Universidad Autónoma Metropolitana
Unidad Azcapotzalco
División de Ciencias Básicas e Ingeniería
Licenciatura en Ingeniería en Computación

Aplicación que ayuda a la seguridad de puertos de un *Switch* de capa 2.

Modalidad: Proyecto tecnológico.

Trimestre 2018 Invierno.

Alumno.
Jair Michel Paredes Estrada.

Assessor.
M. en C. Arturo Zúñiga López.

Abril de 2018

Declaratoria

Yo, Jair Michel Paredes Estrada, doy mi autorización a la Coordinación de Servicios de información de la Universidad Autónoma Metropolitana, Unidad Azcapotzalco, para publicar el presente documento en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Jair Michel Paredes Estrada

Yo, Arturo Zúñiga López, declaro que aprobé el contenido del presente Reporte de Proyecto de Integración y doy mi autorización para su publicación en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

M. en C. Arturo Zúñiga López.

Resumen.

Las redes de computadoras evolucionan constantemente y desarrollan nuevas formas de defensa a ataques que pueden comprometer la red o la información que circula en ella. Por esta razón las empresas se preocupan en desarrollar herramientas y características para hacer más robustos y seguros los equipos de la red.

Existen dos tipos de *Switches* que operan en diferentes capas según el módelo de referencia *OSI*, unos trabajan en capa dos y otros trabajan en capa tres. Los que trabajan en capa dos generalmente lo hacen en redes *LAN* (Red de Área Local) y los que trabajan en capa 3 pueden trabajar tanto en *LAN* como en *WAN* (Red de Área Metropolitana). Este proyecto se basa en los que trabajan en capa dos y en concreto en la seguridad de los puertos de estos dispositivos

El puerto de seguridad es una característica de los *Switches* que ayuda a mantener la seguridad y/o las políticas de seguridad en una empresa u organización, pero esta característica tiene una desventaja al configurarla en los equipos para que apague el puerto cada que exista una violación de seguridad y si el administrador de redes no se entera de que cambio el estado de los puertos puede conllevar a afectaciones operativas en la red. El puerto de seguridad no tiene la capacidad de regresar el puerto a su estado normal u original, es necesaria una herramienta o programa que mitigue este inconveniente para que no afecte a la operación de la red y tener consecuencias graves.

El programa implementado realiza una conexión vía *TELNET* para entrar a los equipos e investigar que puertos cayeron en el inconveniente, una vez con la información de los puertos se realiza otra conexión para regresar a su estado normal a cada puerto, introduciendo los comandos adecuados para limpiarlos y posteriormente prenderlos y apagarlos, de esta manera el programa mitiga el inconveniente del puerto de seguridad. También se generan reportes cuando existe en más de una ocasión en el mismo día el inconveniente de seguridad.

De esta manera se obtiene un programa de computo que cumple con una tarea muy específica en los *Switches*, ya que es poco probable que una solución como esta se encuentre implementada o que existan proyectos relacionados con este tema. De esta manera el proyecto contribuye a que la red tenga una mayor disponibilidad, seguridad y confianza tanto para los usuarios como para las empresas u organizaciones, y finalmente ahorre tiempo.

Tabla de contenido.

Introducción.

En la actualidad las redes de computadoras han tomado mucha importancia en la vida de las personas y en las empresas, ya que las personas se pueden comunicar a grandes distancias y obtienen información de la red.

Hoy en día es muy poco común que una persona o una empresa no esté conectada a internet, por tal motivo los dispositivos de red tienen la necesidad de cambiar y evolucionar, ya que la red crece a un ritmo acelerado. Tal ha sido su crecimiento que los dispositivos que interactúan en la red evolucionan constantemente, dichos dispositivos cuentan con herramientas y características para mantener la integridad, seguridad y disponibilidad.

Las tecnologías cambian constantemente y se adaptan a las necesidades de la red, por ejemplo, los equipos de red hace algunos años no soportaban direccionamiento *IPv6* solamente *IPv4* y después de que las direcciones ipv4 estén casi por agotarse, tuvieron que añadir tanto a los protocolos de enrutamiento como a los equipos que sean capaces de manejar direccionamiento ipv6. Esto ha provocado que los equipos sean más robustos y, por ende, permite tener más tecnologías y características para que la red sea confiable y segura.

El dispositivo de red llamado *Switch* puede funcionar en la capa 2 y 3 del modelo de referencia OSI, los que funcionan en capa dos generalmente se utilizan para trabajar en una red local, estos equipos cuentan con algunas características como, por ejemplo, poder separar una red local y varias redes virtuales locales con la función llamada *VLAN* (Red Virtual de Área Local). Los que funcionan en capa tres son capaces de enrutar tráfico además de las características de los de capa dos.

Este proyecto se centra en los equipos que trabajan en capa dos, a estos se les puede conectar un host o dispositivo final, para esto se necesita poner el puerto en modo acceso, esto quiere decir, que solo va a pasar tráfico de una sola *VLAN* y de un solo host, pero a esto equipos también se les puede conectar otro *Switch*, para esto generalmente los puertos se deben poner en *Trunk*, esto quiere decir, que por esos puertos va a pasar tráfico de diferentes *VLANS* y diferentes hosts que estén conectados.

La seguridad es uno de los ámbitos más importantes en las redes, ya que los usuarios tienen sus dispositivos conectados y en estos guardan datos, imágenes, documentos, etc. Por tal motivo existe la necesidad de protegerlos, y no solo los dispositivos de los usuarios, también existe la necesidad de proteger los dispositivos de red, así como los datos que viajan por la red mediante credenciales, algoritmos y criptografía para asegurar al usuario que no va a tener algún robo de información y en el caso de la criptografía si existe un robo de información que esta sea irreconocible para el atacante.

Las grandes compañías como Cisco que se dedican a desarrollar e implementar protocolos, herramientas y características para los dispositivos que interactúan en la red. Se preocupan por mantener la seguridad y cumplir con las políticas de seguridad. Algunas de estas herramientas se enfocan en proteger dispositivos como los *Switches* o *Routers*, tratando de hacer la red sea más segura.

Ya que es peligroso que tanto los hosts (dispositivos finales) como los dispositivos de red se vean comprometidos, por usuarios que no pertenecen a la red como un hacker o usuarios mal intencionados que quieran robar y hacer mal uso de los datos.

Las redes de computadoras son atacadas por hackers o personas mal intencionadas, con el fin de robar información o afectar la operación de la red con fines personales, uno de los ataques más comunes es el famoso *Man in The Midle* (MTM), este ataque consiste en conectar un dispositivo a un red que va a estar obteniendo todo el tráfico de la red poniendo su tarjeta de red en modo promiscuo, y con un analizador de protocolos ver el contenido de los paquetes que esta capturado y robar información valiosa tanto para una empresa como para una persona. Con este tipo de ataques a las redes, las compañías están obligadas a desarrollar características y herramientas para tratar de parar en la mayor medida posible estos ataques

Para contribuir a la seguridad en los quipos de red Cisco desarrolló una característica para los *Switches* de capa dos, llamada Port Security (Puerto de seguridad), dicha característica se configura en los puertos de acceso y permite registrar una o más direcciones físicas en el puerto en el cual este activado.

De esta manera los puertos permiten pasar el tráfico a través de la red de las direcciones físicas retenidas, pero si en estos puertos se conecta un dispositivo diferente al registrado, al tratar de pasar tráfico lo detecta y se toma la acción generalmente de apagar el puerto para detener cualquier intento de tratar de acceder a la red.

El puerto de seguridad tiene muchas ventajas como evitar que un dispositivo desconocido entre en la red, informa del número de veces que se detecta una dirección física desconocida, etc. Sin embargo, el puerto de seguridad tiene inconvenientes y uno de ellos se da cuando se inhabilita un puerto por parte de un *Switch*, ya que éste, no avisa ni lo vuelve a activar y si el administrador no se percata de dicha situación, esto conlleva a consecuencias que pueden llegar a afectar la red.

El puerto de seguridad se puede configurar de muchas maneras, por ejemplo, las direcciones físicas que guarda se pueden asignar estáticas o para que las aprenda dinámicamente, también limitar el número que aprende de estas. La forma en que va a tomar la acción si es que pasa trafico desconocido, puede ser proteger el puerto, restringiéndolo o apagándolo.

El primero no apaga el puerto sino simplemente tira los paquetes que no pertenecen a ellas direcciones físicas guardadas, el segundo también tira los paquetes, pero además genera alertas avisado que existe direcciones físicas desconocidas tratando de acceder a la red, y el ultimo simplemente apaga el puerto evitando que pase cualquier tipo de trafico incluyendo las direcciones físicas guardadas y las desconocidas.

La finalidad de esta propuesta es que, a partir de la configuración extraída de un *Switch*, se detectarán los puertos que están dados de baja por el inconveniente del puerto de seguridad. A partir de lo anterior se realizará una limpieza del puerto, que consiste en limpiar las direcciones físicas retenidas por el puerto, para posteriormente apagarlo y prenderlo, así el puerto vuelve a su estado normal.

Justificación.

En las redes de computadoras existe un dispositivo para conectar más de una computadora, en el mismo segmento de red llamado *Switch*, además en la redes medianas y grandes no basta con tener solamente uno, ya que tienen un número limitado de puertos para poder conectar computadoras, así que estos dispositivos tienen la capacidad de conectarse con otros para ampliar el número de equipos que se pueden conectar al mismo segmento de red, sin embargo, esto implica tener una mayor seguridad.

El puerto de seguridad se ha convertido en una característica básica que se debe implementar en los *Switches* de acceso, pero esta característica una vez que toma acciones para proteger el o los puertos, no regresa a su estado normal y el técnico o administrador de red debe regresar el puerto a su estado normal, este proceso se tiene que hacer manualmente por lo que podría conllevar bastante tiempo de espera en la recuperación del puerto.

Para una empresa grande o una universidad, que manejan gran cantidad de dispositivos, sería muy útil contar con un programa, que de forma automática restablezca el puerto de acceso, ya que las veces que se va a presentar el inconveniente van a ser mayores que en una red pequeña o mediana, además de que las personas expertas en redes no van a tener la necesidad de lidiar con el inconveniente del puerto de seguridad.

Por otra parte, cuando un puerto de acceso lo apaga el *Switch*, es porque, generalmente se viola una política de seguridad de la red, que no necesariamente conlleva un riesgo, por ejemplo, que alguien llegue a conectar un dispositivo de red y que éste no tenga permisos para hacerlo, porque el usuario desconoce las políticas de seguridad. Así que el programa a desarrollar generará una bitácora de los conteos que observa del puerto y a partir de ahí tomará la decisión de abrir un puerto o simplemente dejarlo cerrado.

Antecedentes.

Este proyecto es muy particular, porque no es algo que se haga comúnmente o muchas personas trabajen en algo similar, sino que persigue es un objetivo muy específico que se centra en las redes de computadoras y más aún en una característica que ocupan los equipos de la red.

Los trabajos que más se acercan a este proyecto son trabajos relacionados con la seguridad en las redes de computadoras, ya que hablan de cómo proteger la red, los equipos de red y los tipos de ataques que existen. También los trabajos relacionados que se acercan a este proyecto son relacionados con la seguridad en los *Switches* de capa dos.

En CCNA Security el cual es un libro que emite cisco, explica el funcionamiento del puerto de seguridad y como es que se puede configurar, los comandos necesarios para ello, la forma dinámica de configurar direcciones físicas de manera segura, etc. En este libro es le los poco que hablan de esta característica ya que, al ser desarrollada por cisco, no existen muchos textos o libros que hablen de ella.

Seguridad en redes de datos es un trabajo que tiene mucha relación con este proyecto, ya que habla de la evolución de la seguridad en las rede de datos, como es que han ido cabiendo las formas de ataque como los métodos para defenderse de ellos.

También toca el tema de las redes privadas virtuales (*VLAN*), que contribuyen mucho a la seguridad en los *Switches*, al separar en un solo equipo más de una red de manera virtual para tener diferentes redes.

EL trabajo toca el tema de los *sniffers* en una red, los cuales sirven para capturar tráfico en una red a la cual están conectados. Para evitar que alguien llegue y conecte algo así en la red se necesita de una característica que detenga el tráfico del este, porque el funcionamiento de un *sniffer* es poner en modo promiscuo una tarjeta de red para capturar todo el tráfico de la red.

Poner en modo promiscuo una tarjeta de res se significa que captura todos los paquetes que se envían a la red, aunque no sean los suyos, pero para poder ponerse en ese modo primero tiene que avisar a todos los hosts que existe, al hacer esto la característica del puerto de seguridad bloquearía el intento de anunciarse a todos los demás hosts.

Objetivo General.

Desarrollar e implementar una aplicación que ayude a la seguridad de puertos de un *Switch* de capa de 2, Observando una bitácora y tomando una decisión en base a esta.

Objetivos Específicos.

Diseñar un módulo para el almacenamiento y distribución de datos.

Diseñar un módulo que establezca la conexión entre el programa y un *Switch*.

Programar un módulo que obtenga la configuración asociada con el *Switch* con el que se estableció la conexión previamente.

Diseñar un módulo para inspeccionar la configuración con la finalidad de encontrar el puerto que este dado de baja por el inconveniente del puerto de seguridad.

Marco teórico.

Los dispositivos de red llamados *Switches* tienen muchas características y bondades para poder construir una red, algunas de estas características son que es capaz de separar dominós de colisión, en su configuración de full dúplex en un puerto omite el uso de CSMA, el poder conectar muchos hosts, etc. Pero también cuentan con características y herramientas para hacer más robustos a estos equipos, algunas de las características son, *Spaning Tree*, *Port Fast*, *VLAN*, etc. Pero estos equipos implementan mecanismos y herramientas que mantienen la seguridad de este y por ende mantienen la red segura.

En un *Switch* se puede conectar cualquier usuario a un puerto sin importar si este pertenece a la red o no, y más aún si tiene permisos o no, simplemente conecta el cable al puerto y del puerto al host, de esta manera el usuario ya está en la red y si existe algún servicio de configuración automática, es mucho más fácil estar dentro de la red porque ya solo es activar en el host asignar dirección automática.

Por tal motivo existe la necesidad de alguna manera impedir que cualquier usuario o un usuario no autorizado entre a la red, una manera de proteger un puerto ante esta situación y siendo restrictivo ante los usuarios que pertenecen a la red y los usuarios que están autorizados a pertenecer a la red, es conectar un tipo especial de cable, un puerto especial o alguna política de seguridad, pero estas soluciones afectan al usuario directamente.

La solución que se encontró es restringir a que un puerto guarde direcciones físicas ya sea dinámicamente o estáticamente, dinámicamente quiere decir que el puerto guarda las direcciones, las primeras n direcciones físicas que el puerto reciba tráfico, dependiendo del límite con el cual se configure el puerto, estáticamente el configurador asigna manualmente las direcciones físicas que quiere que el puerto permita pasar el tráfico, de esta manera se restringe a que un host o varios dependiendo de la configuración puedan acceder a la red, y la limitación es que solo los host guardados en el puerto.

