Trabajo fin de máster

**NetScanAlert**

*Estado: BORRADOR INICIAL*

Autor: David Oliver Gutiérrez

Tabla de contenido

[1.Resumen / Abstract 3](#_Toc200534273)

[RESUMEN 3](#_Toc200534274)

[ABSTRACT 3](#_Toc200534275)

[2.Introducción 4](#_Toc200534276)

[3.Estado del arte 5](#_Toc200534277)

[4.Metodología 7](#_Toc200534278)

[Enfoque de trabajo 7](#_Toc200534279)

[Herramientas utilizadas 7](#_Toc200534280)

[Planificación temporal (cronograma o Gantt) 7](#_Toc200534281)

[Gestión de versiones y pruebas 8](#_Toc200534282)

[5.Análisis del sistema 9](#_Toc200534283)

[Requisitos funcionales, no funcionales, deseables y fuera de alcance 9](#_Toc200534284)

[Casos de uso 11](#_Toc200534285)

[6.Diseño del sistema 14](#_Toc200534286)

[Justificación de decisiones de diseño 14](#_Toc200534287)

[Justificación de herramientas desarrollo seleccionadas 15](#_Toc200534288)

[Arquitectura del sistema 16](#_Toc200534289)

[7.Implementación 17](#_Toc200534290)

[Estructura de carpetas 17](#_Toc200534291)

[8.Pruebas y validación 18](#_Toc200534292)

[9.Despliegue y uso 19](#_Toc200534293)

[10.Conclusiones y líneas futuras 20](#_Toc200534294)

[11.Bibliografía 21](#_Toc200534295)

[12.Anexos 22](#_Toc200534296)

[Preparación del entorno de desarrollo: 22](#_Toc200534297)

## 1.Resumen / Abstract

*Breve descripción del trabajo: objetivo, metodología, resultados. En español e inglés.*

### RESUMEN

Desarrollo de sistema de escaneo y monitoreo continuo de redes con el fin de detectar de forma automatizada equipos no autorizados y con capacidad de reportarlos a través de sistemas de mensajería instantánea.

Objetivos principales:

Mejora el control de los equipos que se conectan a la red.

Mejorar tiempos de respuesta ante intrusiones.

Fácil despliegue y mantenimiento.

Escalable.

Bajo coste.

Metodología:

Resultados:

### ABSTRACT

*Traducir el apartado anterior cuando esté completado.*

## 2.Introducción

*Contexto y motivación del proyecto -*

*Problema que se aborda -*

*Objetivos generales y específicos -*

*Alcance del trabajo -*

*Organización del documento*

El proyecto viene motivado por la necesidad de **controlar de forma eficiente y fácil qué equipos se conectan a las redes locales**, tanto empresariales como personales. Un equipo que se conecta a la red es un potencial problema de seguridad, si no ha sido previamente autorizado y validado por los administradores de la red.

No obstante, los administradores no pueden estar constantemente analizando qué equipos hay conectados en cada momento, más si cabe en el caso de redes corporativas dispersas por un territorio amplio. Se trata de un **problema** grave y es responsabilidad de estos controlar qué hacen los equipos conectados a su red.

Se hace necesario algún sistema que sea capaz de reportar estos accesos a la red, con el fin de que los administradores puedan validar el acceso, o tomar las medidas necesarias para evitar que se conecte a la red.

Existen sistemas de Control de Acceso a la Red (NAC) que permiten esta funcionalidad, con capacidad incluso de validar o bloquear los accesos de los equipos en función de determinadas características de estos (dominio al que pertenece, sistema operativo con soporte, actualizaciones, antivirus activo y actualizado, etc). A pesar de que esto pudiera parecer una solución óptima, el problema es que estos sistemas pueden ser propietarios o de complicada puesta en marcha y requieren de electrónica de red gestionable, lo cual resulta en un elevado coste de implementación.

Entre los **objetivos principales** están el mejorar el control de los equipos que se conectan a la red, así como mejorar tiempos de respuesta ante intrusiones. También se pretende que sea un sistema de fácil despliegue y mantenimiento, escalable y de bajo coste.

