples maneras. Algunos de los usos más comunes son el desarrollo y pruebas, gracias a la facilidad y rapidez para crear y configurar un equipo con el que

codificar y probar aplicaciones, o la ampliación de centros de datos gracias su sencilla conectividad a la red de una organización.

El número de máquinas virtuales usadas es escalable tanto verticalmente como horizontalmente, además de permitir el uso de sistemas operativos tanto Windows como Linux.

3.4.3. Azure SQL Databases

Azure SQL Database [18] es un servicio de bases de datos relacional escalable en la nube. Este administra las bases de datos SQL de manera automatizada, encargándose de las actualizaciones, el aprovisionamiento y las copias de seguridad, siendo ejecutado siempre en la última versión estable del motor de base de datos de SQL Server y en un sistema operativo revisado con una muy alta disponibilidad.

Debido a su flexibilidad y capacidad de respuesta, permite adaptar rápidamente el proceso *sin servidor* y el nivel de *Hiperescala* a las necesidades correspondientes. Por otro lado, también gestiona la protección de los datos mediante controles integrados y detección de amenazas, mientras mantiene una durabilidad y un rendimiento máximos.

3.4.4. Azure Web Apps

El servicio de Azure Web App [19] está basado en HTTP y permite alojar aplicaciones web, APIs REST y back ends móviles. Permite el desarrollo en múltiples lenguajes de programación, ya sea .NET, .NET Core, Java, Ruby. Node.js, PHP o Python. Estas aplicaciones se ejecutan y escalan de forma sencilla independientemente de si el entorno se basa en Windows o en Linux.

Este servicio proporciona a sus aplicaciones la potencia, seguridad, equilibrio de carga, autoescalado y gestión automatizada requerida por el cliente. También permite la implementación continua desde Azure DevOps, Github, o Docker Hub entre otros, la gestión de paquetes, entornos de preparación, dominios personalizados y certificados TLS/SSL.

3.5. Microsoft Office 365

Microsoft Office 365 es el modelo de servicio SaaS de Microsoft [20]. Este abarca un conjunto de servicios entre los cuales figuran las aplicaciones de Microsoft 365 (PowerPoint, Word, Excel, OneNote, Publisher y Access), correo y calendario (Outlook, Exchange, Bookings), reuniones de voz (Teams), redes sociales e intranet (SharePoint y Yammer), archivos y contenido (OneDrive, Stream, Sway, Lists y Forms), herramientas de administración del trabajo (Planner, Power Apps, Power Automate, Power Virtual Agents y To Do) y análisis avanzados(MyAnalitycs, Power BI Pro).

Estos servicios se ofrecen en base a suscripciones abonadas de forma periódica, los cuales varían en función del plan y la licencia que haya sido obtenida.

3.5.1. Planes de Office 365

Office 365 se divide distintos planes en función del tipo de negocio, siendo Business y Enterprise los enfocados a organizaciones. Estas se diferencian en la cantidad máxima de miembros que pueden soportar, limitando la de Office 365 Business a 300 usuarios, mientras que Office 365 Enterprise no tiene limitaciones. También existen algunas diferencias a nivel de cumplimiento las cuales pueden ir variando.

En Office 365 cada usuario requiere de una licencia que define las funcionalidades de las que va a disponer el usuario.

El plan Enterprise es el más común en el entorno corporativo, ofreciendo los tipos de licencias ProPlus E1, E3 y E5. Estas dos primeras son las más básicas, ofreciendo acceso a las aplicaciones de escritorio con ProPlus, y correo y videoconferencia con E1, mientras que las licencias E3 y E5 ofrecen a cambio de un mayor coste funcionalidades más avanzadas.

El plan Business está enfocada a consumidores finales y pequeños y medianos negocios. En este caso las suscripciones toman el nombre de Microsoft 365 Business Basic (acceso a servicios), Microsoft 365 Apps for Business (uso de aplicaciones de escritorio y OneDrive) y Microsoft 365 Business Standard (conjunto de ambas).

3.6. Azure AD

Azure Active Directory [21] es un servicio que permite administrar las identidades y el acceso en la nube de Microsoft. Esto ayuda a los usuarios a iniciar sesión y acceder a los recursos tanto externos (Microsoft 365, Azure Portal u otras aplicaciones SaaS), como internos (aplicaciones de la red corporativa o aplicaciones en la nube desarrolladas por la organización). Este va dirigido principalmente a:

- **Administradores de IT**: Este perfil de usuario puede controlar el acceso a aplicaciones y recursos según los requisitos de la organización. También es posible automatizar el aprovisionamiento de usuarios entre instancias de Windows Server AD y las aplicaciones en la nube. Por otro lado, proporciona herramientas que permiten proteger las identidades y credenciales de los usuarios y ayudan a cumplir los requisitos de acceso.
- **Desarrolladores**: Mediante la configuración de inicio de sesión único (SSO), un usuario puede acceder a las distintas aplicaciones con las mismas credenciales. Azure AD además proporciona varias APIs con las que personalizar las aplicaciones que utilicen datos de la organización.
- **Suscriptores de Microsoft 365,Office 365, Azure o Dynamics CRM Online**: Los suscriptores utilizan Azure AD, ya que cada Tenant de cualquiera de estos servicios es automáticamente un Tenant de Azure AD.

3.6.1. Azure AD y Active Directory

A pesar del nombre engañoso, Azure AD no es lo mismo que un Active Directory en la nube. Las diferencias entre estos son las siguientes:

(Windows Server)Active Directory	Azure Active Directory
LDAP	REST API's
NTLM/Kerberos	OAuth/SAML/OpenID/etc
Structured directory	Flat Structure
GPO's	No GPO's
Super fine-tuned access controls	Predefined roles
Domain/forest	Tenant
Trusts	Guests

Tabla 3.1: Diferencias AAD y AD

3.6.2. Roles y permisos

Azure AD trae una gran cantidad de términos, los cuales pueden ser malinterpretados con los de Active Directory. En este caso, el rol con el nivel de privilegios más alto es el de Global Admin, aunque también puede encontrarse con el nombre de Company Admin. Este puede administrar cualquier cosa relacionada con la suscripción de Azure AD.

Además de este rol, existen decenas de roles con permisos de administrador limitados [22]. Algunos ejemplos son:

- **Administrador de Aplicaciones**: Puede crear y administrar todos los aspectos de los registros de aplicaciones y aplicaciones empresariales
- **Administrador de Usuarios**: Pueden administrar todos los aspectos de usuarios y grupos, incluido el restablecimiento de credenciales.
- **Administrador de Autenticación**: Puede ver y configurar la información de los métodos de autenticación de los usuarios no administradores.

Azure Resource Manager es el servicio de implementación y administración, con el cual gestionar la creación, actualización y eliminación de recursos de la cuenta de Azure. Este se apoya en Azure AD para la gestión de identidades, por lo que un usuario privilegiado en Azure AD también tiene acceso a Azure Resource Manager.

3.6.3. Identidades de Azure AD

Existen varios tipos de identidades en Azure AD para el manejo de usuarios y otros objetos en la nube:

- **Cloud Only Identity**: Los usuarios son creados y administrados directamente en la nube. No cuenta con un servicio de Directorio Activo On Premises o no se ha vinculado este con Azure AD. En este caso, todo lo relacionado con identidades y autenticación es gestionado por Azure AD, administrando las cuentas desde Office 365 Admin Center o mediante Windows Powershell. En la Figura 3.5 se puede ver la estructura de esta identidad y como un administrador realiza un cambio en el Centro de Administración de Office 365.

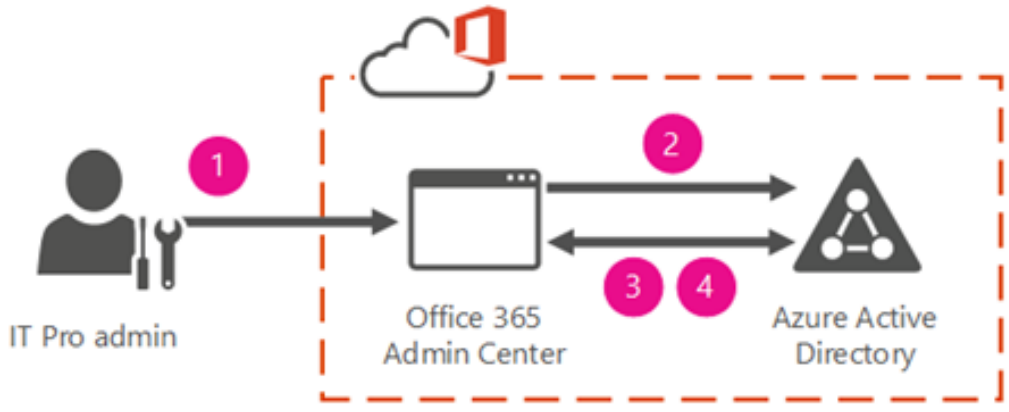

Figura 3.5: Cloud Only Identity

- **Synchronized Identity**: Organizaciones que cuentan con Active Directory y desea sincronizar las identidades o usuarios (de forma parcial o total) con la nube. Esto deriva en la generación de escenarios híbridos con un cierto nivel de integración. Para realizar esta tarea se utiliza Azure AD Connect, servicio que se configura normalmente en un servidor dedicado y permite la sincronización de credenciales con Azure. De esta forma, los usuarios acceden tanto a recursos locales, como a recursos en la nube con un mismo usuario y contraseña.

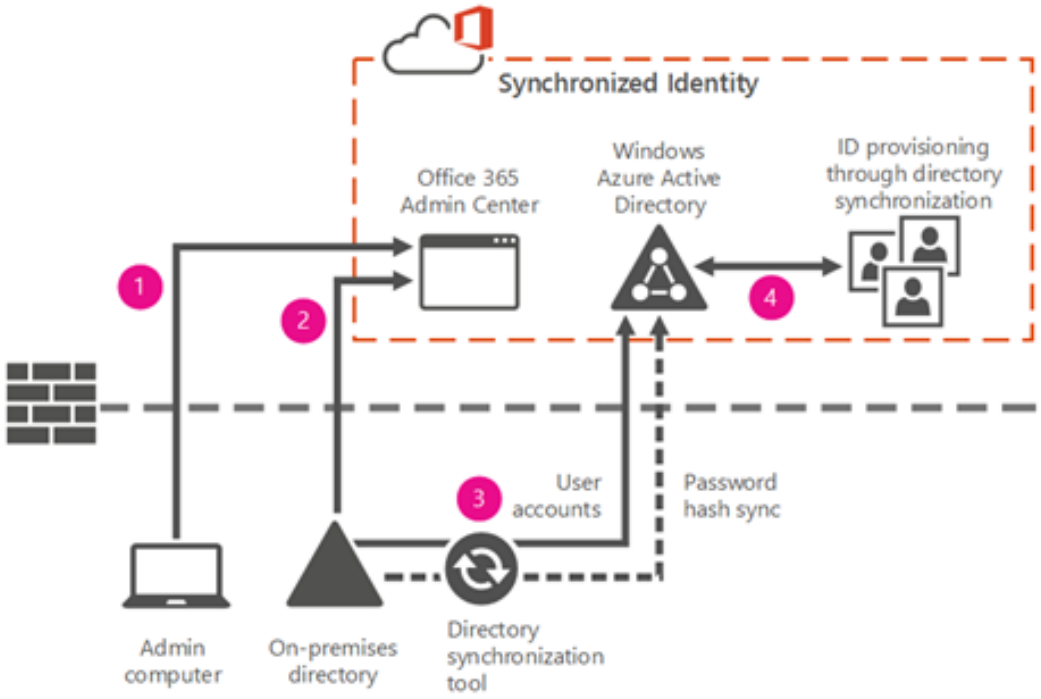

Figura 3.6: Synchronized Identity

- **Federated Identity**: Es un escenario más avanzado a nivel de autenticación que los anteriores. Es requerido para habilitar funciones como el Single Sing On (SSO), lo que permite a los usuarios tener acceso a múltiples aplicaciones, recursos y sistemas con una única cuenta. Este modelo necesita la sincronización de identidades, siendo el que las valida un directorio On Premises. Para esto es necesario el servicio adicional Active Directory Federation Services (AD FS) [23], que permite la administración de identidades federadas y su acceso de forma segura dentro de los límites de la organización.