De esta manera se asegura que ningún host desconocido pueda acceder a la red, ya que la acción que se toma cuando un host desconocido se conecta es descartar el tráfico, descartar el tráfico y alertar o apagar el puerto para que el trafico ya no pase.

Estas medidas de seguridad conllevan a ciertos inconvenientes que, si no son detectados y atendidos a tiempo, pueden llegar a afectar a la red de manera grave. En el caso de que se configure para solo descartar tráfico, el atacante puede seguir intentando acceder a la red la veces que el desee y, por ende, saturar al dispositivo de peticiones y en algún momento tirar el dispositivo o peor aún lograr acceder a la red.

En la configuración de descartar y alertar, también el usuario puede intentar entrar las veces que el desee, pero en esta configuración el equipo de red alerta cada que recibe un evento, entonces el equipo se saturaría de alertas y puede que se llene la memoria del equipo y este se caiga afectando la red.

La configuración de apagar el puerto es una de las configuraciones que más se usan, porque ya no le permite al usuario acceder las veces que el desee, si no que le limita a solo hacer un intento de acceder.

La ultima configuración conlleva a un inconveniente, porque si el puerto necesita estar siempre en operación o pasa mucho tiempo antes de que se detecte, conllevaría a una afectación en la red o a que un servicio deje de funcionar que anteriormente no tenía ningún problema.

Por tal motivo la solución que se va emplear para este inconveniente es ingresar al equipo, una vez dentro del equipo obtener la configuración de los puertos que están apagados por el inconveniente del puerto de seguridad, ya que no es conveniente obtener la configuración de

todos los puertos apagados, debido a que algunos puertos es posible que deban estar apagados por algún motivo.

Después de obtener dicha configuración hay que analizarla para obtener solamente los puertos y no configuración de mas, una vez con los puertos se conectara de nuevo al equipo para regresar a su estado normal, esto consiste en tirar los comandos para prender y apagar el puerto seguido de tirar el comando que limpia las direcciones físicas que tenga el puerto.

GNS3 es un simulador de redes que se usa de manera profesional para crear maquetas y entornos de desarrollo para probar y crear redes de computadoras, este simulador soporta varios fabricantes de dispositivos en los cuales se encuentra *CISCO*, ***HUAWEI***, etc. También de estos fabricantes soporta una variedad completa de dispositivos *Routers*, *Switches*, *Fire Walls*, etc.

Telnetlib es una librería creada para hacer conexiones vía *TELNET* a cualquier equipo en el que este configurado con este protocolo. Utiliza el puerto 23 para establecer la conexión que es el puerto por default para este protocolo. Esta librería tiene la capacidad de introducir comandos mediante *TELNET*, pero también es capaz de leer la salida que arroja el comando introducido.

Re es una librería que se ocupa para hacer expresiones regulares dentro de *Python*, funciona introduciendo patrones de búsqueda para que los compare en una cadena y obtener el resultado dependiendo de la expresión regular que se diseñe.

Pandas es una librería de *Python* que no es nativa y sirve para manejar archivos *Excel* o *CSV* de manera fácil, esta librería es muy útil cuando se necesita escribir, leer, modificar archivos de hojas de cálculo y también cuando existe la necesidad de pasar los datos de dichos tipos de archivos para operarlo o analizarlos en *Python*, se ayuda de otras librerías como *CSV* que si es nativa para complementar su función.

CSV es un documento de formato libre y sencillo para representar datos en forma de tablas, su sintaxis dice que las columnas se separan por comas y las filas por saltos de línea. Los programas que son capaces de leer hojas de cálculo como Excel y libre Office son capaces de entender esta sintaxis y acomodan la información para visualizarla de mejor manera gráficamente.

XLSX es un documento que pertenece a la suite de Office llamada Excel, este ocupa una sintaxis definida por Office. Sus bondades es que te permite realizar formulas, gráficos, macros para manejar la información de mejor manera. Los programas que son capaces de leer hojas de cálculo como *Excel* y libre *Office* son capaces de leer este tipo de archivos.

Desarrollo del proyecto.

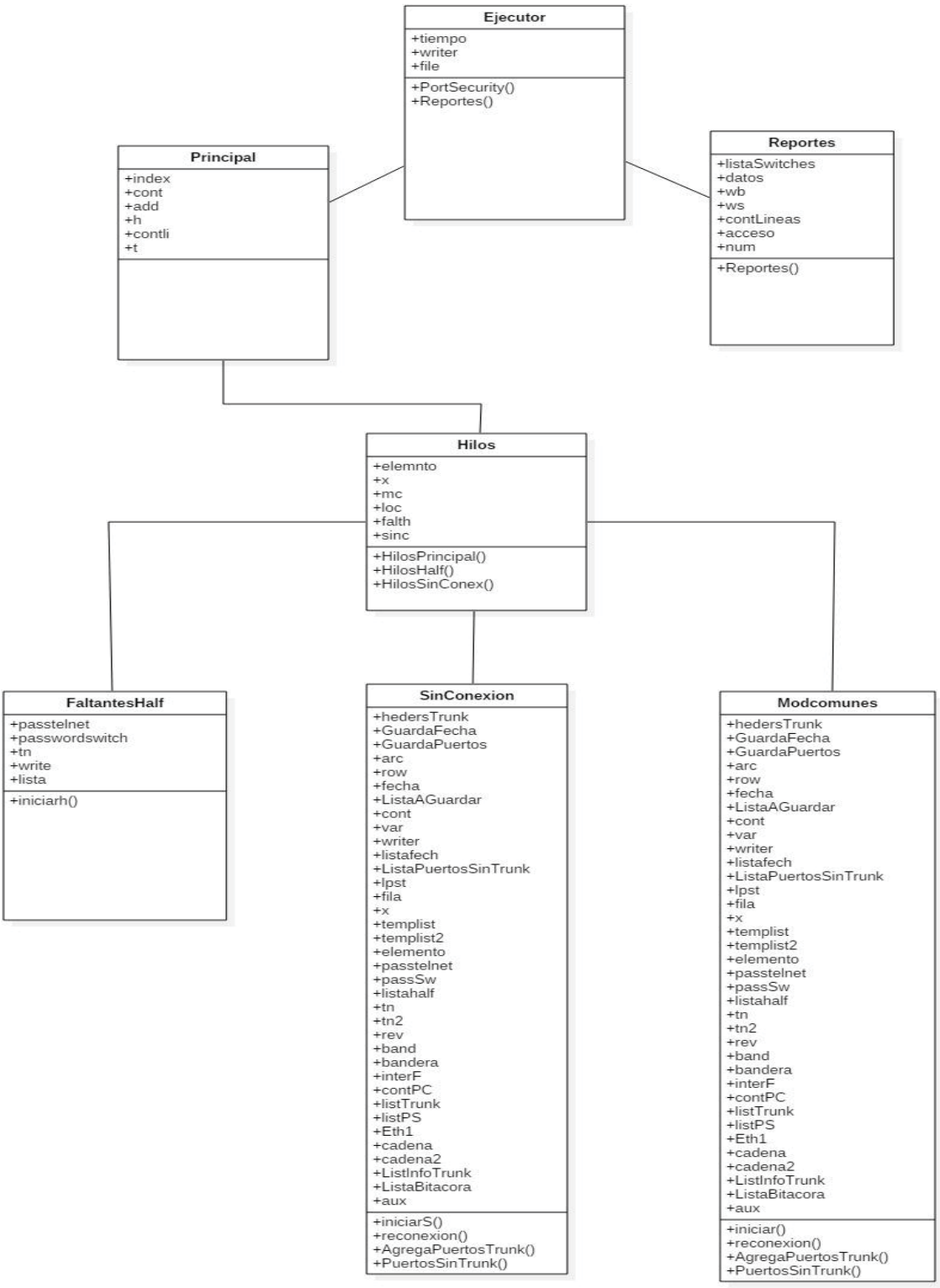


Figura1. Diagrama de clases.

La primera tarea que se realizó fue instalar Python 2.7, también instalar las librerías que se ocuparan para el proyecto. El siguiente paso es instalar el simulador de redes GNS3 junto

con sus imágenes de *Switches* para poder simular el entorno. Otro elemento que se instalo es una máquina virtual con el sistema operativo Linux debían.

Ya instalado lo anterior comencé a construir la maqueta poniendo *Switches* y la Pc virtual con debían, los equipos se configuraron para poder acceder por medio de telnet con los comandos VTY 0 4, *Password* cisco, *Login*. Después se configuro la contraseña para el *Enable*, con los comandos *Enable*, *Secret* cisco. También se configuro la interface para la gestión del equipo, en este caso es la interface VLAN 1 con su dirección IP correspondiente.

Una vez montada la maqueta probé la conectividad de la computadora virtual con los *Switches* que respondieran los pings. Una vez que la maqueta se probó, comencé a estructurar el programa en las clases que se divide y las funciones con las que se compone.

Como se muestra en la figura 1, el programa consta de siete clases, se decidió empezar a programar la clase *Modcomunes* porque es la clase más importante del programa y la más complicada de programar, ya que su función es obtener los datos del *Switch* y posteriormente limpiar los puertos para que regresen a su estado normal. Al momento de estar programando la clase se tuvieron dos variantes para desarrollarla, hacer las conexiones vía *TELNET* o *SNMP*, pero existió un problema con la emulación de los *Switches* para hacer la conexión vía *SNMP* así que se decidió por *TELNET* realizar la conexión.

Continuando con la clase *Modcomunes* al programar los hilos se optó por usar la librería estándar de Python, porque al ser nativa es conveniente usarla ya que cuenta con menos errores y bugs que librerías hechas por personas ajenas.

En el transcurso de la programación de esta clase surgió una disyuntiva la cual fue hacer una base de datos o manejar todo con archivos *Excel* o *CSV*, ya que una base de datos es más robusta y confiable que los archivos, el problema es que depende de que en el sistema operativo tangan instalada una base de datos, en cambio los archivos sin necesidad de instalar nada los sistemas operativos ya contienen aplicaciones para poder abrir los archivos anteriormente mencionados, así que se decidió hacerlo mediante archivos para que fuese compatible en cualquier sistema operativo sin necesidad de instalar nada o crear la base de datos.

Una vez que se terminó de programa y probar la clase anterior, se continuó programando la clase *Hilos*, porque si se tienen muchos equipos es importante que el programa pueda realizar conexiones en paralelo para no depender de que primero termine una conexión y empiece otra, y esto se hace para el rendimiento del programa, así como su desempeño cuando la lista de equipos sea muy grande y no tarde mucho tiempo en hacer el trabajo el programa.

Al estar probando las dos clases anteriores surgieron algunos errores que no se contemplaban en ese momento, de los errores más graves que se presentaron fue cuando no se podía alcanzar el host, también cuando el número de puerto de telnet estaba mal, cuando el

programa no alcanzaba a tirar los comandos y los puertos no se limpiaban por completo o a veces seguían apagados a pesar de que ya se había corrido el programa.

Los primeros dos errores se corrigieron usando excepciones de diferentes tipos, estas excepciones son *IOE* que *Python* cuanta nativamente con ellas, además se agregó una excepción más con la librería socket para cachar errores como el *time out* de una conexión y cuando el socket es incorrecto.

Los siguientes dos errores se corrigieron programando las clases *SinConex* y *FaltantesHalf*, la primera clase tenía dos enfoques los cuales eran, programa una clase nueva para este tipo de errores u ocupar la clase que ya se tenía programada.

Se eligió ocupar la clase que ya se tenía programada porque la función de esta clase es la misma que la de *Modcomunes*, ocupa las mismas funciones y variables a excepción de *SinConexion* se ocupa para volver a intentar establecer conexión con un equipo que no se pudo en la clase *Modcomunes*, así que se reutilizo código de dicha clase y el único cambio que existió entre las dos fue que la primer clase programa manda a un archivo llamado archivo no conectados y la clase *SinConexion* manda a un archivo llamado imposible conexión.

Después de programar *SinConexion* se continuo programando la clase *FaltantesHalf*, a pesar de que es una clase con una sola función y relativamente sencilla es importante tener esta clase, ya que asegura que ningún equipo después de ejecutar *Modcomunes* se quede sin recibir todos los comandos que necesita , esta clase no fue fácil de establecer cómo es que realiza su función, porque si se hacia la conexión para volver a checar los puertos y el estatus de estos, el programa se iba a ser ineficiente ya que haciendo esto es como si volviéramos a correr el programa.

La otra forma que se pensó fue generando un aviso mediante un archivo de que el programa no había podido completar su tarea con determinado equipo, pero esto implicaría que una persona revise dicho archivo y entre al equipo manualmente para ver el estado de los puertos y esto ocasionaría que de cierta forma el programa ya no sea tan automático.

La solución para que el programa mantuviera su esencia de automatizar el proceso fue que el programa con la salida de la conexión *TELNET*, la analizara para que posteriormente guarde la información de los puertos que no se alcanzó a tirar todos los comandos mientras se ejecuta la clase de *Modcomunes* y más específico en la función reconexión, de esta manera no es necesario volver a analizar todos los puertos ni generar un archivo. Ya con la información de los puertos se guarda en un archivo para que después la clase *FaltantesHalf* la tome e ingrese únicamente los comandos de los puertos que fueron guardados en el archivo.

La ultima clase programada fue *Reportes* porque, es la clase que depende de que todas las demás clases se ejecuten, para aprovechar la información que arrojan y poder hacer un archivo de reportes en formato *Excel*, para su mayor facilidad de lectura en cualquier sistema operativo. Esta clase también fue prueba y error, ya que la forma en que se hacen los reportes

es por fila y columna, esto es, primero se escriben las cabeceras en la fila 0 columna 1, 2, 3, etc. Después se escriben la información de los equipos, incrementando el contador de la columna por cada cabecera escrita en el archivo y también incrementando el contador de la fila para ir escribiendo los datos.

Las clases de las que se compone el programa son las siguientes: clase *Ejecutor*, *FaltantesHalf*, *Hilos*, *Modcomunes*, *Principal*, *Reportes*, *SinConexion*.

Ejecutor: esta clase está encargada de obtener el tiempo del sistema y con el comparar si las horas han llegado después de las 0, si es así, se les borrara a los archivos la información que obtuvieron durante el tiempo que se ejecuta el programa, si aún no son las 0 horas, no se borra la información de los archivos, este también ejecuta dos clases además de lo anterior, la clase principal y la clase reportes.

Principal: esta clase obtiene la información de los equipos que se van a clarear los puertos desde un archivo Excel, la información del archivo Excel se pasa a listas para el mejor de la misma en Python, una vez con la información, se ejecutan cuatro ciclos *while* que tienen como límite cincuenta, el primero lanza cincuenta hilos y espera a que acaben estos hilos para posteriormente lanzar otro cincuenta y así sucesivamente hasta que el límite de líneas que contenga la lista con los equipos, este ciclo es el encargado de ejecutar los hilos que van a obtener los puertos y regresarlos a su estado normal.

