UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Sistema de autenticidad para aplicaciones de análisis de eventos para seguridad.

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Toni Escamilla Pardo

Tutor: Julio Pons Terol

2016/2017

Resumen

En este proyecto se afronta la creación de un sistema distribuido capaz de garantizar, en aplicaciones empleadas para el análisis de eventos de seguridad, la autenticidad de los datos transmitidos entre el host cliente y servidor con el fin de asegurar la integridad de los mismos en la transmisión entre ambas partes, así como evitar la posible alteración en el destino. Para ello, se utilizará la tecnología *blockchain* a modo de base de datos en la que se almacenen los hashes de los eventos generados en el host cliente.

Palabras clave: sistema, blockchain, integridad, ciberseguridad, eventos.

Abstract

This project tackles the creation of a distributed system capable of guaranteeing, in applications used for the analysis of security events, the authenticity of the data transmitted between the client and the server hosts in order to ensure the integrity of the data in the transmission between both parties, as well as to avoid the possible alteration in the destiny. To do this, *blockchain* technology will be used as a database where to store the hashes of generated events in the client host.

Keywords: system, blockchain, integrity, cybersecurity, events.

Tabla de contenidos

1. Introducción

Actualmente, el recurso intangible más importante que existe en el entorno empresarial es la información. Esta información es la principal fuente de ingresos de las empresas, y es aquella que mueve a toda una organización. Su valor depende de diversos factores como los años de investigación, creaciones, ideas, conceptos, reglamentos, entre otros. Sin embargo, no todas las organizaciones poseen el mismo grado de concienciación de la importancia de la información.

Según los datos publicados en el *Informe Anual de Seguridad Nacional 2016*[1], solo en España durante el año 2016 se gestionaron, por parte del Centro Criptológico Nacional, 21.000 incidentes a organizaciones, un 15% más respecto al año 2015. Por lo tanto, y observando los datos proporcionados, los ataques informáticos a empresas son una tendencia que lejos de reducirse, tiende a agravarse con el paso de los años.

Este tipo de ataques se encuentran dirigidos a los sistemas informáticos que una organización posee, encargados de dar soporte a las operaciones empresariales, la gestión y la toma de decisiones, proporcionando a las personas la información que necesitan, siendo estos el conjunto de elementos en los cuales se almacena y procesa la información, el recurso intangible más importante. Por ello, la obtención de este tipo información, por parte de un individuo externo a la organización, representa un importante acaecimiento cuyo impacto puede afectar a la empresa de forma notable.

Pero, ¿es posible detectar ataques cuyo objetivo es vulnerar los sistemas informáticos de una organización? La seguridad 100% es imposible de alcanzar, pero sí es factible aproximarse empleando las técnicas adecuadas a partir de la detección de anomalías producidas en los sistemas a proteger. Hoy en día existen múltiples soluciones llamadas HIDS (Host Intrusion Detection System), o sensores, cuya función principal es monitorizar y analizar las actividades realizadas en un sistema en particular a partir de los datos que estas generan en dicho equipo. Estos datos, se cotejan con unos patrones definidos previamente y, en el caso de que exista coincidencia, se genera una

[1] DEPARTAMENTO DE SEGURIDAD NACIONAL. *Informe Anual de Seguridad Nacional*. 2016. Disponible en http://www.lamoncloa.gob.es/serviciosdeprensa/notasprensa/Documents/140217-Informe_Anual_de_Seguridad_Nacional_2016.pdf

alerta, o evento, relativa a dicha actividad y se envía a un servidor central encargado de realizar el análisis posterior.

Sin embargo, y dado que este tipo de información se envía a través de la red, es posible que exista una modificación de esta durante su transmisión e incluso se realice dicha modificación a posterior en el servidor central con el fin de alterar los datos y ocultar la actividad realizada en el equipo inicial. Esta necesidad en la alteración de la información transmitida puede deberse a la intrusión de un usuario ajeno a la organización en el que su objetivo para realizar esta modificación en la integridad de los datos no sea sino ocultar la penetración en los sistemas de la compañía.

Sin embargo, el punto de partida del proyecto reside en el hecho de la posible necesidad de alterar la información transmitida por alguna de las partes que conforman el sistema de análisis de eventos; la empresa encargada de analizar los eventos o el cliente que ha contratado dicho servicio. Esta posible necesidad parte una supuesta desconfianza entre ambas partes a partir de la información generada, es decir, existe la posibilidad de que un empleado de la compañía dedicada al análisis haya cometido una negligencia durante dicha tarea y decida eliminar los eventos recibidos con el fin de enmascarar la desidia producida. Por otra parte, es posible que la empresa cliente reclame una posible negligencia a la compañía encargada del servicio y para ello, el cliente decida alterar los eventos en sus equipos con el fin de añadir aquellos por los que se reclama.

Las posibles acciones descritas que pueden ser llevadas a cabo por alguna de las dos partes suponen, en ambos casos, una afectación en la imagen de la organización encargada del análisis de los eventos para seguridad.

Por todo ello, el objetivo de este proyecto es explicar e implementar un sistema de autenticidad que garantice la completa integridad de los datos transmitidos entre el sistema informático, en la cual se encuentra instalado el software HIDS, y el servidor central, encargado de realizar el post-análisis de los eventos generados. Adicionalmente, con el sistema descrito, también se pretende garantizar la integridad de la información tras almacenarla en el servidor central, asegurando de esta forma la no alteración de los mismos una vez almacenados.

En la implementación propuesta del sistema de autenticidad, además, se garantiza la legitimidad de la procedencia de dichos datos dada la tecnología empleada para ello.

Destacar que el sistema de autenticidad focaliza su ámbito de aplicación entre dos equipos, es decir, entre pares, permitiendo la verificación bajo demanda de una transferencia de datos realizada entre ambos y no sobre la transferencia de datos realizada en un mismo equipo.

La motivación para la realización de este proyecto es la creación de un nuevo sistema que permite dotar de una capa extra de seguridad a la transferencia de logs entre equipos empleando una tecnología actual y que representa el pilar de un futuro descentralizado, la tecnología *blockchain*[2]. Además, con la realización del sistema de autenticidad para aplicaciones de análisis de eventos para seguridad se pretende ir un paso más allá, demostrando una de las múltiples posibilidades que esta tecnología ofrece.

Adicionalmente, y otro factor de motivación para la realización de este proyecto es la elaboración de un nuevo prototipo cuyo futuro desarrollo permita obtener un producto con fines comerciales y que, a día de hoy, no se evidencia la existencia de este tipo de sistemas en el mercado actual, siendo esto un aliciente extra al punto descrito en el apartado anterior.

Finalmente, destacar que el desarrollo de este sistema puede suponer un aumento del prestigio de la compañía si se reconoce como un sistema válido por aquellas organizaciones en las que se encuentra implantado.

[2] NAKAMOTO, S. *Bitcoin: A Peer-to-Peer Electronic Cash System*. 2008. Disponible en https://bitcoin.org/bitcoin.pdf

2. Estado del arte

En este apartado va a realizarse un análisis de las distintas herramientas disponibles actualmente y que, en su conjunto, ofrecen diversas alternativas las cuales permitirán crear el sistema de autenticidad para aplicaciones de análisis de eventos para seguridad. De las diversas herramientas disponibles, se justificará de forma razonada la elección individual del software que permitirá implementar el sistema de autenticidad. Posteriormente, se aportará una visión global del estado actual de la tecnología empleada para garantizar la integridad de los logs transmitidos entre dos pares, objetivo para el cual se realiza este proyecto.

Para la creación del sistema de autenticidad, se han determinado diversas características que el sistema de autenticidad debe contener a fin de cumplir con el objetivo propuesto. A continuación, se detallan las principales propiedades que debe englobar el sistema implementado:

- Obtención de mensajes de registro individualizados, es decir, los mensajes de registro deben ser específicos para cada una de las actividades realizadas en cada uno de los equipos a fin de conseguir una rápida verificación de los mismos.

- No alteración de los mensajes de registro en la transmisión de estos entre pares.

- Protección frente a modificaciones de la integridad de los mensajes de registro tras su almacenamiento.

- Capacidad de ejecución en sistemas multiplataforma a fin de permitir la interoperabilidad entre los mismos.

Tras la definición de estas características, se ha realizado una búsqueda de herramientas o tecnologías disponibles las cuales cumplan con, al menos, uno de los aspectos descritos y que, adicionalmente, se encuentren en constante desarrollo, indicador que evidencia la evolución del producto y el acogimiento por parte de los distintos usuarios.

Si se centra en la abstracción del sistema de autenticidad, éste debe garantizar la integridad de los mensajes de registro, desde el mismo momento en que son transmitidos en origen, para cada una de las fases que atraviesa, siendo dos de ellas las más cruciales en lo que a la no alteración de los datos contenidos se refiere; transmisión y almacenamiento.

En la fase de transmisión, los mensajes de registro son intercambiados entre el equipo donde se han originado, el equipo origen, y el equipo en el cual se realizará el análisis de estos eventos, el equipo destino. La criticidad de esta fase reside en el hecho de la no alteración de los datos al transmitirse por la red, generalmente compartida, y que, dados unos conocimientos mínimos al respecto, permitirían a un individuo la interceptación de los paquetes que contienen dichos datos, su modificación o inspección, y el posterior envío hacia el equipo destino. Sin embargo, es posible conseguir una transmisión segura en un entorno de red compartido a partir del empleo de una conexión cifrada entre ambos hosts a partir de la utilización de cifrado asimétrico con claves criptográficas. La utilización de transmisiones seguras con cifrado asimétrico garantiza la segunda de las características, la no alteración de los mensajes de registro en la transmisión de estos entre pares.

En la fase de almacenamiento, los mensajes de registro son guardados en el ordenador destino para su análisis a lo largo del tiempo. En este caso, la criticidad reside en el hecho de la posible alteración de los eventos tras ser almacenados, existiendo una discordancia entre los mensajes provenientes del equipo origen y el análisis en el equipo destino. La alteración, entre la que se incluye la posibilidad de borrado de los mismos, es posible mediante el acceso al sistema destino. Este acceso puede ser bien un acceso autorizado por parte del administrador de dicho sistema, o bien, un acceso no autorizado por parte de un individuo, normalmente, ajeno a la organización. Esta es una de las principales desventajas de los sistemas centralizados, caracterizados por tener una estructura en la que los elementos de la red son productores o consumidores de información los cuales precisan de un elemento central para el proceso y gestión de la información. Por ello, como solución alternativa a los sistemas con arquitectura centralizada, se disponen de sistemas con arquitectura distribuida y que poseen diversas ventajas respecto a los sistemas centralizados:

- Mayor rendimiento que un único sistema.

- Tolerancia frente a fallos en cualquier equipo del sistema distribuido.
- Escalabilidad

Con todo ello, y tras valorar distintas tecnologías, se ha optado por la tecnología blockchain, "cadena de bloques" en español, como base para la implementación del sistema de autenticidad puesto que cumple a la perfección con dos de las cuatro características requeridas para la implementación de dicho sistema. Por una parte, permite el cifrado de la transmisión de los datos entre los pares que conforman la red, asegurando la integridad de los eventos durante el envío de los mismos, y por otra parte garantiza un sistema de almacenamiento replicado en el que todos los pares de la red participan de forma igualitaria a partir de la validación de las transacciones realizadas en la cadena de bloques, eliminando la posibilidad de la modificación de los mensajes de registro puesto que la información se encuentra redundada y validada por cada uno de los equipos participantes.

Seguidamente, se indica el software escogido y que, en su conjunto, abarca la totalidad de los requisitos definidos para el diseño e implementación del sistema de autenticidad, así como la implementación de la cadena de bloques:

OSSEC HIDS

OSSEC HIDS (Host Intrusion Detection System, Sistema de Detección de Intrusos en Host) es un software el cual se encarga de monitorizar y detectar anomalías en el sistema a partir del análisis de los mensajes de registro producidos por las distintas actividades en dicho sistema.

OSSEC utiliza el protocolo *syslog* para el envío de los distintos mensajes de registro que va generando. Cada uno de estos mensajes está etiquetado con un código que indica el tipo de software que genera el mensaje y un valor de gravedad. Para la implementación de syslog, emplea *rsyslog*, utilidad de software de código abierto utilizada en sistemas informáticos Unix, y similares a Unix, para reenviar mensajes de registro en una red IP. Rsyslog implementa el protocolo syslog, lo extiende con el filtrado basado en contenido, las capacidades de filtrado enriquecidas, las opciones de configuración flexibles y añade características como el uso de TCP para el transporte.

La elección de OSSEC como software encargado de la recolección de mensajes de registro y su posterior envío se debe a diversos motivos los cuales se detallan a continuación:

- Escalable. La arquitectura con la que OSSEC está concebido se corresponde con una arquitectura cliente-servidor, de tal forma que los agentes instalados en cada uno de los equipos envían los distintos eventos a uno o múltiples servidores. Por ello, es posible aumentar de una forma rápida y sencilla tanto el número de agentes desplegados como el número de servidores.

- Multiplataforma. OSSEC cuenta con múltiples paquetes de instalación para los principales sistemas operativos y que permiten una compatibilidad entre diversos equipos cuyos sistemas operativos difieren entre sí.

- Código abierto y gratuito. OSSEC se encuentra bajo licencia GNU General Public License, que garantiza a los usuarios finales (personas, organizaciones, compañías) la libertad de usar, estudiar, compartir y modificar el software.

- Amplia utilización por parte de los usuarios. OSSEC es un proyecto que se encuentra en constante crecimiento dada las continuas mejoras y evoluciones que se introducen en el desarrollo de este. Adicionalmente, la gran comunidad de usuarios que existe tras este proyecto, proporciona un soporte ante cualquier incidencia o consulta.

- Envío de mensajes de registro relacionados con la seguridad. El principal objetivo de OSSEC es la monitorización de todos los aspectos relacionados con la seguridad a partir del análisis de los registros, comprobación de la integridad de ficheros, supervisión del registro de Windows y la aplicación de políticas centralizadas, alertas en tiempo real, entre otros.

- Configuración en lenguaje XML. La configuración de OSSEC mediante etiquetas basadas en lenguaje XML facilita la aplicación de diversas configuraciones adaptadas a cada uno de los casos requeridos.

Con todas estas características, OSSEC garantiza la primera de las propiedades requeridas por el sistema de autenticidad puesto que los mensajes de registro son

específicos para cada una de las actividades realizadas en cada uno de los equipos donde se encuentra instalado.

MultiChain

MultiChain es una plataforma que se emplea para la creación y despliegue de cadenas de bloque privadas, ya se dentro de una organización o entre varias. El objetivo de MultiChain es facilitar el despliegue de la tecnología de cadenas de bloque a la vez que proporciona privacidad y mecanismos de control.

La elección de MultiChain como plataforma para el despliegue e implementación de la tecnología blockchain viene dada por múltiples características las cuales se detallan a continuación:

- Gestión de permisos. Dado que se trata de una cadena de bloques privada, permite controlar de forma dinámica quien puede conectarse, así como enviar y recibir transacciones, entre otras acciones.
- Personalizable. La forma en la que MultiChain se encuentra estructurado, posibilita el control absoluto sobre cada uno de los aspectos que forman la cadena de bloques.
- Multifuncional. Dada la capacidad de personalización de la cadena de bloques, MultiChain permite utilizar la tecnología blockchain para múltiples utilidades.
- Escalabilidad. Siguiendo con el objetivo de la tecnología blockchain, es posible desplegar nuevas instancias de MultiChain en nuevos equipos con el fin de añadirlos a la cadena de bloques existente pasando a formar parte de la misma.

Una de las características claves de MultiChain es la multifuncionalidad, ya que, tal y como se ha comentado, permite utilizar la cadena de bloques para diversos aspectos. Entre estos aspectos, se encuentra el almacenamiento de datos, funcionalidad para la cual se va a utilizar la cadena de bloques.

Para ello, implementa lo que se conoce como *streams*, que proporcionan una abstracción natural para aquellos casos de uso de la red blockchain que se centran en la recuperación de los datos, el registro de tiempo y el archivado en lugar de la

transferencia de activos entre los participantes de la red. Es decir, los flujos permiten que una cadena de bloques se utilice como una base de datos de propósito general.

Una cadena de bloques puede contener cualquier número de flujos, en la que los datos publicados en cada uno de ellos son almacenados por cada nodo suscrito.

Cada stream es una lista ordenada de ítems en la que cada uno de los ítems presenta las siguientes características:

- Uno o más nodos han firmado digitalmente ese elemento.
- Una clave para permitir su posterior recuperación.
- Datos en formato hexadecimal que pueden alcanzar varios megabytes de tamaño.
- Información sobre la transacción de dicho ítem, incluyendo el ID de la transacción, el hash del bloque completo y el registro de tiempo de la generación del bloque al que pertenece la transacción, entre otros.

Por todo ello, MultiChain se postula una solución eficaz y concreta al despliegue e implementación de la tecnología blockchain.

De manera adicional, la plataforma emplea un protocolo propio llamado protocolo MultiChain el cual se encuentra basado en el protocolo *Bitcoin*, ampliamente utilizado. El protocolo MultiChain extiende las funcionalidades de este último para poder gestionar la cadena de bloques de forma privada.

Finalmente, tras realizar un análisis de los distintos métodos de garantizar la integridad de los mensajes de registro, se ha optado por emplear una función hash. En la Tabla 1 se muestran los resultados obtenidos en dicho análisis, así como una gráfica en la que se muestra la representación de dichos datos.

	MD5	SHA1	SHA256	SHA512
100	0.002000332	0.000999928	0.001000166	0
1000	0.003999949	0.005000114	0.008999825	0.00999999
10000	0.039001226	0.061998367	0.08300066	0.107999802
100000	0.516002417	0.569001675	0.887003899	1.181999445
1000000	5.340008259	6.363994598	9.181000471	11.75299096

Tabla 1. Datos obtenidos en el análisis

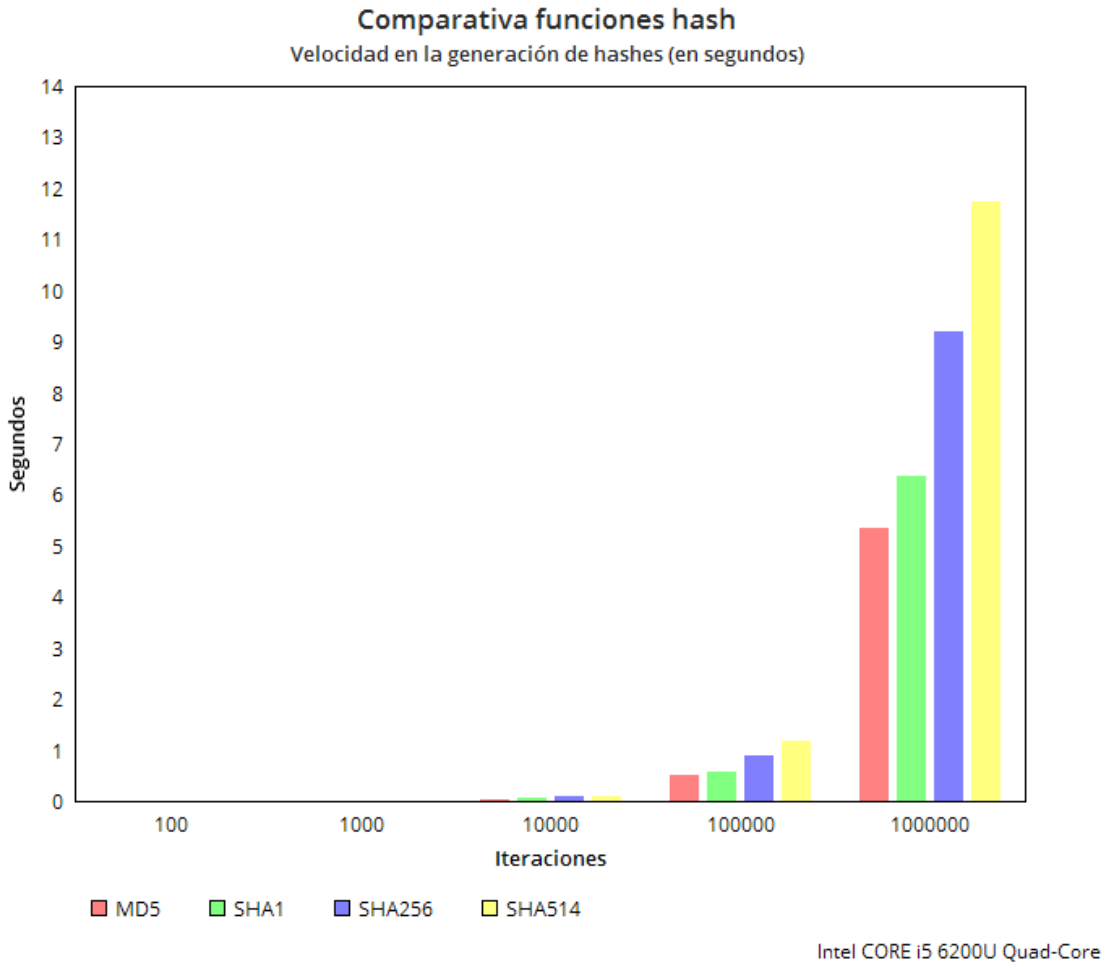

Ilustración 1. Comparativa de diversas funciones hash

Tal y como se observa en la Ilustración 1, se han analizado cuatro funciones hash distintas; *MD5*, *SHA1*, *SHA256* y *SHA514*. Para realizar dicho análisis, se han empleado las funciones de la librería *hashlib* del lenguaje de programación *Python* ya que esta será la librería empleada en uno de los componentes del sistema de autenticidad.

Respecto al procedimiento seguido, este se ha basado en la obtención de un hash cuya entrada es una cadena alfanumérica pseudoaleatoria de 2097 caracteres, tamaño de la que representa el número máximo de caracteres permitidos por rsyslog. Adicionalmente, este proceso se ha repetido en múltiples iteraciones para cada uno de ellos, obteniendo los valores que se muestran en la Tabla 1.

Destacar que el tamaño de la cadena se ha determinado en función del número máximo de caracteres permitidos por rsyslog, utilidad software que implementa el protocolo syslog y que será la utilizada para el envío de los mismos.

Los datos obtenidos en el análisis, evidencian una mínima diferencia entre la obtención de los hashes al realizar un número reducido de iteraciones. Sin embargo, al incrementar el número de iteraciones se aprecia un incremento en el tiempo de generación para cada una de las funciones. Concretamente, se observa que la función SHA512 es la función que presenta un tiempo mayor respecto a las otras funciones, siendo la función MD5 aquella que requiere un menor tiempo para la obtención de los hashes.

A partir de mediciones reales en sistemas de obtención y procesamiento de mensajes de registro, se calcula que el sistema de autenticidad tendrá una frecuencia máxima de generación de hashes aproximada de 100 hashes/segundo, es decir, 100 iteraciones en un segundo como valor máximo. Si se observan los datos obtenidos, se evidencia que la función hash cuyo tiempo es menor para 100 iteraciones es la función hash SHA512.

Adicionalmente, otros de los factores a considerar para realizar la elección de la función hash a utilizar, son los bytes que se obtienen en la salida a partir de una misma entrada y la posibilidad de obtener dos hashes idénticos a partir de dos o más entradas diferentes, es decir, la posibilidad de producirse colisiones en la función. En la Tabla 2 se observa un resumen.

Función Hash	Tamaño de salida (bytes)	Colisiones encontradas
MD5	16	Sí
SHA1	20	Sí
SHA256	20	No
SHA512	64	No

Tabla 2. Valores de los factores a considerar

Según se observa en la tabla superior, el tamaño de salida no afecta al valor máximo establecido para la transacción en la cadena de bloques dado el reducido número de bytes que las funciones producen, por lo que un tamaño de salida más elevado proporciona mayor seguridad puesto que las posibles combinaciones aumentan de forma exponencial, existiendo 2^{64} posibilidades para el caso de SHA512. Sin embargo, sí se han encontrado colisiones en las funciones MD5 y SHA1, lo cual puede afectar de forma grave al sistema de autenticidad si se produjesen dos hashes idénticos.

Con todo ello, tras el análisis realizado, la función hash escogida ha sido SHA512, función hash que destaca por su elevada seguridad, debido al tamaño de la cadena en su salida, y la no existencia de colisiones detectadas a día de hoy. En esta función la entrada es el evento producido por OSSEC y la salida es una cadena alfanumérica única de 64 bytes. Esta cadena, en formato hexadecimal, actuará como clave de cada uno de los ítems, cuyo valor asociado a cada uno será el propio mensaje de registro en el mismo formato a partir del cual se ha obtenido dicho hash.