El **alcance** del sistema desarrollado, sin llegar a ser propiamente un NAC, sí que permite estar informado de forma rápida de cada equipo que se conecta a la red, con el fin de poder tomar las medidas necesarias en el caso de ser un dispositivo no autorizado, con un coste relativamente bajo y una curva de aprendizaje sencilla para administradores de red.

3.Estado del arte

*Revisión de tecnologías o soluciones existentes -*

*Comparación y análisis crítico -*

*Justificación de las elecciones técnicas -*

**Sistemas NAC:**

Estos permiten autenticación, autorización y verificar la identidad del dispositivo o usuario que intenta conectarse. Pueden evaluar cumplimiento, es decir, revisar si el dispositivo tiene antivirus, firewall activado, parches actualizados, etc. Dependiendo de lo que detecte, pueden dinámicamente permitir el acceso completo, limitarlo (por ejemplo, solo a una red de cuarentena) o denegar totalmente el acceso.

El problema de sistemas NAC propietarios, como por ejemplo los siguientes, es que tienen un elevado coste, ya que requieren del propio sistema, así como de la infraestructura de red gestionable (switchs, firewalls,…):

* Cisco ISE
* Aruba ClearPass
* FortiNAC (de Fortinet)
* Forescout

Existen sistemas NAC basados en software libre pero requieren conocimiento técnico avanzado para instalar y mantener, algunos ejemplos son los siguientes:

* PacketFence
* FreeNAC (menos activo)
* OpenNAC (proyecto descontinuado o inactivo)

**Sistemas de Monitoreo de redes:**

Existen sistemas de monitoreo general de redes como Nagios, Zabbix o LibreNMS, los cuales permiten un monitoreo avanzado de los dispositivos de red, pero más enfocados a la verificación del correcto funcionamiento de estos (disponibilidad, consumo de recursos, espacio en disco, memoria, etc) y envían alertas. Se podría realizar una adaptación de estos para que se reportara la detección de equipos desconocidos en la red.

**OpenWRT + Scripts personalizados:**

Si se dispone un router compatible con OpenWRT y conocimientos técnicos, es posible instalar scripts para monitorear nuevos dispositivos y configurar notificaciones por Telegram, correo o SMS. En el caso de no se disponga de un router de este tipo, o no sea gestionable por el administrador de la LAN, no sería una solución viable.

**Nmap + Cron:**

Es posible igualmente realizar escaneos programados con Nmap, detectar cambios en dispositivos conectados, enviar alertas por correo por ejemplo si se detecta alguno nuevo.

Esta opción requiere conocimientos técnicos de linux, nmap, cron etc.

Es la opción que parece a priori más versátil y en la que se basará el sistema a desarrollar en este trabajo, añadiéndole una capa que facilite la gestión de redes a monitorizar y el envío de alertas mediante el sistema de mensajería Telegram.

## 4.Metodología

Enfoque de trabajo

Se utilizará una metodología por fases tipo “mínimo producto viable + mejoras iterativas” con el fin de conseguir avances de forma ordenada y evitar complicaciones antes de tiempo.

### Herramientas utilizadas

Se ha considerado la utilización las herramientas siguientes:

Virtualización sobre **Oracle Virtualbox**. Se considera que el uso de un entorno virtualizado es clave a la hora de realizar un desarrollo sobre un Sistema Operativo lo más limpio posible, de cara a las diferentes fases del ciclo de vida de software: Desarrollo, pruebas, despliegue y mantenimiento.

El desarrollo se realizará sobre una máquina **Ubuntu Desktop 24.04-2 LTS**.

Para el repositorio de código fuente se utiliza **GitHub**: **https://github.com/davidog7/netScanAlert**

### Planificación temporal (cronograma o Gantt)

A continuación, se muestra la planificación temporal prevista. Se ha optado por realizar este seguimiento utilizando la herramienta en línea y gratuita Trello.

https://trello.com/invite/b/68078a83628bf34186791f91/ATTIe0cb9b822227b3bfdc347c8d1b3e55ed575724AD/netscanalert

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

### Gestión de versiones y pruebas

## 5.Análisis del sistema

Requisitos funcionales y no funcionales

Casos de uso o descripciones funcionales

Diagramas de análisis (si aplica)

Perfil de usuario y contexto de uso

### Requisitos funcionales, no funcionales, deseables y fuera de alcance

**Requisitos Funcionales (RF) ¿Qué debe hacer el sistema?**

* RF-01: Detección Automática de Dispositivos

El sistema debe identificar equipos nuevos que se conectan a la red local mediante técnicas activas (ARP, ICMP).