El segundo *while* tiene la misma función que el primero que va de cincuenta en cincuenta, pero su diferencia es que este *while* trabaja con las líneas de un archivo que se llama incompletos full, también se pasan a una lista, este archivo contiene los equipos que no se les puedo tirar los comandos para extraer la información de los puertos, posteriormente a estos equipos se volverá a conectar para ingresarle los comandos y obtener la información, este ciclo ejecuta el hilo encargado de obtener la configuración y regresar el puerto a su estado normal.

El tercer *while* hace tiene la misma función que el primero que va de cincuenta en cincuenta, pero su diferencia es que este *while* trabaja con las líneas de un archivo que se llama *IncompletosHalf*, también se pasan a una lista, este archivo contiene los equipos que no se les puedo tirar los comandos para regresar a su estado normal el puerto, cualquier comando que no se haya ingresado, este archivo guarda la información de equipo junto con sus puertos para posteriormente volverle a tirar los comandos, este ciclo se encarga de ejecutar los hilos *IncompletosHalf*.

El cuarto *while* tiene la misma función que el primero que va de cincuenta en cincuenta, pero su diferencia es que este *while* trabaja con las líneas de un archivo que se llama archivo no conectados, este archivo contiene los equipos que no se pudo establecer la conexión por cualquier motivo, este archivo también se pasa a una lista en Python, este ciclo ejecuta los hilos sin conexión.

Hilos: este archivo contiene tres clases *HilosPrincipal*, Hilos*Half* e *HilosSinConex*. La primera clase se encarga de ejecutar una función llamada identificación. Esta función

compara el modelo de equipo y dependiendo del modelo lo manda a su respectiva clase si es que se tienen diferentes modelos, antes de ejecutar la clase correspondiente al modelo se pone un *lock adquire* (comando de Python) y al finalizar la ejecución de clase se pone un *lock release* porque estas clases va a ingresar a recursos compartidos que en este caso son los archivos donde se guarda la información.

La segunda clase se encarga de ejecutar *HilosHalf*, esta ejecuta la función también llamada identificación, porque también si existen modelos diferentes de equipos, este los manda a sus respectivas clases, también contiene un *lock adquiere* (comando de Python) y un *lock reléase* (comando de Python) por los recursos compartidos, esta se encarga de ejecutar la clase que mete los comandos que faltaron en los equipos para regresar el puerto a su estado normal.

La tercera clase se encarga de ejecutar hilos sin conexión, esta ejecuta la función sin conexión, esta función también compara el modelo de equipo por si hay modelos diferentes, esta se encarga de ejecutar la clase que intenta a hacer conexión de nuevo con los equipos que en el primer intento no se logró establecer la conexión.

Modcomunes: se encarga de hacer la conexión, analizar los puertos y volver a hacer la conexión para regresar a su estado normal los puertos. Esta contiene cuatro funciones, agrega puertos *Trunk*, puertos sin *Trunk*, reconexión e iniciar.

La primera función está encargada de obtener los puertos *Trunk* del equipo, esto es, obtiene de la configuración los puertos que están en modo *Trunk* y además que estén apagados, esto se hace para reportar los puertos que en ese momento se encontraron con este estatus. Una vez con los puertos detectados, se obtiene la fecha y la hora del sistema, también se obtiene la fecha y la hora del archivo en el cual se va a guardar esta información, dicho archivo se llama puertos *Trunk*, en este archivo se guarda la información del quipo la fecha y hora, también los puertos que se encontraron.

Esta fecha se obtiene porque al correr más de una vez el programa se va guardando en el archivo, tantas fechas, horas y puertos como se corra el archivo, separándolos con una línea por cada vez de ejecución del programa.

Una vez con la fecha del archivo y la del sistema se guardan en variables respectivamente, se busca en el archivo de puertos *Trunk* si el equipo ya existe o no, en el caso de que ya exista se accede al archivo en busca de dicho equipo, se guarda la información de este, más la fecha, la hora y el o los puertos y se actualiza el archivo. Si el equipo no está en el archivo se agrega su información más la fecha, hora y los puertos para que quede en el archivo.

Si es la primera vez que se va a poner un equipo en el archivo, esta función contiene un *Except* para este caso, en el cual se crea el archivo se guarda de este primer equipo su información más la fecha, hora y puertos, además a este archivo se agregan las cabeceras correspondientes a la información, por ejemplo, *host name*, modelo, *IP*, fecha, puertos. Serían las cabeceras para el archivo puertos *Trunk*.

La segunda función se encarga de guardar en un archivo llamado bitácora los puertos en este estado, este archivo contiene la información del equipo, fecha, un contador de las veces que

se ha detectado el puerto apagado y contiene los puertos que se encontraron apagados por cada equipo. Este archivo también guarda las fechas y horas de las veces en que el programa se ejecuta, al igual que el anterior tendrá tantas fechas y horas como el programa se ejecute.

Esta función también cuenta con un *Except* (comando de Python) que funcionara cuando sea la primera vez que se va a meter un equipo al archivo, se crea el archivo se meten los datos del equipo, la fecha y hora, el contador y los puertos, también se agregan las cabeceras para cada campo de información.

La tercera función se encarga de hacer la reconexión, esta función recibe la lista del equipo, la lista con las interfaces para que haga nuevamente la conexión al equipo correcto, tira los comandos configure terminal, interface el tipo de interface y el número de interface, *Shutdown*, no *Shutdown*, *end*, *Clear mac address-table*.

Todos estos comandos se usan para regresar el puerto a su estado normal, terminando de tirar los comandos se obtiene la salida del buffer de la conexión y se guarda en una cadena para analizar si todos los comandos se ingresaron o falto alguno, se revisa la cadena y en caso de que todos los comandos se ingresaron completos la función ya no hace nada, pero si falto algún comando, guarda en un archivo llamado faltantes *half* el equipo y los puertos para volver a tirar los comandos después de que termine esta clase.

La cuarta función se encarga de tirar los comandos para obtener la configuración de los puertos *Trunk* y de los puertos con *err-disbale*, tirando los comandos show interfaces status *err-disable* y *show interfaces status | Include Trunk*, se obtiene esta configuración y se separa en dos cadenas, una de las cadenas tiene los puertos *Trunk* y la otra los puertos *err-disable*.

Para separar los puertos de la configuración se usan expresiones regulares, esta es la expresión regular que se utiliza (Et|Gi|Fa|fa) [0-9] (/+ [0-9]+) +, una vez ya con los puertos separados, los puertos *Trunk* se pasan a su respectiva función y los puertos *err-disbale*, se comparan con puertos *Trunk* para que no se vaya a tocar ninguno y después se manda a la función puertos sin *Trunk*.

En esta función se compara si los primeros dos comednos se ingresaron, en caso de que no se hayan ingresado se manda a un archivo llamado incompletos full, este archivo si no está creado se crea ya que tiene la opción *a+*, que agrega y si no está el archivo lo crea.

En el caso de que suceda una desconexión, la función manda a un archivo llamado sin conexión para que posteriormente se les de otra oportunidad de conexión a esos equipos.

FaltantesHalf: Esta clase contiene una función solamente, esta función recibe la lista con la información del Switch y además recibe una lista con los puertos en los cuales no se ingresaron los comandos completos, con la información del equipo hace una conexión al mismo t con los puertos, ingresa los comandos a cada uno, estos comandos son: *Shutdown*, no *Shutdown*, *End*, *Clear mac address table*. Esta función ya no compara si todos los comandos se ingresaron completos, ya que es una función que ingresa los comandos sin hacer otra tarea y, por ende, los ingresa de una menara muy rápida para que no se tenga problemas de desconexión.

En el caso de que suceda una desconexión, la función manda a un archivo llamado sin conexión para que posteriormente se les de otra oportunidad de conexión a esos equipos.

Sin conexión: Esta clase tiene exactamente la misma funcionalidad que la clase *Modcomunes* las mismas funciones a excepción que si el programa no alcanza a tirar algún comando, este ya no manda a un archivo incompletos. Otra de las diferencias es que, si no se logra establecer conexión con un equipo, se manda a un archivo el cual contiene los equipos que definitivamente no se pudieron conectar junto con su información. También contiene las funciones que agregan los puertos *Trunk* y las que agregan los puertos que contienen *err-disable*.

Reportes es una clase que checa el archivo bitácora, dicho archivo contiene la información del *Switch* mas la información de los puertos junto con un contador que lleva las veces que el puerto se encuentra con *err-disable*, con esta información checa el contador y según el valor indicado por el administrador de redes o del programa va a generar un reporte con base a el contador, si se define un límite de tres, por ejemplo, esta clase va y checa el contador y si el valor es mayor a tres genera un archivo Excel con los datos del *Switch* junto con sus puertos para tener un registro de los puertos.

El programa se realizó de manera gradual, la primera clase que se programo fue la clase de *Modcomunes*, ya que primero tenía que probar la conexión con los equipos vía telnet, esta clase fue de bastante prueba error, debido a que algunas veces no se conectaba la máquina virtual con los *Switches*, a veces las direcciones IP no coincidían.

Después de lograr establecer la conexión y que respondiera el *Switch* al tirar cualquier comando, programé las funciones que agregan los puertos *Trunk* y los puertos con *err—dissble*, estas funciones no fueron prueba y error porque son cadenas, pero si existieron errores que suceden normalmente cuando manejas cadenas y listas, como los índices en las listas, tipos incompatibles entre cadenas, y en Python al copiar listas tiene una sintaxis en específico y si no la sigues las listas no se copian separadas si no se copian como si fuera una sola, ay que si modificas alguna se modifican todas las que copiaste.

Después de la clase *Modcomunes* programe las clases principal e hilos. Ya que estas clases van juntas, ya que principales manda a llamar a hilos para ejecutarse en paralelo, la clase principal no existió problema alguno en programarla, solo fue el pensar cómo iba a ejecutar de cincuenta hilos en cincuenta hilos o dependiendo de las necesidades puede ejecutar de más en más equipos o de menos en menos equipos.

Hilos no represento problema en programarlos solo fue seguir la sintaxis que Python tiene para los hilos, el único detalle que tuve al prográmala fue el uso compartido de recursos y esta pasaba cuando los hilos querían acceder al mismo documento para escribir o consultar, algunas veces no lograban hacerlo, para esto emplee un método en los hilos llamado *lock*, este método evita que los hilos intente acceder al recurso compartido y si no pueden termine se ejecución, si algún hilo está ocupando el recurso los demás los deja en espera hasta que lo libere el que lo está ocupando para que pueda seguir su tarea y terminar de manera exitosa el hilo.

Después de programar la clase *Principal*, *Hilos* y *Modcomunes*, se programaron las clases *SinConexión* y *FaltantesHalf*, ya que estas dos clases tienen código de la clase *Modcomunes*, *SinConexion* contiene exactamente las mismas funciones que *Modcomunes*, a excepción que si no se logra establecer la conexión se manda a otro archivo.

Al estar programando y manejando archivos existió la necesidad de ocupar una librería extra para los archivos, dicha librería se llama pandas, se usa para manjar archivos csv, xlsx y archivos de hojas de cálculo con mayor precisión. Esta librería se llevó su tiempo aprenderla para poder manejar el contenido de los archivos en Python.

Pandas implico errores, porque se aprendió la forma en que se lee el archivo, después la forma en que Python manejaba estos archivos que es con un *data frame*, este *data frame* se puede operar de múltiples formas, pero como se necesitaba en el programa no era tan operable, así que se tuvo que pasar de un *data frame* a una lista para poderla operar de mejor manera y que se adaptase al programa.

Resultados y discusión de resultados.

La topología resultante para este proyecto se muestra en la figura 2, en la cual podemos apreciar que consta de 20 *Switches*, la computadora virtual con *Linux*, y una computadora virtual que pertenece a *GNS3*. Esta última se usa para simular el inconveniente del puerto de seguridad en cada uno de los equipos y por el contrario la máquina virtual de debían se utiliza para mitigar dicho inconveniente.

Parte de la configuración de los equipos se muestra en la figura 3, en la cual podemos ver que contienen la interface usada para gestión que en este caso es la *VLAN*1, también contiene dos puertos que se encuentran administrativamente abajo, esto quiere decir que se apagaron manualmente en la configuración del equipo, esto puertos son *Ethernet* 2/3 y *Ethernet* 1/3.

Los demás puertos del equipo no se usan y están encendido por default por *GNS3*. Esta configuración se encuentra en los demás equipos a excepción que algunos tienen un puerto menos administrativamente abajo.

En la figura 4 se muestra la configuración del puerto de seguridad que es el *Ethernet* 3/3, este es de acceso como se ve en la Figura, también tiene configurada una dirección física estática para que esta solo pueda pasar tráfico, tiene configurado número máximo de direcciones físicas con 1 y que la acción que tome al registrar una dirección física diferente se apague el puerto. Por último, podemos notar que el puerto de seguridad está habilitado.