3. Tecnología blockchain

Técnicamente, una cadena de bloques, o *blockchain* en inglés, es una base de datos distribuida que facilita el registro de transacciones y el seguimiento de activos. La implementación de la blockchain se hizo pública en el año 2009, junto con la publicación del protocolo Bitcoin, ya que es la tecnología que sustenta dicha criptomoneda. Su nombre proviene de la forma en la que el almacenamiento de las transacciones se produce; bloques de transacciones se unen entre ellos para formar una cadena, tal y como se observa en la Ilustración 2, y se dice que es distribuida puesto que la cadena de bloques se encuentra en una red descentralizada *peer-to-peer*. Cada uno de los nodos que componen la red, almacenan de forma completa la cadena de bloques, y estos se mantienen sincronizados de tal forma que ésta se encuentre actualizada en todos y cada uno de dichos nodos.

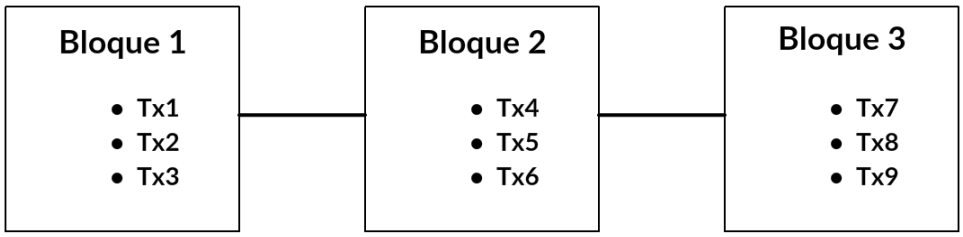

Ilustración 2. Estructura de la cadena de bloques

Según se ha comentado, un bloque es un conjunto de transacciones, marcado con un sello de tiempo, y una huella digital (*hash*) del bloque anterior de tal forma que estos mantengan relación y formen la cadena de bloques. Cada uno de los bloques que forman la blockchain contienen una estructura determinada, Ilustración 3, compuesta por el hash del bloque actual, el hash del bloque anterior, un sello de tiempo, un número arbitrario empleado una única vez (*nonce*) y las propias transacciones. Adicionalmente, se establece un tamaño máximo de bloque el cual permite homogeneizar el tamaño de estas de tal forma que todos los bloques ocupen un tamaño similar y con ello, contengan aproximadamente el mismo tamaño de datos.

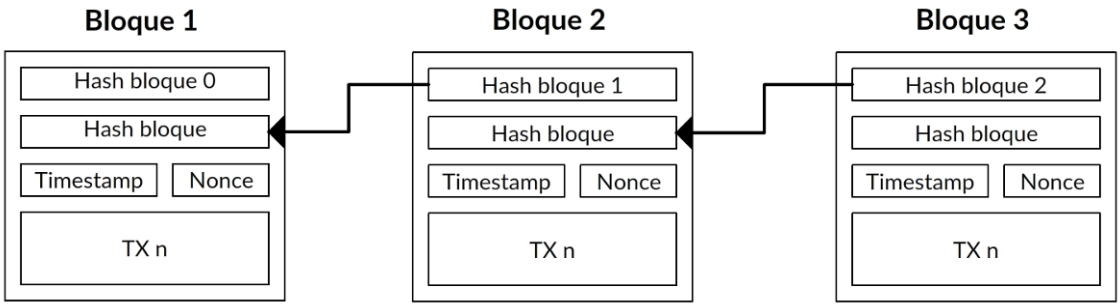

Ilustración 3. Estructura de los bloques de la blockchain

Mediante la inclusión de la huella criptográfica digital del bloque anterior, la cadena de bloques asegura que la integridad de los bloques previos se mantiene intacta, así como la imposibilidad de insertar un nuevo bloque entre dos existentes, ya que para realizar dichas acciones sería necesario alterar todos y cada uno de los bloques que forman la cadena de bloques puesto que una modificación en alguno de los bloques cambiaría el hash de dicho bloque, incluido en el bloque inmediatamente posterior. Esto se realiza sucesivamente en todos los bloques que conforman la cadena de bloques, por lo que sería necesario alterar todos y cada uno de los siguientes bloques para que coincidiesen los valores hash incluidos en cada uno de ellos. De esta manera, cada bloque subsiguiente refuerza la verificación del bloque anterior y, por tanto, de toda la cadena de bloques. Toda cadena de bloques contiene un bloque inicial, llamado *bloque génesis*, el cual se emplea para inicializar la blockchain y este no contendrá ninguna referencia a un bloque anterior al tratarse del primero de estos.

El objetivo de la cadena de bloques es almacenar transacciones. Una transacción, definida a alto nivel, es un traspaso de información entre dos nodos que componen la red. Sin embargo, en la descripción a bajo nivel, una transacción es una estructura de datos firmada digitalmente que expresa el valor de la transferencia realizada y se compone de una lista con transacciones de entrada y una lista con transacciones de salida. Una transacción de entrada es de donde procede el valor recibido, normalmente de una transacción de salida. Estas transacciones de salida asignan un nuevo propietario del valor asociándolo a él mediante la dirección del nuevo propietario.

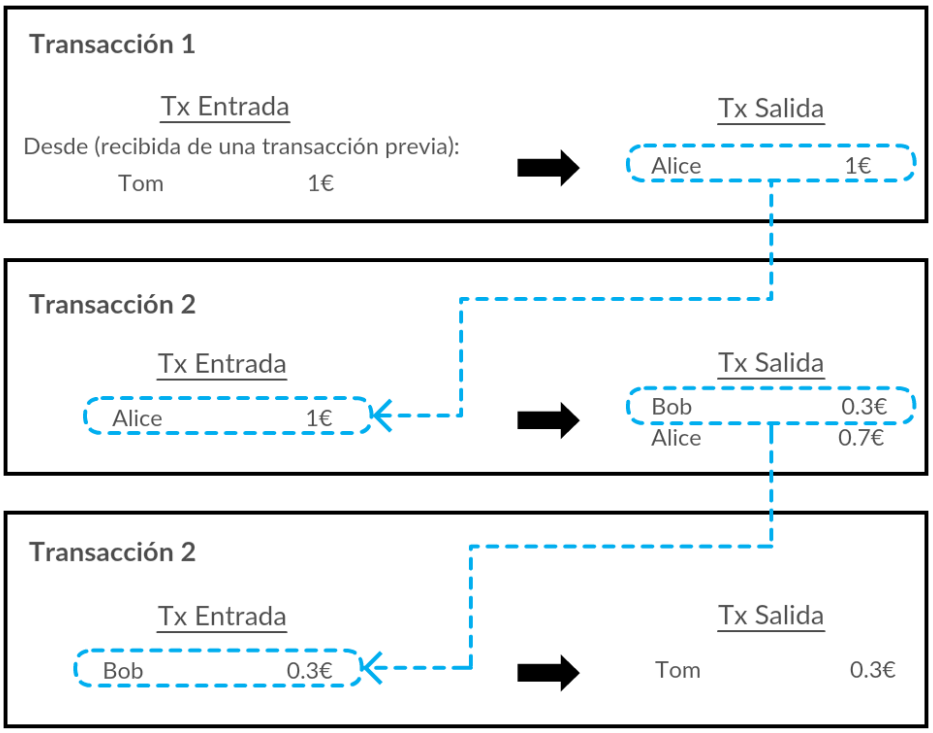

Ilustración 4. Cadena de transacciones, donde la salida de una transacción es la entrada de la siguiente.

Las direcciones de los distintos nodos que componen la red son identificadores únicos para cada uno de estos, y se obtienen a partir de las claves públicas, empleadas para la firma de las transacciones realizadas.

Tras realizar una transacción entre un nodo y otro, esta se irá propagando por toda la red al tratarse de una red peer-to-peer, en la que los nodos emisor y receptor mandan dicha transacción a sus nodos vecinos más cercanos. Cualquier nodo que recibe una transacción válida, y que no ha sido recibida previamente, reenviará esta transacción de forma inmediata a los nodos a los que se encuentra conectado. De esta forma, todos los nodos que pertenecen a la cadena de bloques se mantienen sincronizados y la cadena de bloques se encuentra actualizada en cada uno de ellos. La comprobación de la transacción en cada uno de los nodos, se realiza siguiendo una comprobación de los parámetros preestablecidos propios de la blockchain entre los que se pueden incluir el tamaño máximo de la transacción, los valores posibles a transferir, la estructura de la transacción y su sintaxis y muchos otros.

Sin embargo, una transacción difundida por la red no se encuentra confirmada hasta que forma parte de la cadena de bloques, es decir, hasta que se encuentra en un bloque el cual ha sido añadido a la blockchain. Para ello, existen nodos llamados *mineros* que

se encargan de generar los nuevos bloques de la cadena de bloques, confirmando con ello las transacciones que han sido incluidas en dicho bloque.

El *minado* es el proceso mediante el cual las nuevas transacciones son validadas y almacenadas en la cadena de bloques, y este proceso es realizado por los mineros. Para realizar dicha acción, los mineros deben resolver un problema matemático complejo basado en un algoritmo criptográfico de hash. La solución a este problema se llama *proof of work* y se corresponde con el hash del bloque minado.

Hasta ahora, se ha asumido que los hashes de los bloques son los hashes del bloque completo, lo cual incluye el hash del bloque previo y todas las transacciones contenidas en dicho bloque. No obstante, esto presenta algunos inconvenientes:

- Si una transacción se modifica, el hash debe ser computado de nuevo. Esto fuerza a los nodos a mantener en memoria todo el bloque completo.
- Para verificar si una transacción pertenece a un bloque, el bloque completo debe estar disponible para poder calcular el hash y verificar si dicha transacción pertenece al bloque.

Por ello, los bloques incluyen una parte llamada *cabecera* en la que se incluye el hash del bloque anterior, el sello de tiempo, el valor del nonce y un hash raíz dependiente de las transacciones incluidas en el bloque. El hash del bloque es, por lo tanto, el hash de la cabecera del bloque.

Para obtener el hash raíz, se emplea lo que se conocen como *árboles Merkle*, estructura de datos en forma de árbol binario de valores donde cada nodo padre es el resultado de aplicar una función hash sobre los nodos hijo hasta llegar al nodo raíz, conocido como *Merkle root*, o hash raíz.

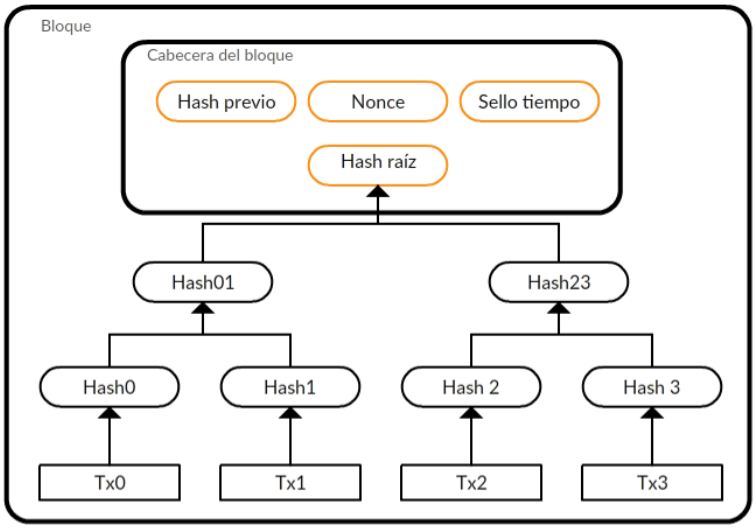

Ilustración 5. Árbol de Merkle de las transacciones en un bloque

De esta forma, un cambio en cualquiera de las hojas, provocará un cambio en el nodo superior hasta llegar al nodo raíz, lo que variará el hash del bloque y, por lo tanto, se considerará inválida dicha modificación.

Tal y como se ha comentado en uno de los puntos anteriores, esta tecnología surgió junto con Bitcoin, conjunto de conceptos y tecnologías que forman la base de un ecosistema en lo que a moneda digital se refiere. Por lo tanto, y dado el objetivo de Bitcoin, la utilización de la cadena de bloques que sustenta dicho sistema es pública. Sin embargo, también existe la posibilidad de utilizar cadenas de bloques privadas, como la que se emplea en este proyecto.

Las cadenas de bloques privadas se caracterizan por la introducción de nuevos aspectos sobre las cadenas de bloques públicas, resolviendo de esta forma problemas asociados al minado, la privacidad y la confianza mediante la gestión de permisos para cada uno de los nodos.

3.1 Implementación de la cadena de bloques en MultiChain

MultiChain es la plataforma empleada para la implementación, y despliegue, de la cadena de bloques. Según se ha comentado, el enfoque aportado a la utilización de la cadena de bloques es un enfoque privado, por lo que seguidamente se comentarán los aspectos más relevantes acerca de la implementación técnica realizada por MultiChain al respecto.

En primer lugar, en lo que se refiere a la creación de la red peer-to-peer, existe un nodo central que se encarga de gestionar y administrar la cadena de bloques. El primero de los nodos participantes en la red debe realizar una conexión a este nodo central, el cual almacenará la dirección de este nuevo nodo conectado. Éste primer nodo almacenará a su vez la dirección del nodo central y la cadena de bloques que dicho nodo le ha transferido.

Cuando un nuevo nodo entre en la red, existe la posibilidad de que se conecte al nodo central o al nodo ya existente en la red. En el primero de los casos, el nodo central proporcionará una lista con los pares vecinos existentes en la red al nuevo nodo conectado de forma que este conozca la lista de los pares en la red, así como la cadena de bloques actual. En el segundo de los casos, el nodo ya existente actuará de igual forma que el nodo central, transfiriendo una lista de pares vecinos al nuevo nodo y la cadena de bloques existente. Esto se repetirá para cada uno de los pares de tal forma que todos los nodos se encuentren interconectados entre sí.

Mediante la interconexión de todos los nodos, se garantiza que todos los pares mantienen la misma versión de la cadena bloques, ya que tal y como se ha comentado, cada vez que un nuevo nodo se añade a la red, éste recibe la cadena de bloques completa. Además, las nuevas actualizaciones se transmitirán a todos los nodos vecinos y estos, a su vez, transmitirán dicha actualización a sus propios nodos vecinos, consiguiendo que la actualización se propague por toda la red.

En segundo lugar, y respecto al proceso de minado, todos los bloques añadidos a la cadena de bloques se encuentran firmados por el nodo minero que ha realizado dicha acción, de tal forma que en todo momento sea posible identificar a dicho nodo y, además, se garantice la integridad de los nuevos bloques dado que el último de los bloques añadidos posee una probabilidad mayor de ser alterado puesto que no se encuentra referenciado por ninguno de estos. Adicionalmente, este proceso se puede restringir para que únicamente un número determinado de bloques puedan ser creados por el mismo minero en un espacio de tiempo dado. Para ello, MultiChain utiliza un parámetro llamado *diversidad de minado* (*mining diversity*), restringido entre $0 \leq$ diversidad de minado ≤ 1, a partir del cual se fuerza un esquema *Round-Robin* en el que los mineros permitidos deben crear los nuevos bloques de manera equitativa y en un orden racional para generar una cadena de bloques válida. Un valor de 1 asegura

que todos los mineros permitidos se encuentran incluidos en la rotación, mientras que un valor de 0 no representa ninguna restricción. Un valor más alto representa mayor seguridad, pero un valor muy cercano a 1 puede producir que la cadena de bloques se quede obstruida si alguno de los mineros se torna inactivo. Los efectos de este valor se pueden calcular en función de varios parámetros, para ello se asume que cada minero tiene una probabilidad independiente *f* de fallo técnico y *c* de colusión maliciosa. La probabilidad *Pr(f)* de que un minero se quede inactivo viene dada por la distribución binomial acumulativa *Pr(Bin(mineros, 1-f) ≤ (espaciado – 1))*. Del mismo modo, la probabilidad *Pr(c)* de que mineros maliciosos minen una cadena alternativa es *Pr(Bin(mineros, 1-c) ≤ (espaciado – 1))*.

Para asegurar que el bloque a añadir a la cadena de bloques es válido, y realizar el esquema Round-Robin, la validación se comprueba de la siguiente forma:

1. Se aplican todos los cambios de permisos, sobre el minado, definidos por las transacciones, en ese mismo bloque, en orden.

2. Se cuentan el número de mineros permitidos tras aplicar los cambios definidos en el punto 1.

3. Se multiplica el número de mineros permitidos por el valor asignado al parámetro diversidad de minado. El resultado se redondea al número mayor para obtener el *espaciado*.

4. Si el minero del bloque ya ha realizado este proceso en uno de los bloques *espaciado – 1* anteriores, el bloque no es válido.

Con todo ello, esta forma de realizar el minado evita los ataques de suplantación de identidad, ya que únicamente se permite un número determinado de mineros y adicionalmente, se requiere de la clave privada para realizar el firmado de los bloques.

Adicionalmente, y al tratarse de cadenas de bloques privadas, MultiChain ofrece la gestión de permisos para distintos aspectos como asegurar que la actividad de la cadena de bloque únicamente es visible a los nodos que participan en la blockchain, controlar las transacciones permitidas y realizar el minado de los nuevos bloques. Este último aspecto referido al minado es el que se ha descrito en el punto superior. Para realizar esta gestión de permisos, se emplea criptografía de clave asimétrica, en la que

cada uno de los nodos genera un par de claves pública-privada, las cuales se utilizan para firmar las transacciones realizadas. Es a partir de la clave privada de donde se obtiene la dirección pública única del nodo.

Además, este tipo de criptografía se utiliza para restringir el acceso de nuevos nodos a la cadena de bloques a aquellos permitidos previamente. Para ello, MultiChain expande el proceso de *handshake*, que ocurre cuando dos nodos de la cadena de bloques se conectan:

1. Ambos nodos muestran su identidad como una dirección pública única en la lista de nodos permitidos.
2. Ambos nodos verifican que la dirección del otro nodo se encuentra en la lista de los nodos permitidos.
3. Ambos nodos se intercambian un mensaje de reto.
4. Cada nodo devuelve el mensaje de reto firmado con su clave privada, demostrando que posee la clave privada correspondiente a la dirección pública presentada.

Este tipo de comportamiento realizado para la conexión entre nodos se extiende a muchas otras operaciones en la red como por ejemplo la restricción de enviar y/o recibir transacciones entre dos nodos.

Finalmente, y tal y como se ha explicado en el apartado 2, MultiChain implementa los streams. Para realiza dicha función, los streams son tratados como transacciones, de tal forma que se expanden por la red, llegando a todos los nodos. Posteriormente, cuando un nodo recibe un stream y este se encuentra suscrito a dicho stream, se procede al indexado de la información de tal forma que el nodo sea consciente de la información almacenada. Por el contrario, y en caso de que dicho nodo no se encuentre suscrito al stream recibido, omitirá la indexación del mismo.

4. Análisis del entorno

El entorno propuesto se compone de cuatro sistemas informáticos en función del rol adquirido por cada una de las mismas. En este entorno se simula la presencia de la compañía cliente y la compañía encargada del análisis de los eventos. El sistema informático a monitorizar en el cliente se compone por dos equipos, *C1.1* y *C1.2*, que se encuentran en las propias oficinas del cliente. Por otro lado, el sistema informático de la compañía encargada de dicho análisis se compone de dos equipos, *Servidor Blockchain* y *Servidor Syslog*, alojados en las oficinas de la compañía. En la Ilustración 6 se observa la estructura del entorno propuesto.

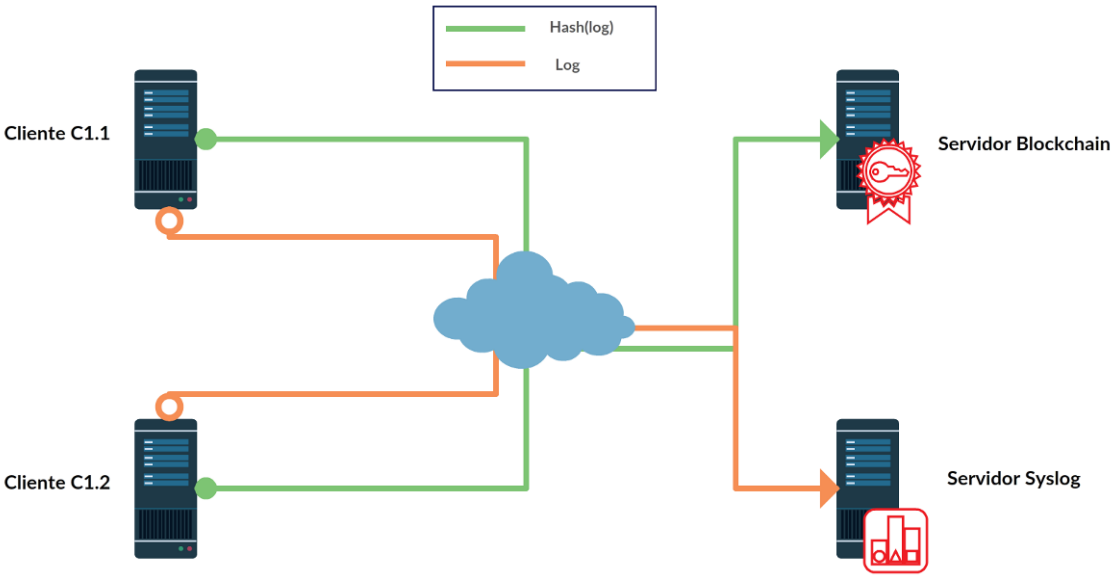

Ilustración 6. Entorno propuesto

La funcionalidad del sistema de autenticidad reside en el intercambio de información entre los distintos componentes que integran dicho sistema, garantizando en todo momento la integridad de los mensajes de registro. Seguidamente, se muestran los diversos elementos del sistema de autenticidad, así como la secuencia seguida por la información desde que el mensaje de registro es generado hasta que dicho evento es validado en la cadena de bloques y, por lo tanto, inmutable dadas las propiedades comentadas anteriormente de la blockchain.

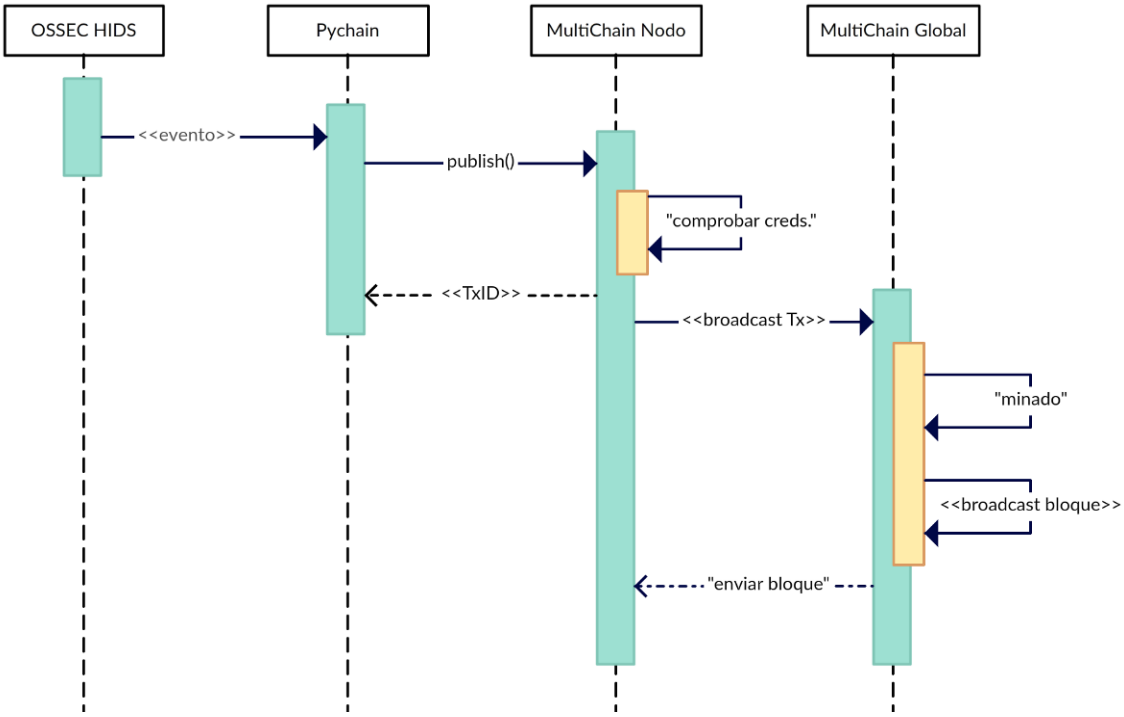

Ilustración 7. Diagrama UML de secuencia del sistema de autenticidad

Tal y como se observa en la Ilustración 7, cuando el software OSSEC HIDS detecta un nuevo evento en el sistema, este envía dicho evento al cliente Pychain, explicado más adelante en el apartado 5.2 Cliente *pychain*, y que se encarga de tratar el evento y publicarlo en el MultiChain ejecutado en el propio equipo. Posteriormente, se hace una difusión de la transacción a todos los nodos participantes de la red. Los nodos de la cadena de bloques almacenarán estas transacciones y uno de los nodos se encargará de realizar el minado del bloque actual. Seguidamente, este nodo realizará una difusión de dicho bloque al resto de nodos de la red, los cuales validarán las transacciones contenidas en el bloque, aceptándolas como válidas.