* RF-02: Inventario de Dispositivos

Mantener una base de datos actualizada con información de los dispositivos (MAC, IP, nombre, SO, fabricante, descripción).

* RF-03: Notificaciones en Tiempo Real

Enviar alertas (Telegram) cuando un dispositivo no autorizado se conecte.

* RF-04: Clasificación de Dispositivos

Diferenciar entre dispositivos conocidos (autorizados) y desconocidos (no autorizados) mediante listas blancas.

* RF-05: Integración con Redes Existentes

Funcionar en redes con o sin electrónica gestionable (ej: switches no gestionables).

* RF-06: Interfaz de Administración

Ofrecer un panel web o CLI para gestionar dispositivos, configurar alertas y generar informes.

* RF-07: Histórico de Conexiones

Registrar fecha/hora de conexión/desconexión de cada dispositivo para auditoría.

* RF-08: Soporte para Múltiples Subredes

Escanear redes en diferentes VLANs o segmentos IP sin necesidad de despliegue físico adicional.

**Requisitos No Funcionales (RNF) ¿Cómo debe funcionar el sistema?**

* RNF-01: Bajo Coste

Debe utilizar software de código abierto y hardware asequible (ej: Raspberry Pi, servidores virtuales o contendores docker).

* RNF-02: Escalabilidad

Soportar desde redes pequeñas (hogar/PYMES) hasta corporativas (hasta 1000+ dispositivos).

* RNF-03: Facilidad de Despliegue

Instalación en menos de 30 minutos, con configuración mínima.

* RNF-04: Compatibilidad

Funcionar en Linux/Windows y soportar dispositivos IoT, BYOD, y sistemas operativos variados (Windows, macOS, Linux, Android/iOS).

* RNF-05: Bajo Impacto en la Red

El escaneo no debe saturar el ancho de banda (ej: usar sondeos espaciados en el tiempo).

* RNF-06: Seguridad

El sistema debe seguir prácticas de seguridad desde el diseño.

* RNF-07: Rendimiento

Detección de nuevos dispositivos en menos de 5 minutos desde su conexión.

* RNF-08: Curva de Aprendizaje Sencilla

Interfaz intuitiva para administradores sin experiencia en NAC complejos.

* RNF-09: Modularidad

Permitir añadir funcionalidades futuras (ej: integración con SIEM, bloqueo automático vía API).

**Requisitos Opcionales (Deseables)**

* RF-09: Bloqueo Automático (si hay hardware gestionable)

Integración con firewalls/switchs para aislar dispositivos no autorizados.

* RF-10: Análisis de Vulnerabilidades

Escaneo básico de puertos abiertos o servicios inseguros en dispositivos.

* RNF-10: Alta Disponibilidad

Configuración en clúster para redes críticas.

**Fuera de alcance**

No será un NAC completo (sin autenticación 802.1X avanzada).

No requiere hardware caro (ej: switches enterprise).

Esta estructura cubre los objetivos de control de seguridad, bajo coste, y facilidad de uso.

### Casos de uso

**Caso de Uso UC-01: Detección Automática de Nuevos Dispositivos**

Actor: Sistema (automático)

Descripción: El sistema identifica dispositivos desconocidos al conectarse a la red.

Flujo:

El sistema escanea la red cada X minutos.

Detecta una MAC/IP no registrada en la base de datos.

Registra el dispositivo como "No autorizado" y almacena sus datos (MAC, IP, fabricante).

Trigger: Envía una alerta al administrador (Telegram).

Variantes:

Si el dispositivo está en la lista blanca, registra su conexión sin alertas.