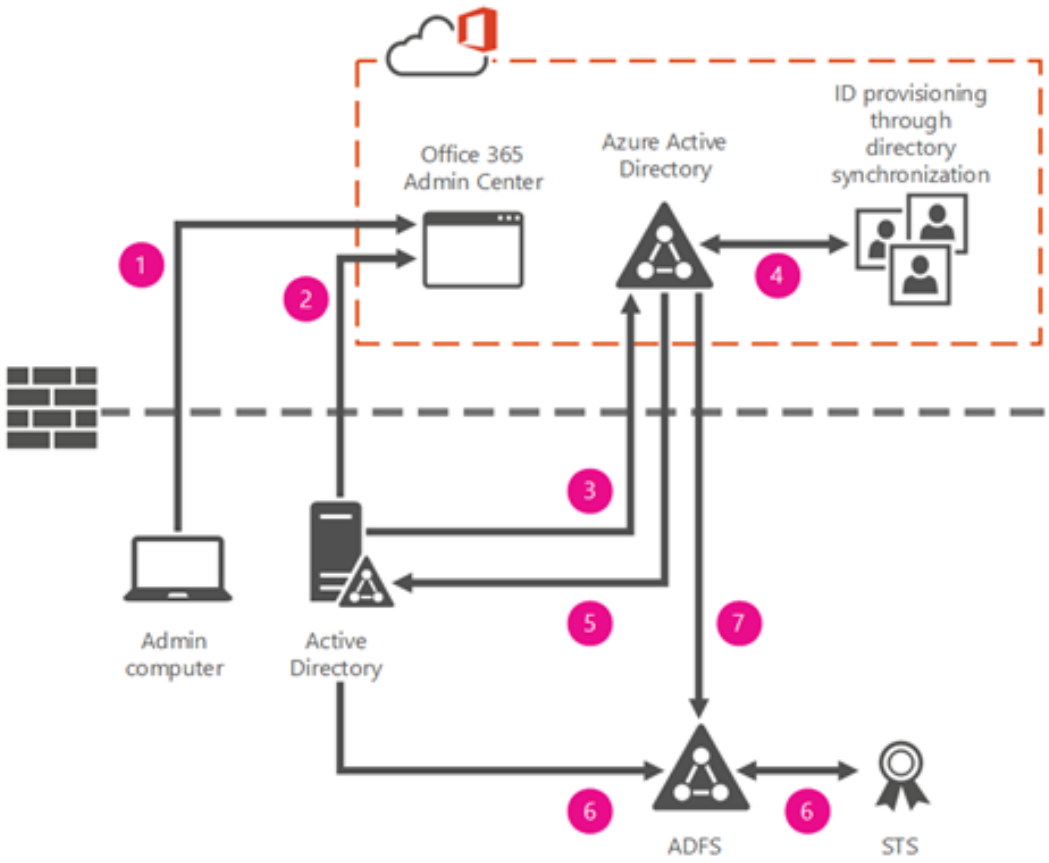

Figura 3.7: Federated Identity

3.6.4. Integración con On-Premises Active Directory

Como se ha comentado anteriormente, existe la posibilidad de Integrar Azure AD con un Directorio Activo desplegado de forma local. Para esto, muchas organizaciones utilizan Active Directory Domain Services (AD DS) para autenticar de las identidades asociadas a los usuarios, los sistemas o las aplicaciones.

Para habilitar que los usuarios del AD puedan utilizar unas únicas credenciales con las que acceder al entorno local y la nube, es necesario sincronizar los hashes de las contraseñas. Esto se puede realizar de tres formas:

- **Password Hash Synchronization (PHS)** [24]: Es uno de los métodos más usados por las organizaciones para lograr una identidad híbrida. Mediante Azure AD Connect se sincroniza el hash obtenido del hash ya existente de la contraseña desde una instancia de Active Directory con una instancia de Azure AD.
- **Pass-through Authentication (PTA)** [25]: Azure AD Connect aporta este servicio en forma de agente que es ejecutado en uno o varios servidores locales unidos a un dominio, el cual valida el inicio de sesión de un usuario en nombre de Azure AD directamente con el Directorio Activo local. Este agente se conecta con Azure AD y escucha las solicitudes de autenticación, requiriendo exclusivamente que los puertos de salida estén abiertos.
- **Active Directory Federation Services (AD FS)**: Al igual que el anterior, este servicio hace uso de agentes instalados en los sistemas operativos Windows Server. Utiliza un modelo de autorización

de control de acceso basado en notificaciones para mantener la seguridad de la aplicación e implementar la identidad federada. Esta autenticación utiliza la transmisión de un token de confianza el cual requiere de ser validado.

3.6.5. Azure AD Tenant

Un Tenant es una instancia dedicada de Azure AD creada de forma automática cuando una organización se suscribe a un servicio en la nube de Azure. Estos Tenants funcionan como contenedores individuales y se identifican bajo el subdominio ***onmicrosoft.com***. El nombre seleccionado para el Tenant debe ser exclusivo y suele corresponderse con el nombre de la organización. Los usuarios pertenecientes a esta de forma predeterminada se autenticarían mediante una dirección de correo con el formato ***usuario@organización.onmicrosoft.com***.

Debido a que normalmente las organizaciones cuentan con un nombre público registrado en DNS, estas suelen validar la propiedad de dicho nombre para que sea posible la autenticación mediante correos con el formato ***@organización.com***, omitiendo el uso de *onmicrosoft.com*.

Capítulo 4

Desarrollo Experimental

I don't need a hard disk in my computer if I can get to the server faster... carrying around these non-connected computers is byzantine by comparison.

Steve Jobs

4.1. Introducción

La herramienta desarrollada se encarga de obtener información de fuentes abiertas relacionada con la utilización de Microsoft Azure en una organización. Esta se enfoca principalmente en la búsqueda de información en el apartado de Azure Storage, Web Apps, SQL Databases y Virtual Machines, además de obtener información sobre el Azure AD utilizado por la organización. También da la posibilidad de comprobar la existencia de usuarios en el entorno.

Como resultado, el programa devuelve un fichero con un JSON con toda la información recolectada, a la vez que lo muestra por el terminal.

4.2. Arquitectura

Como ya se ha comentado anteriormente, el programa ha sido desarrollado en Python 3 y requiere de llamadas a Powershell a lo largo de la ejecución. Estas llamadas no requieren del uso de librerías exclusivas de Microsoft, lo que permite su uso en sistemas operativos Windows y Linux.

Para el desarrollo de la herramienta se ha optado por un paradigma de programación funcional, debido a la necesidad de realizar un programa modular. Esto permite en un futuro la adaptabilidad a nuevas actualizaciones y la implementación o eliminación de módulos, los cuales funcionan de manera independiente.

El programa hace uso de varias herramientas ya desarrolladas las cuales se han modificado con el objetivo de cambiar su funcionamiento o poder ser integradas. Estas son:

- **Cloud_Enum**: Esta herramienta nos permite obtener información sobre Azure Blob Storage, Web Apps, SQL Databases y Virtual Machines. Está basada en la generación de palabras mediante mutaciones gracias a la utilización de diccionarios. Una vez generadas las palabras, realiza fuerza bruta probando las posibles direcciones y procesando el resultado.

- **AADInternals**: AAD Internals es un módulo de PowerShell con una gran cantidad de funciones relacionadas con los directorios activos de Azure (Azure AD) y Office 365. En este caso se utilizarán los módulos **Invoke-AADIntReconAsOutsider**, el cual reúne información sobre el Azure AD de una organización a partir de un dominio, e **Invoke-AADIntUserEnumerationAsOutsider**, que permite comprobar si un usuario existe en ese Azure AD.

- **O365creeper**: Este script permite validar cuentas de correo que pertenezcan a Office 365. Es utilizado como alternativa en caso de no poder usarse AADInternals.

Por otra parte, se han desarrollado dos módulos correspondientes a la utilización de los motores de búsqueda de Google y Shodan para la utilización de dorks y recopilación de información en estos. Además, junto con la herramienta se aportan una serie de diccionarios con los cuales realizar la mutación de directorios. Estos son de distintos tamaños en función del tiempo que se le quiera dedicar a la tarea. El nombre de los archivos es *fuzz.txt* (242 palabras), *storage_small.txt* (112 palabras), *storage_large.txt* (18.425 palabras).

A continuación, se van a listar las bibliotecas de Python necesarias para el correcto funcionamiento de la herramienta.

- **time**: Este módulo provee de varias funciones relacionadas con el tiempo. Permite controlar los tiempos de ejecución del programa.

- **sys**: Esta biblioteca permite el acceso a variables y funciones que interactúan o pertenecen al intérprete.

- **datetime**: Permite la manipulación de fechas y tiempos. También utilizada en el control de los tiempos de ejecución.

- **re**: Provee operaciones relacionadas con expresiones regulares. Permite filtrar cadenas de caracteres y obtener la información necesaria.

- **multiprocessing**: Es un paquete que ofrece simultaneidad local y remota, evitando el Global Interpreter Lock mediante el uso de subprocesos. Es usado en la fuerza bruta de DNS.

- **functools**: Es utilizada en funciones que actúan o devuelven otras funciones. Para este módulo, cualquier objeto invocable puede tratarse como una función.

- **requests**: Gracias a esta biblioteca podemos realizar peticiones HTTP con facilidad. Es utilizada a lo largo de todo el programa y aporta información como el estado de una petición.

- **dnspython**: Conjunto de herramientas DNS para Python. Es utilizada para realizar consultas DNS, transferencias de zona o actualizaciones dinámicas.

- **requests_futures**: Es utilizada por Cloud_Enum. Actúa como complemento de la librería *requests*. Hace uso de *concurrent.futures* de Python 3.2.

- **googleapiclient**: Es la biblioteca de Python de la API de Google. Requerido para realizar búsquedas utilizando este motor.

- **argparse**: Gracias a este módulo es posible desarrollar una interfaz de command-line de forma sencilla. Se utiliza para definir los distintos argumentos requeridos por la herramienta.

- **os**: Proporciona una manera sencilla de interactuar con el sistema operativo.

- **json**: Biblioteca que nos ayuda a dar formato JSON a la salida del programa.
- **socket**: Utilizada en la creación de Sockets. Nos permite realizar acciones como resolver un dominio DNS.
- **subprocess**: Permite generar nuevos procesos, utilizar tuberías y obtener su resultado. Utilizado en la creación de procesos Powershell.
- **shodan**: Esta biblioteca proporciona métodos con los que realizar búsquedas en el motor de Shodan.

Como resultado de todo esto, el programa termina con una estructura como la que se puede apreciar en la Figura 4.1.

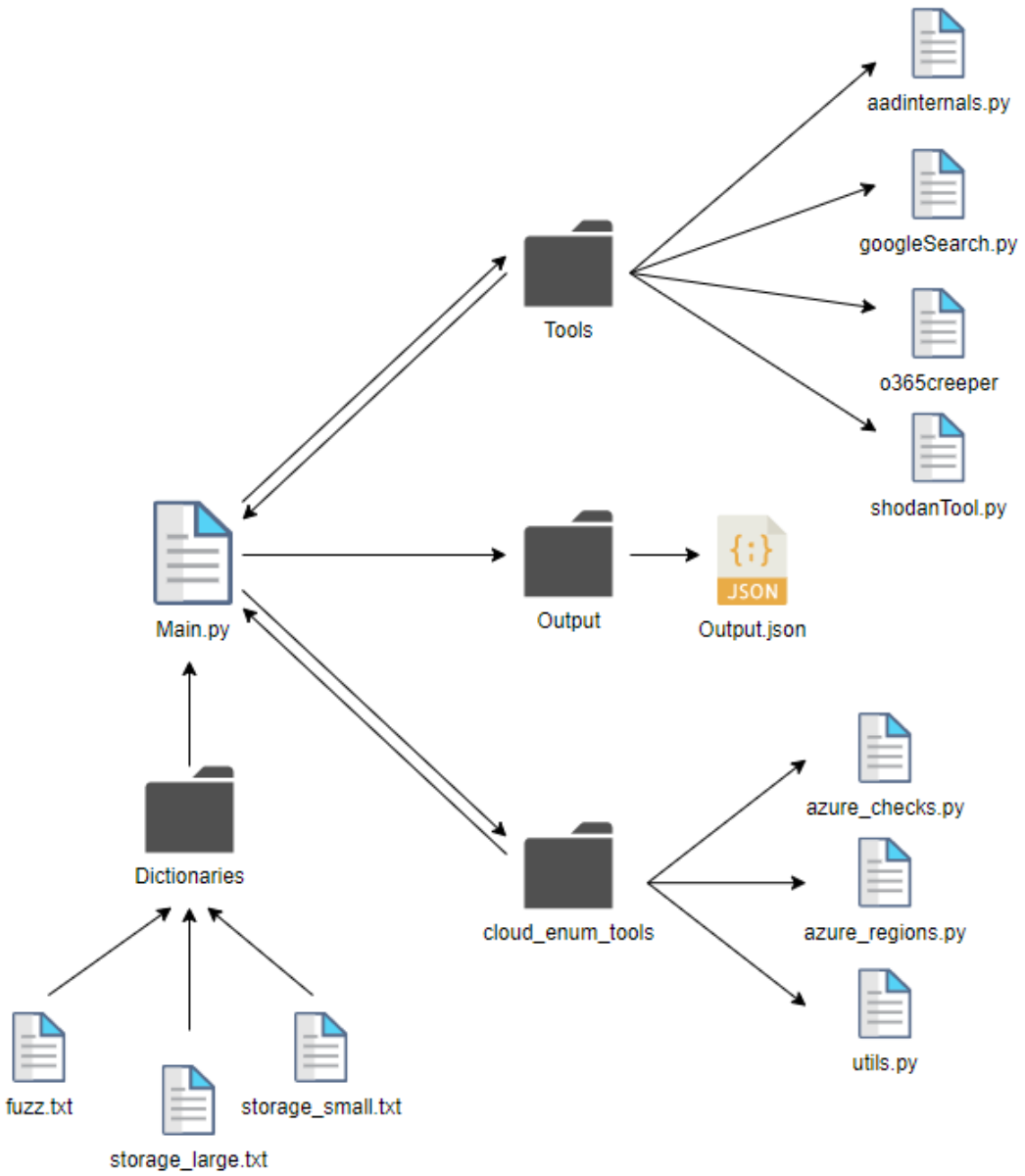

Figura 4.1: Estructura de la herramienta

4.3. Instalación de la herramienta

Para que la herramienta funcione correctamente es necesarios seguir los siguientes pasos:

- **Instalación de Python3**: El programa está desarrollado en Python3 por lo que es necesario que dicho lenguaje se encuentre instalado en el sistema.
- **Instalación de Powershell**: En caso de desplegar la herramienta en un sistema operativo distinto de Windows es necesario instalar Powershell.
- **Bibliotecas de Python3**: Es necesaria la instalación de la bibliotecas listadas anteriormente en caso de no encontrarse en el sistema. Para esto se ha creado un archivo *requeriments.txt* que agrupa todas estas.

```
pip3 install -r requeriments.txt
```

- **Módulos de Powershell**: Para poder tener acceso a las funciones de AADInternals es necesario instalar el módulo correspondiente en Powershell.

```
Install-Module AADInternals
```

- **API de Google**: Para poder realizar las búsquedas en el motor de Google [26], es necesario instalar la API, obtener una API key y crear un Custom Search Engine (CSE). Es posible obtener dicha API key y crear un CSE de forma gratuita, con la limitación de 100 búsquedas diarias. Ambas claves deben escribirse en el archivo *main.py*, al comienzo del método *main*, en las variables `googleAPI` y `cse`. Para instalar la API basta con introducir los comandos mostrados a continuación. En caso de Windows:

```
pip install virtualenv
virtualenv <your-env>
<your-env>\Scripts\activate
<your-env>\Scripts\pip.exe install google-api-python-client
```

En caso de Linux:

```
pip install virtualenv
virtualenv <your-env>
source <your-env>/bin/activate
<your-env>/bin/pip install google-api-python-client
```

- **API de Shodan**: La utilización de está API es gratuita y para obtenerla solo es necesario registrarse en la página web de Shodan [27]. Una vez conseguida la clave, debe introducirse en el archivo *main.py*, al comienzo del método *main*, en la variable `shodanAPI`.