A continuación, se define para cada máquina, el rol establecido, así como los requisitos mínimos para su correcto funcionamiento:

4.1 Máquina 1. Cliente C1.1

Esta máquina se encuentra en el lado del cliente, y se encarga de enviar los mensajes de registro hacia ambos servidores. Adicionalmente, y dado que se encuentra integrada en la red de blockchain, participará de forma activa en el minado de los bloques generados en dicha red.

En lo que ha requisitos se refiere, esta máquina constará de un sistema operativo Debian 9 en modo consola con 4 GB de RAM, aunque el mínimo requerido para el correcto funcionamiento es de 512 MB. Con ello, se asegura el correcto funcionamiento en momentos de elevada carga, permitiendo asegurar la disponibilidad de la misma. Respecto al procesador, éste posee una arquitectura Intel x86 de un núcleo a 2'4 GHz.

El empleo del sistema operativo Debian 9 se debe principalmente a las mejoras en lo que a rendimiento se refiere, así como la presencia de software actualizado, subsanando errores presentes en las versiones anteriores[3].

Además, contará de 100 GB de memoria secundaria donde se almacenarán los mensajes de registro que todavía no hayan sido procesados y preparados para el envío.

Se instalará el software MultiChain junto con la librería Savoir para abordar una adecuada implementación de la red de blockchain. Esta librería para el lenguaje de programación Python realiza una implementación de las distintas llamadas JSON-RPC necesarias para la comunicación con la cadena de bloques en MultiChain empleando este lenguaje de programación. Adicionalmente, y para completar la funcionalidad, se ejecutará un *script* que permitirá el procesamiento del *syslog* tanto para su adición en la cadena de bloques como para su envío al servidor Syslog.

La gestión de dicha máquina se realizará empleando el protocolo SSH en el puerto 22 desde la red interna. Desde la red externa, únicamente se habilitarán los puertos definidos para la comunicación con el servidor Blockchain y Syslog, respectivamente.

4.2 Máquina 2. Cliente C1.2

Réplica de la máquina descrita en el apartado anterior, su funcionalidad reside en el envío de mensajes de registro hacia los servidores Blockchain y Syslog, así como la participación en el minado de los bloques presentes en la red de blockchain.

[3] http://www.phoronix.com/scan.php?page=article&item=debian-9-soon

Al igual que en el caso anterior, esta máquina constará de un sistema operativo Debian 9 en modo consola con 4 GB de RAM y un disco duro de 100 GB. Respecto al procesador, éste posee una arquitectura Intel x86 de un núcleo a 2’4 GHz.

En cuanto el software instalado en la misma, se utilizará MultiChain junto con la librería Savoir para implementar la red de blockchain. Adicionalmente, y para completar la funcionalidad, se ejecutará un *script* que permitirá el procesamiento del *syslog* tanto para su adición en la cadena de bloques como para su envío al servidor Syslog.

La gestión de dicha máquina se realizará empleando el protocolo SSH en el puerto 22 desde la red interna. Desde la red externa, únicamente se habilitarán los puertos definidos para la comunicación con el servidor Blockchain y Syslog, respectivamente.

4.3 Máquina 3. Servidor Blockchain

Esta máquina se utilizará para gestionar y soportar la red de blockchain en la que se encuentran el resto de máquinas del entorno propuesto, así como el minado de los bloques generados en la red.

El sistema operativo que se ha escogido para esta máquina es Debian 9 con el modo consola activado con el fin de optimizar los recursos físicos disponibles los cuales se han establecido en 4 GB de memoria RAM y 100 GB de memoria secundaria. Respecto al procesador, éste posee una arquitectura Intel x86 de un núcleo a 2’4 GHz.

El software adicional a instalar en este host es MultiChain. Sin embargo, y a diferencia del resto de máquinas descritas en los apartados anteriores, los privilegios con los que se empleará dicho software, permiten gestionar la red de blockchain en referencia a la administración de usuarios y sus respectivos permisos, así como la creación de cadenas y flujos de datos.

La gerencia de este equipo se realizará mediante el protocolo seguro SSH a través del puerto 22 desde la red interna con el fin de evitar posibles ataques desde el exterior. Con lo que respecta a los puertos abiertos, únicamente se determinará el

puerto necesario para recibir los hashes de los *syslog* pertenecientes a las máquinas cliente C1.1 y C1.2.

4.4 Máquina 4. Servidor Syslog

Esta máquina será la encargada de recibir los mensajes de registro, previamente enviados por los clientes definidos, y procesarlos para su posterior inclusión en el sistema de análisis de eventos para seguridad.

Este host no formará parte de la red de blockchain puesto que los recursos de los que dispone se emplearán de forma exclusiva al procesado de los *syslog* recibidos y al hospedaje de la aplicación de análisis de eventos.

Por ello, se ha definido la instalación del sistema operativo Debian 9 en modo consola con 8 GB de memoria RAM y 500 GB de disco duro. Si observamos los requisitos de Debian 9, en estos se indica que el mínimo requerido son 512 MB de memoria principal y 2 GB de memoria secundaria. Sin embargo, dada la elevada carga que supone la recepción constante de mensajes y su posterior procesado, así como la ejecución constante de la aplicación utilizada para el análisis de eventos, se ha determinado que la mejor opción es dotar a la máquina de recursos suficientes que aseguren la disponibilidad en un grado cercano al 100% y garanticen la correcta ejecución de las diversas tareas. Respecto al procesador, éste posee una arquitectura Intel x86 de cuatro núcleos a 2’4 GHz.

Para la recepción de los mensajes de registro se empleará un sistema de detección de intrusos de host *open-source* llamado OSSEC, el cual permite procesar los mensajes recibidos desde los clientes para crear alertas en función del contenido de dicho mensaje.

La gestión de dicha máquina se realizará empleando el protocolo SSH en el puerto 22 desde la red interna. Desde la red externa, únicamente se habilitarán los puertos definidos para la comunicación con los clientes.

5. Sistema de autenticidad

En este apartado, se explicará paso a paso el proceso de creación del sistema de autenticidad para aplicaciones de análisis de eventos de seguridad.

Para crear el entorno se han empleado máquinas virtualizadas sobre un hipervisor nativo, llamado *ESXi*, el cual se instala directamente sobre el servidor físico sin necesidad de un sistema operativo host. Las máquinas empleadas para la creación del entorno son las descritas en el apartado 4.

A continuación, se detalla paso a paso el proceso para crear el sistema de autenticidad en su conjunto, haciendo especial hincapié en la configuración de software empleada en cada una de las máquinas. Adicionalmente, y dado que la instalación de las máquinas no entra dentro del alcance del sistema de autenticidad, se ha omitido dicho proceso.

5.1 Instalación y configuración MultiChain

Tomando como referencia el entorno descrito en el apartado 4, y en función de los requisitos de software descritos para cada uno de los equipos implicados en el sistema, la instalación de MultiChain se realizará en las máquinas C1.1, C1.2 y Servidor Blockchain.

Destacar que, al tratarse del mismo sistema operativo Linux en los equipos mencionados, la instalación se realiza de igual forma en cada uno ellos. No obstante, y dado que cada uno de los equipos posee un rol diferente, la configuración sí difiere al respecto.

5.1.1 Descarga e instalación - Linux

Para realizar la descarga e instalación del software, es necesario emplear la línea de comandos e iniciar sesión con el usuario *root*. Para ello, ejecutamos el comando "su", tras el cual nos solicitará la contraseña de dicho usuario, tal y como se observa en la Ilustración 8.

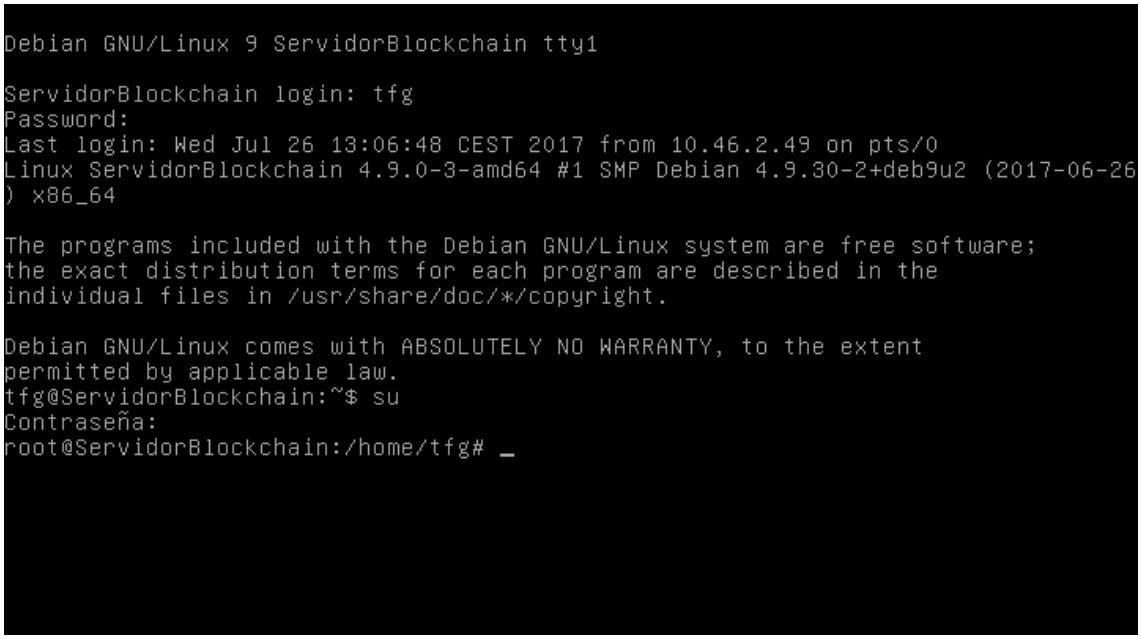

Ilustración 8. Ejecución comando "su"

Actualmente, el directorio establecido es "/home/tfg". El siguiente paso es establecer el directorio actual a "/tmp" mediante la ejecución del comando "cd /tmp".

Ilustración 9. Cambio de directorio actual

A continuación, mediante el comando "wget https://www.multichain.com/download/multichain-latest.tar.gz" se obtiene la última versión disponible de MultiChain.

Ilustración 10. Descarga del software MultiChain con wget

Tras unos segundos, se ha descargado la última versión estable la cual, en el momento de realizar esta descripción, es multichain-*1.0-beta-2*.

Según lo observado en la Ilustración 10, el fichero se ha descargado comprimido en un formato “tar.gz”, por lo que para acceder al contenido es necesario descomprimir dicho fichero. Para ello, ejecutamos el comando “tar -xvzf multichain-latest.tar.gz”.

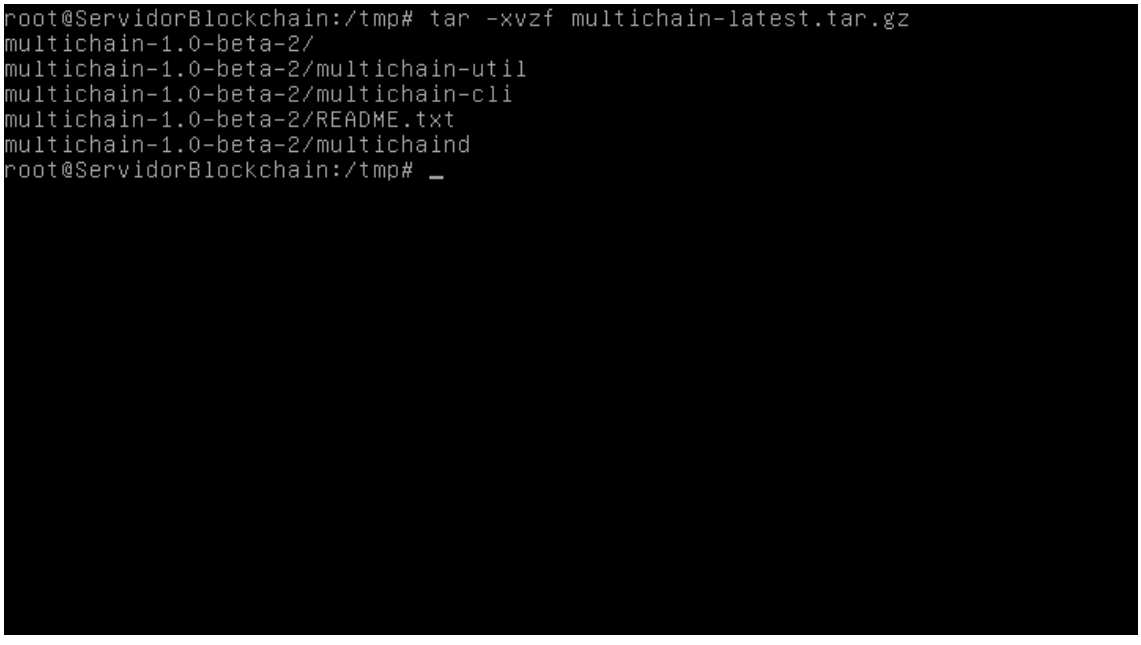

Ilustración 11. Descompresión del fichero descargado

Posteriormente, nos movemos a la carpeta raíz de MultiChain empleando el comando "cd multichain-1.0-beta-2", dentro de la cual existen cuatro ficheros diferentes tal y como se observa en la Ilustración 11.

Los ficheros contenidos en la carpeta raíz representan las utilidades que posteriormente se emplearan para establecer la cadena de bloques, así como la gestión de los distintos nodos, entre otros.

El siguiente paso es permitir el acceso a dichas utilidades desde la línea de comandos independientemente del directorio en el que nos encontremos, para lo cual se debe ejecutar el comando "mv multichaind multichain-cli multichain-util /usr/local/bin" que se encarga de mover los ficheros descritos al directorio "/usr/local/bin".

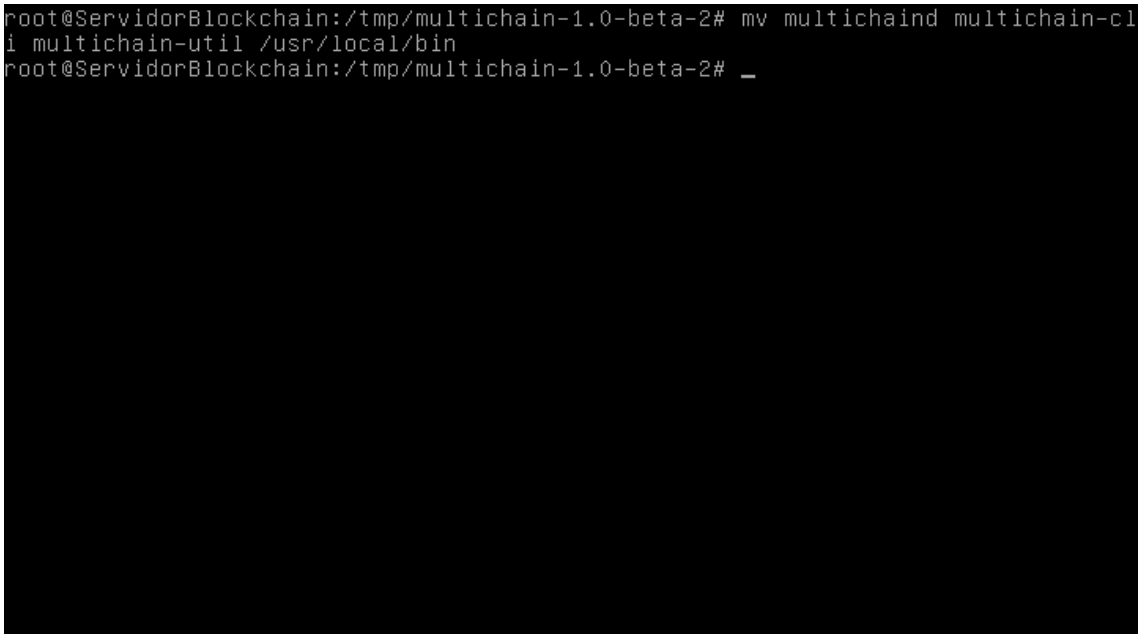

Ilustración 12. Movimiento de ficheros a /usr/local/bin

Finalmente, ejecutamos "exit" para volver al usuario anterior.

Ahora, ya se dispone del software encargado de la gestión de la cadena de bloques en las tres máquinas que la formarán. No obstante, y según se ha comentado anteriormente, cada una de las máquinas posee un rol diferente. Concretamente, la máquina "Servidor Blockchain" será la encargada de administrar toda la red blockchain, mientras que las máquinas "C1.1" y "C1.2" únicamente se encargarán de agregar transacciones con los hashes de los mensajes de registro. Por último, la máquina "Servidor Syslog" es la encargada de recibir los mensajes de registro y procesarlos para la interpretación por parte de la aplicación de eventos de seguridad.

5.1.2 Creación de la cadena de bloques

Tal y como hemos comentado, el host "Servidor Blockchain" es el encargado de gestionar la cadena de bloques, por lo que será en dicho host donde se creará la nueva red blockchain. Para ello, y al igual que en el apartado anterior, se empleará la línea de comandos para tal fin, aunque en este caso se accederá vía *ssh*.

El primer paso es ejecutar el comando "multichain-util create tfg-chain", tal y como se observa a continuación:

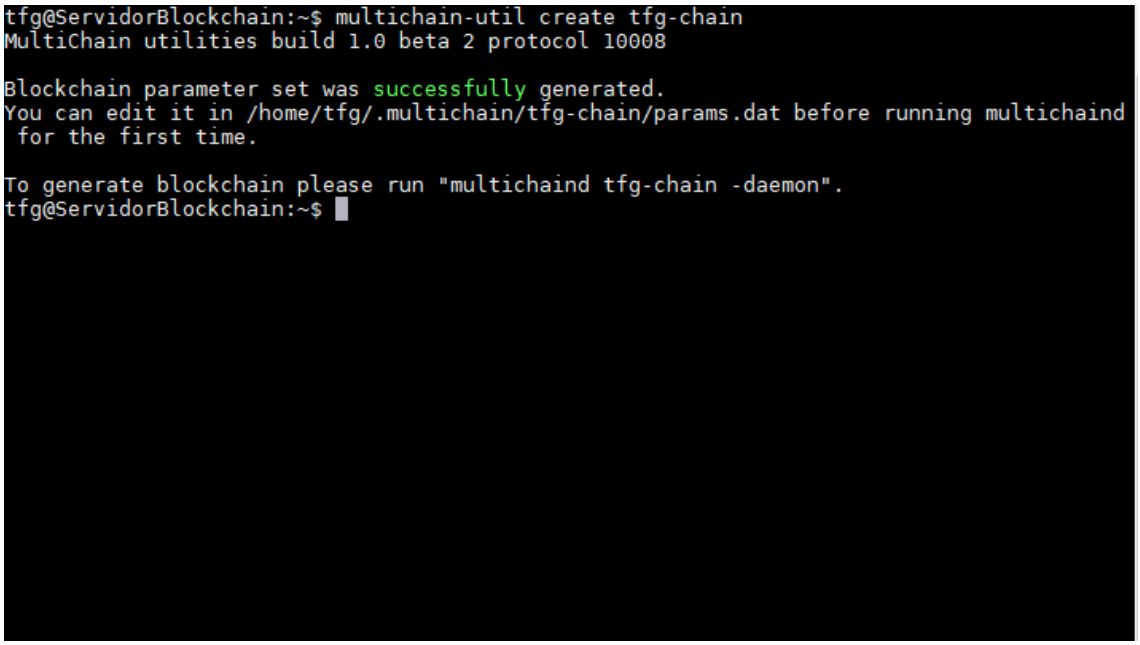

Ilustración 13. Creación cadena de bloques tfg-chain

En este punto, se ha creado la cadena de bloques en la máquina. Sin embargo, no se ha inicializado puesto que existe la posibilidad de modificar la configuración de la misma. La configuración de la cadena de bloques se encuentra en un fichero llamado "params.dat", y este se encuentra en la carpeta "~/.multichain/tfg-chain/params.dat".

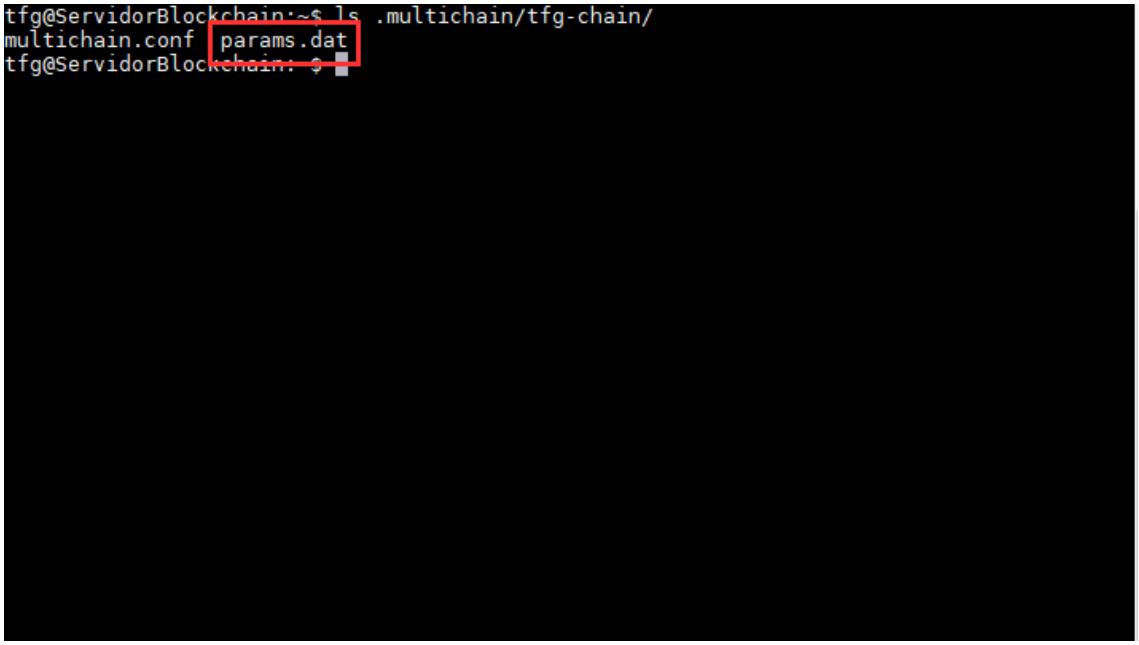

Ilustración 14. Localización del fichero de configuración "params.dat"

En el interior de dicho fichero, se incluyen múltiples valores de configuración y que, en función del valor establecido, permitirá establecer el comportamiento de la cadena de bloques de la cual depende.

```
# ==== MultiChain configuration file ====

# Created by multichain-util
# Protocol version: 10008


# The following parameters can only be edited if this file is a prototype of another con
figuration file.
# Please run "multichain-util clone tfg-chain <new-network-name>" to generate new networ
k.


# Basic chain parameters

chain-protocol = multichain            # Chain protocol: multichain (permissions, nativ
e assets) or bitcoin
chain-description = MultiChain tfg-chain # Chain description, embedded in genesis block
coinbase, max 256 chars.
root-stream-name = root                # Root stream name, blank means no root stream.
root-stream-open = true                # Allow anyone to publish in root stream
chain-is-testnet = false               # Content of the 'testnet' field of API response
s, for compatibility.
target-block-time = 15                 # Target time between blocks (transaction confir
mation delay), seconds. (2 - 86400)
maximum-block-size = 8388608           # Maximum block size in bytes. (1000 - 100000000
0)
```

Ilustración 15. Fragmento del fichero de configuración "params.dat"

A continuación, se detallan los principales parámetros del fichero:

- chain-protocol: Determina el protocolo a emplear, bien el protocolo Multichain, el cual extiende el protocolo Bitcoin, o el propio protocolo Bitcoin.