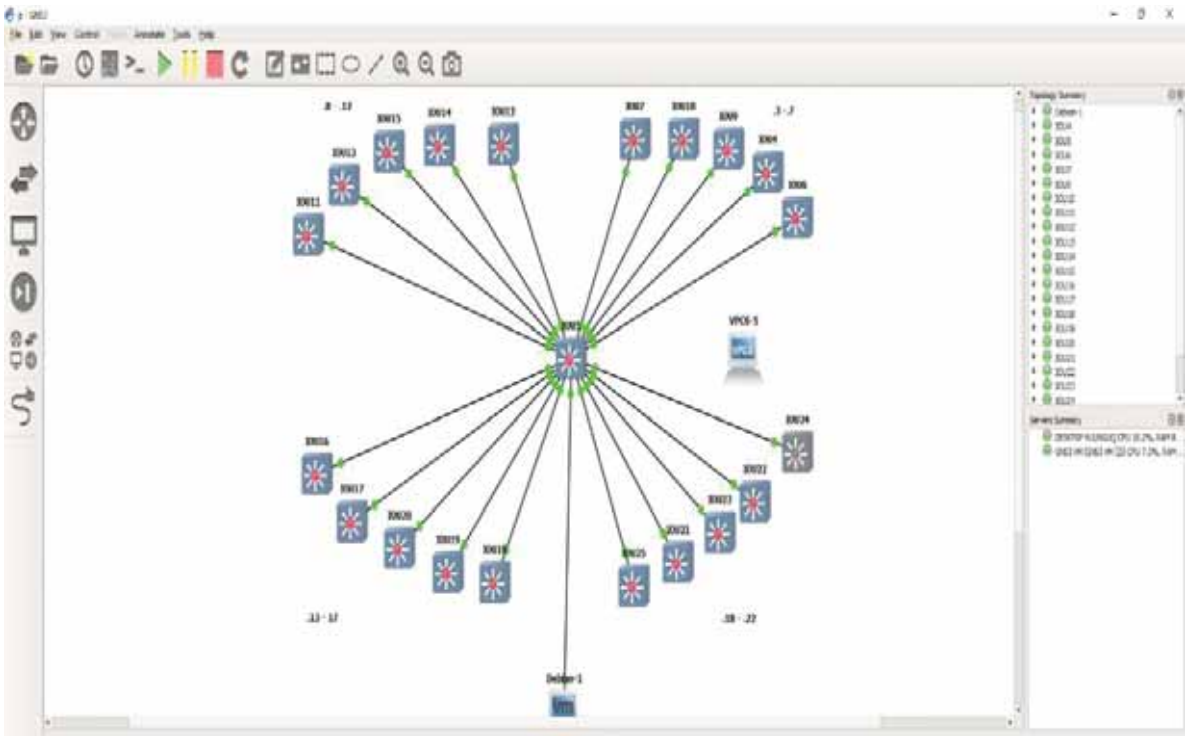


Figura2. Topología de red usada para el proyecto

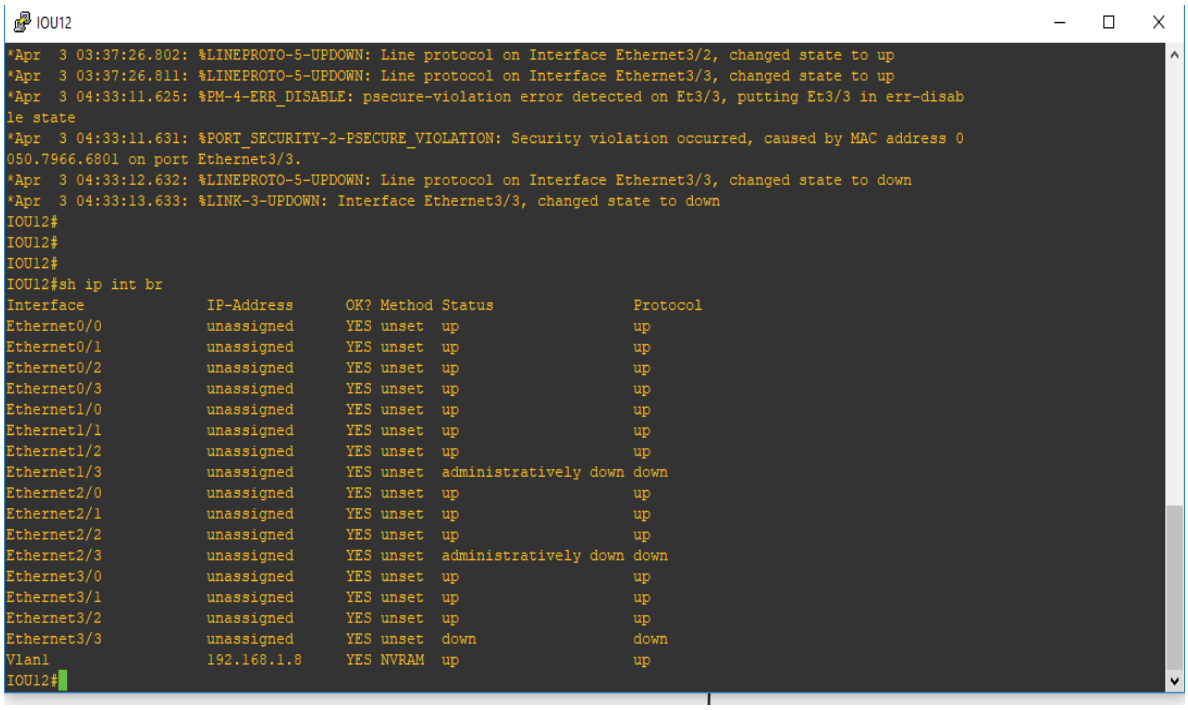


Figura 3. Configuración de *Switches*.

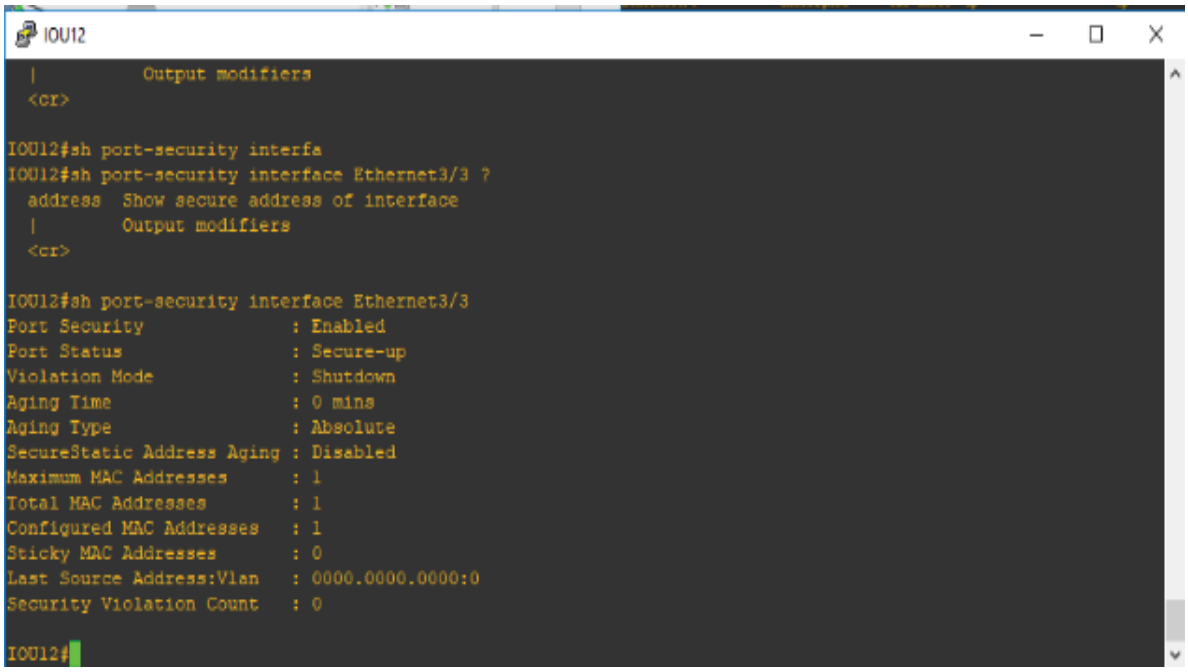


Figura 4. Configuración de *Port Security*.

En la figura 5 se muestran los puertos que se está simulando la acción que toma el puerto de seguridad que como ya se mencionó es la de apagar el puerto, el comando show interface status *err-disabled* muestra los puertos que cayeron en dicha acción, en este caso para fines de simulación solo se configura para que el Ethernet 3/3 caiga en *err-disabled*. También podemos ver que en las cabeceras que arroja el *Switch* se encuentra un que tiene por nombre *reason* y debajo de ella la leyenda *psecure-violation*, indicando el motivo de la violación.

En esta misma figura también muestra la información del estado de las interfaces y cómo podemos ver la interface de *err-disabled* se encuentra *down down*, este estado es como si la interface no tuviera nada conectado y apagada que es lo que hace el puerto de seguridad apagar la interface o puerto.

En la figura 6 podemos ver que se encuentran los puertos configurados en modo Trunk, y además que están apagados administrativamente en este caso, con el comando *show interface status | include Trunk*, este comando se traduce a que de la configuración de las interfaces obtenga los puertos que están configurados de este modo sin importar su estado si está arriba o apagado.

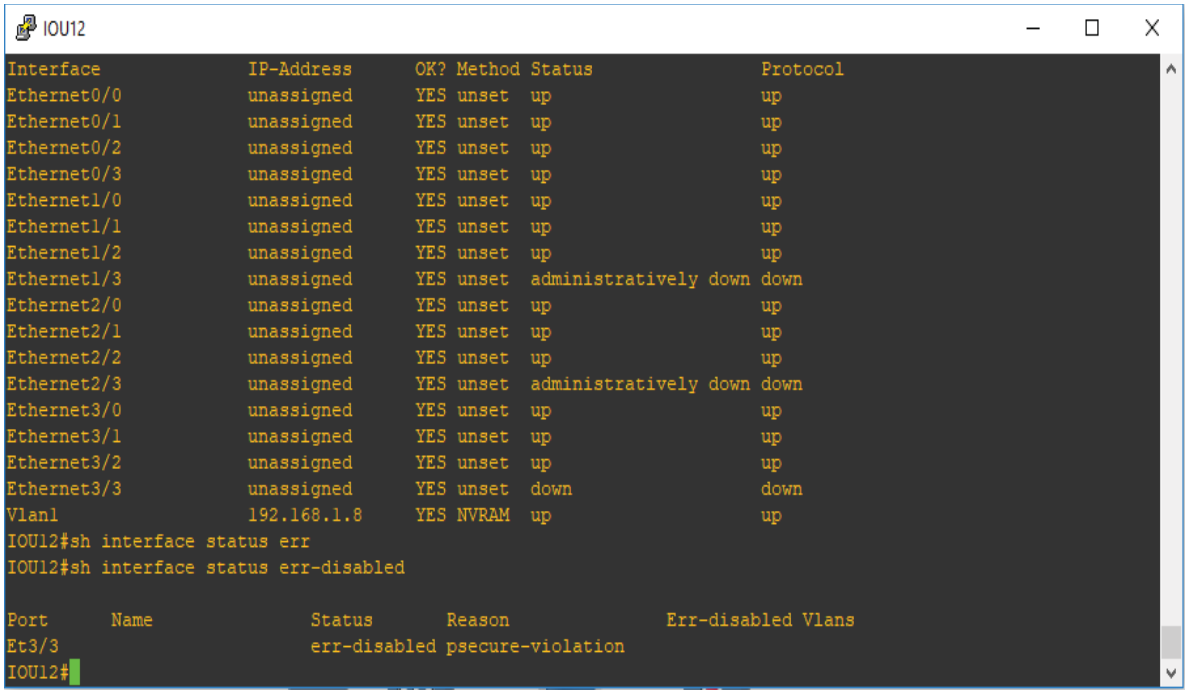


Figura 5. Muestra las interfaces *err-disbaled*.

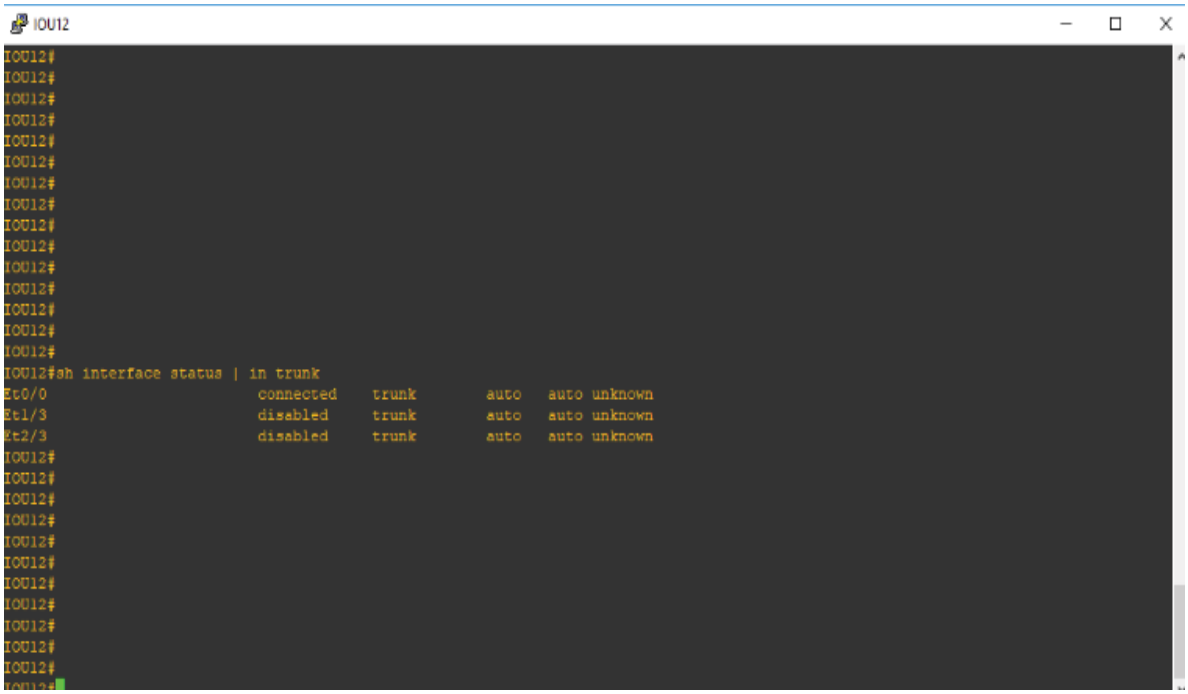


Figura 6. Puertos *Trunk* deshabilitados.

Los archivos mencionados en el desarrollo del programa se encuentran en la figura 7 a excepción bitácora y puertos *Trunk*, ya que se muestra antes de ejecutar el programa y hasta este momento no los ha generado. En la Figura se encuentran archivos no conectados, conexión imposible, equipos, *Incompletosfull IncompletosHalf* que anteriormente fueron descritos cada uno de ellos.

Estos archivos se encuentran en una carpeta y también están vacíos, aunque no del todo cada uno de ellos excepto equipos contienen sus cabeceras y deben mantenerse creados como se muestra en la figura 8, si por algún motivo se elimina alguno de estos, el programa marcara un error ya que no los va a encontrar.

Para el proyecto se usa el formato *CSV*, ya que es un formato simple para manejar y es un formato para manejar hojas de calculo

El archivo llamado equipos contiene la lista de los *Switches* que se tengan en la red que en este caso para el proyecto contiene 20 equipos como se muestra en la figura 9, este archivo también contiene sus respectivas cabeceras las cuales son, *host name* (nombre del equipo), *IP* (dirección *IP* de gestión del equipo), modelo (modelo del equipo), fecha inicio (fecha con la que fue creado el archivo).