4.4. Funcionamiento de la herramienta

La herramienta está desarrollada para ser usada desde una terminal, y, por lo tanto, la selección de opciones e introducción de parámetros se hace desde esta. Los argumentos de los que el programa dispone son los siguientes:

- **-n, –name**: URL de la compañía sobre la que se quiere obtener la información. Introducir este argumento es obligatorio.
- **-m, –mutations**: Ruta del archivo con las palabras que van a ser utilizadas para realizar las mutaciones. Por defecto utiliza el archivo *fuzz.txt* en la carpeta de diccionarios.
- **-b, –brute**: Ruta del archivo con las palabras que van a ser utilizadas para realizar fuerza bruta sobre los containers. Por defecto utiliza el archivo *fuzz.txt* en la carpeta de diccionarios.
- **-t, –threads**: Hilos utilizados por el módulo de Cloud_Enum en la fuerza bruta de HTTP. Por defecto se utilizan 5 hilos.
- **-ns, –nameserver**: Servidor DNS utilizado en la fuerza bruta. Por defecto es utilizado el 8.8.8.8.
- **-qs, –quickscan**: Deshabilita el escaneo de containers.
- **-qm, –quickmutations**: Deshabilita la función de generar más mutaciones con palabras clave encontradas en la investigación del Azure AD.
- **-dl, –dorkslist**: Ruta del archivo que contiene las palabras clave que van a ser utilizadas en las búsquedas mediante dorks. Por defecto utiliza el archivo *dorkingWordsSimple.txt* de la carpeta de diccionarios.
- **-e, –emails**: Ruta del archivo que contiene a los usuarios los cuales van a ser validados.

4.5. Evaluación de la herramienta

Con el objetivo de probar la eficacia de la herramienta y su correcto funcionamiento, se ejecutará sobre múltiples dominios de organizaciones activas las cuales hacen uso de Microsoft Azure. Debido a la cantidad de información que la herramienta recopila, es posible que no se pueda obtener datos en todos los campos sobre una misma empresa. Por esta razón, se ejecutará en distintas organizaciones hasta obtener información en todos los ámbitos, mostrando todo su potencial.

Debido a que el programa obtiene información de fuentes abiertas, está puede utilizarse en empresas reales, sin la necesidad de ser ejecuta en entornos controlados. Esto permite obtener una visión mucho más realista de los obtenible con la herramienta.

Capítulo 5

Desarrollo

Sometimes I'll start a sentence and I don't even know where it's going. I just hope I find it along the way.

Michael Scott

5.1. Introducción

Como ya se ha comentado, la herramienta se ha desarrollado de manera modular. Esto es debido a la necesidad de adaptabilidad ante nuevas actualizaciones y de implementación o eliminación de módulos en un futuro. Por esta razón, en este apartado se explicará el funcionamiento de dichos módulos y como se encuentran integrados en el programa.

5.2. Cloud_Enum

Esta herramienta realiza una de las funciones más importantes del programa, ya que nos permite obtener información sobre Azure Blob Storage, Web Apps, SQL Databases y Virtual Machines. Está basada en la generación de palabras mediante mutaciones gracias a la utilización de diccionarios. Una vez generadas las palabras, realiza fuerza bruta probando las posibles direcciones y procesando el resultado.

Estas mutaciones son generadas con el objetivo de encontrar Accounts, las cuales son siempre concatenadas con el mismo patrón en función del tipo de almacenamiento o servicio. Estos patrones son los siguientes, siendo la cadena *mutation* la que cambia en función de las mutaciones (además de la cadena *region* de las VMs):

```
#Blob Storage and Containers
{mutation}.blob.core.windows.net
{mutation1}.blob.core.windows.net/{mutation2}

#Web App
{mutation}.azurewebsites.net

#Database
{mutation}.database.windows.net
```

```
#Virtual Machine
{mutation}.{region}.cloudapp.azure.com
```

Para realizar la integración de esta herramienta en el programa y que pueda interactuar correctamente con el resto de módulos, se han realizado las siguientes modificaciones:

- **Eliminación de módulos innecesarios**: Ya que la herramienta Cloud_Enum cuenta con funcionalidades similares para otros entornos de nube como AWS o Google Cloud, todo lo relacionado con estos ha sido eliminado, limitando sus funcionalidad exclusivamente a Azure.
- **Modificación de parámetros**: Se ha modificado la forma en la que la herramienta es llamada para que actúe como un módulo, eliminando su método main e integrándola al programa principal.
- **Modificación de métodos**: Debido a que la herramienta ofrece los resultados a través de la consola o mediante la escritura de ficheros, se han modificado los métodos necesarios para que dichos resultados sean devueltos a través de estructuras como listas y diccionarios.

Como resultado de estas modificaciones, la herramienta Cloud_Enum se reduce a tres ficheros, los cuales son azure_checks.py, azure_regions.py y utils.py.

5.2.1. azure_regions.py

Es un archivo de configuración encargado de rastrear las regiones DNS para los recursos de Azure. Este archivo puede modificarse con la intención de eliminar aquellas regiones que no son útiles para el usuario. Las configuradas actualmente son *eastus* y *westeurope*. Esto afecta exclusivamente en la búsqueda de máquinas virtuales.

5.2.2. utils.py

El fichero utils.py contiene las funciones básicas que permiten a la herramienta funcionar correctamente. Entre sus métodos encontramos escritura de logs, procesamiento de URLs y captura de sus respuestas, resolución DNS, formateo de resultados, o manejo de temporizadores. El objetivo de este fichero es funcionar como apoyo para azure_checks.py.

5.2.3. azure_checks.py

Este archivo contiene los métodos encargados de realizar las búsquedas en los diferentes servicios. Hace uso de funciones pertenecientes al archivo utils.py. Cada servicio tiene asociado un método encargado de realizar la búsqueda, el cual depende de la lista de mutaciones que hayan sido formadas, la cantidad de hilos escogidos por el usuario y el servidor DNS seleccionado. Todas estas funciones son invocadas en el método `run_all`, el cual es llamado desde el main del programa.

5.3. AADInternals

Este módulo se divide en dos funciones. Estas son la obtención de información del Azure AD de la organización y la comprobación de la existencia de los correos de usuarios en Azure AD. Ambas requieren del uso de Powershell, ya que la herramienta de AADInternals está programada en este lenguaje.

5.3.1. Obtención de información de Azure AD

Mediante la invocación del módulo `AADIntReconAsOutsider` podemos obtener información sobre el Azure AD de la organización objetivo. Al necesitar ser ejecutado desde Powershell, toda la información interesante es recogida mediante expresiones regulares utilizadas en la salida del script. Los datos recogidos son los siguiente:

- **Tenant brand**: Nombre del Tenant de la organización.
- **Tenant ID**: Identificador de Tenant de la organización. Es un identificador único global diferente al nombre del Tenant o el dominio.
- **DesktopSSO enabled**: Se corresponde con la habilitación del mencionado anteriormente Seamless Single Sing On (SSO). En caso de estar habilitado permite la comprobación de los correos en el Azure AD.
- **Domains**: Listado de todos los dominios verificados y sus tipos de identidades en el Tenant objetivo.
- **Federated Domains**: Listado de los dominios federados de la organización.

Una vez recogida toda esta información, de nuevo, mediante el uso de expresiones regulares, se recopilan palabras claves las cuales pueden ser utilizadas para generar nuevas mutaciones que utilizar en la fuerza bruta de direcciones DNS.

```
for i in listDomains:
        dots = i.count(".")
        if dots == 1:
                aux = re.findall("(.*?)\..*$",i)
                if(len(aux[0]) > 3):
                        names.append(aux[0])
        elif dots == 3 and "onmicrosoft" not in i:
                aux = re.findall(".*\.(.*?)\..*\..*$",i)
                if(len(aux[0]) > 3):
                        names.append(aux[0])
        elif dots == 2 and "onmicrosoft" not in i:
                aux = re.findall("(.*?)\.[\w]{2,3}\.[\w]{2,3}$",i)
                if len(aux) != 0 and len(aux[0]) > 3:
                        names.append(aux[0]))
```

Tras finalizar este proceso, toda la información es organizada y devuelta en una estructura de diccionario.

5.3.2. azure_checks.py

Gracias a la invocación del módulo `AADIntUserEnumerationAsOutsider` es posible comprobar si el correo de un usuario está registrado en el Azure AD. Al igual que en el caso anterior, este script debe ser ejecutado desde Powershell, por lo que se debe filtrar la información obtenida con el uso de expresiones regulares.

Es importante recalcar que el correcto funcionamiento de este método depende de que el valor obtenido en el apartado *DesktopSSO enabled*, explicado anteriormente, sea *True*. En caso de no serlo, la información obtenida no será válida.

Como resultado de la ejecución, junto con cada correo obtendremos una respuesta confirmando o negando si pertenece al Azure AD. Los que hayan sido validados son recogidos y devueltos en una estructura de lista.

5.4. GoogleSearch

El módulo de GoogleSearch nos permite realizar una búsqueda en el motor de Google con la utilización de una API key y un Custom Search Engine (CSE). Esto nos abre un gran abanico de posibilidades ya que habilita el uso de dorks con los que encontrar información.

```
def google_search(search_term, api, cse, **kwargs):

#Using the Google API to search using dorks

    service = build("customsearch", "v1", developerKey=api)
    res = service.cse().list(q=search_term, cx=cse, **kwargs).execute()
    return res
```

Las búsquedas realizadas mediante este método son las siguientes:

- **Archivos con diferentes extensiones que contengan palabras clave en un Blob Storage**: Para realizar esta búsqueda se utilizarán las direcciones verificadas de Blobs obtenidas por Cloud_Enum. En estas se tratará de encontrar archivos con extensiones pdf, docx, xlsx, doc, xls, pptx y ppt que contengan las palabras claves del archivo introducido por el argumento *dorkslist*. A continuación, se puede ver un ejemplo de cómo sería una búsqueda:

```
site:empresa.blob.core.windows.net "password" ext:pdf | ext:docx | ext:
    xlsx | ext:doc | ext:xls | ext:pptx | ext:ppt
```

- **Archivos que contengan palabras clave en un File Storage**: Utilizando de nuevo los resultados de Cloud_Enum, se buscarán archivos que contengan las palabras clave del archivo introducido por el argumento *dorkslist* en direcciones de File Storage. A continuación, se puede ver un ejemplo de cómo sería una búsqueda:

```
site:empresa.file.core.windows.net "password"
```

- **Bases de datos encontradas en pastebin**: Partiendo de las bases de datos validadas por Cloud_Enum, se tratará de encontrar filtraciones en la página de pastebin.com.

```
empresa.database.windows.net site:pastebin.com
```

- **Información sensible en páginas de código**: Con esta búsqueda se trata de encontrar filtraciones de *connection strings* en la página de github.com. En caso de obtener esta clave, mediante la utilización de Azure CLI se puede obtener acceso a una cuenta de Storage, permitiendo listar y descargar todo su contenido [28].

```
site:github.com web.config "DefaultEndpointsProtocol" empresa
site:github.com web.config "StorageConnectionString" empresa
```

Como resultado de estas búsquedas se obtiene una gran cantidad de información, de la que son filtradas las URLs. Estas son recogidas y devueltas en forma de lista.

5.5. O365creeper

Este script permite validar cuentas de correo que pertenezcan al Tenant de Office 365. Para esto, realiza una petición a la URL especifica de Microsoft en cuyo cuerpo contiene el nombre del usuario a validar. En la respuesta a dicha petición se encuentra si el usuario existe o no.

Con el objetivo de integrar esta herramienta, se han realizado una serie de modificaciones, entre las cuales se encuentran: la eliminación de los posibles argumentos con los que funcionaba la herramienta, la reducción de formas de introducir los correos a exclusivamente mediante un fichero y la implementación de una forma de recoger los resultados en una estructura de lista. Tras todos estos cambios, el resultado es el siguiente:

```
def o365emails(inputfile):

    #Validate email accounts that belong to Office 365 tenants

    url = 'https://login.microsoftonline.com/common/GetCredentialType'
    result = []
    with open(inputfile) as file:
        for line in file:
            s = req.session()
            line = line.split()
            email = ' '.join(line)
            body = '{"Username":"'+ email +'"}'
            request = req.post(url, data=body)
            response = request.text
            valid = re.search('"IfExistsResult":0,', response)
            invalid = re.search('"IfExistsResult":1,', response)
            if invalid:
                print(email +' - INVALID')
            elif valid:
                result.append(email)
                print(email +' - VALID')

    return result
```

Este método es utilizado como alternativa a AADInternals, ya que su uso es posible independientemente de si el valor *DesktopSSO enabled* es el adecuado o no. Aun así, el empleo de esta función es menos fiable que la de AADInternals, ya que puede generar una cantidad mayor de falsos positivos.

Como resultado de la ejecución devuelve una lista con los usuarios que han sido validados.

5.6. ShodanTool

El método de Shodan se ha desarrollado con el objetivo de obtener información adicional sobre lo encontrado con la herramienta *Cloud_Enum*. Este motor permite la búsqueda de equipos conectados a internet a través de filtros, la recopilación información sobre los servicios habilitados, la obtención de banners con metadatos que el equipo envía como respuesta, e incluso, la adquisición de información sobre el software instalado.