- anyone-can-*: Conjunto de parámetros empleado para la aplicación de restricciones en la utilización de la blockchain a partir de permisos. Es decir, establece si cualquier equipo puede realizar ciertas acciones a la cadena de bloques o, por el contrario, es necesario el establecimiento de permisos para cada uno de los nodos en cada una de las acciones a realizar.

- target-block-time: Tiempo medio, en segundos, entre bloques. Por lo tanto, este será el tiempo hasta que una transacción esté confirmada.

- máximum-block-size: Tamaño máximo, en bytes, que puede tener un bloque.

- skip-pow-check: Omite la comprobación de si los hashes del bloque demuestran suficiente prueba de trabajo.

- pow-minimum-bits: Dificultad inicial de la prueba de trabajo, es decir, el número de bits necesarios para calcular los hashes.

- target-adjust-freq: Frecuencia para ajustar el nivel de dificultad de prueba de trabajo, medido en segundos.

- default-network-port: Puerto TCP por defecto utilizado para la comunicación *peer-to-peer* entre los nodos de la cadena de bloques.

- default-rpc-port: Puerto TCP utilizado para realizar las llamadas JSON-RPC.

En este caso, se emplearán los valores por defecto al tratarse de un entorno reducido y dirigido a un fin muy específico, ya que la cadena de bloques no se va a emplear para transacciones de activos sino para el almacenamiento y recuperación de datos.

El siguiente paso que se debe realizar es la inicialización de la cadena de bloques que se ha creado con la acción anterior. Para ello, es necesario ejecutar el comando "multichaind tfg-chain -daemon".

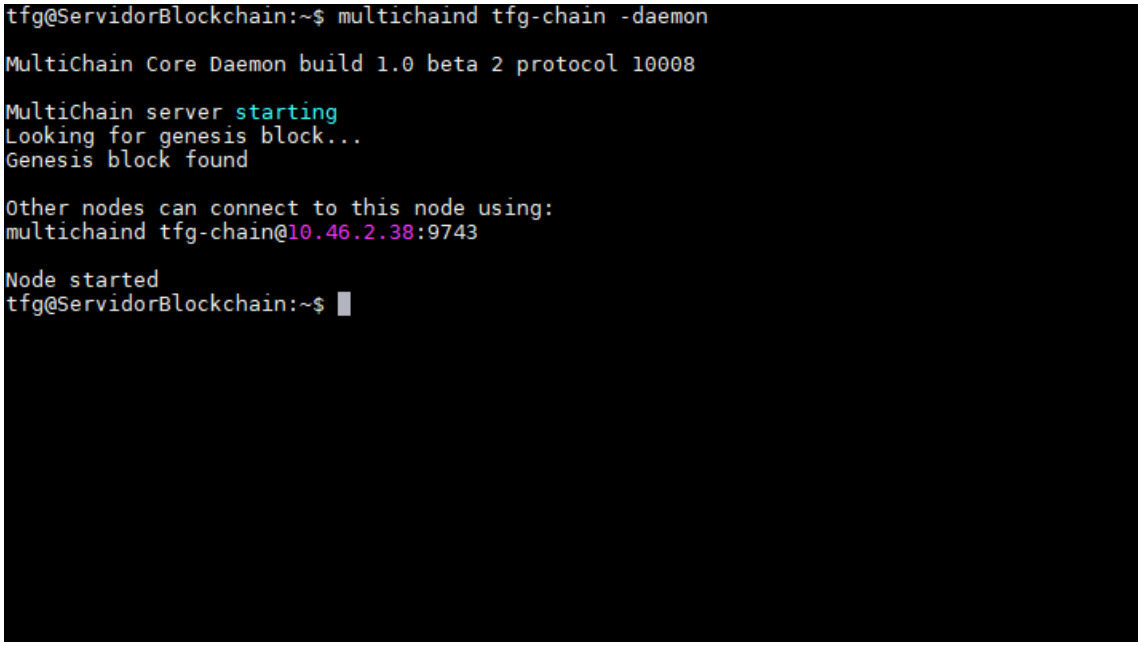

Ilustración 16. Inicialización de la cadena de bloques "tfg-chain"

Si el servidor se ha inicializado correctamente, se mostrará un mensaje en el que se indica que ha sido encontrado el bloque génesis y, después de unos segundos, que el nodo se ha iniciado "Node started". Adicionalmente, se muestra la dirección de dicho nodo, la cual deberán emplear el resto de nodos para conectarse a la cadena creada "tfg-chain". En la Ilustración 17 se remarca la dirección de nodo del host "Servidor Blockchain".

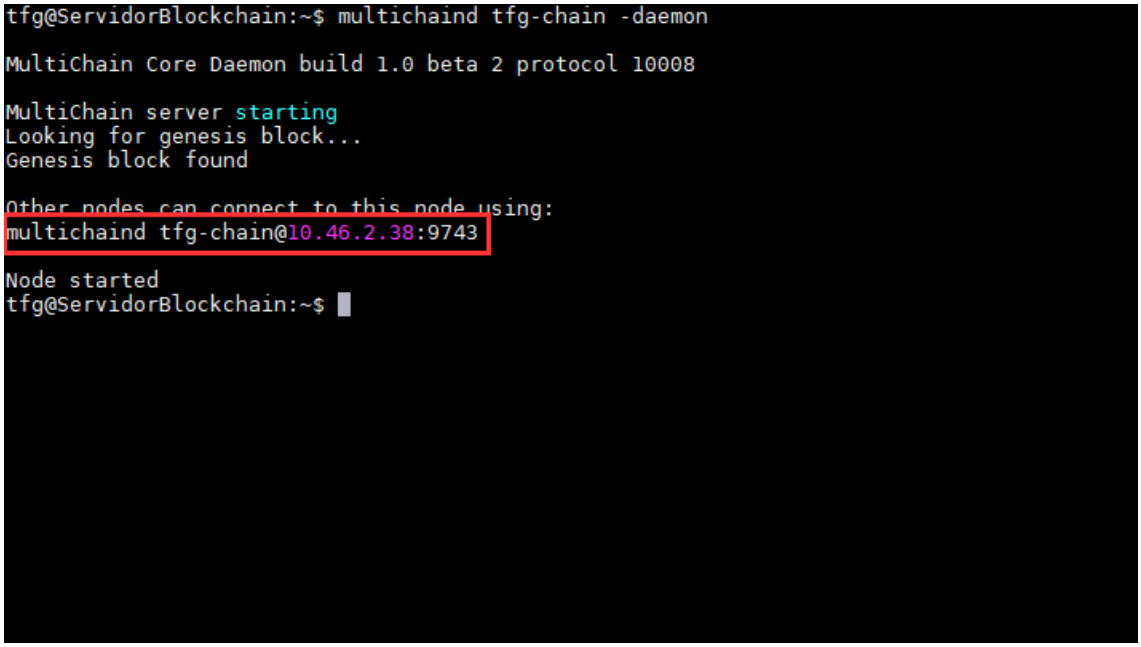

Ilustración 17. Dirección de nodo del host "Servidor Blockchain"

En este punto ya se ha inicializado la red blockchain y se ha establecido el host "Servidor Blockchain" como nodo inicial de la misma, por lo que a partir de ahora este nodo administra la red en lo que a creación de flujos de datos y gestión de usuarios se refiere.

5.1.3 Conexión de los clientes a la cadena de bloques

Tal y como se ha comentado en apartados anteriores, la característica principal de una red blockchain es la validación de las transferencias de datos digitales, ya que no requiere de un intermediario centralizado que identifique y certifique la información, sino que está replicada en diversos nodos independientes entre sí. Por lo tanto, y con el fin de generar un sistema de autenticidad fiable y que cumpla con la característica principal, es necesario añadir diversos clientes.

Siguiendo con el escenario propuesto, el número de clientes que se van a conectar a la cadena de bloques es dos, es decir, las máquinas "C1.1" y "C1.2" serán añadidas a dicha red. Para realizar esta acción, el acceso se realiza mediante ssh.

El primer paso que se debe realizar es utilizar el terminal para ejecutar el comando "multichaind tfg-chain@10.46.2.38:9743".

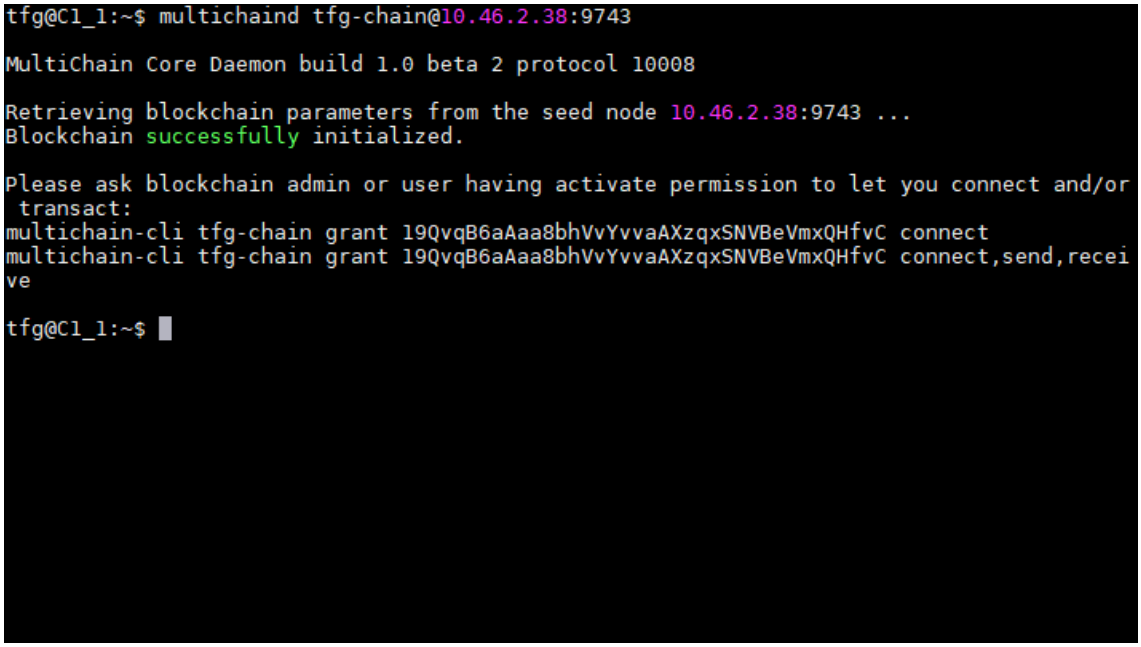

Ilustración 18. Intento de conexión de "C1.1" como nodo de la cadena de bloques

Mediante este comando, el host "C1.1" procede a conectarse a la cadena de bloques "tfg-chain" en el host "Servidor Blockchain", es decir, intenta establecerse como uno de los nodos que forman la red blockchain. Sin embargo, y dado que hemos establecido el parámetro de configuración como "anyone-can-connect = false", es necesario que el nodo encargado de gestionar el blockchain, "Servidor Blockchain", otorgue los permisos necesarios al host "C1.1".

En la Ilustración 18 se observa como al propio host "C1.1" se le ha otorgado una dirección de monedero, la cual se emplea para identificar al host.

Para otorgar los permisos de conexión a la cadena de bloques "tfg-chain", se ejecuta el comando "multichain-cli tfg-chain grant 19QvqB6aAaa8bhVvYvvaAXzqxSNVBeVmxQHfvC connect" en el nodo "Servidor Blockchain", puesto que como ya hemos comentado, es el encargado de administrar el blockchain.

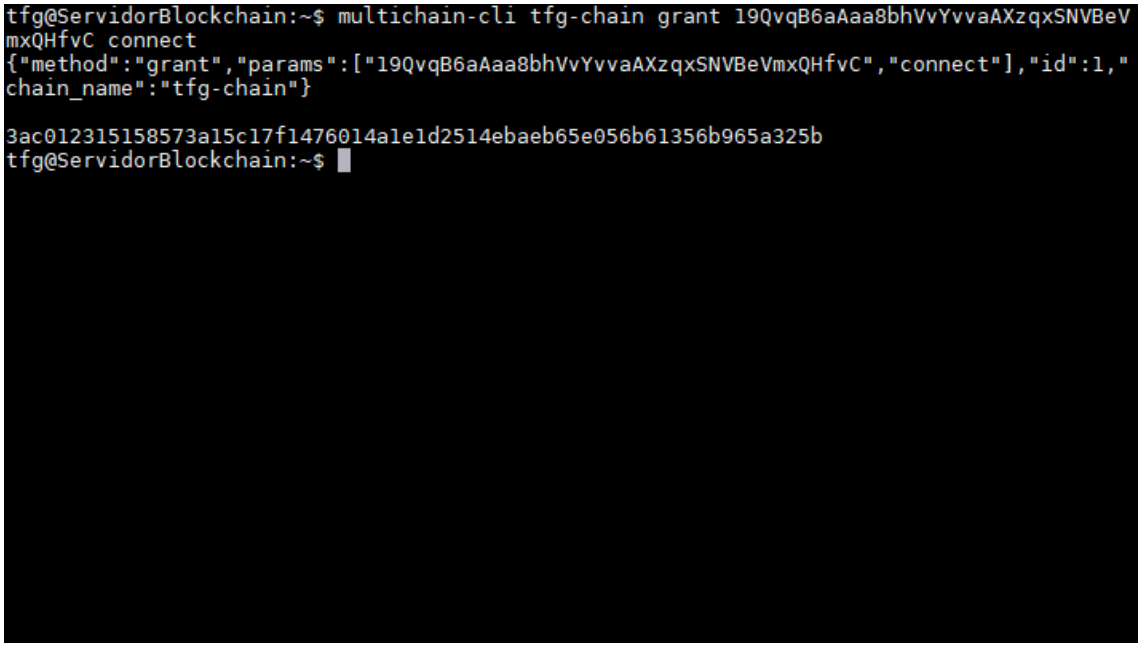

Ilustración 19. Otorgación de permisos al nodo "C1.1"

Una vez se ha otorgado el permiso de conexión, es necesario ejecutar el comando "multichaind tfg-chain -daemon" para iniciar el nuevo nodo y conectarse al ya existente en "Servidor Blockchain".

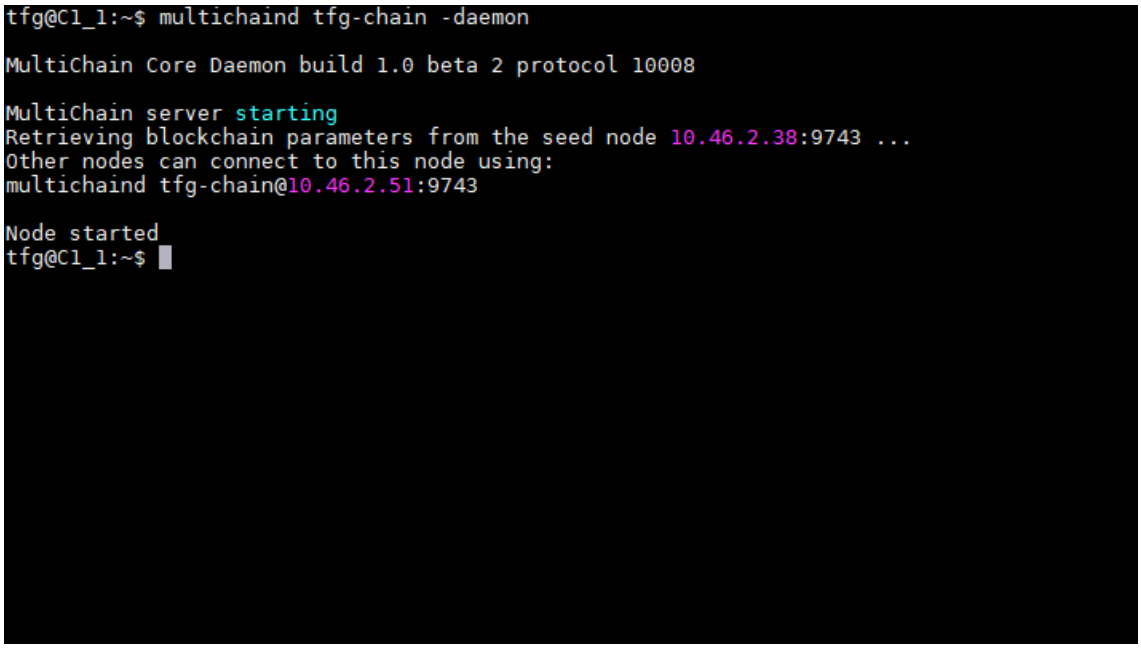

Ilustración 20. Establecimiento de conexión con el nodo principal

Si el servidor se ha inicializado correctamente, se mostrará un mensaje en el que se indica que se han recibido los parámetros de configuración de la cadena de bloques "tfg-chain" desde el nodo principal y, después de unos segundos, que el nodo se ha iniciado "Node started". Adicionalmente, se muestra la dirección de

dicho nodo, la cual deberán emplear el resto de nodos en caso de requerir la conexión con este nodo. En la Ilustración 21 se remarca la dirección de nodo del host "Servidor Blockchain".

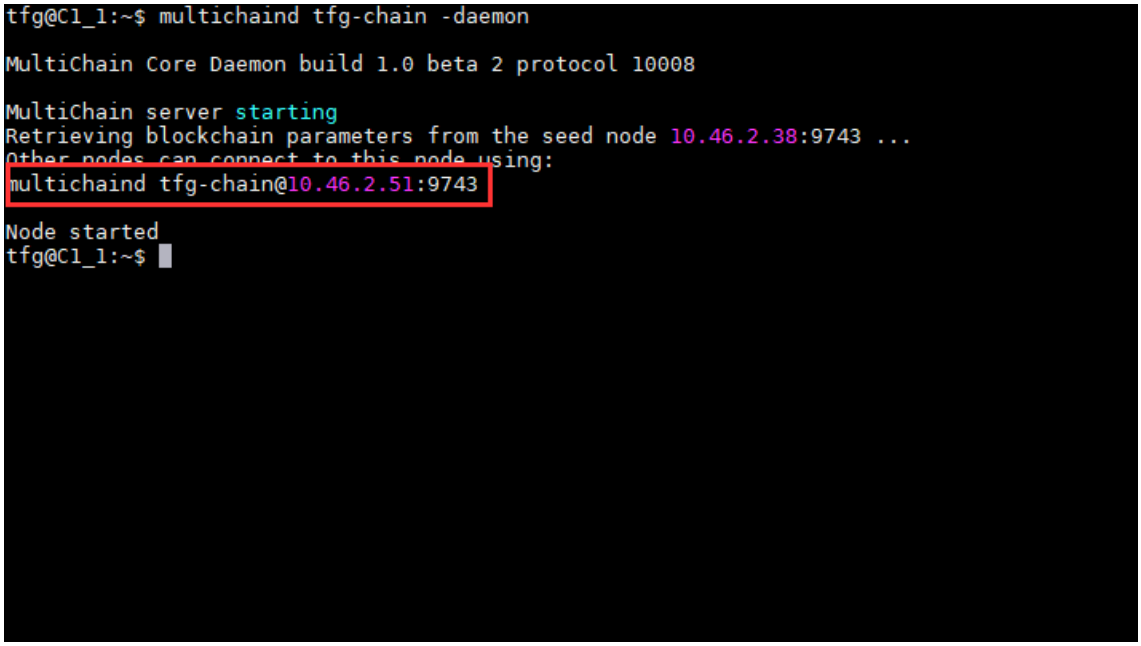

Ilustración 21. Dirección del nodo "C1.1"

En este punto, la red blockchain está establecida a partir de los nodos "Servidor Blockchain" y "C1.1".

Con lo que respecta al segundo cliente, el host "C2.2", el proceso de configuración requiere de las mismas acciones que las descritas en este apartado, por lo que se ha omitido su muestra con el fin de no añadir información redundante.

Una vez añadido el cliente, el escenario actual es el que se observa en la Ilustración 22.

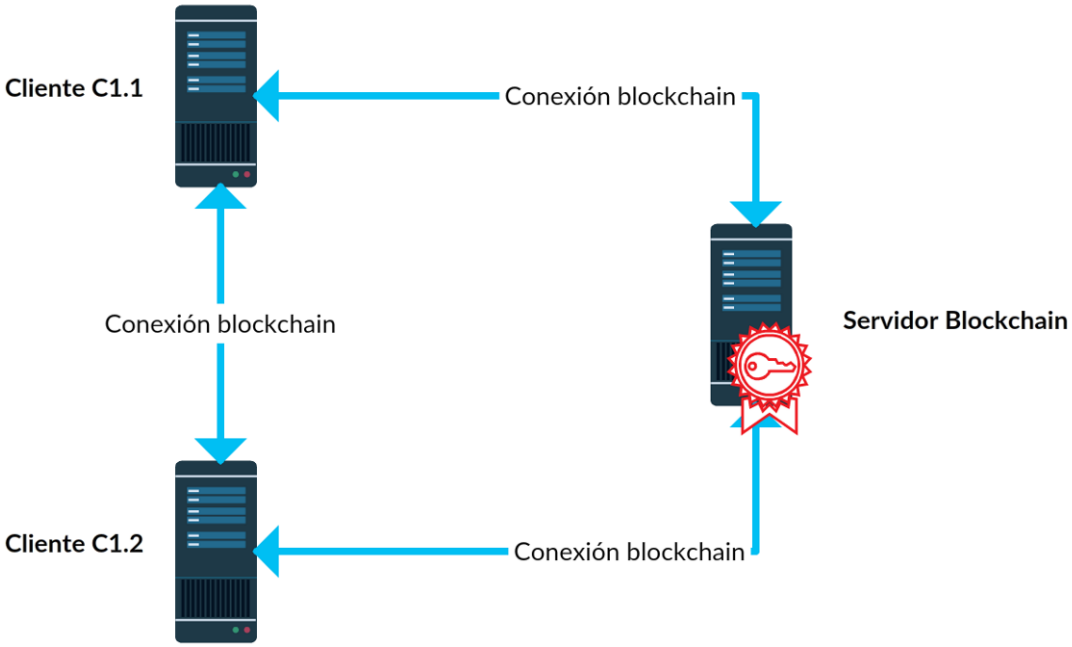

Ilustración 22. Esquema de red blockchain "tfg-chain"

5.1.4 Almacenamiento de datos y recuperación

Según se ha comentado anteriormente, el uso que se le va a dar a la cadena de bloques creada es el almacenamiento de datos y su posterior recuperación con el fin de garantizar la autenticidad e integridad de los mensajes de registro transmitidos entre las máquinas cliente y el servidor "Servidor Syslog".

La creación del stream, que se encargará de almacenar los hashes de los logs procedentes de las máquinas clientes, se realizará empleando el blockchain "tfg-chain". Adicionalmente, y dado que únicamente el nodo administrador "Servidor Blockchain" posee los permisos requeridos para ello, la creación se realizará empleando dicho host.

Para ello, ejecutaremos el comando "multichain-cli tfg-chain create stream logHashes false '{"description":"[tfg-chain LogHashes Stream] Todos los hashes de los logs recibidos por los clientes se incluiran aqui"}'".

Ilustración 23. Creación del flujo "logHashes"

Destacar que el valor "false" hace referencia al hecho de que cualquier nodo debe poseer de forma explícita el permiso de escritura en dicho stream para poder publicar en él. De esta forma, se incrementa la seguridad al restringir los posibles nodos que pueden escribir sobre el flujo y se obtiene un control total del mismo. Para observar los permisos del flujo ejecutamos el comando "multichain-cli tfg-chain listpermissions logHashes.*"

Si se observa la Ilustración 24 únicamente el nodo administrador tiene la posibilidad de escribir en el flujo, así como de administrarlo.

