# Informe Técnico Avanzado: Ingeniería de Evasión de Protecciones Cloudflare en Arquitectura ARM64 (Raspberry Pi 4)

## Resumen Ejecutivo

El presente documento constituye un análisis exhaustivo y una guía técnica detallada diseñada para ingenieros de software y especialistas en automatización que enfrentan el desafío de extraer datos de infraestructuras web protegidas por Cloudflare, utilizando hardware de recursos restringidos y arquitectura específica, concretamente la Raspberry Pi 4 (ARM64).

La investigación aborda la problemática desde una perspectiva de ingeniería inversa defensiva y ofensiva. Se deconstruyen los mecanismos de detección de Cloudflare —incluyendo el análisis de huellas digitales TLS (Transport Layer Security), desafíos interactivos Turnstile y heurísticas de comportamiento del navegador— y se contrastan con las limitaciones inherentes al entorno ARM64. A diferencia de las soluciones desplegadas en servidores x86 estándar, la Raspberry Pi presenta desafíos únicos relacionados con la compatibilidad de binarios precompilados, la gestión de memoria RAM limitada y la falta de aceleración gráfica por hardware para el renderizado de desafíos Canvas/WebGL.

Este informe desestima las estrategias convencionales que han demostrado ser ineficaces en este entorno específico (como el uso estándar de Selenium WebDriver o undetected-chromedriver sin modificaciones) y propone una arquitectura de "Bypass Híbrido". Esta arquitectura integra el uso de nodriver (una implementación asíncrona sobre el protocolo Chrome DevTools) para la resolución de desafíos JavaScript y la obtención de credenciales de sesión, junto con curl_cffi para la ejecución de peticiones HTTP de alto rendimiento con suplantación de huellas TLS. Asimismo, se analizan soluciones contenerizadas mediante Docker y estrategias de ofuscación visual mediante servidores de visualización virtual (Xvfb) para mitigar la detección de navegadores "headless".

---

## 1. El Panorama de Amenazas: Cloudflare vs. Arquitectura ARM64

### 1.1 Deconstrucción de la Defensa en Profundidad de Cloudflare

Para diseñar un sistema de evasión eficaz, es imperativo comprender la sofisticación del adversario. Cloudflare no emplea un único punto de control, sino una defensa en profundidad

que evalúa la legitimidad de cada solicitud entrante basándose en una amalgama de señales técnicas y comportamentales. En el contexto de 2025, estas defensas han evolucionado más allá de la simple verificación de direcciones IP o User-Agents.

El primer anillo de seguridad es el **Análisis de la Huella Digital TLS (JA3/JA4)**. Cuando un cliente establece una conexión segura (HTTPS) con un servidor protegido por Cloudflare, se produce un intercambio inicial de paquetes conocido como "handshake". Durante este proceso, el cliente envía información sobre los cifrados que soporta, las curvas elípticas para el intercambio de claves y las extensiones TLS. Los navegadores legítimos (Chrome, Firefox, Safari) tienen patrones muy específicos y predecibles en este intercambio. Las librerías de scraping tradicionales en Python, como requests o urllib, así como herramientas automatizadas basadas en OpenSSL, presentan huellas digitales radicalmente diferentes.[1] Cloudflare identifica estas discrepancias instantáneamente. Si un cliente declara ser "Google Chrome v120" en su cabecera HTTP User-Agent pero su huella TLS corresponde a la librería OpenSSL de Python, la solicitud es bloqueada con un error 1020 o sometida a desafíos computacionales extremos antes de siquiera servir el contenido HTML.