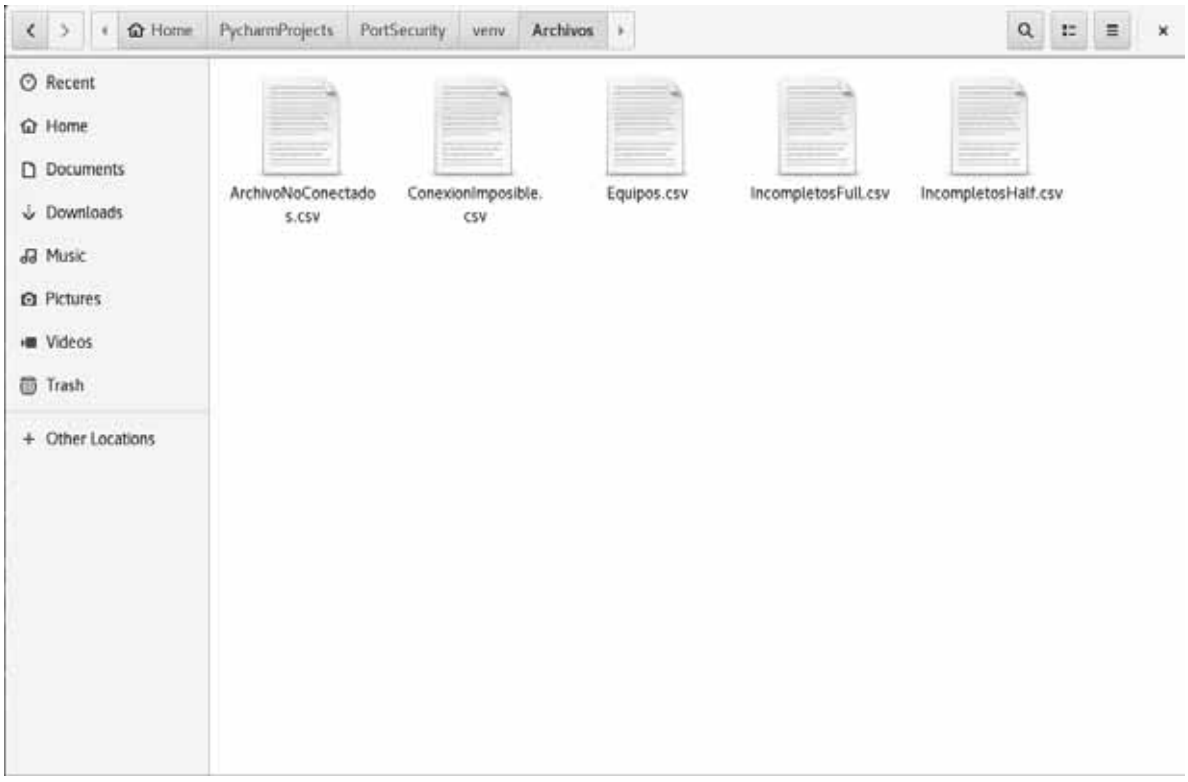


Figura 7. Archivos antes de ejecutar el programa.

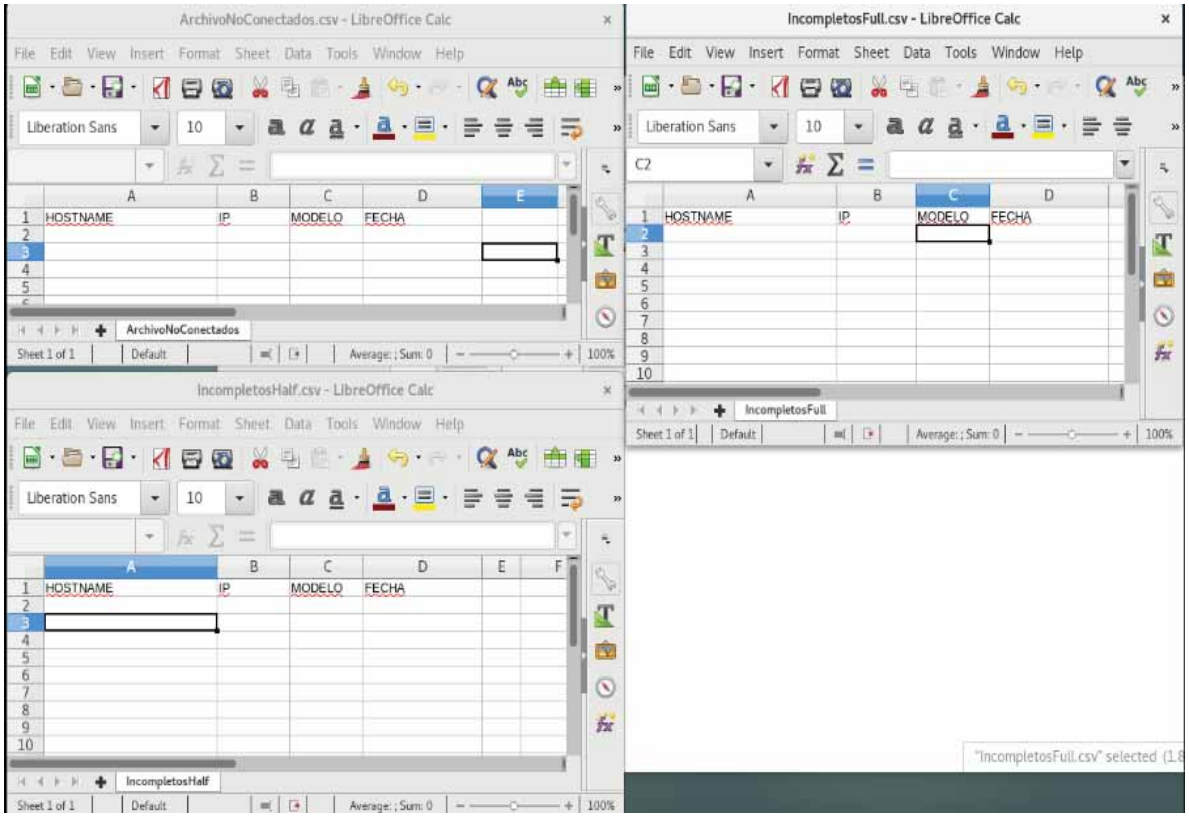


Figura 8. Cabeceras de archivos.

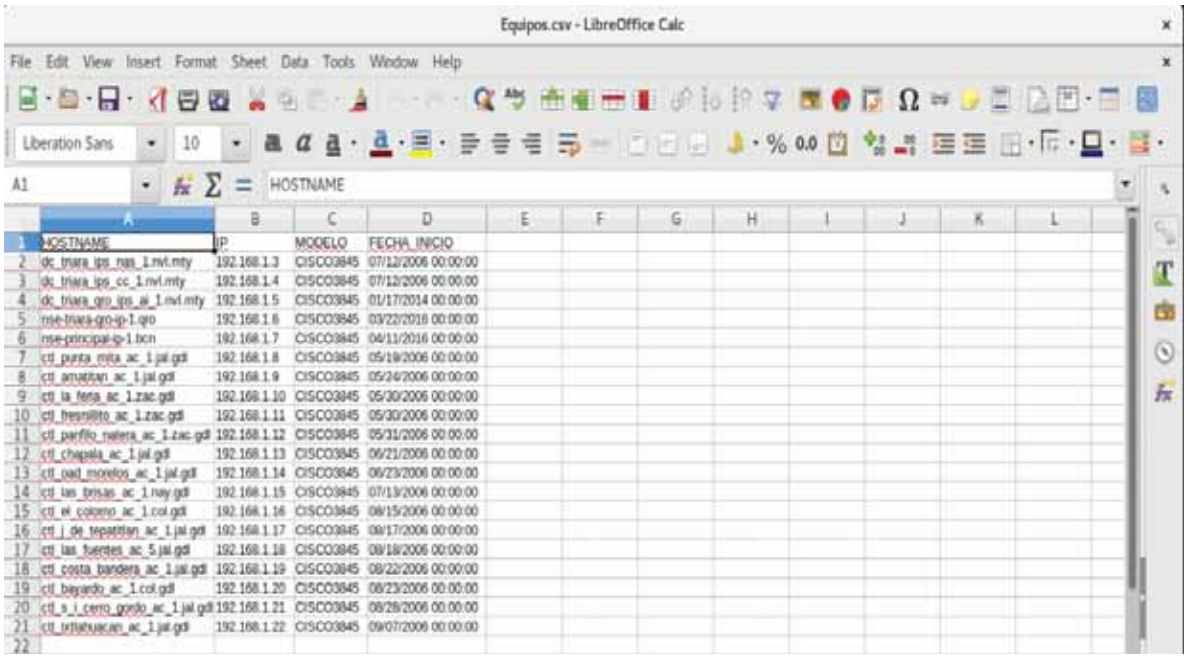


Figura 9. Contenido del archivo equipos.

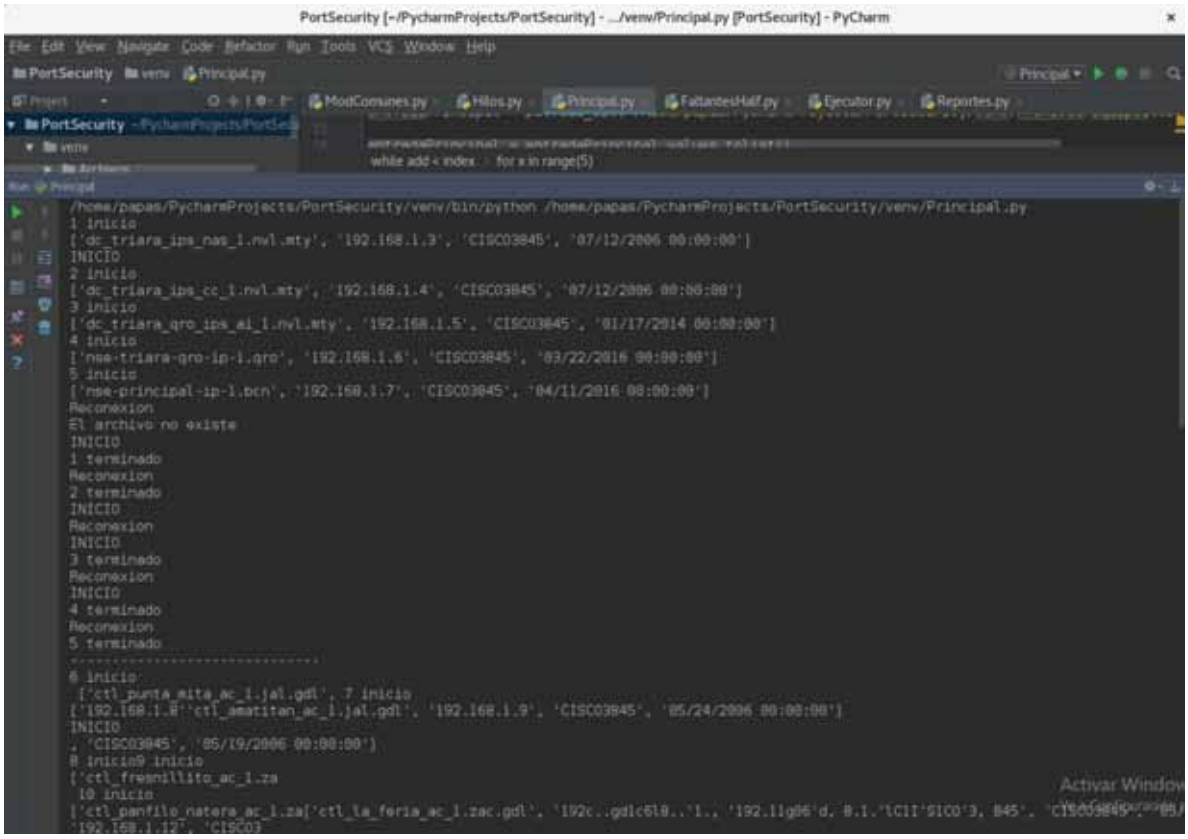


Figura 10. Ejecución del programa.

Al correr el programa se dejaron algunos *Prints* para ver lo que estaba haciendo el programa como un tipo log y seguirle los pasos por si hay algún error ver que paso, donde se quedó o donde se paró. Esto se muestra en la figura 10, contiene los hilos con su identificador que en este caso es una secuencia empezando por el uno. El programa para la simulación corre en paralelo 5 hilos y hasta que no termine la ejecución de los primeros cinco no inician los otros cinco y así sucesivamente.

Esto lo podemos ver en la figura 10 que se separan por una línea punteada, cada hilo muestra la información del equipo, también una leyenda con un inicio junto con su id del hilo y tiene una leyenda de terminado también junto con su id de hilo. Todo esto para saber si la o las funciones que ejecuta el hilo terminan su tarea o que pasa en caso de que exista algún error.

Podemos ver que la ejecución de los primeros cinco hilos fue exitosa porque los hilos muestran la información del equipo más la leyenda inicio y los mismos cinco hilos también imprimen la leyenda terminado que quiere decir que la ejecución fue exitosa, además no marca ningún erro a la salida del programa. La leyenda reconexión es de una de las funciones que ejecutan los hilos, que se usa para meter los comandos para limpiar los puertos y dejar en su estado normal el o los puertos, esta impresión de pantalla también se dejó a propósito a manera de log y monitorear lo que el programa este haciendo.

Hay una excepción en la Figura 10 después de imprimir la leyenda reconexión, ya que en la primera que en este caso pertenece al hilo uno, está la leyenda archivo no existe, pero esto no es un error, también esta se dejó a manera de log porque es la primera vez que el programa se corre o ejecuta. Como el hilo ejecuta las funciones que agrega al archivo bitácora y puertos *Trunk* la información, avisa que el archivo no existe y lo crea para que se puedan agregar más equipos en ambos archivos.

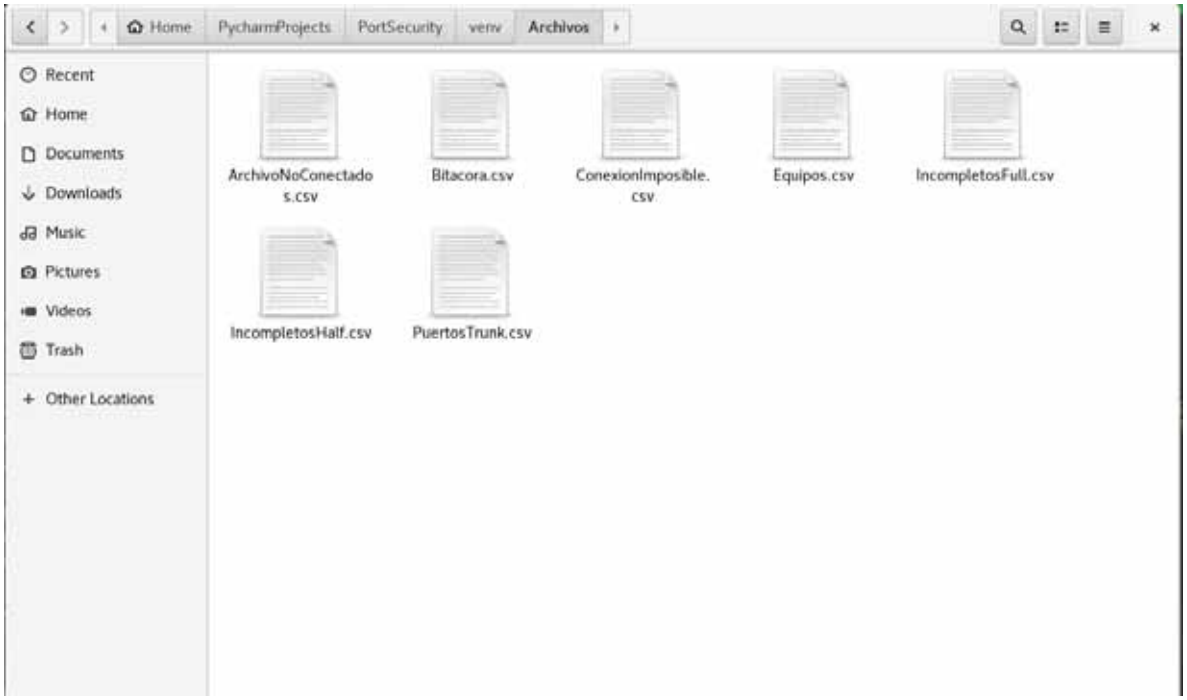


Figura 11. Archivos después de ejecutar el programa.