Si la IP cambia, pero la MAC es conocida, actualiza el registro.

**Caso de Uso UC-02: Notificación de Intrusos**

Actor: Sistema → Administrador

Descripción: Alertar sobre dispositivos no autorizados.

Flujo:

El sistema genera una alerta con detalles (MAC, IP, hora de conexión).

Envía la notificación por canales configurados (Telegram).

El administrador recibe la alerta y decide acciones (ignorar, investigar o bloquear manualmente).

**Caso de Uso UC-03: Gestión de Listas de Dispositivos**

Actor: Administrador

Descripción: Clasificar dispositivos autorizados.

Flujo:

El administrador accede al panel web o terminal.

Consulta la lista de dispositivos detectados (filtros por estado: "Desconocido", "Autorizado").

Selecciona un dispositivo y:

Opción 1: Lo añade a la lista blanca (ej: equipo legítimo).

Opción 2: Lo marca como "Bloqueado" (registra MAC en lista negra para investigar, bloquear o que no se reciban más alertas sobre el mismo).

Variantes:

Importar listas masivas via CSV (ej: todos los equipos corporativos).

**Caso de Uso UC-04: Generación de Informes de Auditoría**

Actor: Administrador

Descripción: Obtener reportes históricos de conexiones.

Flujo:

El administrador selecciona un rango de fechas.

El sistema genera un CSV con.

Dispositivos únicos conectados.

Hora de detección.

Dispositivos desconocidos.

**Caso de Uso UC-05: Integración con Firewall (Opcional)**

Actor: Sistema → API de Firewall

Descripción: Bloqueo automático de amenazas.

Flujo:

El sistema detecta un dispositivo desconocido o en lista negra.

Ejecuta una API call al firewall (ej: pfSense, Cisco) para bloquear su MAC/IP.

Registra la acción en logs y notifica al administrador.

Precondición: Firewall con API habilitada.

## 6.Diseño del sistema

Arquitectura general del sistema

Diseño de base de datos (si aplica)

Estructura del software: frontend, backend, APIs

Justificación de decisiones de diseño -

### Justificación de decisiones de diseño

**Escaneo de redes remotas o subredes a las que no se tiene acceso directo:**

Detectar equipos que se conectan a una red cuando tienen un firewall activado puede ser un desafío, ya que los firewalls están diseñados para bloquear solicitudes no autorizadas y ocultar los dispositivos. Sin embargo, hay varias técnicas que se pueden utilizar dependiendo del nivel de acceso a la red y las herramientas disponibles:

Aunque un firewall puede bloquear ciertos puertos, Nmap tiene técnicas para evadir restricciones:

nmap -sS -Pn -T4 [Rango de IP] *# Escaneo SYN sigiloso (no completa la conexión TCP)*

nmap -sT -Pn [IP] *# Escaneo TCP completo (más lento, pero puede pasar algunos firewalls)*

nmap -sU -p 53,67,68 [IP] *# Escaneo UDP (útil para servicios como DHCP/DNS)*

nmap -O --osscan-guess [IP] *# Detección de SO basada en comportamientos*

nmap -f --mtu 24 [IP] *# Fragmenta paquetes para evitar detección*

nmap --script firewall-bypass *# Usa scripts de Nmap para probar bypass*

**Escaneo de la red local:**

Los firewalls no bloquean tráfico ARP en la red local, por lo que se puede detectar dispositivos incluso si están "invisibles" en capas superiores:

arp-scan --localnet

nmap -sn -PR [Rango de IP] *# Escaneo ARP (solo en la misma subred)*

**Otras opciones:**

Sería posible realizar otras opciones como monitorear la red inspeccionando el tráfico DHCP, DNS, analizar el tráfico de routers, switchs o firewall de red. Estas opciones se ha determinado que requieren conocimiento y configuraciones más avanzadas por lo que no se adaptan al objetivo de facilidad y implantación y mantenimiento. Sin duda serían opciones más eficientes en la detección.