Para funcionar, es necesario proveer una API key, la cual se puede conseguir de forma gratuita registrándose en la página oficial de Shodan.

```
def dataShodan(ip,shodan_api):

    #This method uses the Shodan API to obtain information about the
        provided IP

    api = Shodan(shodan_api)
    output = {}
    try:
        host = api.host(ip)
        hostIP = ip
        organization = host.get('org', 'n/a')
        vulnerabilities = host.get('vulns')

        hostPorts = {}
        for item in host['data']:
            if "product" in item:
                producto = item["product"]
            else:
                producto = item["_shodan"]["module"]
            hostPorts[item['port']] = {"Product":producto,"Data":item['data
                ']}
        output = {"IP":hostIP,"Organization":organization,"Vulnerabilities"
            :vulnerabilities,"Ports":hostPorts}
    except:
        print('Error IP:' + ip)

    return output
```

La información que obtenemos gracias a esta herramienta una vez filtrada es la siguiente:

- **IP**: Dirección IP correspondiente al equipo objetivo.
- **Organization**: Organización a la que pertenece el equipo objetivo.
- **Vulnerabilities**: Vulnerabilidades del equipo objetivo. Recopila los CVEs a los que el sistema es vulnerable y los lista.
- **Ports**: Listado de puertos abiertos en el equipo objetivo. Sobre cada uno de estos puertos se obtiene el servicio que este presta y la respuesta que emite al realizar una petición.

Gracias a este método podemos obtener información muy interesante de cara a un penetration testing, como son vulnerabilidades en los sistemas, versiones de software en funcionamiento o puertos expuestos a internet. Tener acceso a estos datos puede ayudar a la hora de realizar un modelado de amenazas en el que definir la estrategia a seguir.

Toda la información obtenida con el motor de búsqueda es devuelta en forma de diccionario.

5.7. Main program

El main es el cuerpo principal del programa en el que se integran y cooperan todos los módulos comentados anteriormente. Además de esto, este fichero también cuenta con el desarrollo de una serie de métodos que sirven tanto para obtener información relevante, como para que el programa funcione correctamente.

5.7.1. Mutaciones

Para el correcto funcionamiento de la herramienta *Cloud_Enum* es necesario la utilización de tres métodos con los que generar las mutaciones. Estas funciones se encargan de leer las palabras de diccionario seleccionado por el usuario, eliminar los caracteres que no son válidos para formar las direcciones, y combinar las mutaciones resultantes con las palabras clave, formando las cuentas de almacenamiento que serán probadas en la fuerza bruta de DNS.

5.7.2. Identificación de Azure AD/Office 365

Con este método es posible identificar si una organización utiliza Azure AD, Office 365 y el servicio de correo de Microsoft. Para comprobar está información es necesario utilizar las siguientes URLs sustituyendo *domain.com* por el dominio de la organización el cual quiere ser identificado.

```
#Azure AD
https://login.microsoftonline.com/getuserrealm.srf?login=username@domain.
    com.onmicrosoft.com&xml=1

#Office 365
https://login.microsoftonline.com/getuserrealm.srf?login=test@domain.com&
    xml=1

#Office 365 email
https://outlook.office365.com/autodiscover/autodiscover.json/v1.0/
    test@domain.com?Protocol=Autodiscoverv1
```

Como resultado de la ejecución, con respecto a Azure AD, a través del campo `NameSpaceType` podemos comprobar si una organización hace uso de esta tecnología si el valor encontrado es `Managed`.

Con la segunda URL, es posible conocer si la organización utiliza Office 365, además de obtener cierta información sobre sus dominios federados. Los datos relevantes recogidos de la respuesta son `NameSpaceType`, `AuthURL`, `STSAuthURL` y `MEXURL`.

Respecto a la última URL, en caso de devolver información, indica que la organización está haciendo uso del servicio de correos. Sobre esta petición se recoge toda la respuesta, la cual muestra un intento fallido de comprobar si el usuario `test@domain.com` existe.

5.7.3. Identificación de AD FS

Esta función permite obtener información sobre los dominios federados de una organización. Para esto son usadas las URL mostradas a continuación en las que se debe sustituir *federated.com* por el dominio federado deseado.

```
#Get information about the technologies used
https://federated.com/adfs/ls/idpinitiatedsignon.aspx

#Endpoints to bypass blocking policies
https://federated.com/adfs/services/trust/2005/windowstransport
https://federated.com/adfs/services/trust/13/windowstransport
```

Con la primera de las URL, se puede acceder a un portal de inicio de sesión en el que pueden verse ciertas tecnologías y productos utilizados por la organización. Esto puede servir tanto para conocer el uso de estas opciones, o como punto de acceso sobre el que ejecutar un ataque de fuerza bruta o password spraying.

Las dos siguientes URLs corresponden a endpoints los cuales pueden servir para evadir políticas de bloqueo. También son útiles como vectores de ataque mediante la ejecución, de nuevo, de un ataque de fuerza bruta o password spraying.

5.7.4. Otras utilidades

Además de todos los métodos explicados anteriormente, son necesarias una serie de funciones básicas para la ejecución del programa. Estas son:

- **Lectura y escritura de ficheros**: Esta utilidad es requerida tanto para la lectura de los ficheros introducidos mediante argumentos por el usuario, como para la escritura del fichero con la estructura JSON.

- **Resolución de dominios**: Con este método es posible obtener la IP de un dominio. Es necesario ya que algunas herramientas requieren de esta resolución para funcionar.

- **Comprobación de estados**: Esta función realiza peticiones HTTP y HTTPS con el objetivo de determinar el código de respuesta de dicha página.

- **Gestión de parámetros**: Gracias a este método es posible manejar los distintos parámetros introducidos por el usuario de forma sencilla.

5.8. Flujo de ejecución

A continuación, se muestra un diagrama de flujo sobre la ejecución del programa y sus pasos.

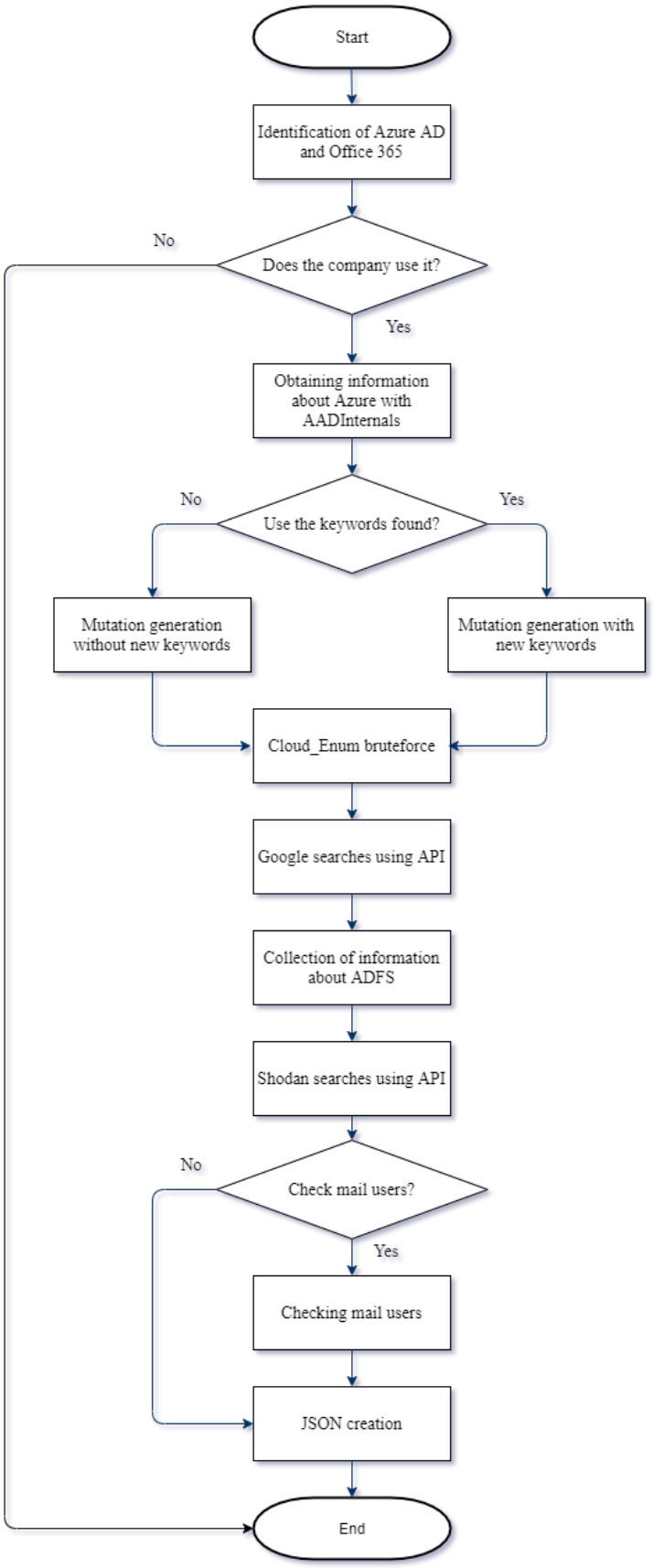

Figura 5.1: Execution Flow

Capítulo 6

Resultados

The advance of technology is based on making it fit in so that you don't really even notice it, so it's part of everyday life.

Bill Gates

6.1. Introducción

En este apartado se mostrarán algunos ejemplos de resultados al ejecutar la herramienta sobre organizaciones reales. Se tratará de cubrir todos los apartados de la herramienta e ilustrar su correcto funcionamiento. Por último, se realizará una comparativa con las herramientas ya existentes.

6.2. Resultados de la herramienta

Como resultado de la ejecución de la herramienta es devuelto un JSON cuya estructura será explicada por apartados, donde se mostrará la información recopilada y ejemplos reales. Además, se incluirán varios ejemplos de ejecuciones completadas en los anexos del documento.

6.2.1. Azure/O365 Identification

Esta sección contiene información sobre la identificación del uso de Azure, AD FS, Office 365 y correo de Microsoft.

```
"Azure/O365 Identification": {
  "Azure AD Check": {
    "URL": "https://login.microsoftonline.com/getuserrealm.srf?login=username@uah.es
      .onmicrosoft.com&xml=1",
    "Status": 200,
    "NameSpaceType": "Managed"
  },
  "Office 365 Check": {
    "URL": "https://login.microsoftonline.com/getuserrealm.srf?login=test@uah.es&xml=1",
    "Status": 200,
    "NameSpaceType": "Federated",
    "AuthURL": "https://fs.uah.es/adfs/ls/?username=test%40uah.es&amp;wa=wsignin1.0&amp
      ;wtrealm=urn%3afederation%3aMicrosoftOnline&amp;wctx=",
    "STSAuthURL": "https://fs.uah.es/adfs/services/trust/2005/usernamemixed",
    "MEXURL": "https://fs.uah.es/adfs/services/trust/mex"
  },
  "Office 365 Email Check": {
    "URL": "https://outlook.office365.com/autodiscover/autodiscover.json/v1.0/test@uah.es
      ?Protocol=Autodiscoverv1",
    "Status": 404,
    "Body": "{\"ErrorCode\":\"UserNotFound\",\"ErrorMessage\":\"The given user was not
      found\"}"
  }
```


Figura 6.1: Azure/O365 Identification

Como se puede apreciar en la Figura 6.1, con la primera consulta se obtiene información sobre la utilización de Azure AD. Si el parámetro `NameSpaceType` contiene *Managed* significa que la organización utiliza Azure.

Con la segunda búsqueda, podemos saber si la organización utiliza Office 365 en caso de dar resultado. Además, nos arroja información sobre el uso de AD FS en caso de que el parámetro `NameSpaceType` contenga *Federated*. Si hace uso de este, obtenemos los demás campos con información posiblemente relevante relacionada con el AD FS.

Por último, con la tercera URL, podemos comprobar si la organización utiliza el correo de Microsoft. Si se obtiene una respuesta de error sobre que el usuario no ha sido encontrado, significa que se hace uso de este servicio.

6.2.2. AAD Data

En este apartado encontramos información sobre el Tenant de la organización, sus dominios verificados y dominios federados.

```
"AAD Data": {
  "Tenant Brand": "Universidad de Alcala",
  "Tenant Id": "ced2c552-7d1f-4731-aa3a-2f0ec9629e26",
  "DesktopSSO enable": "False",
  "Verified Domains": [
    "alu.uah.es",
    "alumni.uah.es",
    "aut.uah.es",
    "edu.uah.es",
    "fgua.es",
    "o365.uah.es",
    "one.uah.es",
    "pas.uah.es",
    "senmes.es",
    "spaceweather.es",
    "uah.es",
    "universidaddealcala.mail.onmicrosoft.com",
    "universidaddealcala.onmicrosoft.com"
  ],
  "FQDN Domains": [
    "fs.uah.es"
  ]
```


Figura 6.2: AAD Data

6.2.3. Azure Storage Accounts

En esta sección se encuentra información sobre los servicios de Azure Blob Storage utilizados por la organización.


```
"Azure Storage Accounts": {
  "alumnibackup.blob.core.windows.net": {
    "IP": "20.60.19.36",
    "Organization": "Microsoft Corporation",
    "Vulnerabilities": null,
    "Ports": {
      "80": {
        "Service": "http",
        "Data": "HTTP/1.1 400 The requested URI does not represent any resource on the
          server.\r\nContent-Length: 244\r\nContent-Type: application/xml\r\nServer: Windows
          -Azure-Blob/1.0 Microsoft-HTTPAPI/2.0\r\nx-ms-request-id: a1e95455-001e-003a-245c
          -ac3a2b000000\r\nDate: Sat, 18 Sep 2021 07:12:13 GMT\r\n\r\n"
      },
```


Figura 6.3: Azure Storage Accounts

Como se puede apreciar en la Figura 6.3, se puede ver la dirección del Blob, la IP, organización a la que pertenece el sistema, vulnerabilidades encontradas y todos los puertos abiertos y servicios que estos ofrecen. También se recoge la cabecera de la respuesta, ya que puede aportar información relevante como la versión del software.