En la figura 11 podemos ver de nuevo la carpeta que contiene los archivos que genera y usa el programa, pero esta vez se agregaron dos archivos llamados bitácora y puertos *Trunk*, dichos archivos son generados por el programa después de la ejecución exitosa, de otra manera se llenarían alguno de los otros archivos que no son equipos, bitácora y puertos *Trunk*.

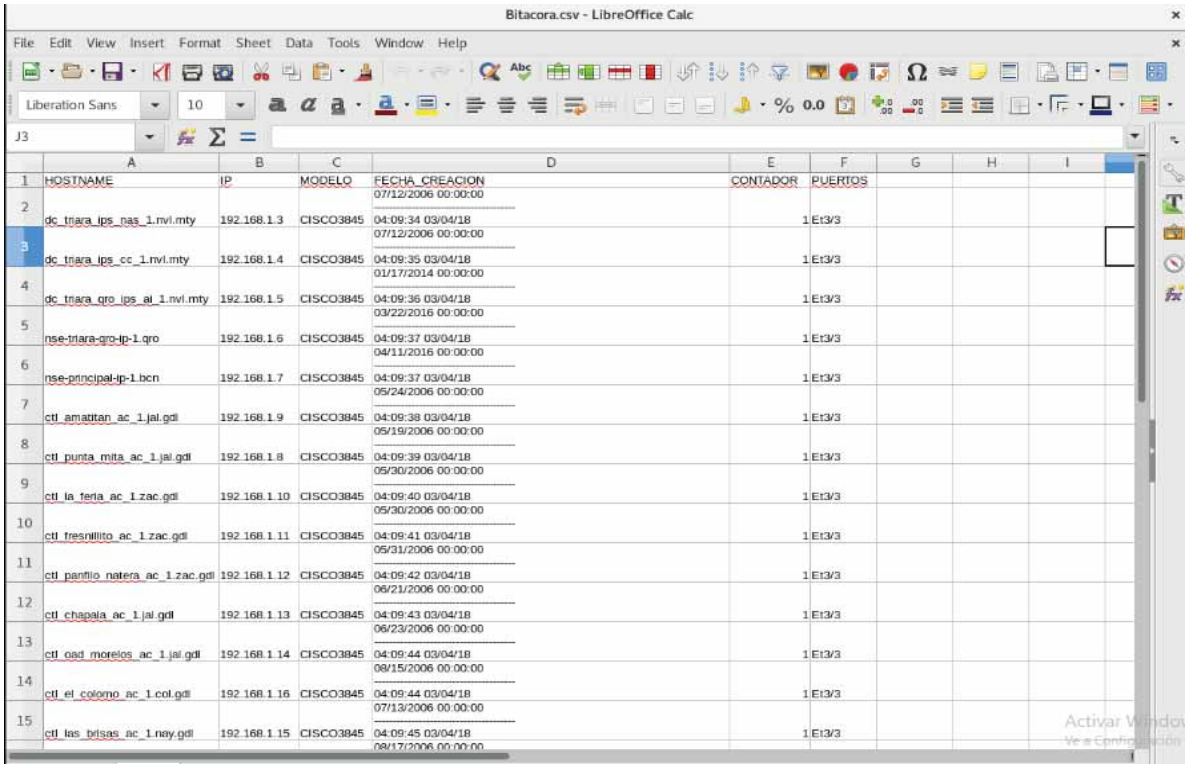
El contenido del archivo bitácora se muestra en la figura 12, vemos que contiene todos los equipos que se detectaron con el inconveniente del puerto de seguridad, el archivo contiene las cabeceras de la información del *Switch* la cual es *host name*, *IP*, modelo, contador, puertos. Es importante notar que el contador de cada uno de los equipos se encuentra en uno, esto es porque la primera vez que se corrió el archivo se simulo que los veinte equipos tuvieran el puerto Ethernet 3/3 apagado por violación de seguridad o se intentó entrar a la red con un equipo de dirección física diferente a la registrada en ese puerto.

Por ende, en la cabecera puertos de cada equipo reporta el puerto el cual ya fue limpiado de direcciones físicas diferentes a la configurada y en su estado normal, esto quiere decir, que el puerto de estar apagado por el por el puerto de seguridad pasa a encendido por la ejecución del programa.

Para ver que el programa si funciono o si realizo de manera exitosa su función, la figura 13 muestra que al tirar el comando show interface status *err-disbaled*, ya no muestra nada porque el puerto o los puertos ya están limpios y funcionando. Y además también se muestran las interfaces, podemos notar que las interfaces Ethernet 1/3 y Ethernet 2/3 siguen apagadas, pero esto es normal ya que estos puertos están administrativamente apagados para la simulación del programa, ya que son puertos *Trunk* y como se explicó en el desarrollo del programa.

Informa los puertos que son configurados en modo *Trunk* y esta deshabilitados. Pero también podemos ver que el puerto *Ethernet* 3/3 esta *up up*. Esto quiere decir, que el puerto se encuentra encendido y funcionando correctamente para pasar el tráfico por el puerto solo con su dirección física configurada estáticamente.

También en la Figura se muestra que no tiene dirección *IP* la interface, pero al ser un *Switch* de capa dos no debe tener dirección *IP* a menos que se vaya a usar para algo en específico o se vaya a activar la función de ruteo en él, de otra manera el trafico pasa si ningún problema.



	A	B	C	D	E	F	G	H	I
1	HOSTNAME	IP	MODELO	FECHA_CREACION	CONTADOR	PUERTOS			
2	dc_triara_ips_nas_1.nvl.mty	192.168.1.3	CISCO3845	07/12/2006 00:00:00 04:09:34 03/04/18	1	Et3/3			
3	dc_triara_ips_cc_1.nvl.mty	192.168.1.4	CISCO3845	07/12/2006 00:00:00 04:09:35 03/04/18	1	Et3/3			
4	dc_triara_qro_ips_ai_1.nvl.mty	192.168.1.5	CISCO3845	01/17/2014 00:00:00 04:09:36 03/04/18	1	Et3/3			
5	nse-triara-qro-ip-1.qro	192.168.1.6	CISCO3845	03/22/2016 00:00:00 04:09:37 03/04/18	1	Et3/3			
6	nse-principal-ip-1.bcn	192.168.1.7	CISCO3845	04/11/2016 00:00:00 04:09:37 03/04/18	1	Et3/3			
7	ctl_amatitan_ac_1.jal.gdl	192.168.1.9	CISCO3845	05/24/2006 00:00:00 04:09:38 03/04/18	1	Et3/3			
8	ctl_punta_mita_ac_1.jal.gdl	192.168.1.8	CISCO3845	05/19/2006 00:00:00 04:09:39 03/04/18	1	Et3/3			
9	ctl_la_feria_ac_1.zac.gdl	192.168.1.10	CISCO3845	05/30/2006 00:00:00 04:09:40 03/04/18	1	Et3/3			
10	ctl_fresnillito_ac_1.zac.gdl	192.168.1.11	CISCO3845	05/30/2006 00:00:00 04:09:41 03/04/18	1	Et3/3			
11	ctl_panfilo_natera_ac_1.zac.gdl	192.168.1.12	CISCO3845	05/31/2006 00:00:00 04:09:42 03/04/18	1	Et3/3			
12	ctl_chapala_ac_1.jal.gdl	192.168.1.13	CISCO3845	06/21/2006 00:00:00 04:09:43 03/04/18	1	Et3/3			
13	ctl_oad_morelos_ac_1.jal.gdl	192.168.1.14	CISCO3845	06/23/2006 00:00:00 04:09:44 03/04/18	1	Et3/3			
14	ctl_el_colomo_ac_1.col.gdl	192.168.1.16	CISCO3845	08/15/2006 00:00:00 04:09:44 03/04/18	1	Et3/3			
15	ctl_las_brisas_ac_1.nay.gdl	192.168.1.15	CISCO3845	07/13/2006 00:00:00 04:09:45 03/04/18	1	Et3/3			

Figura 12. Contenido del archivo bitácora.

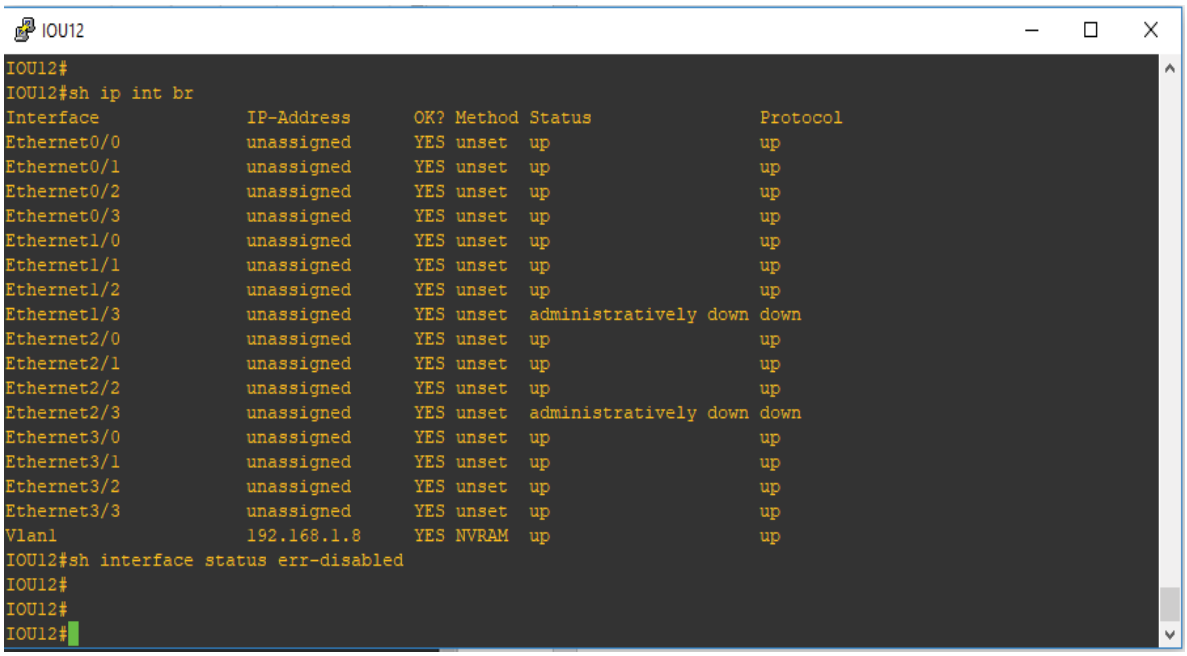


Figura 13. Comando para checar *err-disabled*.

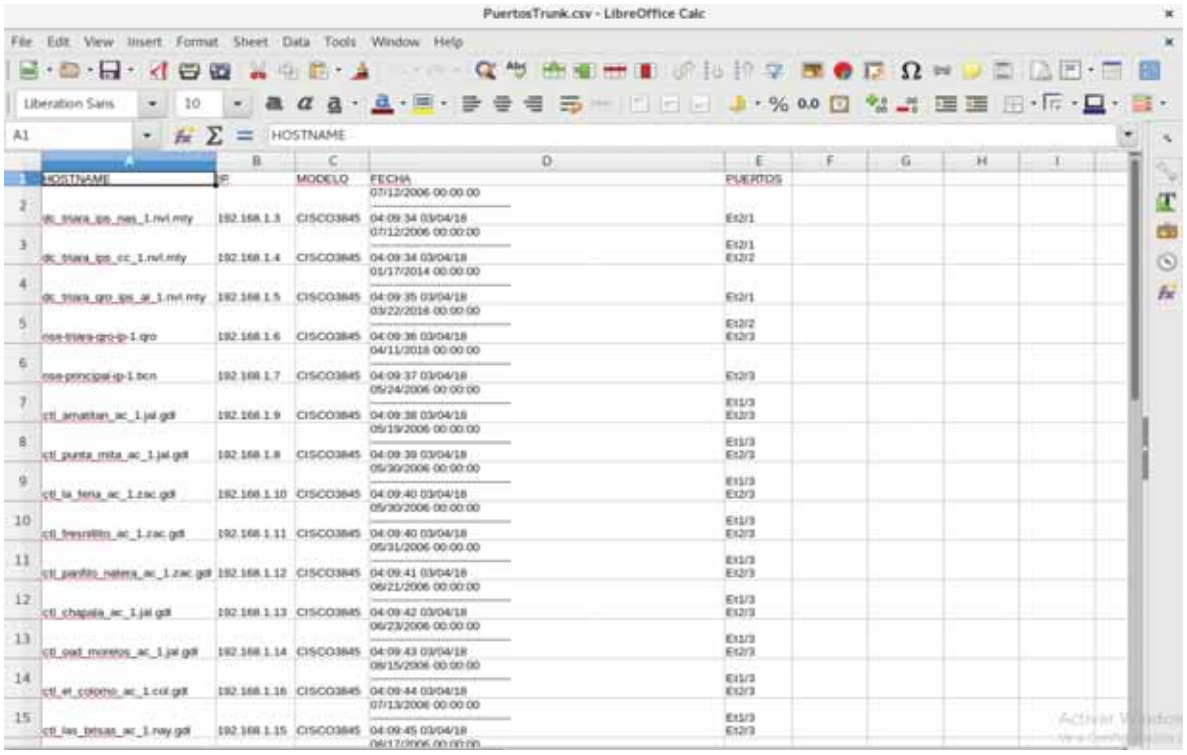


Figura 14. Contenido del archivo puertos *Trunk*.

Los puertos *Trunk* se reportan para evitar que el programa llegue a afectar alguno, solamente se reportan los que se encuentran administrativamente apagados o *down down*, esto se hace para que en instante que se meta el programa garantizar que no se movió nada a excepción de los puertos con *err-disabled*. El archivo contiene cabeceras parecidas a el archivo de bitácoras las cuales son *host name*, *IP*, modelo, fecha, puertos.

A diferencia de del archivo bitácora el de puertos *Trunk* no contiene un contador de las veces que se ejecuta el programa y encuentra el puerto en un estado en el que no debe estar, sino simplemente reporta la fecha y la hora en que ingreso el programa y los puertos *Trunk* que estaban deshabilitado en ese momento para tener un registro y un control de las ejecuciones del programa.

Siguiendo con la ejecución del programa se simulo que el programa no llega a tirar los comandos completos cuando el programa hace la reconexión para limpiar y regresar a su estado normal el puerto. Esto se realizó poniendo un contador en 0, ya que esta función de reconexión checa la salida de telnet y busca los comandos de cada puerto y en caso de que falte algún comando se manda al archivo de *IncompletosHalf*. Esto se muestra en la figura 15, se pueden ver las cabeceras de la información de los equipos las cuales son *host name*, *IP*, modelo, fecha y el o los puertos que les hizo falta algún comando.