### Dificultades técnicas

Para la validación de un dispositivo se han barajado diferentes opciones como la validación de equipos uniéndolos a un dominio Microsoft o acceso mediante SSH utilizando claves públicas para el caso de equipos Linux. Estas opciones, aunque más seguras, añaden una complejidad de gestión al sistema que va en contra las *RNF-03: Facilidad de Despliegue* y *RNF-08: Curva de Aprendizaje Sencilla*. No se descartan estas opciones, pero se valorarán como futura mejora del sistema o para un uso avanzado de la herramienta.

Por lo anterior, se establecerá un equipo como “legítimo” o “autorizado” en base únicamente a que su mac sea conocida. Este sistema tiene dos limitaciones principales, la primera es que debido al funcionamiento básico y limitaciones de los protocolos de red (ARP) no es posible conocer directamente la MAC de un dispositivo fuera de la red local. Por otro lado, es posible que un equipo malicioso haga spoofing de mac o se la cambie a una mac autorizada.

Se van a establecer dos escenarios de funcionamiento del sistema:

En primer lugar, se establecerán los escaneos sobre equipos en la LAN (red local) en la que se encuentre la herramienta ejecutándose.

En el segundo escenario la herramienta realizará escaneos sobre otras redes, remotas, cuyos equipos solo podrán ser validados en base a la IP con la que sean detectados.

Como se ha comentado, para ambos escenarios será una mejora prevista del sistema no solo validar la Mac o la IP sino que el sistema tenga acceso al equipo remoto para validarlo de una forma segura.

### Justificación de herramientas desarrollo seleccionadas

**Uso de Python**

Se considera adecuado para el desarrollo de la herramienta el uso del lenguaje Python ya se adapta a los requisitos técnicos expuestos:

RF-05 (Integración con redes existentes): Python tiene excelentes librerías para networking como scapy, python-nmap y socket que simplifican la implementación de escaneos ARP/ICMP.

RF-08 (Soporte para múltiples subredes): La capacidad de Python para manejar operaciones de red de bajo nivel y su soporte para concurrencia (con asyncio o hilos) lo hacen ideal para escaneos distribuidos.

RNF-01 (Bajo coste): Python es open-source y tiene un ecosistema maduro de herramientas gratuitas.

RNF-03 (Facilidad de despliegue): Los scripts Python son portables y fáciles de instalar en Raspberry Pi, contenedores Docker o servidores virtuales.

RNF-04 (Compatibilidad): Python funciona en Linux, Windows y macOS, y puede interactuar con dispositivos IoT mediante sockets o APIs REST.

RNF-06 (Seguridad): Python permite implementar prácticas como logging seguro, manejo de excepciones y validación de inputs.

RNF-07 (Rendimiento): Para el escaneo periódico de redes pequeñas/medianas, Python es más que suficiente (la limitante suele ser la red, no el lenguaje).

Rápido desarrollo de prototipos: La sintaxis clara y las librerías disponibles (scapy para paquetes, requests para APIs) aceleran la implementación.

Mantenibilidad: El código Python es legible, lo que facilita las futuras ampliaciones (ej: añadir RF-09 de bloqueo automático).

Ecosistema de librerías: Existen paquetes para todo lo necesario: notificaciones (python-telegram-bot, smtplib (email)), procesamiento de datos csv para los archivos o pandas (si crece la complejidad).

**Uso de archivos de texto**

Se ha determinado que para alcance inicial se utilizarán archivos de texto para la configuración de los escaneos, inventario de dispositivos y clasificación de estos.

El uso de un sistema gestor de bases de datos como PostgreeSQL, SQLite o MariaDB, mejoraría el rendimiento y seguridad del sistema en su conjunto, a cambio de añadir complejidad a la instalación inicial, consumo de recursos y portabilidad del sistema.

### Arquitectura del sistema

En base a las motivaciones anteriormente expuestas se ha determinado una arquitectura del sistema inicialmente simple:

Se basará en varios scripts desarrollados en Python que lanzarán los escaneos de forma periódica en base a la configuración que lea de archivos de texto alojados en el propio sistema.

Se modulará el sistema de forma que sea escalable y se permita el posterior almacenamiento de la configuración en bases de datos de forma segura, aunque esto requiera mayor cantidad de recursos y conocimiento del usuario a la hora de portar la herramienta a otro sistema.