6.2.4. Azure Blob Containers

Apartado que recoge información correspondiente a los contenedores y archivos pertenecientes a los Blobs anteriores.

```
"Azure Blob Containers": {
  "https://schultenstorage.blob.core.windows.net/storage/?restype=container&comp=list": [
    "https://schultenstorage.blob.core.windows.net/storage/ISO 90012015 Certificaat Schulten
      Opleidingen &amp;amp; Certificeringen bv.pdf",
    "https://schultenstorage.blob.core.windows.net/storage/ISO9001SchultenOC.pdf",
    "https://schultenstorage.blob.core.windows.net/storage/Privacyverklaring Schulten
      Opleidingen en Certificeringen BV V2.pdf",
    "https://schultenstorage.blob.core.windows.net/storage/ProtocolCOVID-19V1.pdf",
    "https://schultenstorage.blob.core.windows.net/storage
      /algemenevoorwaardenschultenopleidingencertificeringenbv.pdf"
  ]
},
```


Figura 6.4: Azure Blob Containers

En caso de encontrar un contenedor válido dentro de los Blobs encontrados anteriormente, es capaz de listar todos los archivos que este contenga.

6.2.5. Azure Websites

Esta sección recoge información sobre los servicios de Azure Web App utilizados por la organización.


```
"alumni-api.azurewebsites.net": {
  "Status": {
    "HTTP status": 200,
    "HTTPS status": 200
  },
  "Information": {
    "IP": "137.135.91.176",
    "Organization": "Microsoft Corp",
    "Vulnerabilities": null,
    "Ports": {
      "80": {
        "Service": "Microsoft IIS httpd",
        "Data": "HTTP/1.1 404 Site Not Found\r\nContent-Type: text/html\r\nServer:
          Microsoft-IIS/10.0\r\nDate: Fri, 17 Sep 2021 03:27:51 GMT\r\nConnection:
          close\r\nContent-Length: 2778\r\n\r\n"
      },
```


Figura 6.5: Azure Websites

La información recogida es la dirección de la web, la IP, organización a la que pertenece el sistema, vulnerabilidades encontradas y todos los puertos abiertos y servicios que estos ofrecen. También se recoge la cabecera de la respuesta, ya que puede aportar información relevante como la versión del software. Además, se comprueba el acceso a estas y se muestra mediante el estado de la respuesta.

6.2.6. Azure Databases

Apartado encargado de almacenar la información sobre los servicios de Azure Data Bases utilizados por la organización.

```
"staging-alumni.database.windows.net": {
  "IP": "52.236.184.163",
  "Organization": "Microsoft Corporation",
  "Vulnerabilities": null,
  "Ports": {
    "443": {
      "Service": "https",
      "Data": "HTTP/1.1 500 Internal Server Error\r\nTransfer-Encoding: chunked\r\nContent
        -Type: application/json; charset=utf-8\r\nServer: Microsoft-HTTPAPI/2.0\r\nDate:
        Mon, 13 Sep 2021 13:20:02 GMT\r\n\r\n"
    },
```


Figura 6.6: Azure Databases

La información recogida es la dirección de la base de datos, la IP, organización a la que pertenece el sistema, vulnerabilidades encontradas y todos los puertos abiertos y servicios que estos ofrecen. También se recoge la cabecera de la respuesta.

6.2.7. Azure Virtual Machines

Sección encargada de almacenar la información sobre los servicios de Azure Virtual Machines utilizados por la organización.


```
"Azure Virtual Machines": {
  "vodafone.westeurope.cloudapp.azure.com": {
    "IP": "51.137.100.85",
    "Organization": "Microsoft Limited UK",
    "Vulnerabilities": null,
    "Ports": {
      "80": {
        "Service": "Microsoft IIS httpd",
        "Data": "HTTP/1.1 200 OK\r\nContent-Type: text/html\r\nLast-Modified: Wed, 10 Jan
          2018 11:39:11 GMT\r\nAccept-Ranges: bytes\r\nETag: \"cb2817a278ad31:0\"\r\nServer:
          Microsoft-IIS/10.0\r\nX-Powered-By: ASP.NET\r\nDate: Sat, 11 Sep 2021 06:48:42
          GMT\r\nContent-Length: 703\r\n\r\n"
      },
```


Figura 6.7: Azure Virtual Machines

La información recogida es la dirección de la máquina virtual, la IP, organización a la que pertenece el sistema, vulnerabilidades encontradas y todos los puertos abiertos y servicios que estos ofrecen. También se recoge la cabecera de la respuesta.

6.2.8. Active Directory Federation Services

En este apartado se recoge información sobre el AD FS y los demás dominios federados que utilice la organización.

```
"Active Directory Federation Services": {
  "sts.gruposantander.de": {
    "https://sts.gruposantander.de/adfs/ls/idpinitiatedsignon.aspx": {
      "Status": 200,
      "Options": [
        "ClaimsXray"
      ]
    },
    "https://sts.gruposantander.de/adfs/services/trust/2005/windowstransport": {
      "Status": 503
    },
    "https://sts.gruposantander.de/adfs/services/trust/13/windowstransport": {
      "Status": 503
    }
  },
```


Figura 6.8: Active Directory Federation Services

En la primera URL se pueden observar ciertas tecnologías utilizadas por la organización, mientras que la segunda y tercera corresponden a endpoints los cuales pueden servir para evadir políticas de bloqueo. Estas direcciones pueden contener portales de inicio de sesión, los cuales pueden servir como punto de entrada en una posible auditoría.

6.2.9. Google Searchs

Información obtenida de las múltiples búsquedas en el motor de Google con el uso de dorks y la información ya recopilada.


```
"Google Searchs": {
  "Blob Storage": {
    "site:damrexelprod.blob.core.windows.net \"password\"  ext:pdf | ext:docx | ext:xlsx |
      ext:doc | ext:xls | ext:pptx | ext:ppt": [
      "https://damrexelprod.blob.core.windows.net/medias/4fe1c896-bde4-4d7f-b9c1
        -14d7cc954a6a",
      "https://damrexelprod.blob.core.windows.net/medias/a656f78c-de1e-4393-8319
        -d278d03d6d36",
      "https://damrexelprod.blob.core.windows.net/medias/52a62510-fcfa-4909-ae3c
        -b372b30c7c14",
      "https://damrexelprod.blob.core.windows.net/medias/fee4cccf-1cae-4fe0-8628
        -4c4ba2746f5d",
      "https://damrexelprod.blob.core.windows.net/medias/9f45ca34-e418-4027-9f31
        -41d0d4b715f9",
      "https://damrexelprod.blob.core.windows.net/medias/3daeabd5-4a53-44e3-baf9
        -309c30158346",
      "https://damrexelprod.blob.core.windows.net/medias/492eaa20-091b-485a-9a4d
        -d352beceff57",
      "https://damrexelprod.blob.core.windows.net/medias/145013ea-a3b4-412a-af44
        -46688ce22858",
      "https://damrexelprod.blob.core.windows.net/medias/8af0f65f-5b0e-4cb0-8536
        -2899f6d05ff0",
      "https://damrexelprod.blob.core.windows.net/medias/3e281f2e-1679-4111-a786
        -e12ed9f755cf"
    ]
  },
  "Files Storage": {
    "site:damrexelprod.file.core.windows.net  \"password\"": []
  },
  "Databases Storage": {},
  "Sensitive Information": {
    "site:github.com web.config \"DefaultEndpointsProtocol\" \"damrexelprod\"": [],
    "site:github.com web.config \"StorageConnectionString\" \"damrexelprod\"": []
  }
```


Figura 6.9: Google Searchs

Con estas búsquedas se trata de encontrar información sensible como usuarios y contraseñas filtradas, bases de datos o datos relevantes sobre configuraciones.

6.2.10. Valid Emails

En esta lista se recogen los correos de usuario que han sido validados en caso de haberlos introducido.

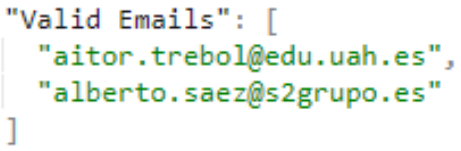

Figura 6.10: Valid Emails

6.3. Comparativa con otras herramientas

Actualmente, las herramientas que podemos encontrar en internet, sobre las que ya se ha hablado a lo largo del trabajo, funcionan de forma correcta y ayudan a obtener información importante sobre los servicios cloud de una organización. Sin embargo, estas herramientas cuentan con funcionalidades u objetivos limitados, cubriendo un único ámbito o aspecto dentro de toda la plataforma. El programa desarrollado es capaz de abarcar un terreno mucho más amplio, obteniendo información sobre muchos más servicios de la plataforma. Además, el proyecto no solo consta de la implementación de las herramientas existentes, sino de su integración y trabajo conjunto, pudiendo obtener una información más avanzada sobre cada uno de estos servicios de lo que podrían obtener las herramientas mediante su ejecución aislada.

Por último, en el programa desarrollado también han sido implementados nuevos métodos no recogidos en herramientas ya desarrolladas, lo que permite obtener una información mucho más completa sobre servicios tanto contemplados, como aun no explorados.

Capítulo 7

Conclusiones y líneas futuras

En este apartado se resumen las conclusiones obtenidas y se proponen futuras líneas de investigación que se deriven del trabajo.

7.1. Conclusiones

Con este proyecto se ha resuelto el objetivo establecido, elaborar una herramienta completa y automática para obtener información de fuentes abiertas sobre la utilización de Microsoft Azure en una organización.

Se ha concluido que la gran mayoría de ataques realizados a sistemas de Microsoft Azure, ignorando vulnerabilidades 0-day, parten del robo de credenciales o información sensible con la que acceder a estos. Esta información es obtenible principalmente mediante filtraciones de datos o ataques de ingeniería social.

Reunir la información con la que realizar estos ataques es cada vez más complicado debido al rápido crecimiento de las tecnologías y la cantidad masiva de información que manejan hoy en día las organizaciones. Por esta razón, es esencial desarrollar métodos con los que poder recopilar esta de manera rápida y efectiva antes de plantear los posibles vectores de ataque en una auditoría.

7.2. Líneas futuras

Durante el desarrollo de esta herramienta se han planteado las siguientes ideas posibles a implementar en un futuro:

- **Abarcar otros proveedores de Cloud Computing**: Aunque este proyecto abarca exclusivamente Microsoft Azure, se plantea el desarrollo de otras herramientas o módulos con los que obtener información de otros proveedores como Amazon o Google. De esta forma, cubrir de la manera más eficiente posible y completa la recopilación de información sobre entornos de Cloud Computing.
- **Mantenimiento de la herramienta**: Debido al crecimiento constante de la plataforma Microsoft Azure, es necesario mantener las funciones que realiza la herramienta a lo largo de las actualizaciones y cambios que este proveedor implemente.
- **Implementación de dorks y motores distintos**: Es interesante investigar e implementar nuevas formas de obtener información sobre organizaciones y usuarios mediante la utilización de nuevos

dorks. Por otro lado, la herramienta utiliza exclusivamente el motor de búsqueda de Google. Esto da pie a la implementación de más motores como Bing, Baidu o Yandex, de los cuales obtener nueva información la cual puede no encontrarse indexada en Google.

- **Investigar nuevas formas de obtener información**: Es posible continuar con la investigación de nuevos métodos con los cuales obtener información relevante relacionada con Microsoft Azure que permita vulnerar la seguridad de una organización. Esto abarca tanto formas de obtener información que no hayan sido todavía recogidas, como herramientas ya desarrolladas o implementadas tras la realización de este proyecto.
- **Añadir distintos outputs**: Por ejemplo, añadir nuevos formatos de salida además de JSON, como podría ser XML, o la creación de una aplicación web por la que mostrar los resultados obtenidos.
- **Dar un enfoque más ofensivo**: También es posible extender las funciones de la herramienta y abarcar métodos más agresivos los cuales permitan obtener nueva información relevante. Estos resultados podrían ser utilizados por la propia herramienta para tratar de vulnerar los sistemas de una organización.

Bibliografía

[1] "Amazon lidera un mercado global de la nube que abarca ya 100.000 millones de dólares," https://bigdatamagazine.es/amazon-lidera-un-mercado-global-de-la-nube-que-abarca-ya-100-000-millones-de-dolares.

[2] "Cloud computing - statistics on the use by enterprises," https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Cloud_computing_-_statistics_on_the_use_by_enterprises.

[3] "Los mejores proveedores en cloud computing del mercado," https://ayudaleyprotecciondatos.es/cloud-computing/proveedores/.

[4] "Cloudbrute," https://github.com/0xsha/CloudBrute.

[5] "cloudenum," https://github.com/initstring/cloud_enum.

[6] "Blobhunter," https://github.com/cyberark/blobhunter.

[7] "AAD Internals," https://github.com/Gerenios/AADInternals.

[8] "o365creeper," https://github.com/LMGsec/o365creeper.

[9] "MSOLSpray," https://github.com/dafthack/MSOLSpray.

[10] "MicroBurst," https://github.com/NetSPI/MicroBurst.

[11] "ROADtools," https://github.com/dirkjanm/ROADtools.

[12] "Tipos de Computación en la Nube – una Extensa Guía Sobre Soluciones y Tecnologías de la Nube en 2021," https://kinsta.com/es/blog/tipos-de-computacion-en-la-nube/.

[13] "Tipos de cloud computing," https://www.redhat.com/es/topics/cloud-computing/public-cloud-vs-private-cloud-and-hybrid-clouds.

[14] "Azure products," https://azure.microsoft.com/en-us/services/.