El segundo anillo es el **Desafío Interactivo y Cognitivo (Turnstile)**. Turnstile representa un cambio de paradigma respecto a los CAPTCHAs tradicionales. En lugar de obligar al usuario a identificar semáforos o pasos de cebra, Turnstile ejecuta una serie de comprobaciones JavaScript en segundo plano para evaluar el entorno de ejecución. Busca evidencias de automatización: la presencia de objetos globales como navigator.webdriver, la consistencia entre la plataforma del sistema operativo y el navegador declarado, y la capacidad de renderizado gráfico. En la Raspberry Pi 4, esto es crítico, ya que la emulación de gráficos o la falta de una GPU dedicada pueden generar inconsistencias en las pruebas de Canvas o WebGL que Turnstile utiliza para "huellas dactilares" de hardware.

El tercer anillo es el **Análisis Comportamental y de Reputación**. Cloudflare monitorea patrones de tráfico, movimientos del ratón y tiempos de interacción. Un script que carga una página e inmediatamente intenta acceder a un endpoint de API, o que mueve el ratón en líneas perfectamente rectas (vectores no humanos), será marcado como sospechoso. Además, la reputación de la dirección IP (ASN) juega un papel fundamental; las IPs residenciales tienen mayor confianza que las IPs de centros de datos.[1]

## 1.2 Las Restricciones Críticas de la Raspberry Pi 4

Implementar soluciones de evasión en una Raspberry Pi 4 introduce una capa adicional de complejidad debido a su arquitectura de hardware.

**Arquitectura ARM64 (Aarch64):** La gran mayoría de las herramientas de "hacking" y scraping, así como los binarios de navegadores modificados, se desarrollan y compilan primariamente para arquitecturas x86_64 (Intel/AMD). Al intentar ejecutar estas herramientas en una Raspberry Pi, los ingenieros se encuentran frecuentemente con errores de "formato ejecutable incorrecto" o la imposibilidad de encontrar "wheels" (paquetes precompilados) de

Python compatibles en PyPI.[3] Esto obliga a menudo a compilar herramientas desde el código fuente, un proceso lento y propenso a errores de dependencias en un entorno Linux ARM.

**Limitaciones de Memoria y CPU:** La Raspberry Pi 4, típicamente con 4GB u 8GB de RAM, no tiene la capacidad de ejecutar múltiples instancias concurrentes de navegadores modernos basados en Chromium. Cada pestaña de Chrome con aislamiento de sitio y renderizado activo puede consumir cientos de megabytes de RAM. Las estrategias de scraping que dependen de lanzar un navegador completo para cada solicitud (como Selenium Grid tradicional) saturarán rápidamente la memoria del dispositivo, provocando que el sistema operativo mate los procesos (OOM Killer) o que el sistema se vuelva inestable debido al uso excesivo de la memoria de intercambio (swap) en la tarjeta SD, que es lenta.[5]

**Renderizado Gráfico:** Las pruebas de antibots modernas utilizan la API WebGL para dibujar gráficos invisibles y comprobar cómo la tarjeta gráfica renderiza la imagen. La GPU VideoCore VI de la Raspberry Pi tiene capacidades limitadas y drivers que difieren significativamente de las GPUs de escritorio (NVIDIA/AMD/Intel). Estas diferencias en el renderizado pueden crear una "huella digital" única que permite a Cloudflare identificar que el tráfico proviene de una Raspberry Pi, clasificándolo potencialmente como tráfico de bot o IoT no deseado.[7]

---

# 2. Autopsia de Estrategias Fallidas: Por Qué lo Convencional Falla en ARM

Es vital documentar y comprender por qué las aproximaciones estándar fracasan en este entorno específico. El usuario ha indicado que "estrategias ya testeadas no han salido bien", lo cual es consistente con la literatura técnica actual sobre scraping en ARM.

## 2.1 La Trampa de Selenium y Undetected-Chromedriver (UC)

La librería undetected-chromedriver (UC) ha sido durante años el estándar de oro para evadir la detección básica de navigator.webdriver. Funciona parcheando el binario del chromedriver para renombrar variables internas que delatan la automatización.