Este archivo lo ocupa la clase *FaltantesHalf* en la cual, los comandos se meten todos sin importan si solo falto uno a cada puerto del equipo.

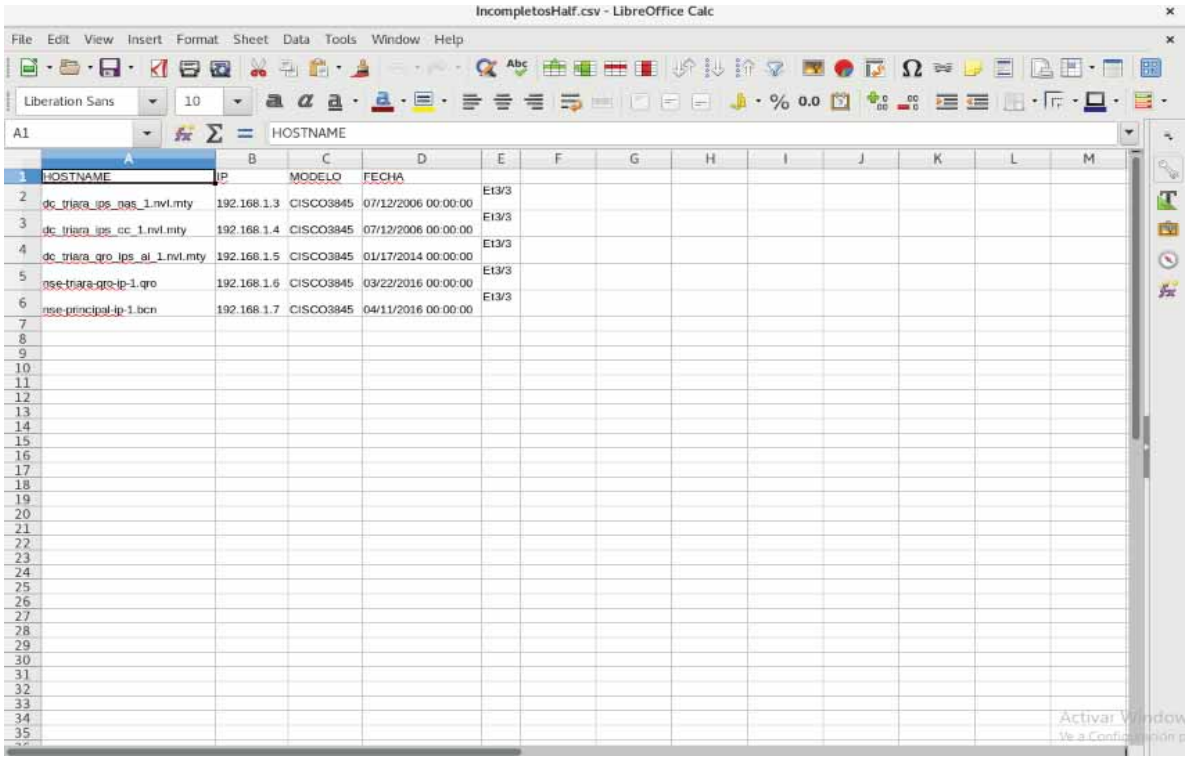
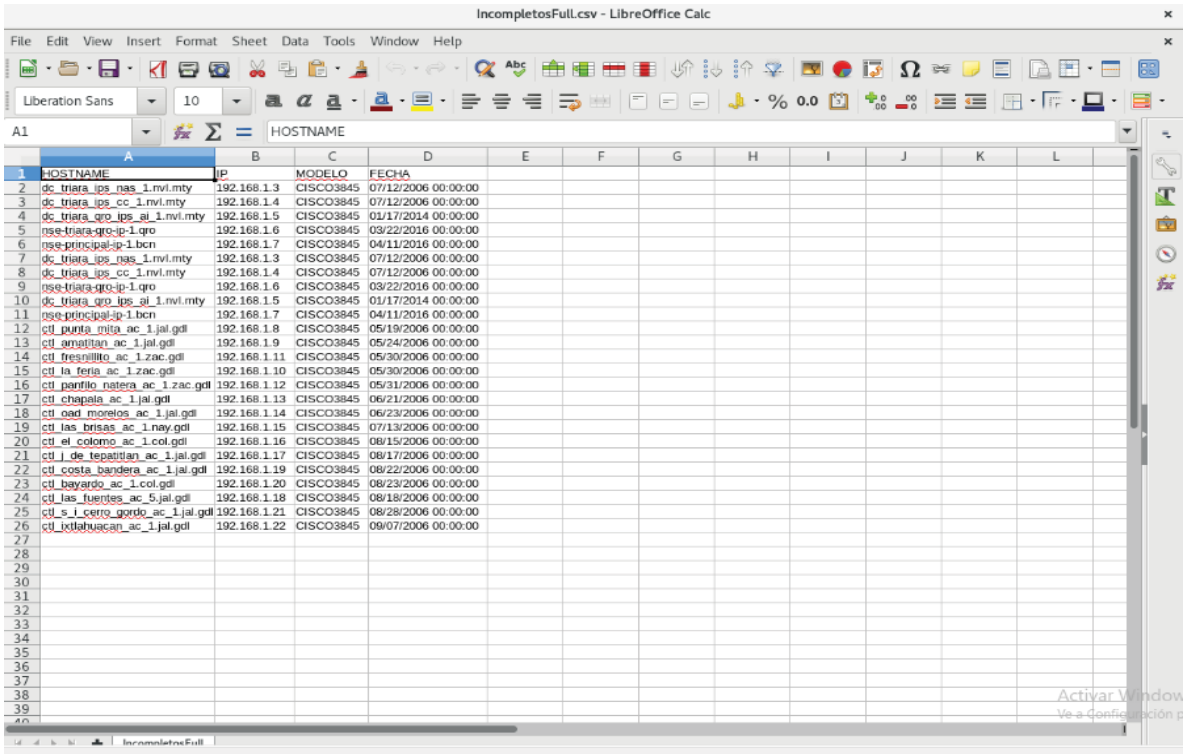


Figura 15. Contenido del archivo *Incompletoshalf*.

También se simulo que no se logró ingresar los comandos principales que obtienen la información de los puertos con *err-disabled* y los puertos *Trunk*. Esto se checa con la funcion de la clase, *Modcomunes* y principal, que con la salida de la conexión de *TELNET* revisa y cuenta que los dos comandos que obtienen la información se hayan ingresado, en caso de no haberse ingresado, el programa los manda al archivo *IncompletosFull* que se muestra e la figura 16.

La figura 15 muestra los resultados de la simulación explicada anteriormente, el archivo contiene la información de los equipos su *host name*, *IP*, modelo, fecha. En este caso se simulo que los veinte equipos no lograron tirar alguno de los comandos en la función reconexión.

Anteriormente en la figura 8 se mostró el archivo sin conexión antes de la ejecución del programa, después de la ejecución del programa se muestra el resultado en la figura 15.



	A	B	C	D
1	HOSTNAME	IP	MODELO	FECHA
2	dc_triara_ips_nas_1.nvl.mty	192.168.1.3	CISCO3845	07/12/2006 00:00:00
3	dc_triara_ips_cc_1.nvl.mty	192.168.1.4	CISCO3845	07/12/2006 00:00:00
4	dc_triara_qro_ips_ai_1.nvl.mty	192.168.1.5	CISCO3845	01/17/2014 00:00:00
5	nse-triara-qro-ip-1.qro	192.168.1.6	CISCO3845	03/22/2016 00:00:00
6	nse-principal-ip-1.bcn	192.168.1.7	CISCO3845	04/11/2016 00:00:00
7	dc_triara_ips_nas_1.nvl.mty	192.168.1.3	CISCO3845	07/12/2006 00:00:00
8	dc_triara_ips_cc_1.nvl.mty	192.168.1.4	CISCO3845	07/12/2006 00:00:00
9	nse-triara-qro-ip-1.qro	192.168.1.6	CISCO3845	03/22/2016 00:00:00
10	dc_triara_qro_ips_ai_1.nvl.mty	192.168.1.5	CISCO3845	01/17/2014 00:00:00
11	nse-principal-ip-1.bcn	192.168.1.7	CISCO3845	04/11/2016 00:00:00
12	ctl_punta_mita_ac_1.jal.gdl	192.168.1.8	CISCO3845	05/19/2006 00:00:00
13	ctl_amatitan_ac_1.jal.gdl	192.168.1.9	CISCO3845	05/24/2006 00:00:00
14	ctl_fresnillito_ac_1.zac.gdl	192.168.1.11	CISCO3845	05/30/2006 00:00:00
15	ctl_la_feria_ac_1.zac.gdl	192.168.1.10	CISCO3845	05/30/2006 00:00:00
16	ctl_panfilo_natera_ac_1.zac.gdl	192.168.1.12	CISCO3845	05/31/2006 00:00:00
17	ctl_chapala_ac_1.jal.gdl	192.168.1.13	CISCO3845	06/21/2006 00:00:00
18	ctl_oad_morelos_ac_1.jal.gdl	192.168.1.14	CISCO3845	06/23/2006 00:00:00
19	ctl_las_brisas_ac_1.nay.gdl	192.168.1.15	CISCO3845	07/13/2006 00:00:00
20	ctl_el_colomo_ac_1.col.gdl	192.168.1.16	CISCO3845	08/15/2006 00:00:00
21	ctl_j_de_tepatitlan_ac_1.jal.gdl	192.168.1.17	CISCO3845	08/17/2006 00:00:00
22	ctl_costa_bandera_ac_1.jal.gdl	192.168.1.19	CISCO3845	08/22/2006 00:00:00
23	ctl_bayardo_ac_1.col.gdl	192.168.1.20	CISCO3845	08/23/2006 00:00:00
24	ctl_las_fuentes_ac_5.jal.gdl	192.168.1.18	CISCO3845	08/18/2006 00:00:00
25	ctl_s_i_cerro_gordo_ac_1.jal.gdl	192.168.1.21	CISCO3845	08/28/2006 00:00:00
26	ctl_ixtlahuacan_ac_1.jal.gdl	192.168.1.22	CISCO3845	09/07/2006 00:00:00

Figura 16. Contenido del archivo incompletos full.

Otra simulación que se hizo fue cambiar el puerto de conexión con el que se conecta telnet. telnet por default se conecta en el puerto 23 de *TCP*, para la simulación se puso el puerto no bien definido 55555, esta se hace para que no pueda establecer la conexión al intentar buscar en el *Switch* a telnet por el puerto 23, si no lo hace por el puerto 55555 y el equipo rechaza la conexión porque no tiene configurado ese puerto para telnet.

La figura 17 muestra los resultados de la simulación explicada anteriormente, el archivo contiene la información de los equipos su *host name*, *IP*, modelo, fecha. En este caso se simulo que cinco equipos no lograron establecer conexión con el equipo.

Anteriormente en la Figura 8 se mostró el archivo sin conexión antes de la ejecución del programa, después de la ejecución del programa se muestra el resultado en la figura 17.

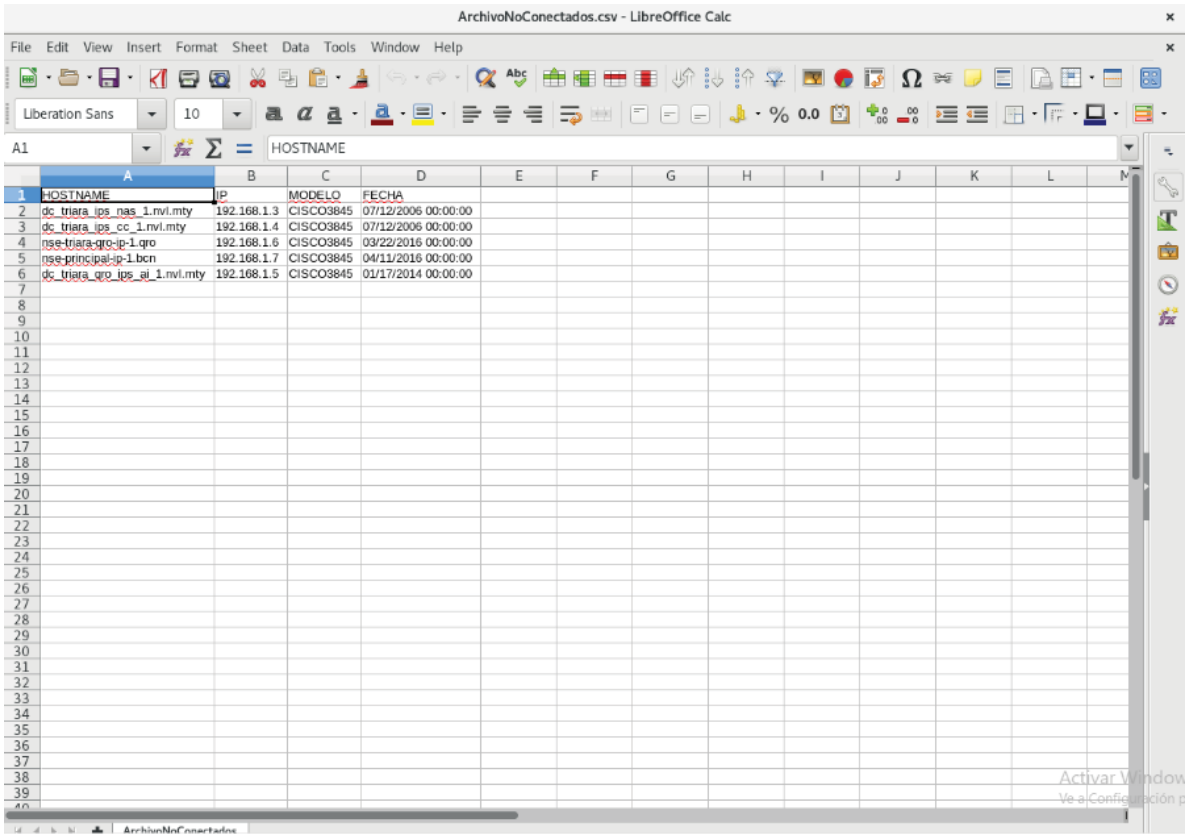


Figura 17. Contenido del archivo, *ArchivoNoConectados*.

La ultima simulación realizada fue con el archivo no conectados, ya que este archivo y su respectiva clase, intenta volver a establecer una conexión con los equipos que están contenidos en el mismo, esta simulación al igual que en la de archivos no conectados se cambió el puerto por default de telnet, aunque se haya simulado de esta manera, el programa también cacha si la ip de gestión está mal, si la red no se puede alcanzar, si la conexión es rechazada o si expira el tiempo en que trata de hacer la conexión según un valor ya establecido.