[15] "Introducción a los servicios principales de Azure Storage," https://docs.microsoft.com/es-es/azure/storage/common/storage-introduction.

[16] "Azure storage types," https://www.dremio.com/data-lake/adls/.

[17] "Máquinas virtuales Windows en Azure," https://docs.microsoft.com/es-es/azure/virtual-machines/windows/overview.

[18] "Azure SQL Database," https://azure.microsoft.com/es-es/products/azure-sql/database/#overview.

[19] "App Service overview," https://docs.microsoft.com/en-us/azure/app-service/overview.

[20] "Introducción a Office 365," https://aprendiendoexchange.com/introduccion-office-365.

[21] "¿Qué es Azure Active Directory?" https://docs.microsoft.com/es-es/azure/active-directory/fundamentals/active-directory-whatis.

[22] "Roles integrados de Azure AD," https://docs.microsoft.com/es-es/azure/active-directory/roles/permissions-reference.

[23] "Introducción a AD FS," https://docs.microsoft.com/es-es/windows-server/identity/ad-fs/ad-fs-overview.

[24] "¿Qué es la sincronización de hash de contraseñas con Azure AD?" https://docs.microsoft.com/es-es/azure/active-directory/hybrid/whatis-phs.

[25] "What is pass-through authentication?" https://oxfordcomputertraining.com/glossary/pass-through-authentication/.

[26] "Using Google Search API With Python," https://linuxhint.com/google_search_api_python/.

[27] "Shodan," https://www.shodan.io/.

[28] "Azure Pentest Book," https://pentestbook.six2dez.com/enumeration/cloud/azure.

Apéndice A

Código

A.1. Introducción

En el siguiente apéndice se puede ver el código desarrollado para el correcto funcionamiento de la herramienta dividido en sus distintos módulos.

A.2. main.py

```
1  from requests.api import request
2  from tools import aadinternals
3  from tools import shodanTool
4  from tools import googleSearch
5  from tools import o365creeper
6  from cloud_enum_tools import azure_checks
7  import requests
8  import argparse
9  import os
10 import sys
11 import re
12 import json
13 import socket
14
15
16 def parse_arguments():
17     """
18     Handles user-passed parameters
19     """
20     desc = "Azure OSINT Tool"
21     parser = argparse.ArgumentParser(description=desc)
22
23     # Grab the current dir of the script, for setting some defaults below
24     script_path = os.path.split(os.path.abspath(sys.argv[0]))[0]
25
```

```
26      parser.add_argument("-n", "--name", help="name of the company URL.
            example: github.com", type=str, required=True, action="store")
27      parser.add_argument("-e", "--emails", help="check if the emails of a
            file exist", type=str, required=False, action="store", default="null
            ")
28      parser.add_argument('-m', '--mutations', type=str, action='store',
            default= script_path + '/Diccionarios/fuzz.txt', help='Mutations.
            Default: Diccionarios/fuzz.txt')
29      parser.add_argument('-b', '--brute', type=str, action='store', default=
             script_path +'/Diccionarios/fuzz.txt', help='List to brute-force
            Azure container names. Default: Diccionarios/fuzz.txt')
30      parser.add_argument('-t', '--threads', type=int, action='store',
            default=5, help='Threads for HTTP brute-force. Default = 5')
31      parser.add_argument('-ns', '--nameserver', type=str, action='store',
            default='8.8.8.8', help='DNS server to use in brute-force.')
32      parser.add_argument('-qs', '--quickscan', action='store_true', help='
            Disable containers scans')
33      parser.add_argument('-qm', '--quickmutations', action='store_true',
            help='Disable all second-level mutations')
34      parser.add_argument('-dl', '--dorkslist', action='store',default=
            script_path +'/Diccionarios/dorkingWordsSimple.txt', help='Words
            used in google dorking')
35
36      args = parser.parse_args()
37      return args
38
39
40  def checkCompany(name):
41      """
42      This method checks if a company uses Azure AD/Office365.
43      If the NameSpaceType indicates "Managed", then the company is using
            Azure AD
44      """
45      #Azure AD
46      url1 = "https://login.microsoftonline.com/getuserrealm.srf?login=
            username@"+ name +".onmicrosoft.com&xml=1"
47      #Office 365
48      url2 = "https://login.microsoftonline.com/getuserrealm.srf?login=test@
            "+ name +"&xml=1"
49      #Office 365 email
50      url3 = "https://outlook.office365.com/autodiscover/autodiscover.json/v1
            .0/test@"+name+"?Protocol=Autodiscoverv1"
51
52      dictionary1 = {"URL":url1}
53      dictionary2 = {"URL":url2}
54      dictionary3 = {"URL":url3}
55
```

```
56      try:
57          r1 = requests.get(url1,timeout=3)
58          status1 = r1.status_code
59      except:
60          status1 = 408
61          print("Error in URL " + url1)
62
63      dictionary1["Status"] = status1
64      if (status1 == 200):
65          body1 = r1.text
66          nameSpace1 = re.findall("<NameSpaceType>(.*?)<\\/NameSpaceType>",
                body1)
67          dictionary1["NameSpaceType"] = nameSpace1[0]
68
69
70      try:
71          r2 = requests.get(url2,timeout=3)
72          status2 = r2.status_code
73      except:
74          status2 = 408
75          print("Error in URL " + url2)
76
77      dictionary2["Status"] = status2
78      if (status2 == 200):
79          body2 = r2.text
80          nameSpace2 = re.findall("<NameSpaceType>(.*?)<\\/NameSpaceType>",
                body2)
81          dictionary2["NameSpaceType"] = nameSpace2[0]
82          authURL = re.findall("<AuthURL>(.*?)<\\/AuthURL>",body2)
83          if len(authURL) != 0:
84              dictionary2["AuthURL"] = authURL[0]
85          stsauthURL = re.findall("<STSAuthURL>(.*?)<\\/STSAuthURL>",body2)
86          if len(stsauthURL) != 0:
87              dictionary2["STSAuthURL"] = stsauthURL[0]
88          mexURL = re.findall("<MEXURL>(.*?)<\\/MEXURL>",body2)
89          if len(mexURL) != 0:
90              dictionary2["MEXURL"] = mexURL[0]
91
92
93      try:
94          r3 = requests.get(url3,timeout=3)
95          dictionary3["Status"] = r3.status_code
96          dictionary3["Body"] = r3.text
97      except:
98          dictionary3["Status"] = 408
99          print("Error in URL " + url3)
100
```

```
101
102     result = {"Azure AD Check":dictionary1,"Office 365 Check":dictionary2,"
            Office 365 Email Check":dictionary3}
103     return result
104
105 def checkStatus(site):
106     """
107     Check the response of site with http and https
108     """
109     name_http = "http://" + site + "/"
110
111     try:
112         r = requests.get(name_http,timeout=3)
113         http_status = r.status_code
114     except:
115         http_status = 408
116     finally:
117         print(name_http + " " + str(http_status))
118
119     name_https = "https://" + site + "/"
120     try:
121         s = requests.get(name_https,timeout=3)
122         https_status = s.status_code
123     except:
124         https_status = 408
125     finally:
126         print(name_https + " " + str(https_status))
127
128     status = [[name_http,http_status],[name_https,https_status]]
129     return status
130
131
132 def read_mutations(mutations_file):
133     """
134     Read mutations file into memory for processing.
135     """
136     with open(mutations_file, encoding="utf8", errors="ignore") as infile:
137         mutations = infile.read().splitlines()
138
139     print("[+] Mutations list imported: {} items".format(len(mutations)))
140     return mutations
141
142
143 def clean_text(text):
144     """
145     Clean text to be RFC compliant for hostnames / DNS
146     """
```

```
147     banned_chars = re.compile('[^a-z0-9.-]')
148     text_lower = text.lower()
149     text_clean = banned_chars.sub('', text_lower)
150 
151     return text_clean
152 
153 
154 def build_names(base_list, mutations):
155     """
156     Combine base and mutations for processing by individual modules.
157     """
158     names = []
159 
160     for base in base_list:
161         # Clean base
162         base = clean_text(base)
163 
164         # First, include with no mutations
165         names.append(base)
166 
167         for mutation in mutations:
168             # Clean mutation
169             mutation = clean_text(mutation)
170 
171             # Then, do appends
172             names.append("{}{}".format(base, mutation))
173             names.append("{}.{}".format(base, mutation))
174             names.append("{}-{}".format(base, mutation))
175 
176             # Then, do prepends
177             names.append("{}{}".format(mutation, base))
178             names.append("{}.{}".format(mutation, base))
179             names.append("{}-{}".format(mutation, base))
180 
181     print("[+] Mutated results: {} items".format(len(names)))
182 
183     return names
184 
185 def readFile(file):
186     """
187     Read from file
188     """
189     lista = []
190     with open(file,"r+") as f:
191         lista = f.readlines()
192 
193     lista = [str(x).strip() for x in lista]
```

```
194
195     return lista
196
197 def writeFile(file,text):
198     """
199     Write to file
200     """
201     f = open(file,"w")
202     f.write(text)
203     f.close()
204
205 def checkFederated(federated):
206     """
207     Get information from federated domains.
208     """
209     #Get information about the technologies used
210     url1 = "https://"+federated+"/adfs/ls/idpinitiatedsignon.aspx"
211
212     #Endpoints to bypass blocking policies
213     url2 = "https://"+federated+"/adfs/services/trust/2005/windowstransport
        "
214     url3 = "https://"+federated+"/adfs/services/trust/13/windowstransport"
215
216     try:
217         request1 = requests.get(url1,timeout=3)
218         status1 = request1.status_code
219     except:
220         status1 = 408
221
222     try:
223         request2 = requests.get(url2,timeout=3)
224         status2 = request2.status_code
225     except:
226         status2 = 408
227
228     try:
229         request3 = requests.get(url3,timeout=3)
230         status3 = request3.status_code
231     except:
232         status3 = 408
233
234     if(status1 == 200):
235         body = request1.text
236         options = re.findall("<option value=.*?>(.*?)<\\/option>",body)
237     else:
238         options = []
239
```

```
240     data = {url1:{"Status":status1,"Options":options},url2:{"Status":
            status2},url3:{"Status":status3}}
241     return data
242
243 def resolveDomain(domain):
244     """
245     Get the IP of a domain
246     """
247     result = socket.gethostbyname(domain)
248     return result
249
250
251 if __name__ == "__main__":
252     #APIs
253     shodanAPI = ""
254     googleAPI = ""
255     cse = ""
256
257     #Parse Arguments
258     args = parse_arguments()
259
260     #Identifies if a company use Azure
261     identificationData = checkCompany(args.name)
262
263     if identificationData["Azure AD Check"]["NameSpaceType"] == "Managed":
264         #Obtain data from the Azure Active Directory
265         dataAAD = aadinternals.reconAsOutsider(args.name)
266         basicName = re.findall("[\w-]+",args.name)[0]
267
268         #Read mutations from file
269         mutations = read_mutations(args.mutations)
270
271         #Generate mutations
272         if args.quickmutations:
273             #Only with the company name
274             names = build_names([basicName], mutations)
275         else:
276             #Using all the relevants words obtained from AAD
277             auxList = dataAAD["listNames"]
278             auxList.append(basicName)
279             auxList = list(set(auxList))
280             names = build_names(auxList, mutations)
281
282         #Cloud_enum-----------------------------------------------
283         try:
284             dataEnum = azure_checks.run_all(names, args)
285         except KeyboardInterrupt:
```

```
286             print("Couldn't get storage Information")
287
288         # Google API-------------------------------------------------------
289         try:
290             fileTypes = ["pdf", "docx", "xlsx", "doc", "xls", "pptx", "ppt
                    "]
291             blobResultFiles={}
292             filesResultFiles={}
293             dbsResult = {}
294
295             for i in dataEnum["valid_accounts"]:
296                 typesString = " ext:"+fileTypes[0]
297                 for j in range(1,len(fileTypes)):
298                     typesString = typesString + " | ext:"+ fileTypes[j]
299                 wordlist = readFile(args.dorkslist)
300                 for k in wordlist:
301                     search = 'site:' + i +' "' + k + '" '+typesString
302                     res = googleSearch.google_search(search, googleAPI, cse
                            )
303                     blobResultFiles[search] = []
304                     if "items" in res:
305                         links = []
306                         for l in res["items"]:
307                             links.append(l["link"])
308                         blobResultFiles[search] = links
309
310                 for k in wordlist:
311                     accounts = i
312                     fileaccounts = accounts.replace("blob","file")
313                     search = 'site:' + fileaccounts + '  "'+ k + '"'
314                     res = googleSearch.google_search(search, googleAPI, cse
                            )
315                     filesResultFiles[search] = []
316                     if "items" in res:
317                         links = []
318                         for l in res["items"]:
319                             links.append(l["link"])
320                         filesResultFiles[search] = links
321         except:
322             print("Error using Google API. Blob and Files")
323
324         try:
325             for i in dataEnum["valid_databases"]:
326                 search = i + " site:pastebin.com"
327                 res = googleSearch.google_search(search, googleAPI, cse)
328                 dbsResult[search] = []
329                 if "items" in dbsResult:
```

```
330                     links = []
331                     for i in dbsResult["items"]:
332                         links.append(i["link"])
333                     dbsResult[search] = links
334             except:
335                 print("Error using Google API. Databases")
336 
337             try:
338                 dork1 = 'site:github.com web.config "DefaultEndpointsProtocol"
                        "'+ basicName+'"'
339                 defaultEndpointsProtocol = googleSearch.google_search(dork1,
                        googleAPI, cse)
340                 dork1_results = []
341                 if "items" in defaultEndpointsProtocol:
342                     links = []
343                     for i in defaultEndpointsProtocol["items"]:
344                         links.append(i["link"])
345                     dork1_results = links
346 
347                 dork2 = 'site:github.com web.config "StorageConnectionString"
                        "'+ basicName+'"'
348                 storageConnectionString = googleSearch.google_search(dork2,
                        googleAPI, cse)
349                 dork2_results = []
350                 if "items" in storageConnectionString:
351                     links = []
352                     for i in storageConnectionString["items"]:
353                         links.append(i["link"])
354                     dork2_results = links
355 
356                 sensitiveInformation={dork1:dork1_results,dork2:dork2_results}
357 
358             except:
359                 print("Error using Google API. Sensitive Information.")
360                 sensitiveInformation = {}
361 
362             # ADFS------------------------------------------------------
363             dataFederated = {}
364 
365             for i in dataAAD["listFQDN"]:
366                 data = checkFederated(i)
367                 dataFederated[i] = data
368 
369             # SHODAN----------------------------------------------------
370             shodan_accounts = {}
371             for i in dataEnum["valid_accounts"]:
372                 dir = resolveDomain(i)
```

```
373             shodan_accounts[i] = shodanTool.dataShodan(dir,shodanAPI)
374
375         shodan_websites = {}
376         data_status = {}
377         for i in dataEnum["valid_websites"]:
378             status = checkStatus(i)
379             dir = resolveDomain(i)
380             shodan_websites[i] = {"Status":{"HTTP status":status[0][1],"
                    HTTPS status":status[1][1]},"Information":shodanTool.
                    dataShodan(dir,shodanAPI)}
381
382         shodan_databases = {}
383         for i in dataEnum["valid_databases"]:
384             dir = resolveDomain(i)
385             shodan_databases[i] = shodanTool.dataShodan(dir,shodanAPI)
386
387         shodan_vms = {}
388         for i in dataEnum["valid_vms"]:
389             dir = resolveDomain(i)
390             shodan_vms[i] = shodanTool.dataShodan(dir,shodanAPI)
391
392         # EMAILS------------------------------
393         validUsers = []
394         if(dataAAD["desktopSSO"] == "True" and args.emails != "null"):
395             validUsers = aadinternals.userEnumeration(args.emails)
396         else:
397             if(args.emails != "null"):
398                 validUsers = o365creeper.o365emails(args.emails)
399
400         # Creating Json
401         try:
402             results = {"Azure/O365 Identification":identificationData,
403                        "AAD Data":{"Tenant Brand":dataAAD["tenantBrand"],"
                               Tenant Id":dataAAD["tenantId"],"DesktopSSO
                               enable":dataAAD["desktopSSO"],"Verified Domains
                               ":dataAAD["listDomains"], "Federated Domains":
                               dataAAD["listFQDN"]},
404                        "Azure Storage Accounts":shodan_accounts,
405                        "Azure Blob Containers":dataEnum["containers_data
                               "],
406                        "Azure Websites":shodan_websites,
407                        "Azure Databases":shodan_databases,
408                        "Azure Virtual Machines":shodan_vms,
409                        "Active Directory Federation Services":
                               dataFederated,
410                        "Google Searchs":{"Blob Storage":blobResultFiles,"
                               Files Storage":filesResultFiles,"Databases
```

```
                                Storage":dbsResult,"Sensitive Information":
                                sensitiveInformation},
411                         "Valid Emails": validUsers
412                         }
413
414             myjson = json.dumps(results)
415             writeFile(".\\Output\\output.json",myjson)
416             print("--------Final JSON--------\n")
417             print(myjson)
418
419         except:
420             print("Error in json")
421
422     else:
423         print("This company do not use Azure")
```