## 7.Implementación

Componentes principales del sistema

Ejemplos de código relevantes y explicados

Herramientas, frameworks y librerías empleadas

Dificultades técnicas superadas (al apartado anterior)

### Estructura de carpetas

Se muestra a continuación la estructura de carpetas y archivos que se ha determinado:

/NetScanAlert/

├── config/

├── networks.txt # Subredes a escanear

└── telegram.conf # Configuración de notificaciones

├── data/

├── devices.csv # Inventario de dispositivos (MAC,IP,Nombre,SO,Fabricante,Estado)

├── whitelist.txt # MACs autorizadas (una por línea)

├── blacklist.txt # MACs bloqueadas (una por línea)

└── history.log # Registro histórico de conexiones

├── src/

├── scanner.py # Módulo de detección

├── inventory.py # Gestión de dispositivos

├── notifier.py # Envío de alertas

└── cli.py # Interfaz de línea de comandos

└── requirements.txt # Dependencias de Python

## 8.Pruebas y validación

Tipo de pruebas realizadas (unitarias, de integración, etc.)

Resultados obtenidos

Evaluación del cumplimiento de requisitos

Pruebas con usuarios (si las hay)

## 9.Despliegue y uso

Requisitos del sistema

Instrucciones de instalación y ejecución

Manual de usuario (básico)

Consideraciones de seguridad o mantenimiento

## 10.Conclusiones y líneas futuras

Evaluación global del trabajo

Lecciones aprendidas

Mejoras o extensiones posibles

Posible impacto o aplicación del sistema

## 11.Bibliografía

Referencias empleadas, tanto técnicas como académicas

## 12.Anexos

Diagramas completos

Manual extendido (si aplica)

Capturas de pantalla

Código fuente (mejor referenciado desde un repositorio)

### Preparación del entorno de desarrollo:

El desarrollo se realizará sobre una máquina virtual **Ubuntu Desktop 24.04-2 LTS sobre VirtualBox**:

*apt update*

*apt ugrade*

*/media/david/VBox\_GAs\_7.1.4/autorun.sh (desde usuario normal y reiniciar)*

*Portapapeles bidireccional*

*Configuración de reenvío de puerto 2222 a 22*

*Carpeta compartida:*

*Ruta: H:\Datos\OneDrive\master\documentación\Modulo11 TFM\NetScanAlert*

*Nombre: NetScanAlert*

*Punto de Montaje: / (lo monta en /media/sf\_NetScanAlert*

*Instalado Visual Studio Code desde centro de aplicaciones de Ubuntu.*

*Instalar GIT: sudo apt install git-all*

*Instalar SSH: sudo apt install openssh-server*

Para el repositorio de código fuente se utiliza **GitHub**: **https://github.com/davidog7/netScanAlert**

*sudo su*

*cd /media/sf\_NetScanAlert*

*ssh-keygen -t ed25519 -C "davidog7@hotmail.com"*

*cat ~/.ssh/id\_ed25519.pub*

*Copia y pega en GitHub:* [*https://github.com/settings/ssh/new*](https://github.com/settings/ssh/new)

*git init*

*git config --global user.name "David Oliver"*

*git config --global user.email davidog7@hotmail.com*

*git config user.name # para verlo*

*git config user.mail # para verlo*

*git clone git@github.com:davidog7/netScanAlert.git*

*cd netScanAlert*

*git remote set-url origin git@github.com:davidog7/netScanAlert.git*

*git status*

*git push --set-upstream origin master*

*Lo que se hace a diario:*

***sudo su***

***cd /media/sf\_NetScanAlert/netScanAlert***

***sudo code --no-sandbox --user-data-dir=/tmp/vscode-root /media/sf\_NetScanAlert/netScanAlert***

***cerrar code***

***git add . (añade todos los archivos creados a la rama)***

***git rm nombre del archivo (borrar archivos del repositorio)***

***git commit -m “agregada funcionalidad X“ (hasta ahora todo es en local)***

***git push (para subirlo a github.com)***

***git status (para ver que está todo sincronizado)***