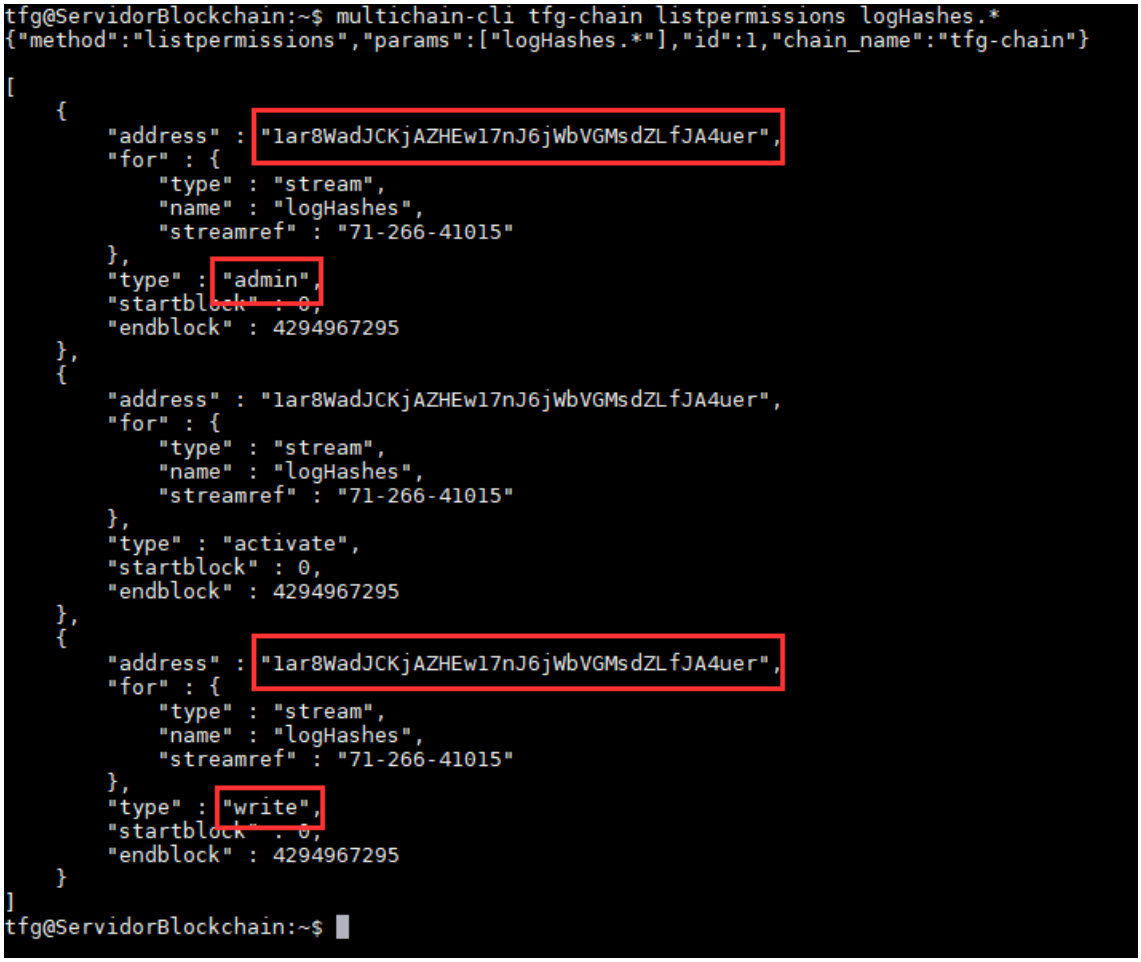

Ilustración 24. Permisos del nodo administrador sobre el flujo "logHashes"

El siguiente paso es visualizar si el flujo se encuentra accesible desde las máquinas cliente. Para ello, ejecutamos el comando "multichain-cli tfg-chain liststreams" desde las máquinas "C1.1" y "C1.2":

Ilustración 25. Visualización del stream *"logHash" desde el nodo "C1.1"*

A continuación, y dado que los nodos cliente no poseen el permiso de emisión y recepción de transacciones en el blockchain "tfg-chain", se les debe otorgar puesto que el stream "logHash" se encuentra en dicha cadena de bloques. Para asignar sendos permisos, se ejecuta el comando "multichain-cli tfg-chain grant 19QvqB6aAaa8bhVvYvvaAXzqxSNVBeVmxQHfvC receive,send" en el nodo administrador para otorgar el permiso de emisión y recepción de transacciones al nodo "C1.1". Se ejecutará este mismo comando, cambiando la dirección del monedero, para asignar dichos permisos al cliente "C2.2".

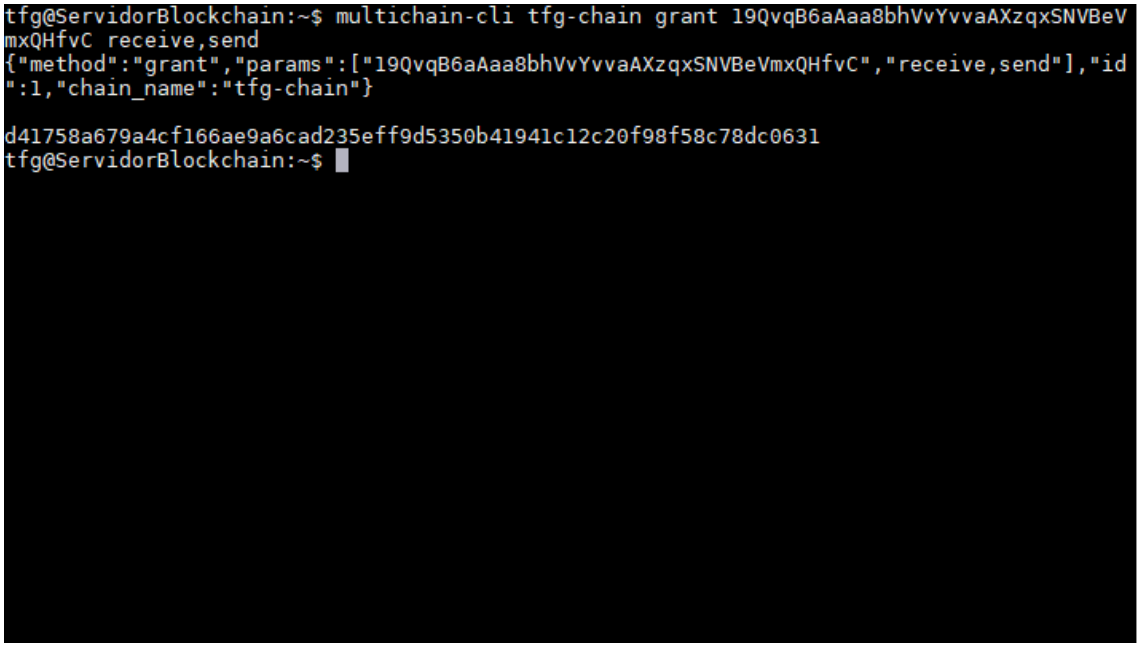

Ilustración 26. Asignación de permisos de emisión y recepción en "tfg-chain" al nodo "C1.1"

Posteriormente, se debe asignar el permiso explícito de escritura en el flujo "logHashes" a ambos clientes, por lo que se ejecuta el comando "multichain-cli tfg-chain grant 19QvqB6aAaa8bhVvYvvaAXzqxSNVBeVmxQHfvC logHashes.write".

De nuevo, y al igual que en la asignación de permisos anterior, dicho comando se debe ejecutar para el cliente "C1.2" incluyendo la dirección del monedero del mismo.

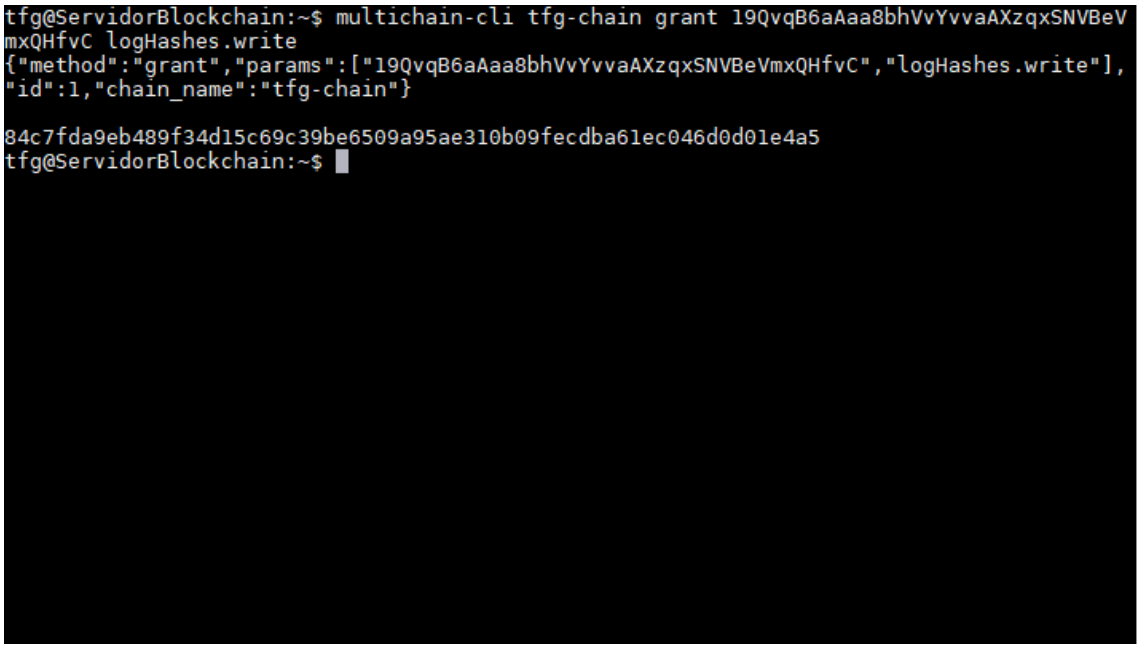

Ilustración 27. Asignación del permiso de escritura en "logHashes" al nodo "C1.1"

El último paso que se debe realizar es la suscripción de los nodos cliente “C1.1” y “C1.2” al flujo “logHashes” con el fin de publicar nuevos ítems, los cuales, en este caso serán los hashes de los logs. Mediante la ejecución del comando “multichain-cli tfg-chain subscribe logHashes” en ambos clientes, se realiza la suscripción al flujo mencionado.

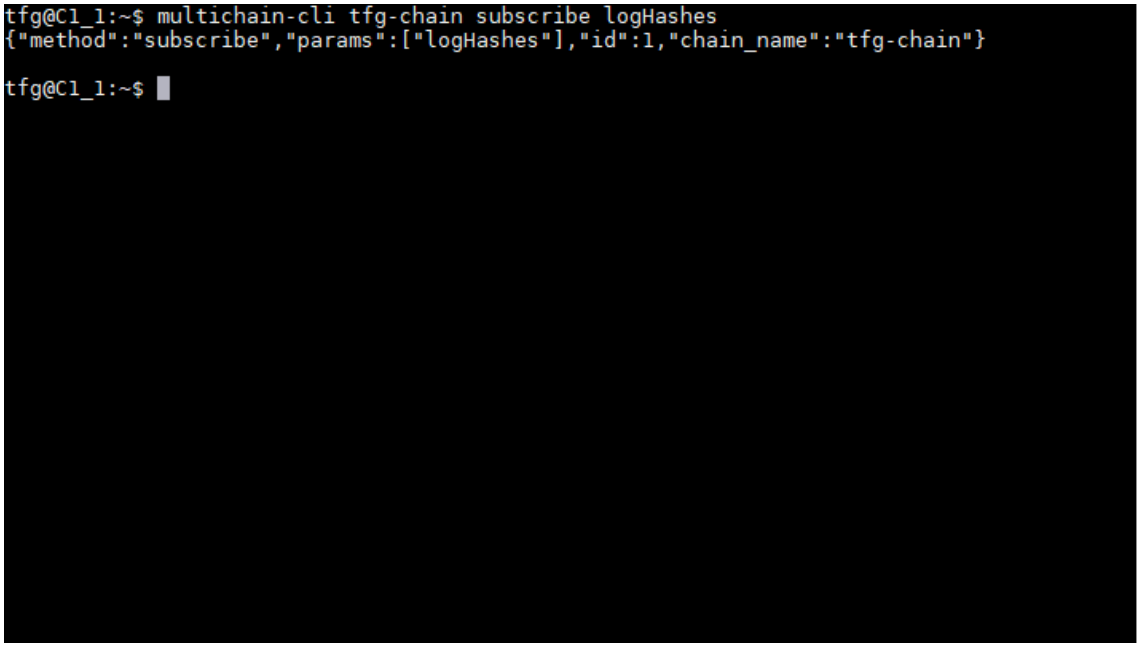

Ilustración 28. Suscripción del cliente “C1.1” al flujo “logHashes”

Finalmente, mediante el comando “multichain-cli tfg-chain listpermissions logHashes.*” se comprueban los permisos de cada uno de los clientes en el flujo.

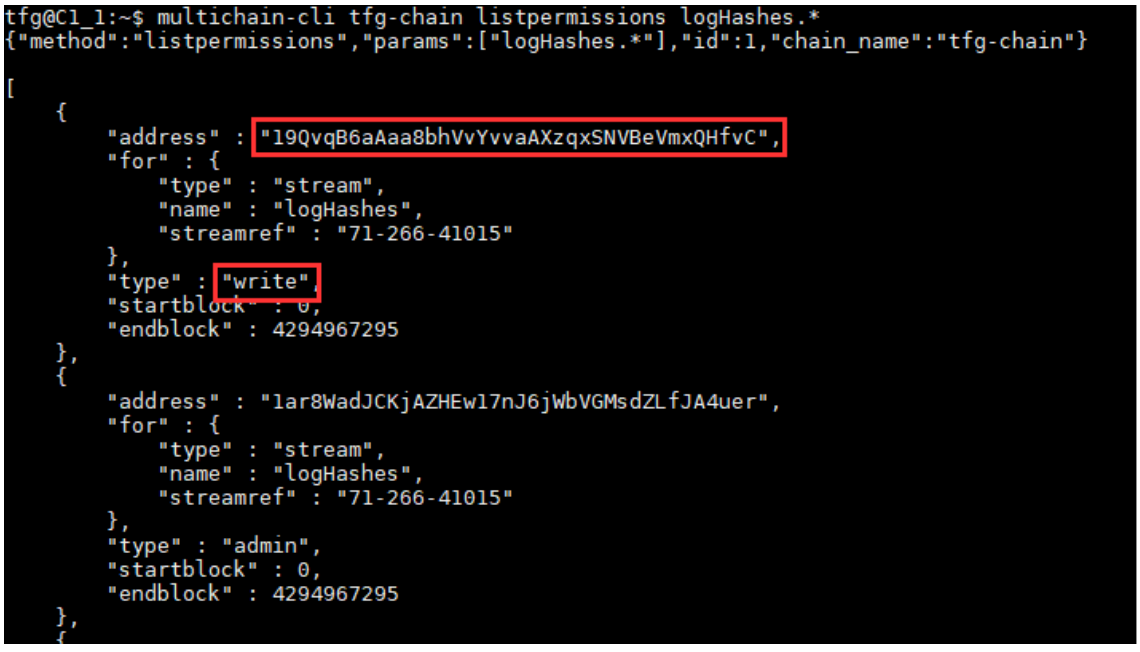

Ilustración 29. Permiso otorgado de escritura al nodo “C1.1”

En este punto, tanto los nodos cliente “C1.1” y “C1.2” como el nodo administrador “Servidor Blockchain” poseen la capacidad de realizar transacciones con ítems, en cuya definición se encuentran los hashes de los logs, empleando la cadena de bloques “tfg-chain”.

5.1.5 Minado colaborativo entre los nodos

Para la verificación de las transacciones, se empleará el minado colaborativo entre todos los nodos participantes en la cadena de bloques a partir de un minado de tipo Round-Robin. De esta forma, se evita la posibilidad de que un mismo nodo monopolice dicho proceso. Para ello, se deberán otorgar los permisos de minado a todos los nodos pertenecientes en la red desde el nodo administrador “Servidor Blockchain” ejecutando el comando “multichain-cli tfg-chain grant 19QvqB6aAaa8bhVvYvvaAXzqxSNVBeVmxQHfvC mine”. En este caso, se ha asignado dicho permiso al nodo C1.1.

Finalmente, en todos los nodos de la red se ejecutará el comando “multichain-cli tfg-chain setruntimeparam miningturnover 1” para maximizar el grado de aleatoriedad del minado.

Ilustración 30. Ejecución del comando para maximizar la aleatoriedad del minado

En este punto, el minado de los bloques se realizará de forma colaborativa entre todos los nodos de tal forma que dicho proceso será realizado de forma equitativa. Esto

permite garantizar la independencia de la cadena de bloques puesto que ninguno de los nodos obtiene el control total sobre el proceso de minado.

5.2 Cliente *pychain*

El cliente *pychain* es un script escrito en Python mediante el cual, las máquinas "C1.1" y "C1.2", insertan los hashes de los logs en el stream *logHashes* y con ello, en la red blockchain, previamente creada en el apartado 5.1 Instalación y configuración MultiChain, llamada "tfg-chain".

Previamente a la escritura del código que conforma el script en su totalidad, se ha definido el flujo de actividad que pychain debe seguir a partir del diseño un diagrama UML en el que se muestra el tratamiento de la información con el fin de generar e insertar el hash correspondiente al log recibido.

En la Ilustración 31, se observa el diagrama generado y que sirve de base para la escritura del código que conforma el script.

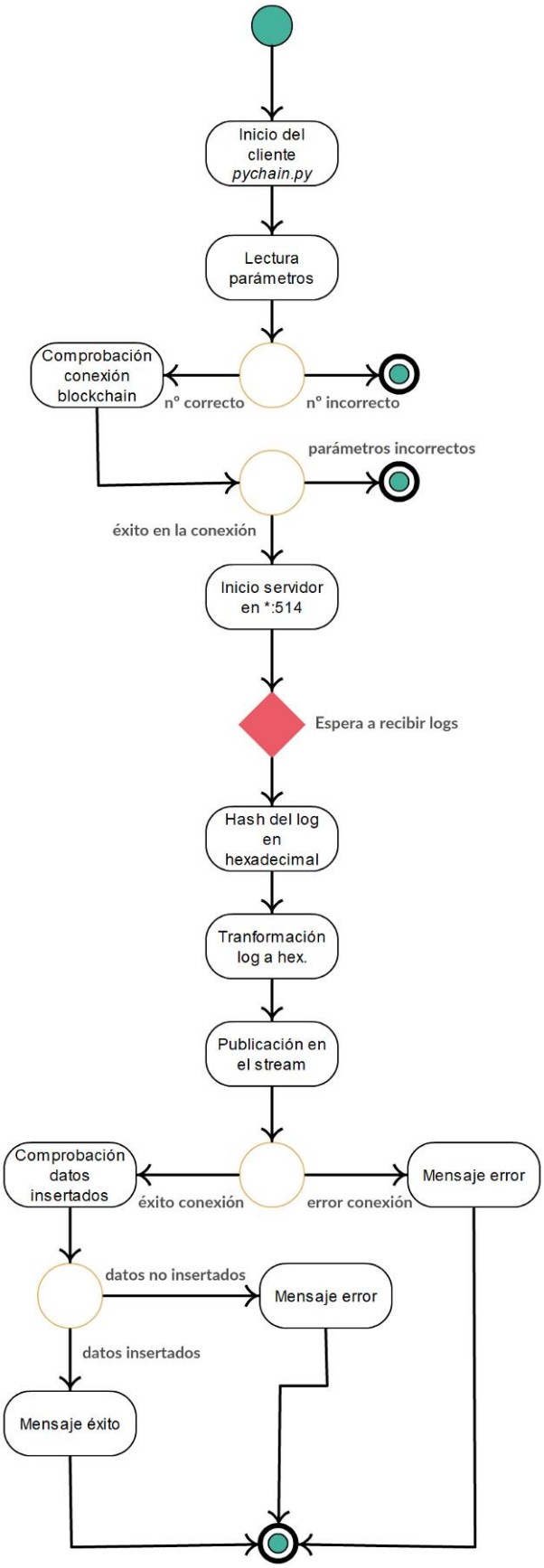

Ilustración 31. Diagrama de flujo UML del script pychain

En dicho diagrama se observa que, en primer lugar, el usuario procede a la iniciación del script al cual se le introducen una serie de parámetros que se utilizan posteriormente para establecer la conexión con la red blockchain. Dichos parámetros son necesarios para el correcto funcionamiento del mismo, por lo que se comprueba si el número de parámetros introducidos coincide con el esperado. En caso afirmativo, se sigue con el flujo esperado. En caso contrario, se muestra un mensaje de ayuda y se finaliza la ejecución del script.

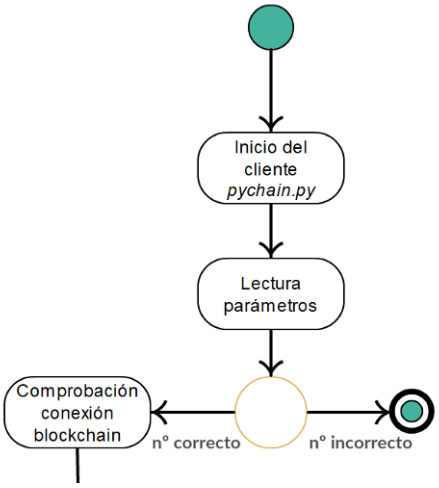

Ilustración 32. Lectura de parámetros y comprobación del número de estos

Si se prosigue con el flujo, el siguiente paso es la comprobación funcional de los parámetros, de tal forma que se verifica si es posible establecer la conexión con la cadena de bloques indicada. Si no es posible establecer dicha conexión, se finaliza la ejecución del mismo. En caso afirmativo, el siguiente paso es iniciar el servidor que recibirá los logs para obtener el hash de estos y añadirlos al stream. Para iniciar el servidor se realiza la asignación del puerto UDP 514 en todas las interfaces del host.

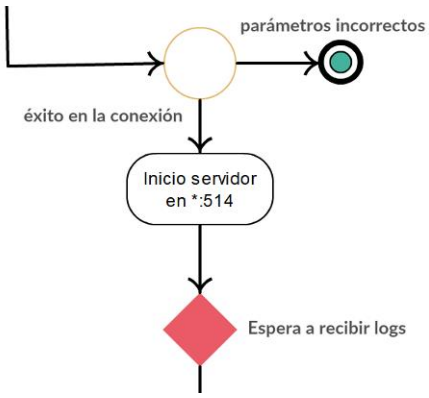

Ilustración 33. Comprobación de la conexión con los parámetros

Cuando el servidor se ha iniciado, se mantiene a la espera de recibir los logs mediante el protocolo syslog de forma concurrente de manera que múltiples equipos puedan utilizar “pychain” como servidor centralizado para el envío de logs.

Destacar que, aunque “pychain” ejecuta un servidor para el recibimiento de logs, se trata de un cliente de la red de blockchain, de ahí que se trate como tal.

Tras recibir el log en el puerto UDP 514, el siguiente paso es obtener el hash SHA1 de dicho log y transformarlo a hexadecimal, ya que es el formato requerido para realizar la inserción en el stream.

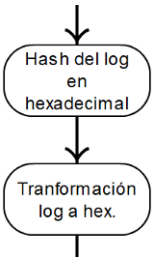

Ilustración 34. Obtención del hash y transformación a hexadecimal

Posteriormente, se realiza la publicación del valor hexadecimal del hash en el stream “logHashes”. Para ello, se emplea dicho dato como clave y adicionalmente, se inserta el log en forma hexadecimal como valor. De esta forma, es posible mantener una relación entre el hash del log y el log propiamente dicho, tal y como se ha comentado en el apartado 2.

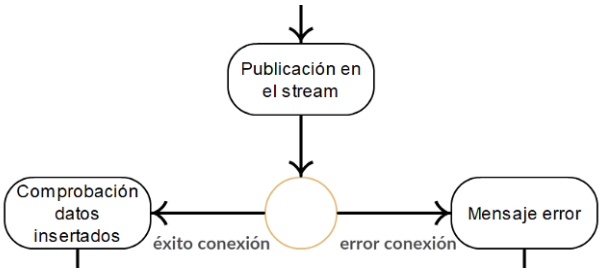

Ilustración 35. Publicación de la información

Finalmente, se comprueba si se ha realizado de forma correcta la transacción, en cuyo caso se muestra un mensaje de éxito. Si, por el contrario, no se ha realizado de forma correcta la transacción, y con ello, la inserción de los datos en el stream, se muestra un mensaje de error.

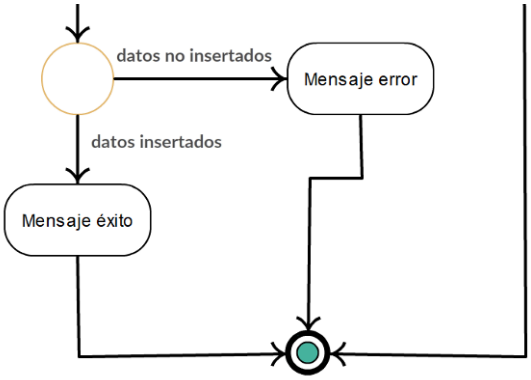

Ilustración 36. Comprobación si inserción es correcta

Tal y como se observa en la descripción realizada del diagrama de flujo, la ejecución del script consta de cinco fases completas y diferenciadas, en las cuales, cada una de ellas, realiza una función específica que en su totalidad permiten la obtención de los valores requeridos para la inserción de los mismos empleando la tecnología blockchain.