Sin embargo, en la Raspberry Pi (ARM64), UC presenta fallos catastróficos estructurales:

1. **Gestión de Binarios:** UC intenta descargar automáticamente el binario de chromedriver correspondiente a la versión del navegador instalado. Su lógica de detección de sistema a menudo falla en Linux ARM64, intentando descargar binarios x64 que no son ejecutables en la Pi, o no encontrando binarios compatibles para las versiones de Chromium suministradas por los repositorios de Raspberry Pi OS o Ubuntu ARM.[3]
2. **Incompatibilidad con Snap/Flatpak:** En distribuciones modernas como Ubuntu para RPi, Chromium se instala a menudo mediante Snap. UC tiene dificultades extremas para interactuar con los binarios encapsulados en Snap debido a las restricciones de

permisos y rutas de archivo no estándar.

3. **Detección Heurística Avanzada:** Cloudflare ha aprendido a identificar las firmas de memoria de los binarios parcheados por UC. Además, el uso de UC no soluciona el problema de la huella digital gráfica o de las inconsistencias del sistema operativo que son evidentes en una Raspberry Pi.
4. **Sobrecarga de Rendimiento:** Ejecutar UC implica levantar todo el stack de Selenium WebDriver. En la CPU ARM de la Pi, la comunicación JSON-Wire-Protocol añade una latencia perceptible que puede afectar a los tiempos de espera (timeouts) estrictos de los desafíos de Cloudflare, provocando fallos en la resolución de Turnstile.[8]

## 2.2 La Insuficiencia de Playwright "Vanilla" y Stealth

Playwright es superior a Selenium en velocidad y fiabilidad, pero su versión base no está diseñada para la evasión.

- **Fugas de Identidad:** Playwright, por diseño, se anuncia como un agente de automatización. Sus modificaciones al navegador (como la inyección de scripts de utilidad en cada frame) son triviales de detectar para Cloudflare.
- **Problemas de WebKit/Firefox en ARM:** Mientras que Chromium suele funcionar aceptablemente en ARM, los motores WebKit y Firefox parcheados que Playwright descarga a menudo tienen dependencias de librerías del sistema (como versiones específicas de bibliotecas multimedia o de renderizado de fuentes) que no están presentes o son incompatibles con las versiones de las distros ARM, causando fallos al iniciar el navegador.[9]
- **Plugin Stealth Desactualizado:** El popular plugin puppeteer-extra-plugin-stealth (a menudo portado a Playwright) se basa en técnicas de evasión de 2020-2022. Cloudflare ha parcheado la mayoría de estas evasiones. Por ejemplo, simplemente sobrescribir navigator.webdriver = false ya no es suficiente; los sistemas modernos verifican la integridad de la propiedad (si es configurable, si lanza errores al intentar modificarla, etc.).[1]

## 2.3 El Bloqueo de Peticiones HTTP Directas

Intentar replicar las peticiones de red utilizando librerías como requests o httpx después de inspeccionar el tráfico de red es inútil contra sitios protegidos por Cloudflare con nivel de seguridad alto.

- **Desvinculación TLS:** Aunque se copien todas las cookies (incluida cf_clearance) y cabeceras HTTP de una sesión de navegador válida, si se usan en requests, Cloudflare detectará que la huella TLS del cliente (OpenSSL) no coincide con la huella TLS esperada para el User-Agent del navegador. Esta discrepancia es una "prueba de vida" fallida inmediata.[2]

---

# 3. Preparación y Endurecimiento del Entorno (Raspberry Pi 4)

Antes de implementar el código, el sistema operativo de la Raspberry Pi debe ser optimizado para soportar la carga de trabajo de un navegador moderno y las herramientas de compilación necesarias para las librerías de evasión.

## 3.1 Selección y Configuración del Sistema Operativo

Se recomienda encarecidamente utilizar una versión de 64 bits de Raspberry Pi OS (basada en Debian Bookworm o superior) o Ubuntu Server 22.04/24.04 LTS ARM64. La arquitectura de 64 bits es obligatoria para la compatibilidad con versiones modernas de Chromium y para la gestión eficiente de direcciones de memoria grandes requeridas por los motores de JavaScript V8.