La figura 18 muestra el resultado de la simulación, al no poder realizar nuevamente la conexión este archivo contiene la información del *Switch* como, el *host name*, *IP*, modelo, la razón por la cual nos e pudo establecer la conexión. A diferencia de los demás archivos este no contiene cabeceras ya que este archivo solamente sirve como registro de los equipos que de plano no se obtuvo gestión de estos.

La razón va a depender de él porque no se pudo establecer la conexión, está la imprime en el archivo las excepciones que se incluyeron en las clases y funciones que ejecuta el programa, dependiendo en que excepción caiga va a ser el motivo de no poder haber establecido la conexión.

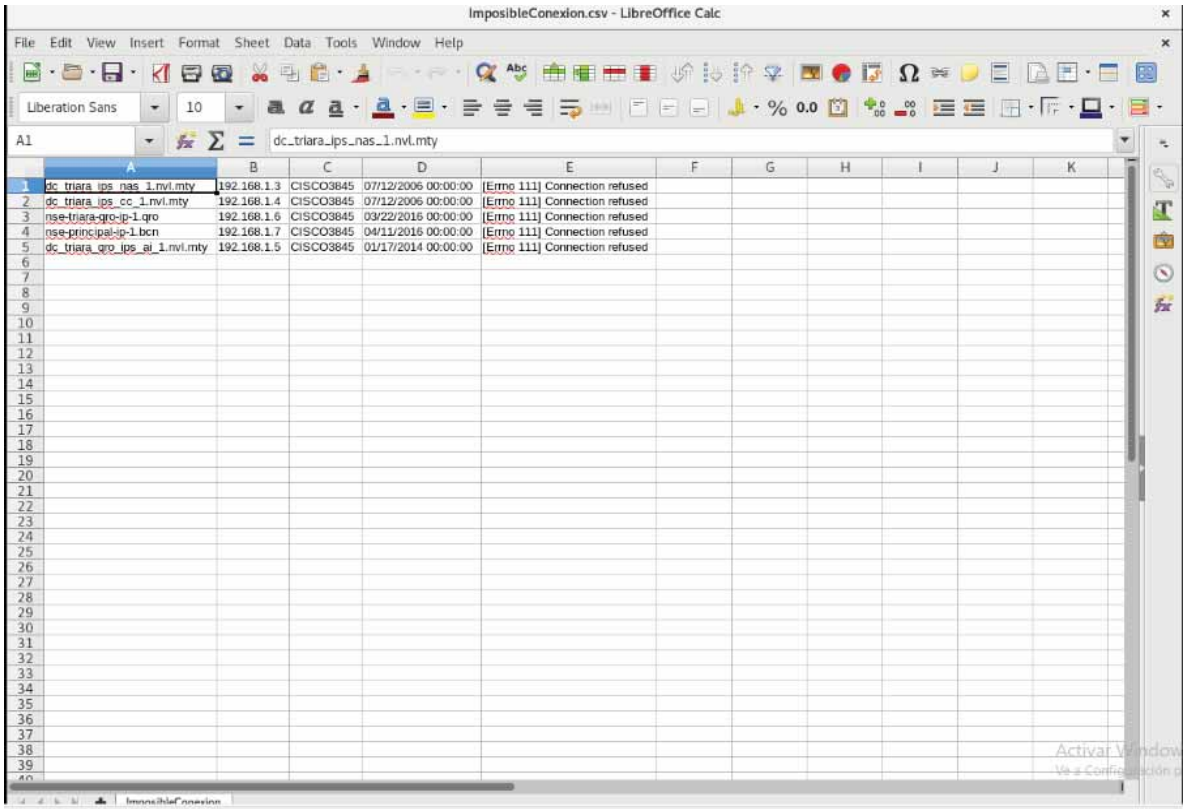


Figura 18. Contenido del archivo imposible conexión

En la figura 7 se muestra la carpeta la cual contiene los archivos que genera el programa y los que ya deben de estar generados y la figura 11 muestra de igual manera lo anterior pero después de la ejecución. En ninguna de estas dos imágenes se muestra el documento que genera la clase reportes ye esto es porque reportes es la última clase en ejecutarse por lo cual aprovecha la información generada por el programa.

Se ejecutó el programa varias veces con algunos puertos con *err-disabled*, para poder simular y ejecutar la clase reportes y esta pueda tener información para poder generar el reporte.

La figura 19 muestra la carpeta donde se encuentran los archivos, ya con el archivo llamado archivo reportes, el cual contienen la información de los equipos que supero tres o más veces que se encontró el o los puertos con *err-disabled.*

La figura 20 muestra la información que contiene el archivo. Dicha información es el *host name*, *IP*, modelo, fecha, puertos, contador, razón. El contador se muestra para que el reporte contenga por qué esta ahí el equipo ya que si es menor a tres no debería estar en ese reporte, pero en este caso los contadores son mayores a tres así que si deben estar en ese archivo de reportes.

También contiene el o los puertos que tienen el contador mayor a tres, la razón es la misma para todos los equipos ya que la única razón para que estén en este archivo es como ya sabemos que el contador sea mayor a tres.

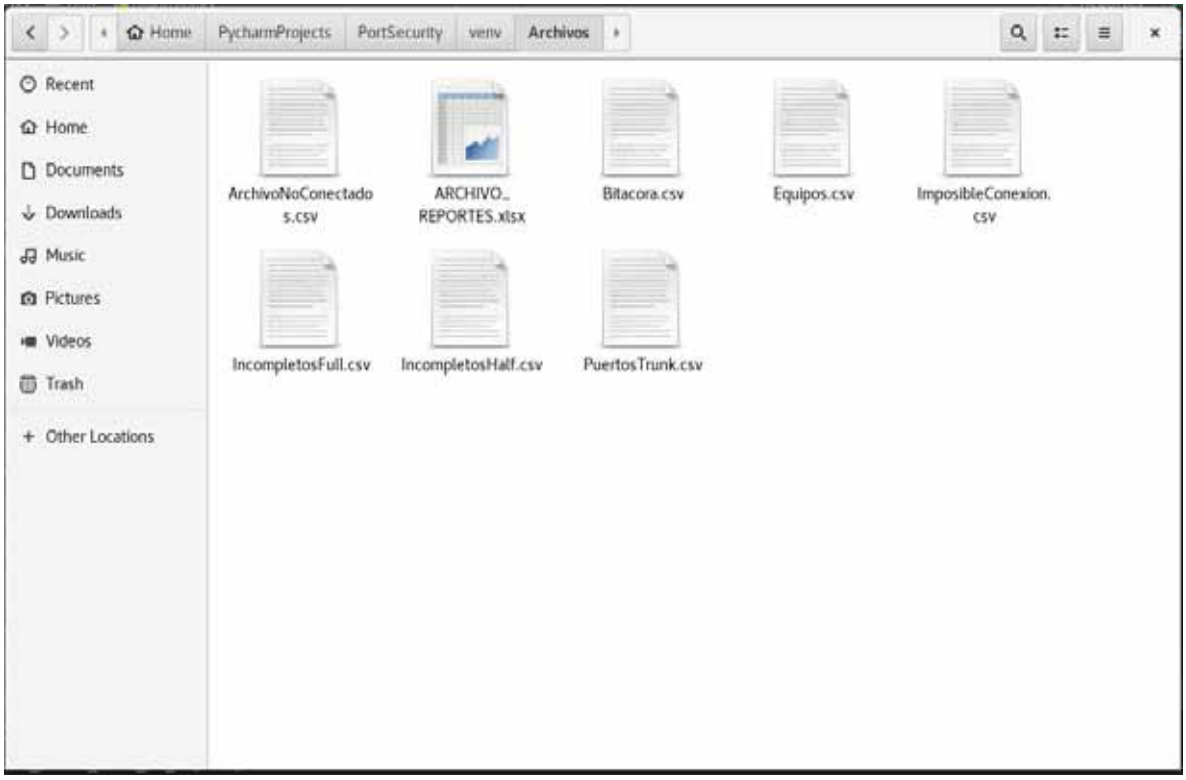


Figura 19. Contenido de carpeta que contiene los archivos.

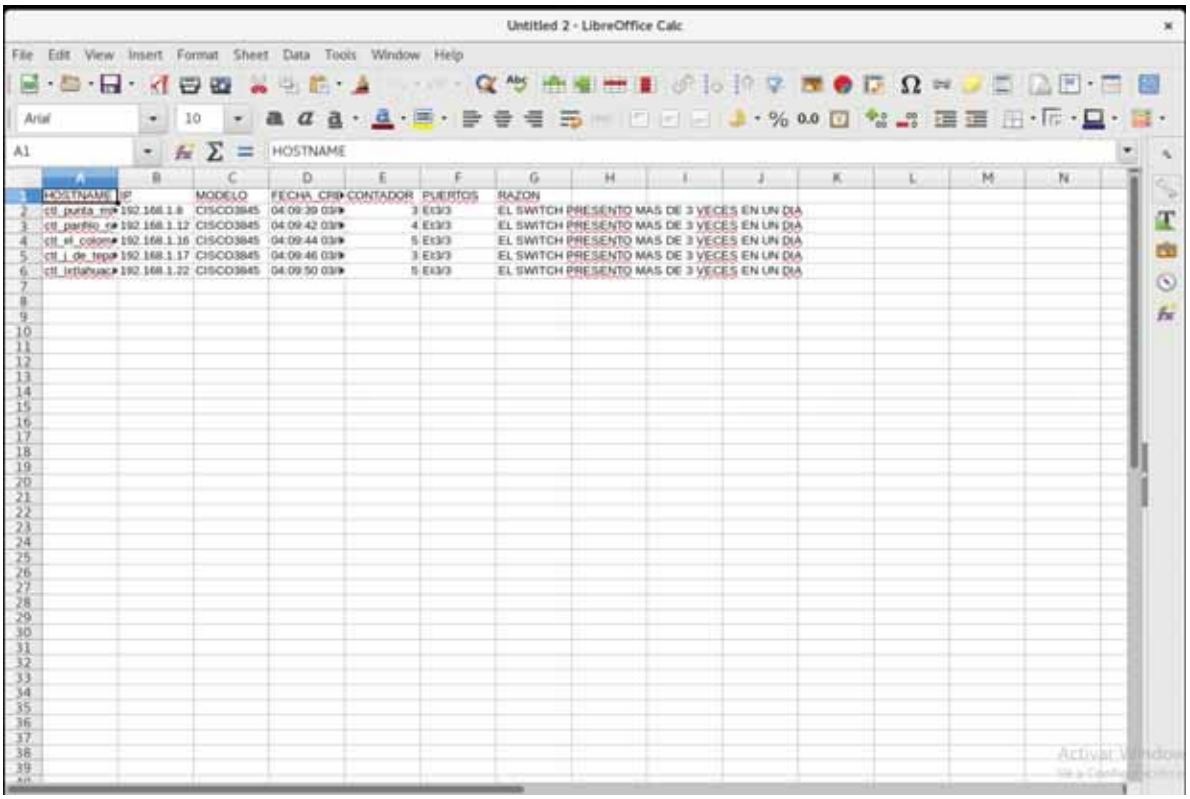


Figura 20. Contenido del archivo de reportes.

Conclusión.

En base a los resultados obtenidos de este proyecto podemos observar, que se ha creado un programa eficiente y que ayuda a la optimización de un proceso que manualmente demanda gastar tiempo. Además, este programa es compatible en cualquier sistema operativo sin necesidad de instalar nada, gracias a que utiliza archivos *CSV* y *XLSX*.

También podemos notar que el programa hace las conexiones mediante *TELNET*, y esto es porque este protocolo no necesita de mucha configuración para funcionar, ni tampoco de objetos especiales para funcionar como *SNMP*. De esta manera el programa se hace adaptable a cualquier equipo.

Además, cabe resaltar que la principal característica del programa es que está escrito en Python, que es un lenguaje muy versátil para desarrollar este tipo de programas y muy sencillo de entender porque contiene muchas librerías que están orientadas a las redes de computadoras.

Este programa utiliza hilos que ayudan a mejorar el rendimiento del mismo, haciéndolo más rápido y versátil cuando se tienen muchos equipos en la red, porque realiza conexiones en paralelo y con ello se puede reestablecer muchos equipos a la vez.

A pesar de que el desarrollo del proyecto se simuló, cabe mencionar que, en un ambiente real, los únicos detalles que se tendrían que modificar para que el programa funcione correctamente son algunos parámetros de acuerdo a la red en la que se pretenda ejecutar y tomar en cuenta los modelos de los *Switches* que se encuentren en dicha red, ya que en algunos modelos cambian los comandos para reestablecer el puerto de seguridad. Este programa realizaría su trabajo efectivamente, porque *GNS3* es un simulador de redes potente que se apega a las redes reales.

El programa desarrollado en este proyecto es una solución muy específica para una empresa de telecomunicaciones o redes y es difícil encontrar o comprar uno como este, por tal motivo este se creó para que exista la solución a la desventaja del puerto de seguridad, en cualquier equipo que cuente con la característica y además que este configurado con la opción de apagar el puerto y de alguna manera se logre acabar con la escases de una herramienta tan específica, para que en un futuro se pueda tomar esta idea para mejorarla y agregarle nuevas funciones.

Bibliografía.

[1] Keith Barker, Scott Morris, CCNA Security 640-554, Editorial Cisco Press, 2013.

[2] Wendell Odom, CCNA ICND2, Editorial Cisco Press, 2016.

[3] Orellana. B. Luis Alberto, H.V. Rafael Cristóbal. "Seguridad en redes de datos", tesis de ingeniería, Universidad Don Bosco, Soyapango 2003.

[4] Zed Shaw, Learn Python the hard way, Editorial Addison-wesley, 2013

[5] Allen Downey, Think Python, Editorial SoHo Books, 2009

[6] Jason C. Neumann, The Book of GNS3, Editorial No Strach Press, 2015

[7] Wendell Odom, CCNA ICND1, Editorial Cisco Press, 2016.

[8] Brett Slatkin, Effective Python, Editorial Addison-Wesley, 2015

[9] Python, "telnetlib", https://docs.python.org/2/library/telnetlib.html, 2018

[10] Pandas, "pandas", https://pandas.pydata.org/, 2018

[11] Cristina Pérez Berro, Algoritmos y Programación, Nueva librería, 2007

[12] Wes McKinney, Python for Data analysis, O´Reilly Media, 2018

[13] Earl Fleming, Telnet 31 Success Secrets - 31 Most Asked Questions On Telnet - What You Need To Know, Editorial Emero Publishing, 2013

[14] Chris Welsh, GNS3 Network Simulation Guide, Editorial Packt Publishing Ltd, 2013

[15] Rapha'l Hertzog, Roland Mas, El manual del Administrador de Debian, Editorial Lulu.com, 2016

[16] Irving Alexander Bermúdez Silva, Debian GNU/Linux Para El Usuario Final, Editorial Lulu.com, 2008

Anexo A.

En el CD que acompaña este trabajo se encuentra el código fuente escrito en Python 2.7 con nombre Port Security. Además, se proporciona la maqueta de *GNS3* para la versión 2.0.6