Dicho diagrama de flujo UML, se refleja de la siguiente forma en el script que se muestra en la Ilustración 42 y del que se detallan cada una de las fases descritas:

1) Fase 1: Comprobación de la cantidad de parámetros y lectura de los mismos:

```
if __name__ == "__main__":
    try:

        if len(sys.argv) < 5:
            print("Usage: %s rpcUser rpcPasswd rpcPort chainName" % sys.argv[0])
            print("rpcUser and rpcPasswd are stored into ~/.multichain/<chainName>/multichain.conf")
            print("rpcPort is set as default_rpc_port into ~/.multichain/<chainName>/params.dat")
            sys.exit(0)

        rpcuser = sys.argv[1]
        rpcpasswd = sys.argv[2]
        rpchost = '127.0.0.1'
        rpcport = sys.argv[3]
        chainname = sys.argv[4]
```

Ilustración 37. Código del script correspondiente a la fase 1

En el código observado, se emplea la función “len()” para comprobar el tamaño de los parámetros pasados por línea de comandos. Dicha función se encuentra en el interior de una instrucción condicional “if” en la que si el valor es menor a cinco se muestra el mensaje de ayuda y se finaliza la ejecución del script con “sys.exit(0)”.

Si el número de parámetros es el correcto, se realiza la lectura de estos mediante “sys.argv[i]”, donde “i” es la posición de cada uno de ellos en la ejecución del script en línea de comandos.

Cada uno de los parametros leídos, es asignado a una variable diferente.

2) Fase 2: Comprobación de la conexión empleando los parámetros obtenidos e iniciación del servidor en el puerto UDP 514 en todas las interfaces de la máquina:

```
    api = sav.Savoir(rpcuser, rpcpasswd, rpchost, rpcport, chainname)

    try:
        api.getinfo()
    except:
        print("Please, check if blockchain server is up or your params are set successfully")
        sys.exit(0)

    server = SocketServer.UDPServer((HOST,PORT), SyslogUDPHandler)
    print("[----] Server started [----]")
    server.serve_forever(poll_interval=0.5)
```

Ilustración 38. Código del script correspondiente a la fase 2

Para la interacción del script con la cadena de bloques, se emplea una librería llamada Savoir y que es inicializada con la llamada “sav.Savoir(rpcuser, rpcpasswd, rpchost, rpcport, chainname)”, en la cual se utilizan los parámetros leídos en la fase anterior. Dicha llamada corresponde con la conexión entre el script y la cadena de bloques, por lo que se asigna a la variable “api”.

El siguiente paso, es comprobar la conexión, por lo que se realiza una llamada al método “getinfo()”. Este método devuelve información sobre los distintos parámetros de configuración de la cadena de bloques. Por lo tanto, y en el caso de que los parámetros leídos en la fase anterior sean incorrectos, la llamada devolverá una excepción la cual es tratada en el apartado “except”. El tratamiento consiste en mostrar un mensaje por pantalla y finalizar la ejecución del script.

3) Fase 3: Obtención del hash del log recibido y conversión al formato hexadecimal:

```
def handle(self):
    data = bytes.decode(self.request[0].strip())
    socket = self.request[1]
    hex_dig = hashlib.sha512(str(data)).hexdigest()
```

Ilustración 39. Código del script correspondiente a la fase 3

Para realizar esta fase, se emplea la librería nativa de Python llamada "hashlib", ya que permite obtener el hash SHA1 del log y su formato en valor hexadecimal. La obtención del hash SHA1 se realiza mediante el método "hashlib.sha512()", al cual se concatena el método "hexdigest()" para obtener su valor en formato hexadecimal. Dicho valor se almacena en la variable "hex_digest" para poder ser tratado de una forma más sencilla en fases posteriores.

4) Fase 4: Publicación del hash y del log, ambos en formato hexadecimal, en el stream "logHashes":

```
try:
    request = api.publish("logHashes",hex_dig, data.encode("hex"))
```

Ilustración 40. Código del script correspondiente a la fase 4

La publicación del valor hexadecimal del hash se realiza mediante la función de la librería Savoir, "publish()". A dicha función, se le pasan diversos parámetros; el primero de ellos, es el stream en el cual se va a realizar la inserción de los datos. El segundo, se corresponde con la clave de los datos insertados, en cuyo caso se ha definido como el hash en formato hexadecimal del log a insertar. Dicho log se inserta en formato hexadecimal en el tercero de los parámetros mediante "data.encode("hex")".

5) Fase 5: Comprobación de la inserción realizada:

```
if type(request) is not dict:
    print("Successfully inserted into stream. TxID: " + str(request))
else:
    error_message = request['error']['message']
    print "Ups! " + str(error_message)
```

Ilustración 41. Código del script correspondiente a la fase 5

En esta última fase, se realiza la comprobación de la inserción a partir de la respuesta obtenida al realizar la inserción, ya que si la publicación en el stream ha sido satisfactoria, la llamada "publish()" devuelve el identificador de la transacción. En caso contrario, la llamada devuelve un json en cuyo interior se encuentra el mensaje de error. Esta comprobación se realiza mediante el condicional "if" a partir del tipo de datos recibido.

Finalmente, se muestra todo el script completo en la siguiente Ilustración 42.

```
#!/usr/bin/env python

## Syslog Server in Python.
##
## This is a tiny syslog server that is able to receive UDP based syslog
## entries on a specified port and send them to a Blockchain network.

import SocketServer
import hashlib
import sys
import imp

sav = imp.load_source('Savoir', '/usr/local/lib/python2.7/dist-packages/Savoir/Savoir.py')

HOST, PORT = "0.0.0.0", 514

api, rpcuser, rpcpasswd, rpchost, rpcport, chainname = "", "", "", "", "", ""

class SyslogUDPHandler(SocketServer.BaseRequestHandler):

    def handle(self):
        data = bytes.decode(self.request[0].strip())
        socket = self.request[1]
        hex_dig = hashlib.sha512(str(data)).hexdigest()
        print(str(self.client_address[0]) + ": " + str(data) + " --> Hash: " + hex_dig)
        try:
            request = api.publish("logHashes",hex_dig, data.encode("hex"))
            if type(request) is not dict:
                print("Successfully inserted into stream. TxID: " + str(request))
            else:
                error_message = request['error']['message']
                print "Ups! " + str(error_message)
        except:
            print("Please, check if blockchain '" + str(chainname) + "' is up")

if __name__ == "__main__":
    try:

        if len(sys.argv) < 5:
            print("Usage: %s rpcUser rpcPasswd rpcPort chainName" % sys.argv[0])
            print("rpcUser and rpcPasswd are stored into ~/.multichain/<chainName>/multichain.conf")
            print("rpcPort is set as default_rpc_port into ~/.multichain/<chainName>/params.dat")
            sys.exit(0)

        rpcuser = sys.argv[1]
        rpcpasswd = sys.argv[2]
        rpchost = '127.0.0.1'
        rpcport = sys.argv[3]
        chainname = sys.argv[4]

        api = sav.Savoir(rpcuser, rpcpasswd, rpchost, rpcport, chainname)

        try:
            api.getinfo()
        except:
            print("Please, check if blockchain server is up or your params are set successfully")
            sys.exit(0)

        server = SocketServer.UDPServer((HOST,PORT), SyslogUDPHandler)
        print("[----] Server started [----]")
        server.serve_forever(poll_interval=0.5)
    except (IOError, SystemExit):
        raise
    except KeyboardInterrupt:
        print ("Crtl+C Pressed. Shutting down.")
```

Ilustración 42. Código del script "pychain.py"

5.3 Instalación y configuración OSSEC

El último software adicional que se instalará en cada uno de los equipos del sistema de autenticidad es OSSEC, un HIDS, que tal y como se ha comentado en el apartado 2, permite procesar los mensajes de log para crear alertas en función del contenido de dicho mensaje.

Destacar que, al tratarse del mismo sistema operativo Linux en los equipos mencionados, la instalación se realiza de igual forma en cada uno ellos. No obstante, y dado que cada uno de los equipos posee un rol diferente, la configuración sí difiere al respecto.

5.3.1 Descarga e instalación - Linux

Para realizar la descarga e instalación del software, es necesario emplear la línea de comandos e iniciar sesión con el usuario *root*. Para ello, ejecutamos el comando "su", tras el cual nos solicitará la contraseña de dicho usuario, tal y como se observa en la Ilustración 43.

Ilustración 43. Ejecución comando "su"

Actualmente, el directorio establecido es "/home/tfg". El siguiente paso es establecer el directorio actual a "/tmp" mediante la ejecución del comando "cd /tmp".

Ilustración 44. Cambio al directorio "/tmp/"

A continuación, mediante el comando "wget -U ossec https://github.com/ossec/ossec-hids/archive/2.9.1.tar.gz" se obtiene la versión 2.9.1 de OSSEC, siendo esta la última versión estable a la hora de escribir este documento.

Ilustración 45. Descarga de OSSEC HIDS 2.9.1

Según lo observado en la Ilustración 45, el fichero se ha descargado comprimido en un formato "tar.gz", por lo que para acceder al contenido es necesario descomprimir dicho fichero. Para ello, ejecutamos el comando "tar -xvzf 2.9.1.tar.gz".

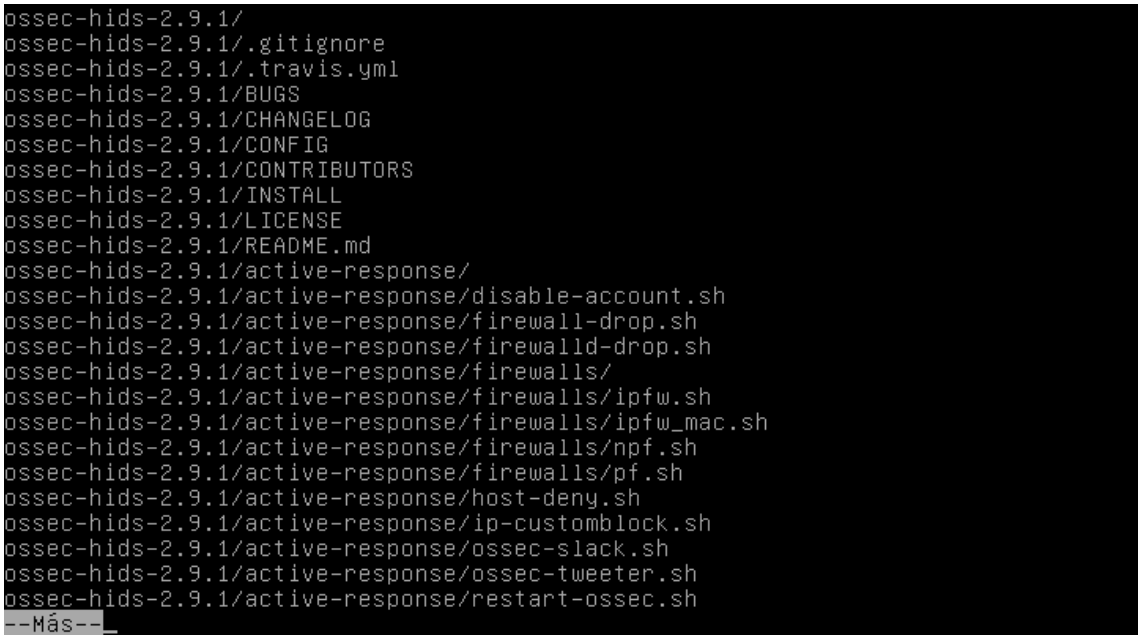

Ilustración 46. Extracción del paquete descargado

El próximo paso a realizar es la instalación de OSSEC. Para ello, únicamente es necesario ejecutar el comando "ossec-hids-2.9.1/install.sh", tras el cual se muestra un asistente para la instalación:

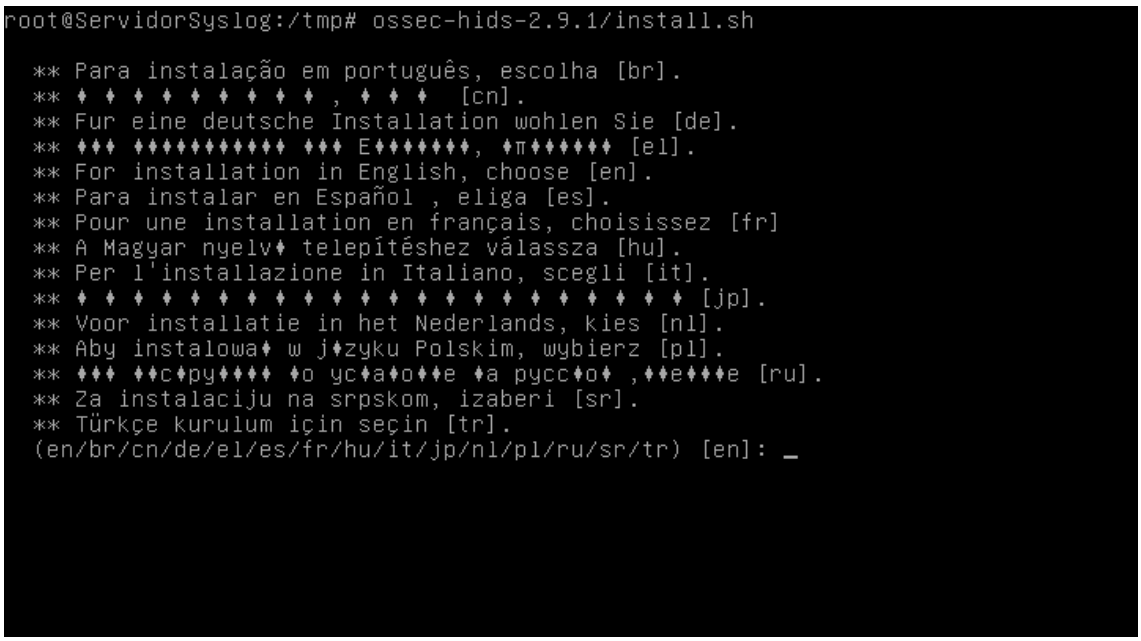

Ilustración 47. Elección del idioma de instalación

Lo primero que se debe realizar es la elección del idioma de dicha instalación. Por defecto viene establecido el idioma inglés, por lo que se presionará la tecla intro para continuar la instalación en el idioma por defecto.

Tras escoger el idioma, se muestra un resumen del equipo en el que se va a realizar la instalación. Se presiona de nuevo la tecla intro para seguir.

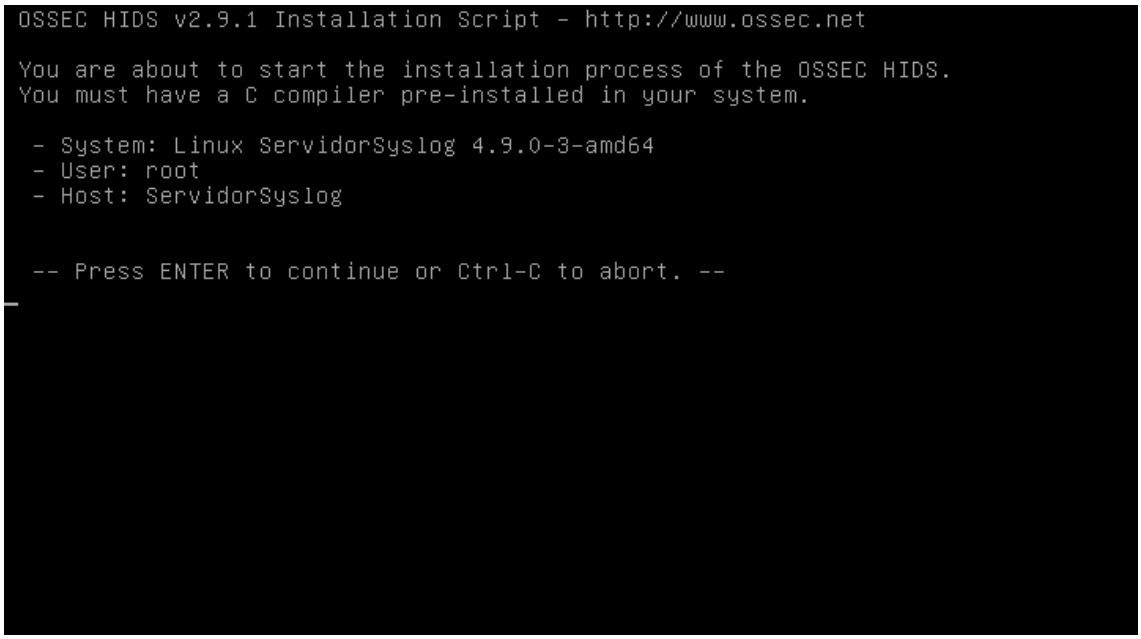

Ilustración 48. Resumen del sistema donde se instalará OSSEC

A continuación, se escoge el tipo de instalación deseada entre las existentes. En este caso, y según las características definidas en el apartado 2 para cada uno de los tipos, se escogerá servidor. Para ello, es suficiente con teclear la palabra "server" y pulsar intro.

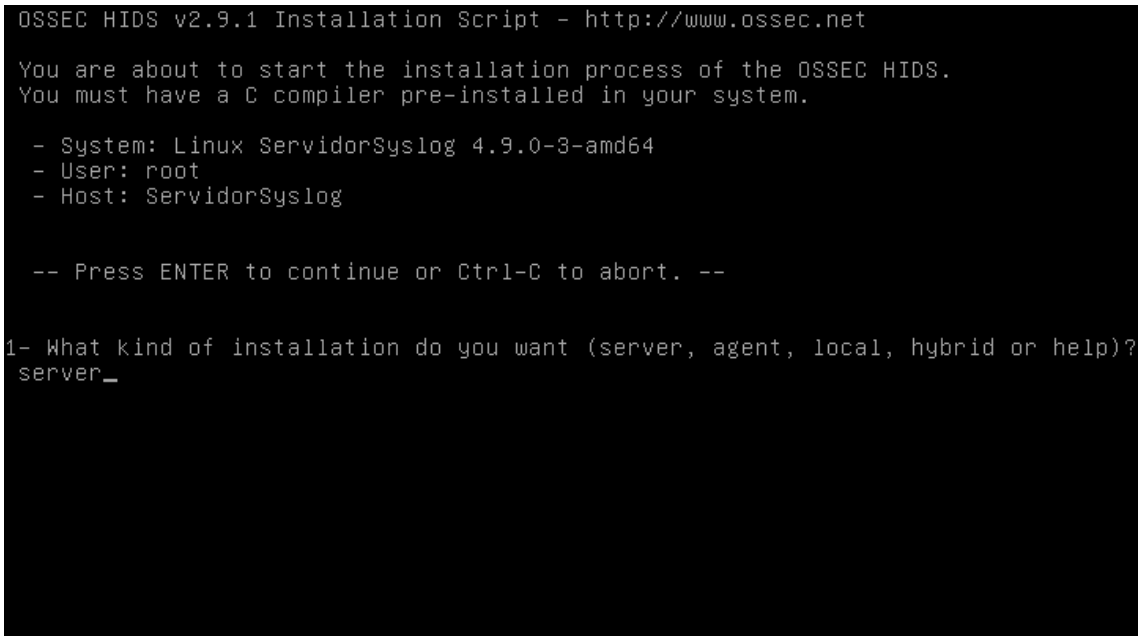

Ilustración 49. Elección del tipo de instalación

La siguiente opción es determinar la ruta de instalación de OSSEC. Por defecto está establecida en "/var/ossec", por lo que simplemente se sigue con la instalación presionando intro.

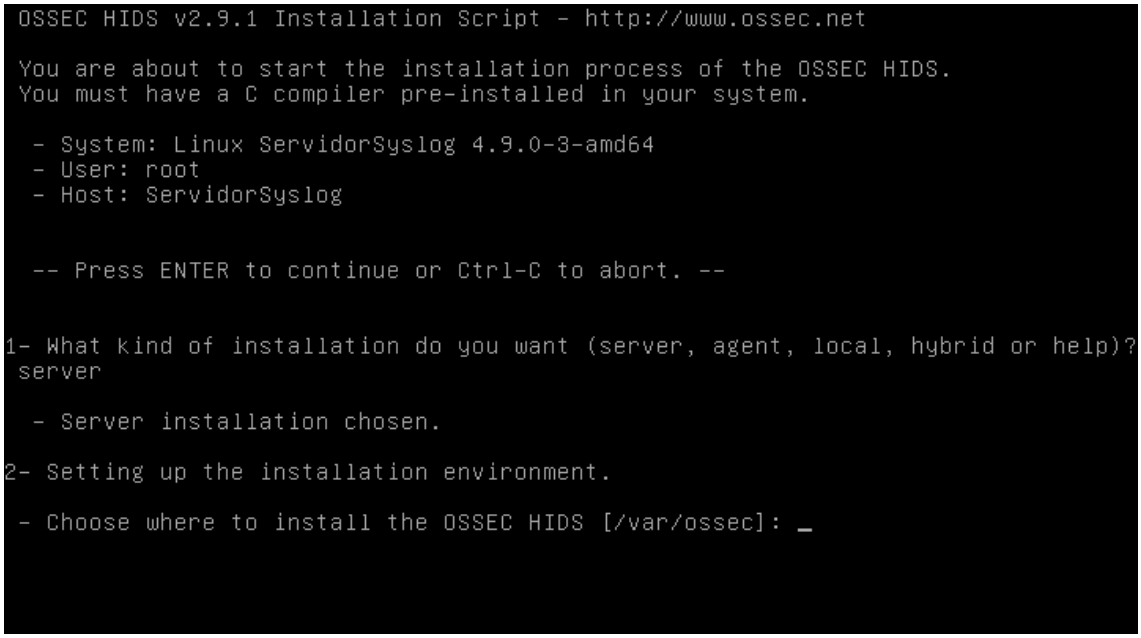

Ilustración 50. Establecimiento de la ruta de instalación

En el paso 3.1, se establece la posibilidad de recibir alertas vía correo electrónico, función que no se empleará en este caso. Para omitir esta parte, se introducirá la letra "n" y se pulsará la tecla intro:

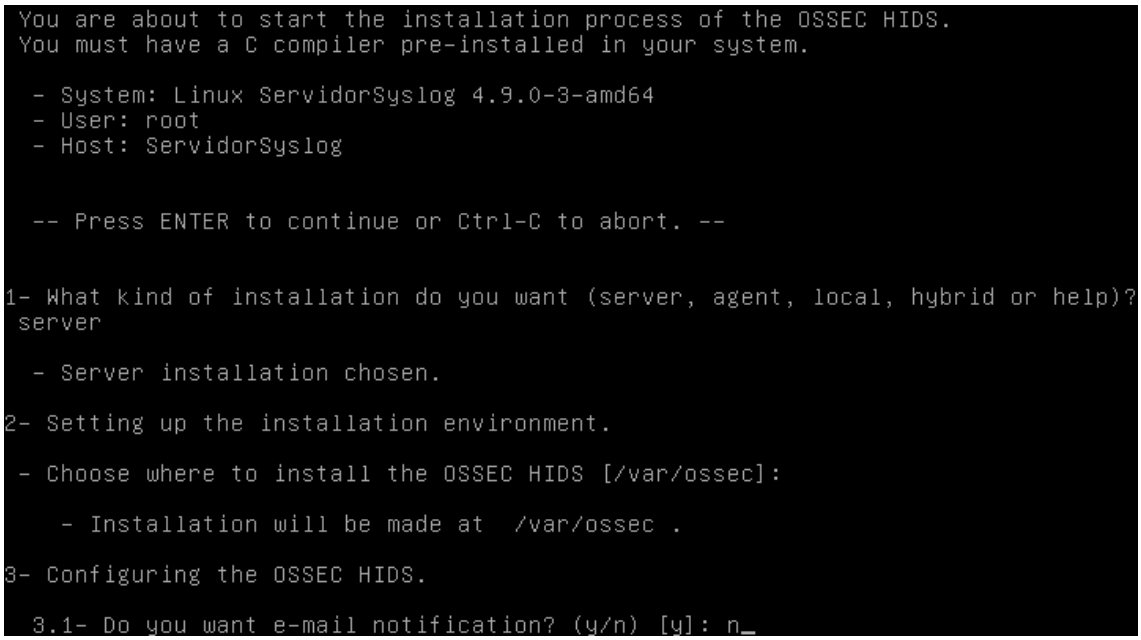

Ilustración 51. Opción de envío de notificaciones por correo electrónico

En el paso 3.2, correspondiente al servicio de comprobación de integridad de ficheros, se presionará la tecla enter para aceptar la instalación de dicho servicio como daemon.