**Tabla 1: Comparativa de Sistemas Operativos para Scraping en RPi4**

| Característica | Raspberry Pi OS (32-bit) | Raspberry Pi OS (64-bit) | Ubuntu Server ARM64 |
|---|---|---|---|
| **Compatibilidad Chromium** | Alta (versiones antiguas) | Alta (versiones modernas) | Alta (vía Snap/Apt) |
| **Soporte Nodriver/UC** | Bajo (problemas de memoria) | Alto | Alto |
| **Rendimiento Python** | Medio | Alto | Alto |
| **Gestión de Memoria** | Limitada a 3GB por proceso | Acceso total a 4GB/8GB | Acceso total |
| **Recomendación** | **NO USAR** | **RECOMENDADO** | **ALTERNATIVA VIABLE** |

## 3.2 Optimización de Memoria (Swap)

El navegador Chromium es voraz con la memoria. En una RPi con 4GB, es fácil agotar la RAM durante la carga de sitios pesados. Se debe aumentar el tamaño del archivo de intercambio (swap) para evitar cierres inesperados.

**Procedimiento Técnico:**

1. Editar el archivo de configuración de swap: sudo nano /etc/dphys-swapfile
2. Modificar la variable CONF_SWAPSIZE=100 a CONF_SWAPSIZE=2048 (2GB).
3. Reiniciar el servicio: sudo /etc/init.d/dphys-swapfile restart.

## 3.3 Instalación de Dependencias Críticas

Para evadir la detección "headless", utilizaremos Xvfb (X Virtual Framebuffer), que simula una pantalla física en memoria. Esto permite ejecutar el navegador en modo "con cabeza" (headed) sin necesidad de un monitor real, lo cual es mucho menos sospechoso para Cloudflare que el flag --headless.[12]

Además, es crucial instalar el navegador Chromium del sistema, ya que nodriver en ARM no debe intentar descargar su propio binario (fallará).

Bash

```
# Actualización de repositorios
sudo apt-get update && sudo apt-get upgrade -y

# Instalación de Chromium y codecs necesarios
sudo apt-get install chromium-browser chromium-codecs-ffmpeg-extra -y

# Instalación de librerías gráficas y Xvfb para evasión visual
sudo apt-get install xvfb libatk-bridge2.0-0 libgtk-3-0 libgbm1 -y

# Instalación de herramientas de compilación para curl_cffi (si fuera necesario compilar)
sudo apt-get install build-essential libssl-dev libffi-dev python3-dev -y
```

---

# 4. Fase 1: El Motor de Interacción (Nodriver en ARM64)

La primera fase de la arquitectura propuesta utiliza nodriver para navegar al sitio objetivo, ejecutar los scripts de Cloudflare, resolver el desafío Turnstile y obtener las cookies de sesión (cf_clearance).

## 4.1 Por Qué Nodriver es la Elección Correcta

nodriver (anteriormente conocido como una evolución asíncrona de undetected_chromedriver) elimina la dependencia del binario chromedriver. Se comunica

directamente con el navegador utilizando el protocolo Chrome DevTools (CDP). Esto tiene ventajas críticas para la Raspberry Pi:

1. **Elimina el Infierno de Versiones:** No es necesario emparejar la versión de chromedriver con la de chromium-browser. Si el navegador se actualiza, nodriver sigue funcionando.
2. **Menor Overhead:** La comunicación directa por websockets es más ligera que la capa HTTP REST de WebDriver.
3. **Mejor Evasión:** Al no usar WebDriver, la variable navigator.webdriver es naturalmente false o indefinida, eliminando el vector de detección más obvio.[8]

## 4.2 Implementación del Script de Sesión

El siguiente código implementa la inicialización de nodriver específicamente configurada para ARM64, utilizando el Chromium del sistema y Xvfb.

Python

```
import asyncio
import nodriver as uc
from pyvirtualdisplay import Display
import os
import json
import random

async def get_cloudflare_session():
    # 1. Configuración de Display Virtual
    # Iniciamos Xvfb para simular un monitor de 1920x1080.
    # Esto engaña a Cloudflare, que verifica las dimensiones de la pantalla.
    # visible=0 significa que se ejecuta en background sin ventana física.
    display = Display(visible=0, size=(1920, 1080))
    display.start()

    browser = None
    try:
        # 2. Definición de Rutas y Argumentos para ARM64
        # Es crítico apuntar al ejecutable del sistema.
        # En RPi OS, suele estar en /usr/bin/chromium-browser o /usr/bin/chromium
        browser_path = "/usr/bin/chromium-browser"

        # Argumentos de lanzamiento críticos para estabilidad en RPi
        launch_args =
```

```
    # 3. Inicialización del Navegador
    # headless=False es MANDATORIO para pasar Turnstile en modos agresivos.
    # Xvfb se encarga de que no necesitemos monitor real.
    browser = await uc.start(
        browser_args=launch_args,
        browser_executable_path=browser_path,
        headless=False
    )

    # 4. Navegación Táctica
    target_url = "https://www.sitio-protegido.com"
    page = await browser.get(target_url)

    # 5. Estrategia de Espera y Resolución de Turnstile
    # Cloudflare analiza el comportamiento durante la carga.
    # Un sleep estático es malo; usar esperas aleatorias y movimientos es mejor.
    print("Analizando desafío Cloudflare...")

    # Simulación de tiempo de lectura/reacción humano
    await asyncio.sleep(random.uniform(5.0, 8.0))

    # Detección del iframe de Turnstile
    # Buscamos el widget y, si es necesario, interactuamos.
    # Nota: Nodriver permite búsqueda inteligente de elementos.
    try:
        # Buscar el contenedor del desafío (el selector puede variar según la implementación del sitio)
        turnstile_iframe = await page.select("iframe[src*='challenges.cloudflare.com']")
        if turnstile_iframe:
            print("Desafío Turnstile detectado.")
            # Mover el ratón aleatoriamente antes de interactuar
            # Nodriver no tiene mouse nativo complejo, pero podemos inyectar JS o usar primitivas
            # Si el desafío es 'non-interactive', solo esperar suele bastar.
            # Si es 'interactive', se requiere clic.

            # Técnica avanzada: Inyección de JS para clic confiable
            # A veces el clic nativo falla si el elemento está oculto o es 0x0
            # Pero Turnstile verifica las coordenadas del evento.
            # Lo ideal es esperar a que el estado cambie solo.

            await asyncio.sleep(random.uniform(3.0, 5.0))
    except Exception as e:
        print("No se detectó iframe explícito o ya se pasó.")
```

```
        # 6. Extracción de Credenciales
        # Una vez superado el desafío, extraemos las cookies y el UA.
        cookies = await browser.cookies.get_all()
        user_agent = await page.evaluate("navigator.userAgent")

        # Filtrar la cookie cf_clearance, que es el "pase VIP"
        cf_cookie = next((c for c in cookies if c.name == "cf_clearance"), None)

        if cf_cookie:
            print(f"ÉXITO: Cookie cf_clearance obtenida: {cf_cookie.value[:10]}...")
            session_data = {
                "cookies": {c.name: c.value for c in cookies},
                "user_agent": user_agent
            }
            # Guardar en disco para la Fase 2
            with open("session.json", "w") as f:
                json.dump(session_data, f)
        else:
            print("FALLO: No se obtuvo cf_clearance. Posible bloqueo o desafío no resuelto.")

    except Exception as e:
        print(f"Error crítico en Nodriver: {e}")
        # Importante: Volcar el HTML o tomar screenshot para depuración
        # await page.save_screenshot("debug_error.png")

    finally:
        # Limpieza rigurosa para liberar RAM en la RPi
        if browser:
            browser.stop()
        display.stop()

if __name__ == "__main__":
    uc.loop().run_until_complete(get_cloudflare_session())
```

## 4.3 Detalles Técnicos de la Implementación

- **Xvfb Integration:** La clase Display de pyvirtualdisplay es el puente con Xvfb. Al inicializar size=(1920, 1080), el navegador cree que está en un monitor FullHD. Esto es vital porque Cloudflare penaliza resoluciones "extrañas" típicas de bots (como 800x600 o 0x0 en headless puro).[12]
- **Ruta del Ejecutable:** La línea browser_executable_path="/usr/bin/chromium-browser" evita que nodriver intente descargar una versión x64 de Chrome, lo cual rompería el

script en la Raspberry Pi.[16]

- **Argumentos de Lanzamiento:** --disable-dev-shm-usage es esencial en entornos Docker o dispositivos con memoria compartida limitada como la RPi, ya que Chrome intenta usar /dev/shm para compartir memoria entre procesos. Si este espacio es pequeño, el navegador crashea.[5]

---

# 5. Fase 2: El Motor de Red (curl_cffi en ARM64)

Una vez obtenida la sesión, usar nodriver para scrapear miles de páginas es ineficiente. La Fase 2 utiliza curl_cffi, una librería Python que envuelve curl-impersonate, permitiendo realizar peticiones HTTP de alto rendimiento que imitan la huella TLS de un navegador real.

## 5.1 El Desafío de la Instalación en ARM64

Hasta 2023, curl_cffi no tenía soporte oficial precompilado para ARM64. Sin embargo, las versiones recientes (0.5.10 en adelante) han mejorado esto. Aún así, en la Raspberry Pi pueden surgir problemas de dependencias de libcurl o OpenSSL.[19]

**Procedimiento de Instalación Robusto:**

1. Intentar instalación directa: pip install curl_cffi --upgrade
2. Si falla, es probable que falten librerías de desarrollo. Instalar:
   Bash
   sudo apt-get install libcurl4-openssl-dev libssl-dev

3. Si persiste el error de "no matching distribution", se debe compilar desde el código fuente o buscar "wheels" específicos en repositorios como piwheels (aunque curl_cffi es complejo de compilar allí). La alternativa segura es usar la versión Dockerizada si la instalación nativa es imposible.

## 5.2 Sincronización de Huellas TLS

Para que el bypass funcione, la petición de curl_cffi debe ser indistinguible de la que haría el navegador usado en la Fase 1.

- **Consistencia de Versión:** Si nodriver usó Chromium 120, curl_cffi debe configurarse con impersonate="chrome120". Una discrepancia aquí (ej. User-Agent dice Chrome 120 pero la huella TLS es de Chrome 100) es una señal de alerta para Cloudflare.[21]
- **Orden de Cabeceras:** curl_cffi maneja automáticamente el orden de las cabeceras pseudo-HTTP (:authority, :method, etc.) para que coincidan con el navegador imitado. No se debe intentar forzar un orden manual con diccionarios de Python estándar, ya que estos no garantizan el orden en la transmisión HTTP/2.

## 5.3 Script de Extracción de Datos (Consumo de Sesión)

Python

```python
from curl_cffi import requests
import json
import time

def scrape_protected_content():
    # 1. Cargar Sesión Persistida
    try:
        with open("session.json", "r") as f:
            session_data = json.load(f)
    except FileNotFoundError:
        print("No hay sesión guardada. Ejecutar Fase 1 primero.")
        return

    cookies = session_data["cookies"]
    user_agent = session_data["user_agent"]

    # Extraer la versión mayor del UA para ajustar el impersonate
    # Ejemplo: "Mozilla/5.0... Chrome/118.0.0.0..." -> 118
    # Esto es una simplificación; en producción usar regex robusta.
    try:
        chrome_version = user_agent.split("Chrome/").split(".")
        impersonate_target = f"chrome{chrome_version}"
    except:
        impersonate_target = "chrome120" # Fallback seguro

    target_url = "https://www.sitio-protegido.com/api/data"

    # 2. Configuración de Petición Stealth
    # Usamos requests.Session() de curl_cffi para mantener conexiones vivas
    s = requests.Session()

    # Configurar cookies
    s.cookies.update(cookies)

    # Configurar headers. IMPORTANTE: No sobrescribir headers que curl_cffi
    # gestiona internamente para la suplantación, salvo los necesarios.
    headers = {
        "User-Agent": user_agent,
```

```
        "Accept":
"text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8",
        "Accept-Language": "en-US,en;q=0.9",
        "Referer": "https://www.sitio-protegido.com/",
        # Sec-Ch-Ua headers son vitales hoy en día
        "Sec-Ch-Ua": f'"Chromium";v="{chrome_version}", "Google Chrome";v="{chrome_version}",
"Not-A.Brand";v="99"',
        "Sec-Ch-Ua-Mobile": "?0",
        "Sec-Ch-Ua-Platform": '"Linux"'
    }

    try:
        # 3. Ejecución de la Petición
        # impersonate garantiza que el handshake TLS coincida con el navegador objetivo
        response = s.get(
            target_url,
            headers=headers,
            impersonate=impersonate_target,
            timeout=15
        )

        # 4. Validación de Respuesta
        if response.status_code == 200:
            print("Datos extraídos correctamente.")
            # Procesar JSON o HTML
            print(response.text[:500])
        elif response.status_code == 403 or response.status_code == 1020:
            print("Acceso Denegado. La cookie puede haber expirado o la IP está quemada.")
            # Aquí se dispararía la lógica para re-ejecutar la Fase 1
        elif response.status_code == 429:
            print("Rate Limit. Pausar y reintentar.")

    except Exception as e:
        print(f"Error de red: {e}")

if __name__ == "__main__":
    scrape_protected_content()
```

---

# 6. Alternativa Contenerizada: FlareSolverr en Docker

Si la implementación manual de nodriver resulta inestable debido a dependencias del sistema,

**FlareSolverr** ofrece una solución encapsulada. FlareSolverr actúa como un servidor proxy: recibe una petición JSON, lanza un navegador interno, resuelve el desafío y devuelve el HTML y las cookies.

## 6.1 Desafíos de Docker en Raspberry Pi

FlareSolverr es pesado. En una RPi4, el contenedor puede tardar mucho en iniciar el navegador, provocando timeouts. Además, la arquitectura de la imagen debe ser compatible con ARM64.[23]

## 6.2 Configuración Optimizada para ARM64

Se debe usar un archivo docker-compose.yml ajustado específicamente para los límites de la RPi. Es crítico aumentar el BROWSER_TIMEOUT y gestionar los límites de memoria para evitar colapsar el host.

**Configuración Recomendada (docker-compose.yml):**

YAML

```yaml
services:
  flaresolverr:
    # Usar la imagen oficial que soporta multi-arch (incluyendo arm64)
    image: ghcr.io/flaresolverr/flaresolverr:latest
    container_name: flaresolverr
    environment:
      - LOG_LEVEL=info
      - LOG_HTML=false
      - CAPTCHA_SOLVER=none
      - TZ=Europe/Madrid
      # Timeout crítico para RPi4: Aumentar a 2 minutos para dar tiempo a la CPU lenta
      - BROWSER_TIMEOUT=120000
    ports:
      - "8191:8191"
    restart: unless-stopped
    # Límites de recursos para proteger el sistema operativo de la RPi
    deploy:
      resources:
        limits:
          cpus: '1.50'
          memory: 1024M
```

```
  # Deshabilitar comprobaciones de salud pesadas si consumen mucha CPU
  healthcheck:
    disable: true
```

**Nota de Implementación:** Al usar FlareSolverr, el script de Python simplemente envía una petición POST a http://localhost:8191/v1/request con la URL objetivo. FlareSolverr devuelve la respuesta ya "limpia". Esta es la opción más sencilla pero menos flexible que la combinación nodriver + curl_cffi.[24]

---

# 7. Estrategias Avanzadas de Contra-Evasión y Huellas Digitales

El mero uso de herramientas sofisticadas no garantiza el éxito si se ignoran las heurísticas avanzadas.

## 7.1 Consistencia Gráfica (Canvas/WebGL)

Cloudflare y herramientas como FingerprintJS generan una imagen oculta usando la API Canvas. En una Raspberry Pi sin monitor, el renderizado por software (SwiftShader) produce una imagen diferente a la de una tarjeta gráfica NVIDIA/Intel típica.

- **Mitigación:** No hay una solución perfecta en RPi sin GPU externa. Sin embargo, asegurar que nodriver lance Chromium con --use-gl=swiftshader o --use-gl=egl garantiza al menos que el navegador sea capaz de renderizar algo, evitando el fallo total de la prueba de Canvas que delataría inmediatamente al bot.[5]

## 7.2 Falsificación de AudioContext

Similar al video, la API de AudioContext se usa para huellas digitales. En entornos Linux sin tarjeta de sonido configurada (como una RPi server), esta API puede devolver valores nulos o errores.

- **Mitigación:** Asegurarse de que las librerías de sonido (libasound2) estén instaladas en el sistema host, incluso si no hay altavoces, para que la API del navegador devuelva datos de "silencio" válidos en lugar de una excepción de error.

## 7.3 Rotación de Proxies y Gestión de Reputación

La dirección IP de la Raspberry Pi es el punto más débil. Si se realizan demasiadas peticiones fallidas (errores 403), la IP entrará en una lista negra temporal o permanente.

- **Estrategia:** Implementar un middleware en curl_cffi que rote proxies residenciales. No usar proxies de datacenter. Configurar la lógica de reintento para que, tras 3 fallos consecutivos, la RPi entre en modo "dormido" por 15-30 minutos para enfriar la

reputación de la huella digital.

---

# 8. Estrategia Operativa y Mantenimiento

Para mantener este sistema operativo a largo plazo, se debe adoptar una mentalidad de "gato y ratón".

1. **Monitorización de Versiones:** Cloudflare actualiza sus desafíos semanalmente. nodriver y curl_cffi se actualizan frecuentemente para contrarrestar esto. Automatizar la actualización de estas librerías con un cron job semanal (pip install --upgrade...) es vital.
2. **Gestión de Excepciones:** El código debe estar preparado para detectar no solo códigos de estado HTTP, sino también el contenido de la respuesta. A veces Cloudflare devuelve un código 200 OK pero el cuerpo es la página del desafío ("Just a moment..."). El script debe buscar palabras clave ("Verify you are human", "Attention Required") en el HTML para detectar falsos positivos de éxito.
3. **Ciclo de Vida del Token:** La cookie cf_clearance tiene una vida útil corta (desde 30 minutos hasta 2 horas, dependiendo de la configuración del sitio). El sistema debe ser capaz de detectar la expiración del token y disparar automáticamente la Fase 1 (Nodriver) para renovarlo, sin intervención humana.

---

# Conclusión

La implementación de un scraper capaz de evadir Cloudflare en una Raspberry Pi 4 es un ejercicio de ingeniería de precisión. Requiere abandonar las herramientas monolíticas de "fuerza bruta" como Selenium en favor de herramientas quirúrgicas como nodriver y curl_cffi. La clave del éxito reside en la **simulación perfecta del entorno**: desde el uso de Xvfb para proveer un contexto visual, hasta la alineación meticulosa de las huellas TLS en la capa de red.

Si bien la arquitectura ARM64 impone restricciones severas, también ofrece una plataforma de bajo consumo ideal para operaciones de recolección de datos persistentes, siempre y cuando se respete la jerarquía de recursos y se implementen las estrategias de evasión híbridas detalladas en este informe.

1.