A.3. shodanTool.py

```
1  from shodan import Shodan
2
3
4  def dataShodan(ip,shodan_api):
5      """
6      This method uses the Shodan API to obtain information about the
           provided IP
7      """
8      api = Shodan(shodan_api)
9      output = {}
10     try:
11         host = api.host(ip)
12         hostIP = ip
13         organization = host.get('org', 'n/a')
14         vulnerabilities = host.get('vulns')
15
16         hostPorts = {}
17         for item in host['data']:
18             if "product" in item:
19                 producto = item["product"]
20             else:
21                 producto = item["_shodan"]["module"]
22             hostPorts[item['port']] = {"Service":producto,"Data":item['data
                   ']}
23         output = {"IP":hostIP,"Organization":organization,"Vulnerabilities
               ":vulnerabilities,"Ports":hostPorts}
24     except:
```

```
25          print('Error IP:' + ip)
26
27      return output
```

A.4. o365creeper.py

```
1  import requests as req
2  import re
3
4
5  def o365emails(inputfile):
6      """
7      Validate email accounts that belong to Office 365 tenants
8      """
9      url = 'https://login.microsoftonline.com/common/GetCredentialType'
10     result = []
11     with open(inputfile) as file:
12         for line in file:
13             s = req.session()
14             line = line.split()
15             email = ' '.join(line)
16             body = '{"Username":"'+ email +'"}'
17             request = req.post(url, data=body)
18             response = request.text
19             valid = re.search('"IfExistsResult":0,', response)
20             invalid = re.search('"IfExistsResult":1,', response)
21             if invalid:
22                 print(email +' - INVALID')
23             elif valid:
24                 result.append(email)
25                 print(email +' - VALID')
26
27     return result
```

A.5. googleSearch.py

```
from googleapiclient.discovery import build


def google_search(search_term, api, cse, **kwargs):
    """
    Using the Google API to search using dorks
    """
    service = build("customsearch", "v1", developerKey=api)
```

```
    res = service.cse().list(q=search_term, cx=cse, **kwargs).execute()
    return res
```

A.6. aadinternals.py

```
import subprocess
import re

def run(cmd):
    """
    Powershell command execution
    """
    completed = subprocess.run(["powershell", "-Command", cmd],
        capture_output=True)
    return completed


def reconAsOutsider(name):
    """
    Using AAD Internals to obtain data from the Azure Active Directory of a
        company
    """

    try:
        data = run("Import-Module AADInternals; Invoke-
            AADIntReconAsOutsider -DomainName " + name)

        #Obtaining all the data with regex
        tenantBrand = re.findall("(?<=Tenant brand:        )[\w ]+",str(data
            .stdout))
        tenantId = re.findall("(?<=Tenant id:           )[\w-]+",str(data.
            stdout))
        desktopSSO = re.findall("(?<=DesktopSSO enabled: )\w+",str(data.
            stdout))
        listDomains = re.findall("(?<=Name   : )[\w.-]+",str(data.stdout))
        fqdn = re.findall("(?<=STS    : )[\w.-]+",str(data.stdout))
        listFQDN = list(set(fqdn))

        #Obtaining relevant words to make mutations
        names = []

        for i in listDomains:
            dots = i.count(".")
            if dots == 1:
                aux = re.findall("(.*?)\..*$",i)
                if(len(aux[0]) > 4):
                    names.append(aux[0])
```

```
37              elif dots == 3 and "onmicrosoft" not in i:
38                  aux = re.findall(".*\.(.*?)\..*\..*$",i)
39                  if(len(aux[0]) > 4):
40                      names.append(aux[0])
41              elif dots == 2 and "onmicrosoft" not in i:
42                  aux = re.findall("(.*?)\.[\w]{2,3}\.[\w]{2,3}$",i)
43                  if len(aux) != 0 and len(aux[0]) > 4:
44                      names.append(aux[0])
45
46          listNames = list(set(names))
47          output = {"tenantBrand" : tenantBrand[0], "tenantId" : tenantId[0],
                "desktopSSO" : desktopSSO[0], "listDomains" : listDomains, "
               listFQDN" : listFQDN, "listNames" : listNames}
48
49      except:
50          output = {"tenantBrand" : [], "tenantId" : [], "desktopSSO" : [], "
               listDomains" : [], "listFQDN" : [], "listNames" : []}
51      finally:
52          return(output)
53
54  def userEnumeration(usersFile):
55      """
56      Using AAD Internals to check if users exist in Azure AD
57      """
58      data = run("Import-Module AADInternals; Get-Content " + usersFile + " |
            Invoke-AADIntUserEnumerationAsOutsider")
59
60      listUsers = re.findall("[0-9\w\-_#.]+@[0-9\w.]+\.[0-9\w]+\s+\w+",data.
           stdout.decode())
61      validUsers = []
62
63      for i in listUsers:
64          aux = str(i).split("  ")
65          if(aux[1]=="True"):
66              validUsers.append(aux[0])
67
68      return validUsers
```

A.7. azure_checks.py

```
1  import re
2  import requests
3  from cloud_enum_tools import utils
4  from cloud_enum_tools import azure_regions
5
```

```
6  BANNER = '''
7  ++++++++++++++++++++++++++
8          azure checks
9  ++++++++++++++++++++++++++
10 '''
11
12 # Known Azure domain names
13 BLOB_URL = 'blob.core.windows.net'
14 WEBAPP_URL = 'azurewebsites.net'
15 DATABASE_URL = 'database.windows.net'
16
17 # Virtual machine DNS names are actually:
18 #   {mutation}.{region}.cloudapp.azure.com
19 VM_URL = 'cloudapp.azure.com'
20
21
22 def print_account_response(reply):
23     """
24     Parses the HTTP reply of a brute-force attempt
25
26     This function is passed into the class object so we can view results
27     in real-time.
28     """
29     if reply.status_code == 404:
30         pass
31     elif 'Server failed to authenticate the request' in reply.reason:
32         utils.printc("    Auth-Only Storage Account: {}\n"
33                      .format(reply.url), 'red')
34     elif 'The specified account is disabled' in reply.reason:
35         utils.printc("    Disabled Storage Account: {}\n"
36                      .format(reply.url), 'red')
37     elif 'Value for one of the query' in reply.reason:
38         utils.printc("    HTTP-OK Storage Account: {}\n"
39                      .format(reply.url), 'orange')
40     elif 'The account being accessed' in reply.reason:
41         utils.printc("    HTTPS-Only Storage Account: {}\n"
42                      .format(reply.url), 'orange')
43     else:
44         print("    Unknown status codes being received from {}:\n"
45               "        {}: {}"
46               .format(reply.url, reply.status_code, reply.reason))
47
48 def check_storage_accounts(names, threads, nameserver):
49     """
50     Checks storage account names
51     """
52     print("[+] Checking for Azure Storage Accounts")
```

```
53
54      # Start a counter to report on elapsed time
55      start_time = utils.start_timer()
56
57      # Initialize the list of domain names to look up
58      candidates = []
59
60      # Initialize the list of valid hostnames
61      valid_names = []
62
63      # Take each mutated keyword craft a domain name to lookup.
64      # As Azure Storage Accounts can contain only letters and numbers,
65      # discard those not matching to save time on the DNS lookups.
66      regex = re.compile('[^a-zA-Z0-9]')
67      for name in names:
68          if not re.search(regex, name):
69              candidates.append('{}.{}'.format(name, BLOB_URL))
70
71      # Azure Storage Accounts use DNS sub-domains. First, see which are
            valid.
72      valid_names = utils.fast_dns_lookup(candidates, nameserver,
73                                          threads=threads)
74
75      # Send the valid names to the batch HTTP processor
76      utils.get_url_batch(valid_names, use_ssl=False,
77                          callback=print_account_response,
78                          threads=threads)
79
80      # Stop the timer
81      utils.stop_timer(start_time)
82
83      # de-dupe the results and return
84      return list(set(valid_names))
85
86  def print_container_response(reply):
87      """
88      Parses the HTTP reply of a brute-force attempt
89
90      This function is passed into the class object so we can view results
91      in real-time.
92      """
93      container = ""
94      files=[]
95      result={}
96
97      # Stop brute forcing disabled accounts
98      if 'The specified account is disabled' in reply.reason:
```

```
99          print("    [!] Breaking out early, account disabled.")
100         return 'breakout'
101
102     # Stop brute forcing accounts without permission
103     if ('not authorized to perform this operation' in reply.reason or
104             'not have sufficient permissions' in reply.reason or
105             'Public access is not permitted' in reply.reason or
106             'Server failed to authenticate the request' in reply.reason):
107         print("    [!] Breaking out early, auth required.")
108         return 'breakout'
109
110     # Stop brute forcing unsupported accounts
111     if 'Blob API is not yet supported' in reply.reason:
112         print("    [!] Breaking out early, Hierarchical namespace account")
113         return 'breakout'
114
115     # Handle other responses
116     if reply.status_code == 404:
117         pass
118     elif reply.status_code == 200:
119         utils.printc("    OPEN AZURE CONTAINER: {}\n"
120                      .format(reply.url), 'green')
121         container=reply.url
122         files = utils.list_bucket_contents(reply.url)
123     elif 'One of the request inputs is out of range' in reply.reason:
124         pass
125     elif 'The request URI is invalid' in reply.reason:
126         pass
127     else:
128         print("    Unknown status codes being received from {}:\n"
129               "        {}: {}"
130               .format(reply.url, reply.status_code, reply.reason))
131
132     if container != "":
133         result = {"container":container,"urls":files}
134         return result
135
136
137
138 def brute_force_containers(storage_accounts, brute_list, threads):
139     """
140     Attempts to find public Blob Containers in valid Storage Accounts
141
142     Here is the URL format to list Azure Blog Container contents:
143     <account>.blob.core.windows.net/<container>/?restype=container&comp=
            list
144     """
```

```
145
146     # We have a list of valid DNS names that might not be worth scraping,
147     # such as disabled accounts or authentication required. Let's quickly
148     # weed those out.
149     print("[*] Checking {} accounts for status before brute-forcing"
150           .format(len(storage_accounts)))
151     valid_accounts = []
152     for account in storage_accounts:
153         reply = requests.get('https://{}/'.format(account))
154         if 'Server failed to authenticate the request' in reply.reason:
155             storage_accounts.remove(account)
156         elif 'The specified account is disabled' in reply.reason:
157             storage_accounts.remove(account)
158         else:
159             valid_accounts.append(account)
160
161     # Read the brute force file into memory
162     clean_names = utils.get_brute(brute_list, mini=3)
163     # Start a counter to report on elapsed time
164     start_time = utils.start_timer()
165
166     print("[*] Brute-forcing container names in {} storage accounts"
167           .format(len(valid_accounts)))
168     for account in valid_accounts:
169         print("[*] Brute-forcing {} container names in {}"
170               .format(len(clean_names), account))
171
172         # Initialize the list of correctly formatted urls
173         candidates = []
174
175         # Take each mutated keyword and craft a url with correct format
176         for name in clean_names:
177             candidates.append('{}/{}/?restype=container&comp=list'
178                               .format(account, name))
179
180         # Send the valid names to the batch HTTP processor
181         containers_data = utils.get_url_batch(candidates, use_ssl=True,
182                               callback=print_container_response,
183                               threads=threads)
184
185     # Stop the timer
186     utils.stop_timer(start_time)
187
188     return containers_data
189
190 def print_website_response(hostname):
191     """
```

```
192     This function is passed into the DNS brute force as a callback,
193     so we can get real-time results.
194     """
195     utils.printc("     Registered Azure Website DNS Name: {}\n"
196                  .format(hostname), 'green')
197
198 def check_azure_websites(names, nameserver, threads):
199     """
200     Checks for Azure Websites (PaaS)
201     """
202     print("[+] Checking for Azure Websites")
203
204     # Start a counter to report on elapsed time
205     start_time = utils.start_timer()
206
207     # Initialize the list of domain names to look up
208     candidates = [name + '.' + WEBAPP_URL for name in names]
209
210     # Azure Websites use DNS sub-domains. If it resolves, it is registered.
211     valid_names = utils.fast_dns_lookup(candidates, nameserver,
212                           callback=print_website_response,
213                           threads=threads)
214
215     # Stop the timer
216     utils.stop_timer(start_time)
217
218     return list(set(valid_names))
219
220 def print_database_response(hostname):
221     """
222     This function is passed into the DNS brute force as a callback,
223     so we can get real-time results.
224     """
225     utils.printc("     Registered Azure Database DNS Name: {}\n"
226                  .format(hostname), 'green')
227
228 def check_azure_databases(names, nameserver, threads):
229     """
230     Checks for Azure Databases
231     """
232     print("[+] Checking for Azure Databases")
233
234     # Start a counter to report on elapsed time
235     start_time = utils.start_timer()
236
237     # Initialize the list of domain names to look up
238     candidates = [name + '.' + DATABASE_URL for name in names]
```

```
239
240     # Azure databases use DNS sub-domains. If it resolves, it is registered
            .
241     valid_names = utils.fast_dns_lookup(candidates, nameserver,
242                             callback=print_database_response,
243                             threads=threads)
244
245     # Stop the timer
246     utils.stop_timer(start_time)
247
248     return list(set(valid_names))
249
250 def print_vm_response(hostname):
251     """
252     This function is passed into the DNS brute force as a callback,
253     so we can get real-time results.
254     """
255     utils.printc("     Registered Azure Virtual Machine DNS Name: {}\n"
256                  .format(hostname), 'green')
257
258 def check_azure_vms(names, nameserver, threads):
259     """
260     Checks for Azure Virtual Machines
261     """
262     print("[+] Checking for Azure Virtual Machines")
263
264     # Start a counter to report on elapsed time
265     start_time = utils.start_timer()
266
267     # Pull the regions from a config file
268     regions = azure_regions.REGIONS
269
270     print("[*] Testing across {} regions defined in the config file"
271           .format(len(regions)))
272
273     for region in regions:
274
275         # Initialize the list of domain names to look up
276         candidates = [name + '.' + region + '.' + VM_URL for name in names]
277
278         # Azure VMs use DNS sub-domains. If it resolves, it is registered.
279         valid_names = utils.fast_dns_lookup(candidates, nameserver,
280                                 callback=print_vm_response,
281                                 threads=threads)
282
283     # Stop the timer
284     utils.stop_timer(start_time)
```

```
285
286     return list(set(valid_names))
287
288 def run_all(names, args):
289     """
290     Function is called by main program
291     """
292     print(BANNER)
293     valid_accounts = []
294     containers_data = []
295     valid_databases = []
296     valid_websites = []
297     valid_vms = []
298     try:
299         valid_accounts = check_storage_accounts(names, args.threads, args.
                nameserver)
300
301         containers_data = []
302         if valid_accounts and not args.quickscan:
303             containers_data = brute_force_containers(valid_accounts, args.
                    brute, args.threads)
304     except:
305         print("Error validating accounts, blobs and containers")
306
307     try:
308         valid_websites = check_azure_websites(names, args.nameserver, args.
                threads)
309     except:
310         print("Error validating websites")
311
312     try:
313         valid_databases = check_azure_databases(names, args.nameserver,
                args.threads)
314     except:
315         print("Error validating databases")
316
317     try:
318         valid_vms = check_azure_vms(names, args.nameserver, args.threads)
319     except:
320         print("Error validating VMs")
321
322     output = {"valid_accounts":valid_accounts, "valid_websites":
            valid_websites, "valid_databases":valid_databases, "valid_vms":
            valid_vms, "containers_data":containers_data}
323
324     return output
```