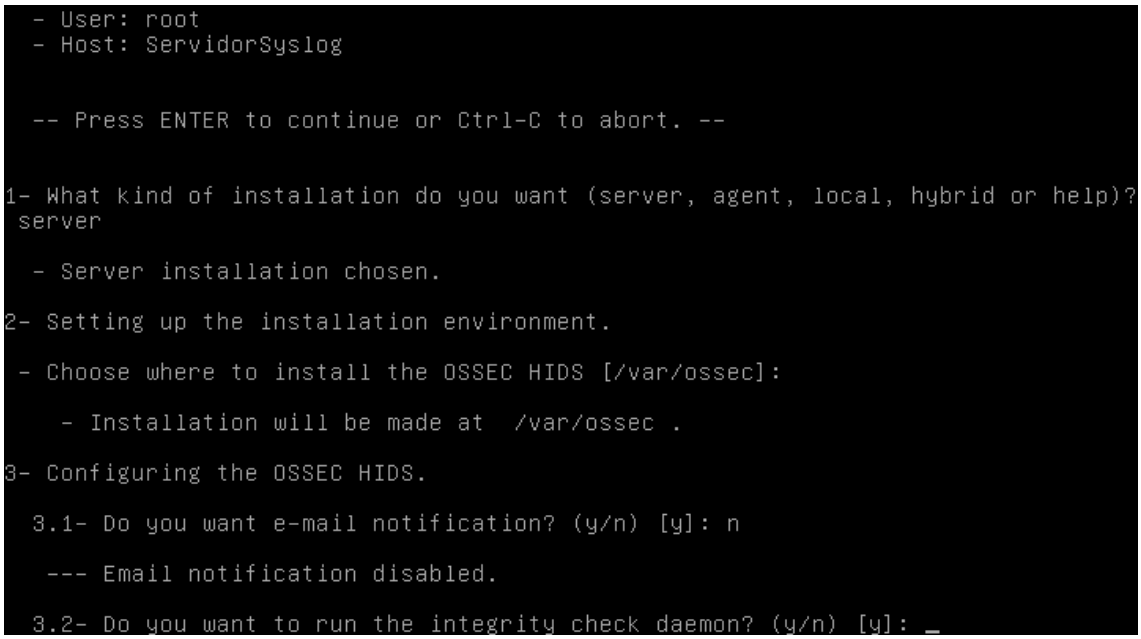

Ilustración 52. Selección de la instalación del servicio de comprobación de integridad

En el paso 3.3, se aceptará la instalación del motor de detección de rootkits presionando la tecla intro.

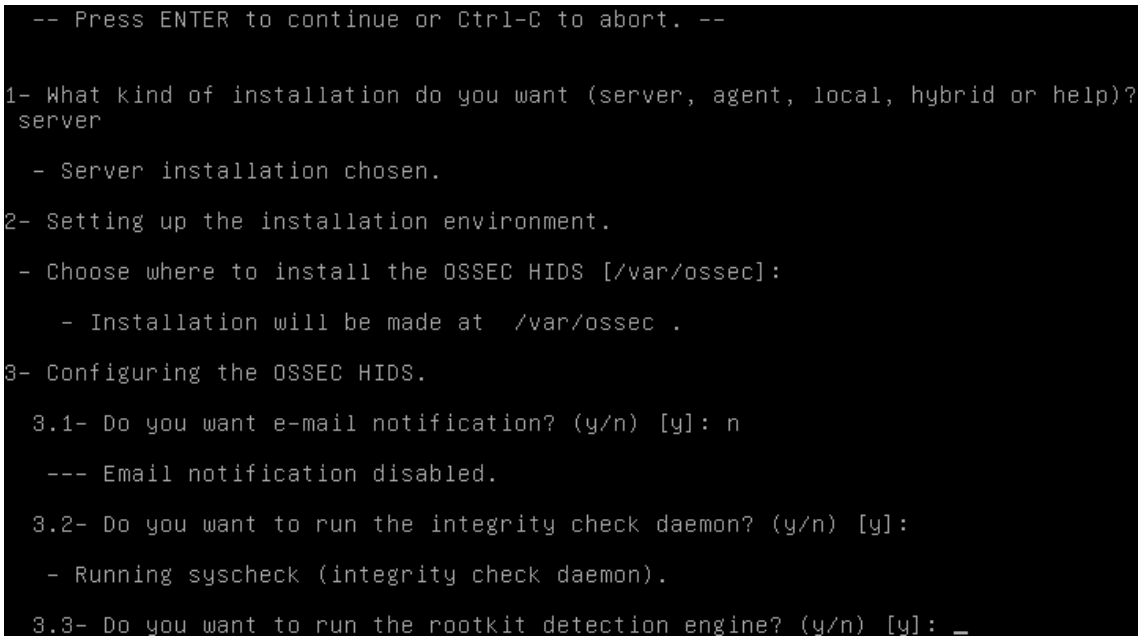

Ilustración 53. Instalación del servicio de detección de rootkits

En el paso 3.4, se ofrece la posibilidad de instalar el servicio de "respuesta activa", el cual ejecutar ciertos comandos a partir de los eventos recibidos. En este caso, se omitirá la instalación introduciendo la letra "n" y presionando intro.

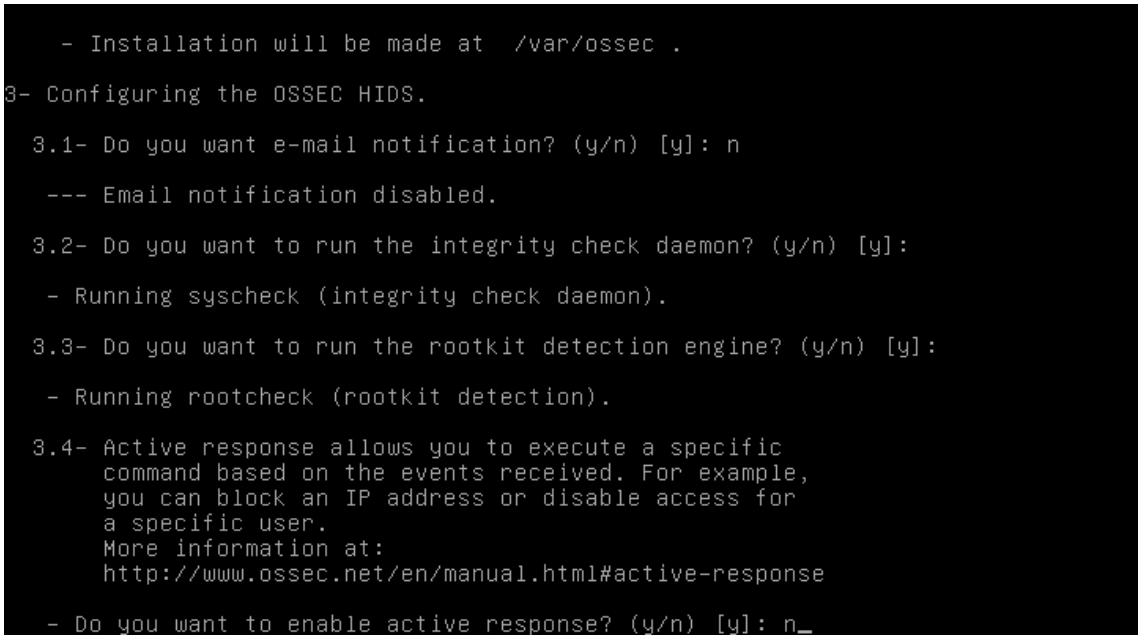

Ilustración 54. Omisión de la instalación del servicio de respuesta activa

Finalmente, en el paso 3.5 se ofrece la posibilidad de habilitar el puerto UDP 514 para recibir logs de otros equipos. Esta característica dependerá para cada uno de los equipos implicados en el sistema de autenticidad. Por ello, en el host "Servidor Syslog" se habilitará dicha posibilidad ya que, de esta forma, se recibirán los logs de OSSEC de las máquinas "C1.1" y "C1.2" para ser procesados y analizados por la aplicación de análisis de eventos para seguridad. Para aceptar la opción, se presiona enter.

Sin embargo, en el resto de hosts, es decir, "C1.1", "C1.2", dicha opción se omitirá ya que este tipo de máquinas no requiere de la recepción de logs de OSSEC de otras máquinas. Para omitir la opción, se introduce la letra "n" y se pulsa enter.

```
  - Do you want to enable active response? (y/n) [y]: n

    - Active response disabled.

 3.5- Do you want to enable remote syslog (port 514 udp)? (y/n) [y]:

  - Remote syslog enabled.

 3.6- Setting the configuration to analyze the following logs:
   -- /var/log/messages
   -- /var/log/auth.log
   -- /var/log/syslog
   -- /var/log/dpkg.log
   -- /var/log/apache2/error.log (apache log)
   -- /var/log/apache2/access.log (apache log)

- If you want to monitor any other file, just change
  the ossec.conf and add a new localfile entry.
  Any questions about the configuration can be answered
  by visiting us online at http://www.ossec.net .


  --- Press ENTER to continue ---
```

Ilustración 55. Finalización de la instalación de servicios opcionales

Finalmente, se presiona de nuevo la tecla intro para proceder a la instalación completa con las opciones definidas anteriormente.

Una vez se ha completado la instalación, se mostrará por pantalla un mensaje que indica que la instalación se ha completado de manera satisfactoria y será necesario presionar de nuevo la tecla intro para salir del asistente.

```
- System is Debian (Ubuntu or derivative).
- Init script modified to start OSSEC HIDS during boot.

- Configuration finished properly.

- To start OSSEC HIDS:
     /var/ossec/bin/ossec-control start

- To stop OSSEC HIDS:
     /var/ossec/bin/ossec-control stop

- The configuration can be viewed or modified at /var/ossec/etc/ossec.conf


   Thanks for using the OSSEC HIDS.
   If you have any question, suggestion or if you find any bug,
   contact us at contact@ossec.net or using our public maillist at
   ossec-list@ossec.net
   ( http://www.ossec.net/main/support/ ).

   More information can be found at http://www.ossec.net

   ---  Press ENTER to finish (maybe more information below). ---
```

Ilustración 56. Finalización de la instalación de OSSEC HIDS

En este punto, ya se tiene instalado el OSSEC HIDS en modo servidor en todos los equipos implicados en el escenario propuesto.

5.3.2 Configuración

La configuración de OSSEC HIDS dependerá de la máquina donde se encuentre instalado, ya que, como se ha comentado en ocasiones anteriores, cada uno de los equipos posee un rol distinto.

Por ello, si el OSSEC HIDS a configurar se encuentra instalado en las máquinas cliente "C1.1" y "C1.2", dicha configuración consiste en la redirección de los eventos generados, a partir de los logs, hacia el cliente "pychain" para que este, a su vez, los introduzca en la cadena de bloques. Adicionalmente, también será necesaria la redirección de dichos eventos hacia la máquina "Servidor Syslog" con el fin de realizar una prospección de los mismos mediante la aplicación de análisis de eventos.

Si el software OSSEC HIDS que se va a configurar se encuentra en el equipo "Servidor Syslog", es necesario que la redirección de los eventos obtenidos a partir de los equipos cliente se realice hacia la aplicación encargada del análisis de los mismos.

A continuación, se detalla la configuración para cada uno de ellos:

OSSEC HIDS – Máquina "Servidor Syslog"

Empleando el mismo terminal que para la instalación, el primer paso a realizar es cambiar el directorio actual a "/var/ossec", directorio en el que se encuentra instalado OSSEC HIDS por defecto.

Una vez cambiado el directorio actual, es necesario editar el fichero "ossec.conf". En dicho fichero se encuentra toda la configuración de OSSEC en formato XML. Para realizar dicha edición, se ejecuta el comando "nano etc/ossec.conf".

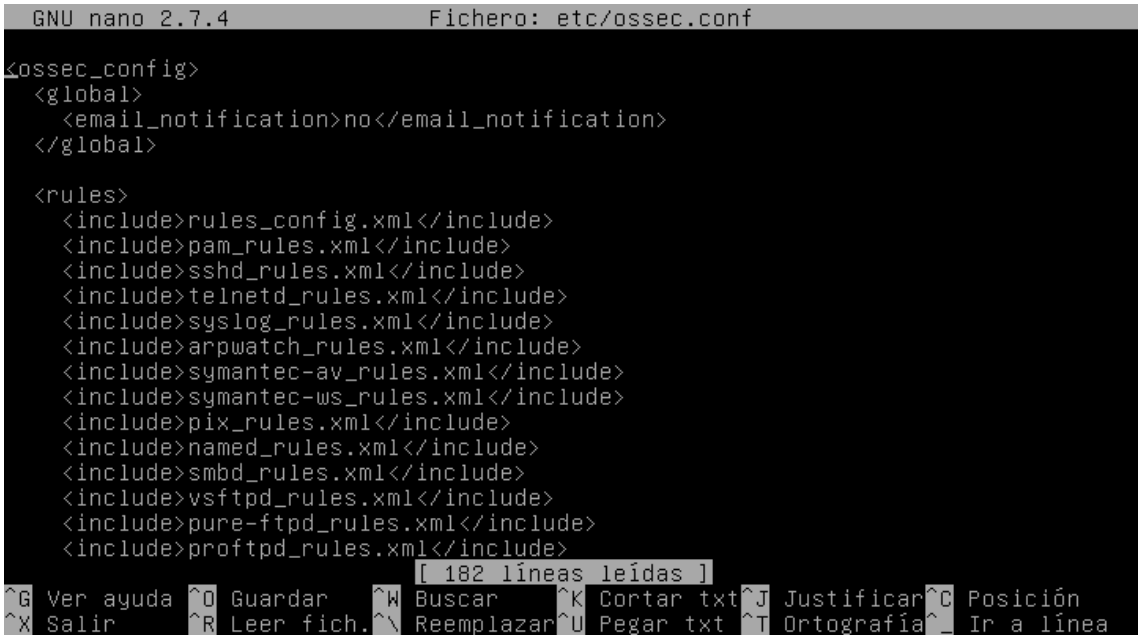

Ilustración 57. Modificación del fichero ossec.conf

Tras la ejecución del comando, se habilitará la posibilidad de edición del fichero de configuración.

Posteriormente, será necesario agregar la línea “<syslog_output><server>127.0.0.1</server><port>10002</port></syslog_output> ” tal y como se observa en la Ilustración 58.

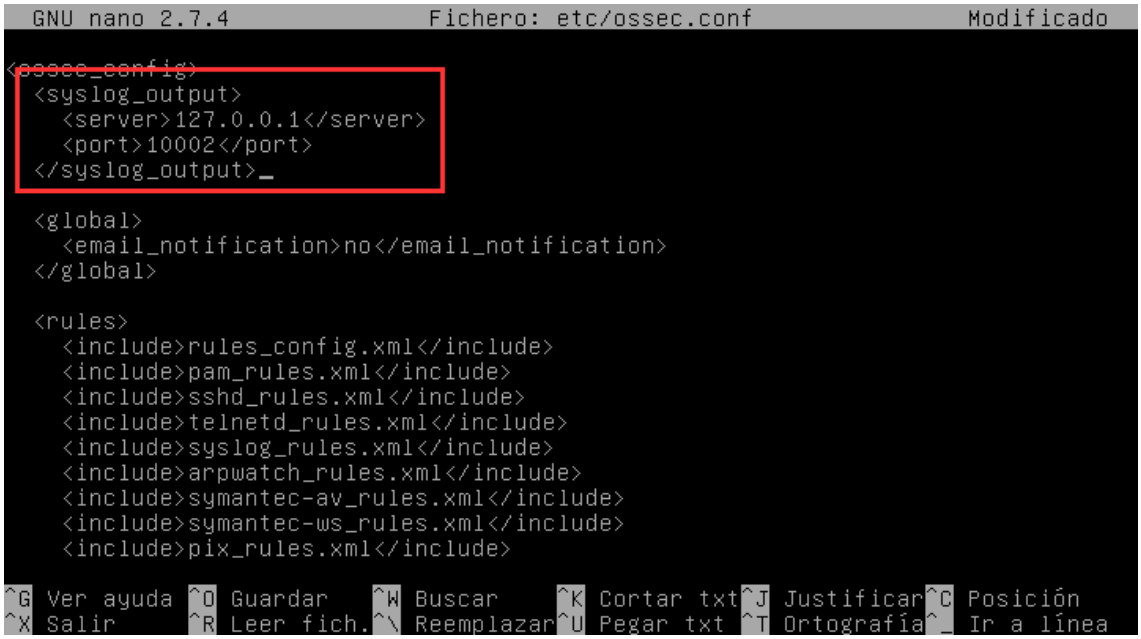

Ilustración 58. Configuración de la salida de los mensajes de registro

A continuación, se detallan cada uno de los parámetros introducidos:

- syslog_output: Etiqueta indicada para establecer la configuración correspondiente al envío de logs hacia un servidor syslog.
- server: Dirección IP del servidor donde se encuentra la aplicación de análisis de eventos. En el caso descrito, la aplicación se encuentra en la misma máquina, por lo que la dirección establecida es la asignada para "loopback".
- port: Puerto UDP en el que la aplicación de análisis de eventos recibe dichos eventos. En el caso descrito, la aplicación se encuentra configurada para la recepción de los mismos en el puerto 10002.

Finalmente, se guarda la nueva configuración y se inicia el servicio. Para iniciarlo, únicamente es necesario ejecutar el comando "bin/ossec-control start".

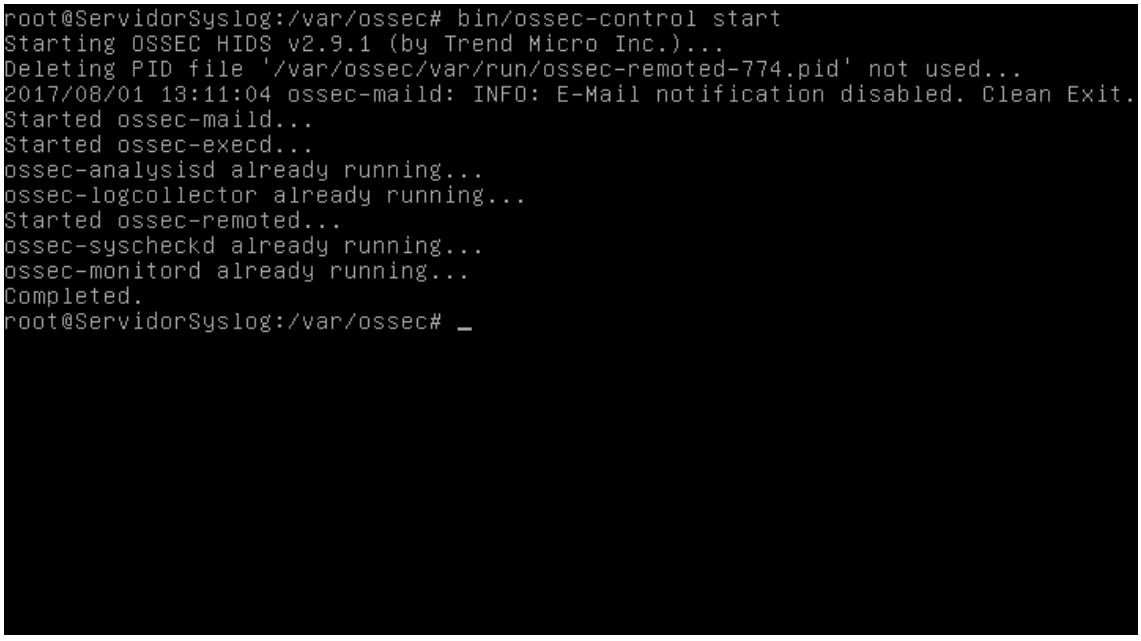

Ilustración 59. Inicio de OSSEC HIDS en el servidor Syslog

En este punto, ya se tiene el equipo "Servidor Syslog" configurado para el reenvío de los eventos hacia la aplicación de análisis de los mismos para seguridad. Adicionalmente, y dado que se habilitó la recepción de dichos eventos en la instalación mediante el puerto 514 UDP, también se encuentra configurado para recibir las alertas desde el resto de máquinas.

Sin embargo, la configuración actual permite la recepción de alertas desde cualquier máquina y no únicamente desde las máquinas clientes. Para modificar este comportamiento y permitir la recepción desde máquinas autorizadas, es necesario

editar de nuevo el fichero “ossec.conf” y desplazarse hasta el apartado etiquetado como “<remote>”.

Existen dos apartados con dicha etiqueta. En este caso, se editará aquella que posee el valor “syslog” en la etiqueta “<connection>”, introduciendo la línea “<allowed-ips>10.46.2.21</allowed-ips><allowed-ips>10.46.2.28</allowed-ips>”

Posteriormente, se guardan los cambios y se reinicia el servicio OSSEC con el comando “bin/ossec-control restart”.

OSSEC HIDS – Máquinas “C1.1” y “C1.2”

En el caso de las máquinas clientes “C1.1” y “C1.2”, la configuración a realizar consiste en reenviar las alertas de OSSEC hacia el cliente “pychain” y hacia el host “Servidor Syslog”.

El primer paso a realizar es acceder a la línea de comandos con permisos de superusuario, por lo que se deberá introducir el comando “su”.

Posteriormente, se cambiará el directorio actual a aquel en el que se encuentra instalado OSSEC. Para ello, se ejecutará el comando “cd /var/ossec”.

Una vez cambiado el directorio actual, es necesario editar el fichero “ossec.conf”. Para realizar dicha edición, se ejecuta el comando “nano etc/ossec.conf”.

En este caso, y dado que las máquinas van a reenviar los eventos a dos servidores syslog diferentes, es necesario establecer dos salidas distintas. Para ello, se agregaran dos líneas diferentes; “<syslog_output><server>127.0.0.1</server><port>514</port></syslog_output>” y “<syslog_output><server>10.46.2.48</server><port>514</port></syslog_output>.

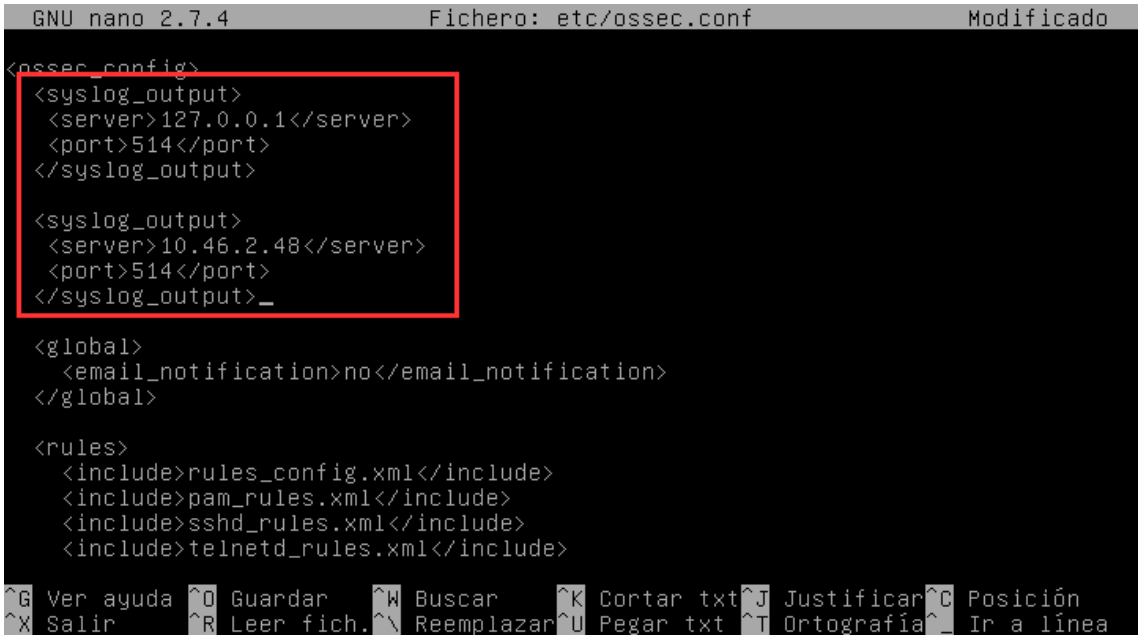

Ilustración 60. Configuración del fichero ossec.conf en C1.1

Mediante la primera de estas líneas, se reenvían las alertas hacia el cliente "pychain", el cual se ejecuta en la propia máquina en el puerto UDO 514. En la segunda, se configura OSSEC HIDS para que reenvíe dichas alertas hacia la máquina "Servidor Syslog", que se encuentra en dicha dirección IP escuchando en el puerto UDP 514.

En este punto, ya se han configurado los equipos "C1.1" y "C1.2" para el reenvío de los eventos hacia las máquinas designadas.

5.4 Verificación de la integridad

La verificación de la integridad es el último de los componentes que forman parte del sistema de autenticidad creado. A partir de dicha verificación, se comprueba la no modificación del contenido de los datos almacenados en el "Servidor Syslog" y la cadena de bloques del tal forma que el sistema de autenticidad complete su función principal, garantizar la integridad de los mensajes de registro transmitidos por el equipo origen hacia el equipo destino y su posterior almacenamiento, todo ello sin ser posible la modificación de los mismos.

5.4.1 Verificación individual de mensaje de registro

Con la verificación individual de los mensajes de registro se pretende verificar la presencia de un log individual en la cadena de bloques. Para realizar esta función, se ha planteado la ejecución de un script en Python alojado en la máquina "Servidor Blockchain" al cual se le pasará como parámetro el mensaje de registro a comprobar

y este devolverá si dicho mensaje se encuentra en la cadena de bloques o si, por el contrario, no se evidencia su presencia y por tanto, ha podido ser alterado en su transmisión o almacenamiento puesto que, tal y como se ha explicado en el apartado 3, la naturaleza de la cadena de bloques impide su modificación tras ser verificada la transacción realizada.

A continuación, se muestra un diagrama de actividad que muestra las distintas acciones que realizará el script para realizar la función para la cual ha sido programado:

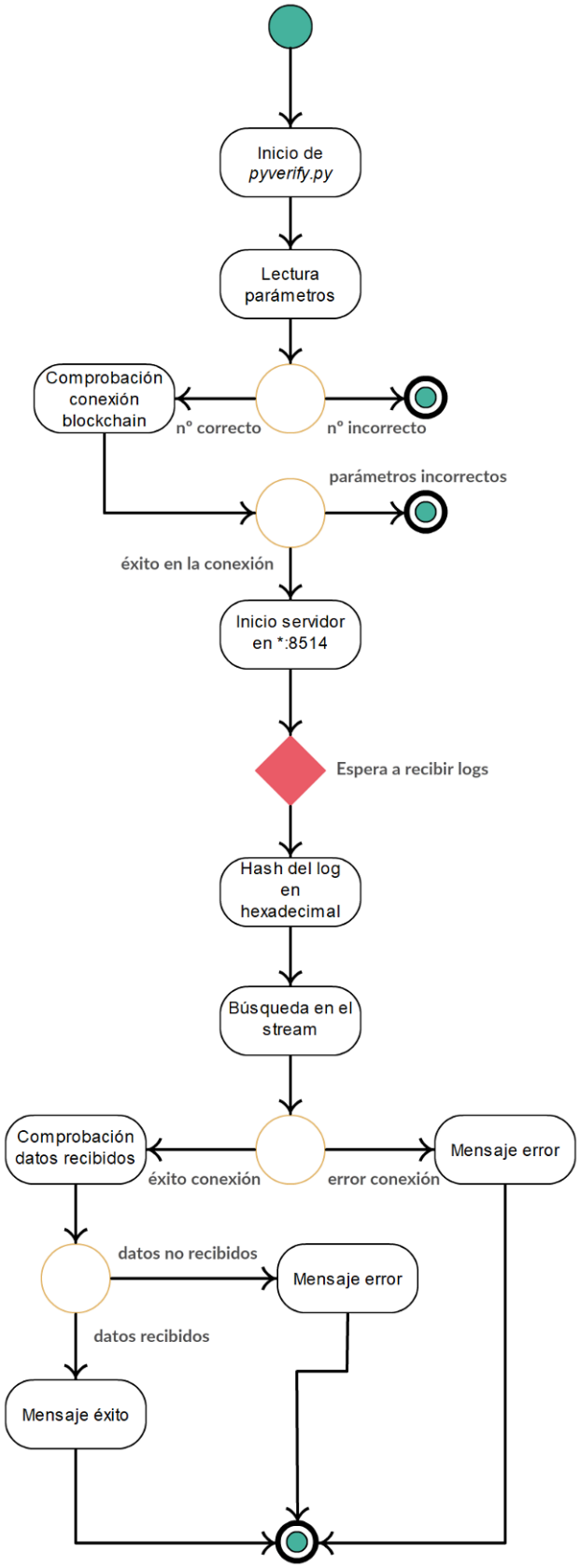

Ilustración 61. Diagrama de actividades de pyverify

Tal y como se observa en el flujo, la ejecución de "pyverify" se compone de diversas fases, las cuales se detallan seguidamente:

1. Lectura del número de parámetros recibidos.
2. Verificación de los parámetros mediante la conexión a la cadena de bloques.
3. Transformación del mensaje de registro obtenido por línea de comandos a formato hexadecimal.
4. Conexión al stream y obtención del valor correspondiente al hash comprobado.
5. Comprobación de la coincidencia de los valores obtenidos.

Estas fases, han dado lugar al script de verificación mostrado en la Ilustración 62.

```
import SocketServer
import hashlib
import sys
import imp

sav = imp.load_source('Savoir', '/usr/local/lib/python2.7/dist-packages/Savoir/Savoir.py')

HOST, PORT = "0.0.0.0", 8514

api, rpcuser, rpcpasswd, rpchost, rpcport, chainname = "", "", "", "", "", ""

class SyslogUDPHandler(SocketServer.BaseRequestHandler):

    def handle(self):
        data = bytes.decode(self.request[0].strip())
        socket = self.request[1]
        hex_dig = hashlib.sha1(str(data)).hexdigest()
        print("Checking log message '" + str(data) +"'...")
        try:
            request = api.liststreamkeyitems("logHashes",hex_dig)
            if len(request) > 0:
                print "Log message found! Integrity has not been modified."
            else:
                print "WARNING! Log message NOT found. Please, check if log is correct."
        except:
            print("Please, check if blockchain '" + str(chainname) + "' is up")

if __name__ == "__main__":
    try:

        if len(sys.argv) < 5:
            print("Usage: %s rpcUser rpcPasswd rpcPort chainName" % sys.argv[0])
            print("rpcUser and rpcPasswd are stored into ~/.multichain/<chainName>/multichain.conf")
            print("rpcPort is set as default_rpc_port into ~/.multichain/<chainName>/params.dat")
            sys.exit(0)

        rpcuser = sys.argv[1]
        rpcpasswd = sys.argv[2]
        rpchost = '127.0.0.1'
        rpcport = sys.argv[3]
        chainname = sys.argv[4]

        api = sav.Savoir(rpcuser, rpcpasswd, rpchost, rpcport, chainname)

        try:
            api.getinfo()
        except:
            print("Please, check if blockchain server is up or your params are set successfully")
            sys.exit(0)

        server = SocketServer.UDPServer((HOST,PORT), SyslogUDPHandler)
        print("<----> Verify Server started <---->")
        server.serve_forever(poll_interval=0.5)
    except (IOError, SystemExit):
        raise
    except KeyboardInterrupt:
        print ("Crtl+C Pressed. Shutting down.")
```

Ilustración 62. Código del script pyverify

5.4.2 Obtención acotada de mensajes de registro

Con la verificación acotada de mensajes de registro se pretende obtener todos los eventos almacenados en la cadena de bloques pertenecientes a un nodo en un rango concreto de fechas y con ello, poder cotejarlos con los eventos almacenados en el Servidor Syslog. Esta función permitiría, por ejemplo, realizar un análisis pericial con el fin de detectar posibles alteraciones del contenido de los eventos, así como posibles eliminaciones posteriores a su almacenamiento. Para realizar esta función, se ha

planificado la ejecución de un script en lenguaje Python al cual se le pasan múltiples parámetros entre los cuales se incluyen la fecha inicial y final a fin de establecer el rango a obtener y la dirección del nodo que ha realizado dicha publicación.

A continuación, se muestra un diagrama de actividad que muestra las distintas acciones que realizará el script para realizar la función para la cual ha sido programado:

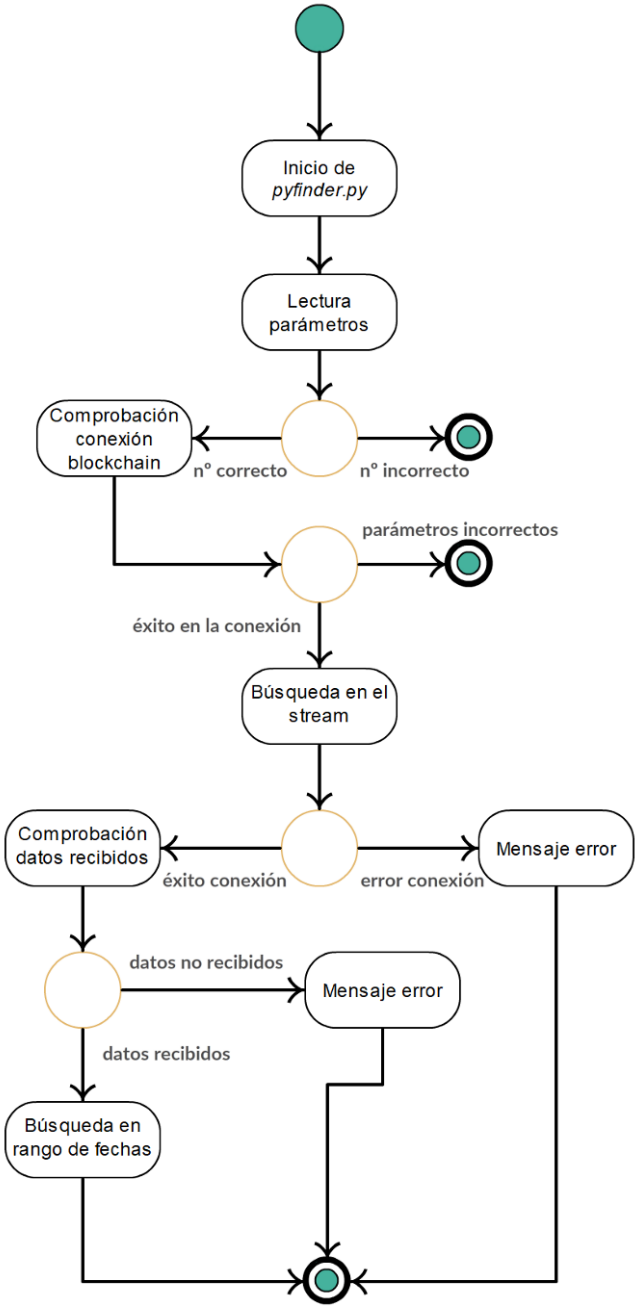

Ilustración 63. Diagrama de actividades de pyfinder

Tal y como se observa en el flujo, la ejecución de "pyfinder" se compone de diversas fases, las cuales se detallan seguidamente:

1. Lectura del número de parámetros recibidos.
2. Verificación de los parámetros mediante la conexión a la cadena de bloques.
3. Conexión al stream y búsqueda de los mensajes de registro del nodo deseado.
4. Comprobación de los datos recibidos y búsqueda de los eventos pertenecientes al rango determinado.

Estas fases, han dado lugar al script de verificación mostrado en la Ilustración 64.

```
import sys
import imp
import datetime

sav = imp.load_source('Savoir', '/usr/local/lib/python2.7/dist-packages/Savoir/Savoir.py')

api, rpcuser, rpcpasswd, rpchost, rpcport, chainname = "", "", "", "", "", ""

if __name__ == "__main__":
    if len(sys.argv) < 5:
        print("Usage: %s rpcUser rpcPasswd rpcPort chainName" % sys.argv[0])
        print("rpcUser and rpcPasswd are stored into ~/.multichain/<chainName>/multichain.conf")
        print("rpcPort is set as default_rpc_port into ~/.multichain/<chainName>/params.dat")
        print("startDate and endDate format must be YYYY-MM-DD")
        sys.exit(0)

    rpcuser = sys.argv[1]
    rpcpasswd = sys.argv[2]
    rpchost = '127.0.0.1'
    rpcport = sys.argv[3]
    chainname = sys.argv[4]
    startDate = raw_input("Start date (YYYY-MM-DD): ")
    endDate = raw_input("End date (YYYY-MM-DD): ")
    publisher = raw_input("Publisher Address: ")

    api = sav.Savoir(rpcuser, rpcpasswd, rpchost, rpcport, chainname)

    try:
        api.getinfo()
    except:
        print("Please, check if blockchain server is up or your params are set successfully")
        sys.exit(0)

    try:
        request = api.liststreampublisheritems("logHashes",publisher,True)
        if type(request) is not dict:
            startDate = startDate.split('-')
            startDate = int(datetime.datetime(int(startDate[0]),int(startDate[1]),int(startDate[2]),0,0).strftime('%s'))
            endDate = endDate.split('-')
            endDate = int(datetime.datetime(int(endDate[0]),int(endDate[1]),int(endDate[2]),0,0).strftime('%s'))
            for x in range(0,len(request)-1):
                item = request[x]
                if int(item['time']) > startDate and int(item['time']) < endDate:
                    print str(item['data'].decode('hex'))
        else:
            print("Ups! " + str(request['error']['message']))

    except KeyboardInterrupt:
        print("Please, check if blockchain '" + str(chainname) + "' is up")
```

Ilustración 64. Código del script pyfinder

Para ejecutar pyfinder de forma correcta, es necesario proporcionar los parámetros de conexión de la cadena de bloques. Posteriormente, el programa habilita la opción de introducir la fecha de inicio y fin del rango determinado, así como la opción de introducir la dirección del nodo deseado. Tras esto, realiza una búsqueda de todas las

transacciones cuyo publicador es el nodo que se ha introducido. Finalmente, realiza una búsqueda en todas las transacciones obtenidas para determinar aquellas que se encuentran comprendidas en el rango establecido.

6. Posibles mejoras

Destacar que las mejoras aquí propuestas se enfocan a proyectar el sistema de autenticidad creado hacia entornos que requieran mayores prestaciones de las presentadas en el actual proyecto.

El sistema de autenticidad desarrollado en este proyecto es capaz de realizar las funciones para las cuales ha sido diseñado con total cumplimiento de los requisitos especificados. Sin embargo, de forma adicional se han estudiado, de forma teórica, una serie de posibles mejoras sobre el sistema de autenticidad propuesto con el objetivo de constituir un sistema de autenticidad con amplias capacidades de interoperabilidad. Las mejoras que a continuación se describen, proporcionan un complemento al sistema de autenticidad creado minimizando la interacción con el mismo de forma que sea el propio sistema el que se adapte a los posibles cambios de configuración en los mensajes de registro.

6.1 Configuración distribuida

Mediante la configuración distribuida se pretende realizar una difusión de la configuración empleada en el manejo de los mensajes de registro de tal forma que no se requiera pausar la cadena de bloques para alterar dicha configuración.

La forma de realizar el proceso de distribución de la configuración consiste en la creación de un nuevo stream que contenga la configuración a aplicar. Esta configuración se almacenará en un ítem clave-valor en el que la clave se mantendrá igual para cada uno de los ítems y el valor, contendrá la configuración en formato *JSON* codificado en hexadecimal. Adicionalmente, y dado que existirán múltiples transacciones con configuraciones diferentes, se deberá obtener la última transacción añadida.

Para evitar posibles alteraciones indeseadas de nodos no autorizados, y empleando los permisos disponibles en MultiChain, se asignarán los permisos pertinentes de escritura únicamente al nodo encargado de establecer la configuración, y los permisos de lectura a los nodos que requieren dicha configuración.

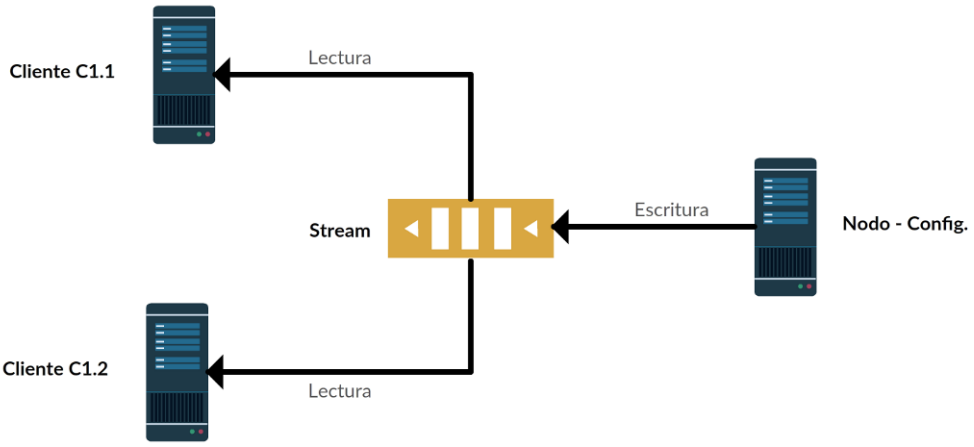

Ilustración 65. Escenario de la configuración distribuida

6.2 Rotación de logs

En la mayoría de los sistemas operativos, al producirse un elevado número de logs, estos se van dividiendo en diversos ficheros. De no producirse esta rotación, los archivos encargados de almacenar los logs tendrían un tamaño desorbitado y cualquier problema con dicho fichero, podría causar una desaparición de todos los logs. Adicionalmente, el simple copiado o gestión de un fichero con un tamaño considerable, dificultaría su manejo.

Este mismo concepto se traslada a la cadena de bloques, ya que, aunque su optimización y eficiencia son elevadas es posible que, dado un número considerable de transacciones, su rendimiento se vea perjudicado.

Por ello, una posible mejora que se puede aplicar al sistema de autenticidad creado es la rotación de logs en la propia cadena de bloques, de tal forma que sea posible realizar búsquedas más eficientes y con un menor coste computacional. Para aplicar esta mejora, se emplea la mejora explicada en el punto anterior, ya que los clientes requieren de una configuración específica para conocer el stream en el que deben realizar la escritura de la información.

Los nodos deberán realizar lecturas constantes de la configuración antes de realizar la inserción de los nuevos datos con el fin de disponer siempre la última configuración distribuida. Adicionalmente, es posible que en el stream existan diversos ítems clave-valor en los que su clave sea compartida, por lo que los nodos obtendrán el último ítem añadido a partir del sello de tiempo asociado a cada elemento.

Además, el nodo administrador de la cadena de bloques deberá ir creando los nuevos streams en base a la configuración aplicada de tal forma que los clientes tengan disponible el stream previamente a la inserción de los datos. El patrón de generación variará en función de la política de rotación a seguir:

- Si la rotación se realiza de forma diaria, el nombre del stream seguirá el patrón *DdíaMesAño*, por ejemplo, D08072017
- Si la rotación se realiza de forma semanal, el nombre del stream seguirá el patrón *WsemanaMesAño*, por ejemplo, W27072017
- Si la rotación se realiza de forma mensual, el nombre del stream seguirá el patrón *MmesAño*, por ejemplo, M072017.

Esta forma de estructurar el nombre de los streams, se empleará para realizar la búsqueda de los mismos en los distintos streams, ya que a partir de la letra inicial de stream será posible obtener todos los logs al respecto.

7. Conclusiones

Como se ha podido observar en los distintos apartados descritos en este proyecto, el sistema de autenticidad creado se compone de múltiples elementos que, en su conjunto, constituyen un conjunto sólido que proporciona el mecanismo necesario para el mantenimiento de la integridad en los mensajes de registro para aplicaciones de análisis de eventos para seguridad.

Adicionalmente, el sistema se encuentra constituido con diversas utilidades las cuales no solo garantizan la integridad en la transmisión de los mensajes de registro y posterior almacenamiento, sino que garantizan la comprobación posterior bajo demanda con el fin de potenciar la confianza en el sistema creado. Esto permite dotar al sistema de autenticidad de un aval para su utilización en temas judiciales como pueda ser un análisis pericial en el que la independencia, por parte de los analistas encargados de examinar los eventos, no se encuentre comprometida.

Con lo que respecta a la tecnología empleada en el núcleo del sistema, la cadena de bloques, y a pesar de ser una tecnología reciente, mediante este proyecto ha sido posible demostrar la utilización de la cadena de bloques para ámbitos alternativos a los utilizados actualmente, como es la gestión de *criptomonedas*.

La creación del sistema de autenticidad no ha estado exenta de diversas problemáticas puesto que, tal y como se ha comentado, actualmente se trata de una tecnología reciente con ámbitos de aplicación reducidos. Entre los múltiples problemas surgidos, destacan aquellos relacionados con la verificación de aquellos eventos que hayan podido ser eliminados en el servidor encargado de almacenar los mensajes de registro tras su análisis, así como la creación de los clientes encargados de generar las transacciones en las que se almacenan los hashes de los mensajes de registro dada la poca documentación existente para la librería empleada.

Sin embargo, y tras realizar múltiples análisis de posibles escenarios y siempre con el objetivo de garantizar la integridad de los mensajes de registro provenientes de las distintas máquinas cliente, se han resuelto las problemáticas asociadas y se ha diseñado, y creado, un sistema de autenticidad que asegura la no modificación, de los

datos relacionados con mensajes de registro de seguridad, en todas las fases posibles durante su ciclo.

Con todo ello, se espera que el producto obtenido fruto de este trabajo se siga perfeccionando hasta obtener un instrumento lo suficientemente desarrollado que permita su puesta en producción en un entorno real.

8. Bibliografía

SWAN, M. *Blockchain*. Sebastopol, California: O'Reilly, 2015

ANTONOPOULOS, A. M. *Mastering Bitcoin*. Sebastopol, California: O'Reilly, 2015

GUPTA, M. *Blockchain for dummies*. Hoboken, Nueva Jersey: Wiley, 2017

MOUGAYAR, W. *The business blockchain*. Hoboken, Nueva Jersey: Wiley, 2016

CROSBY, M., NACHIAPPAN, PATTANAYAK, P., VERMA S., KALYANARAMAN, V. *Blockchain technology beyond bitcoin*. Berkeley, California: University of California Berkeley, 2015. Disponible en http://scet.berkeley.edu/wp-content/uploads/BlockchainPaper.pdf

CUCURULL, J., PUIGGALÍ, J. *Distributed immutabilization of secure logs*. Barcelona, España: Scytl, 2016. Disponible en https://www.scytl.com/wp-content/uploads/2017/01/Distributed-Immutabilization-of-Secure-Logs_Scytl.pdf

DÍAZ, J., SÁNCHEZ A. *Bitcoin: Una moneda criptográfica*. León, España: INTECO, 2014. Disponible en https://www.incibe.es/extfrontinteco/img/File/intecocert/EstudiosInformes/int_bitcoin.pdf

GREENSPAN, G. *MultiChain Private Blockchain*. Reino Unido: Coin Sciences Ltd, 2015. Disponible en https://www.multichain.com/download/MultiChain-White-Paper.pdf

FREEMON, M. *OSSEC Open Source Security*. Springfield, Illinois: University of Illinois, 2008. Disponible en https://edocs.uis.edu/tllos1/Courses/CSC570E/Samples/CSC570E_Research_Paper_OSSEC.pdf

GERHARDS, R. *Request For Comment - The Syslog Protocol*. Großrinderfeld, Alemania: Adiscon GmbH, 2009. Disponible en https://www.rfc-editor.org/rfc/rfc5424.txt

GERHARDS, R. *Syslog-protocol support in rsyslog*. Großrinderfeld, Alemania: Adiscon GmbH, 2009. Disponible en http://www.rsyslog.com/doc/v8-stable/whitepapers/syslog_protocol.html

DASH, P. *Getting Started with Oracle VM VirtualBox*. Birmingham, Reino Unido: Pack Publishing, 2013.

MASLAC, M. *Oracle VM VirtualBox tutorial for complete beginners*. Zagreb, Croacia: Geek University Press, 2016.

PEICEVIC, A. *Linux for Beginners - A Comprehensive Guide*. Zagreb, Croacia: Geek University Press, 2015.

NEGUS, C. *Linux Bible*. Hoboken, Nueva Jersey: Wiley, 2015

ORIYANO, S. *CEHV8 Study Guide*. Indianápolis, Estados Unidos: Sybex, 2014

JACKSON, C. *Network Security Auditing*. Indianápolis, Estados Unidos: Cisco Press, 2010

SMITH, R. W. *LPIC-1 Study Guide*. Indianápolis, Estados Unidos: Wiley, 2013

SCHNEIER, B. *Applied Cryptography*. Indianápolis, Estados Unidos: Wiley, 1996

SCHNEIER, B., FERGUSON, N. *Practical Cryptography*, Indianápolis, Estados Unidos: Wiley, 2003

9. Índice de ilustraciones y tablas

Ilustraciones

Tablas