A.8. azure_regions.py

```
"""
File used to track the DNS regions for Azure resources.
"""
REGIONS = ['eastasia', 'southeastasia', 'centralus', 'eastus', 'eastus2',
           'westus', 'northcentralus', 'southcentralus', 'northeurope',
           'westeurope', 'japanwest', 'japaneast', 'brazilsouth',
           'australiaeast', 'australiasoutheast', 'southindia', '
              centralindia',
           'westindia', 'canadacentral', 'canadaeast', 'uksouth', 'ukwest',
           'westcentralus', 'westus2', 'koreacentral', 'koreasouth',
           'francecentral', 'francesouth', 'australiacentral',
           'australiacentral2', 'southafricanorth', 'southafricawest']


# And here I am limiting the search by overwriting this variable:
REGIONS = ['eastus','westeurope',]
```

A.9. utils.py

Listado A.1: main.py

```
"""
Helper functions for network requests, etc
"""

import time
import sys
import datetime
import re
from multiprocessing.dummy import Pool as ThreadPool
from functools import partial
try:
    import requests
    import dns
    import dns.resolver
    from concurrent.futures import ThreadPoolExecutor
    from requests_futures.sessions import FuturesSession
    from concurrent.futures._base import TimeoutError
except ImportError:
    print("[!] Please pip install requirements.txt.")
    sys.exit()

LOGFILE = False

```

```
24 def init_logfile(logfile):
25     """
26     Initialize the global logfile if specified as a user-supplied argument
27     """
28     if logfile:
29         global LOGFILE
30         LOGFILE = logfile
31 
32         now = datetime.datetime.now().strftime("%d/%m/%Y %H:%M:%S")
33         with open(logfile, 'a') as log_writer:
34             log_writer.write("\n\n#### CLOUD_ENUM {} ####\n"
35                              .format(now))
36 
37 def get_url_batch(url_list, use_ssl=False, callback='', threads=5, redir=
       True):
38     """
39     Processes a list of URLs, sending the results back to the calling
40     function in real-time via the `callback` parameter
41     """
42     containers={}
43     # Start a counter for a status message
44     tick = {}
45     tick['total'] = len(url_list)
46     tick['current'] = 0
47 
48     # Break the url list into smaller lists based on thread size
49     queue = [url_list[x:x+threads] for x in range(0, len(url_list), threads
           )]
50 
51     # Define the protocol
52     if use_ssl:
53         proto = 'https://'
54     else:
55         proto = 'http://'
56 
57     # Using the async requests-futures module, work in batches based on
58     # the 'queue' list created above. Call each URL, sending the results
59     # back to the callback function.
60     for batch in queue:
61 
62         session = FuturesSession(executor=ThreadPoolExecutor(max_workers=
               threads+5))
63         batch_pending = {}
64         batch_results = {}
65 
66         # First, grab the pending async request and store it in a dict
67         for url in batch:
```

```
68              batch_pending[url] = session.get(proto + url, allow_redirects=
                    redir)
69
70          # Then, grab all the results from the queue.
71          # This is where we need to catch exceptions that occur with large
72          # fuzz lists and dodgy connections.
73          for url in batch_pending:
74              try:
75                  # Timeout is set due to observation of some large jobs
                        simply
76                  # hanging forever with no exception raised.
77                  batch_results[url] = batch_pending[url].result(timeout=30)
78              except requests.exceptions.ConnectionError as error_msg:
79                  print("    [!] Connection error on {}:".format(url))
80                  print(error_msg)
81              except TimeoutError:
82                  print("    [!] Timeout on {}. Investigate if there are"
83                        " many of these".format(url))
84
85          # Now, send all the results to the callback function for analysis
86          # We need a way to stop processing unnecessary brute-forces, so the
87          # callback may tell us to bail out.
88
89          for url in batch_results:
90              check = callback(batch_results[url])
91              if isinstance(check,dict):
92                  containers[check["container"]]=check["urls"]
93              elif check == 'breakout':
94                  return
95
96          # Refresh a status message
97          tick['current'] += threads
98          sys.stdout.flush()
99          sys.stdout.write("    {}/{} complete..."
100                           .format(tick['current'], tick['total']))
101         sys.stdout.write('\r')
102
103     # Clear the status message
104     sys.stdout.write('                                \r')
105
106     return containers
107
108 def dns_lookup(nameserver, name):
109     """
110     This function performs the actual DNS lookup when called in a
            threadpool
111     by the fast_dns_lookup function.
```

```
112     """
113     res = dns.resolver.Resolver()
114     res.timeout = 10
115     res.nameservers = [nameserver]
116 
117     try:
118         res.query(name)
119         # If no exception is thrown, return the valid name
120         return name
121     except dns.resolver.NXDOMAIN:
122         return ''
123     except dns.exception.Timeout:
124         print("     [!] DNS Timeout on {}. Investigate if there are many"
125               " of these.".format(name))
126 
127 def fast_dns_lookup(names, nameserver, callback='', threads=5):
128     """
129     Helper function to resolve DNS names. Uses multithreading.
130     """
131     total = len(names)
132     current = 0
133     valid_names = []
134 
135     print("[*] Brute-forcing a list of {} possible DNS names".format(total)
        )
136 
137     # Break the url list into smaller lists based on thread size
138     queue = [names[x:x+threads] for x in range(0, len(names), threads)]
139 
140     for batch in queue:
141         pool = ThreadPool(threads)
142 
143         # Because pool.map takes only a single function arg, we need to
144         # define this partial so that each iteration uses the same ns
145         dns_lookup_params = partial(dns_lookup, nameserver)
146 
147         results = pool.map(dns_lookup_params, batch)
148 
149         # We should now have the batch of results back, process them.
150         for name in results:
151             if name:
152                 if callback:
153                     callback(name)
154                 valid_names.append(name)
155 
156         current += threads
157 
```

```
158         # Update the status message
159         sys.stdout.flush()
160         sys.stdout.write("    {}/{} complete...".format(current, total))
161         sys.stdout.write('\r')
162         pool.close()
163 
164     # Clear the status message
165     sys.stdout.write('                                \r')
166 
167     return valid_names
168 
169 def list_bucket_contents(bucket):
170     """
171     Provides a list of full URLs to each open bucket
172     """
173     finalUrls=[]
174     key_regex = re.compile(r'<(?:Key|Name)>(.*?)</(?:Key|Name)>')
175     reply = requests.get(bucket)
176 
177     # Make a list of all the relative-path key name
178     keys = re.findall(key_regex, reply.text)
179 
180     # Need to remove URL parameters before appending file names
181     # from Azure buckets
182     sub_regex = re.compile(r'(\?.*)')
183     bucket = sub_regex.sub('', bucket)
184 
185     # Format them to full URLs and print to console
186     if keys:
187         printc("      FILES:\n", 'none')
188         for key in keys:
189             url = bucket + key
190             finalUrls.append(url)
191             printc("      ->{}\n".format(url), 'none')
192     else:
193         printc("      ...empty bucket, so sad. :(\n", 'none')
194 
195     return finalUrls
196 
197 def printc(text, color):
198     """
199     Prints colored text to screen
200     """
201     # ANSI escape sequences
202     green = '\033[92m'
203     orange = '\033[33m'
204     red = '\033[31m'
```

```
205     bold = '\033[1m'
206     end = '\033[0m'
207 
208     if color == 'orange':
209         sys.stdout.write(bold + orange + text + end)
210     if color == 'green':
211         sys.stdout.write(bold + green + text + end)
212     if color == 'red':
213         sys.stdout.write(bold + red + text + end)
214     if color == 'black':
215         sys.stdout.write(bold + text + end)
216     if color == 'none':
217         sys.stdout.write(text)
218 
219     if LOGFILE:
220         with open(LOGFILE, 'a')  as log_writer:
221             log_writer.write(text.lstrip())
222 
223 def get_brute(brute_file, mini=1, maxi=63, banned='[^a-z0-9_-]'):
224     """
225     Generates a list of brute-force words based on length and allowed chars
226     """
227     # Read the brute force file into memory
228     with open(brute_file, encoding="utf8", errors="ignore") as infile:
229         names = infile.read().splitlines()
230 
231     # Clean up the names to usable for containers
232     banned_chars = re.compile(banned)
233     clean_names = []
234     for name in names:
235         name = name.lower()
236         name = banned_chars.sub('', name)
237         if maxi >= len(name) >= mini:
238             if name not in clean_names:
239                 clean_names.append(name)
240 
241     return clean_names
242 
243 def start_timer():
244     """
245     Starts a timer for functions in main module
246     """
247     # Start a counter to report on elapsed time
248     start_time = time.time()
249     return start_time
250 
251 def stop_timer(start_time):
```

```
252     """
253     Stops timer and prints a status
254     """
255     # Stop the timer
256     elapsed_time = time.time() - start_time
257     formatted_time = time.strftime("%H:%M:%S", time.gmtime(elapsed_time))
258 
259     # Print some statistics
260     print("")
261     print(" Elapsed time: {}".format(formatted_time))
262     print("")
```

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR