Análisis de Seguridad
O'Reilly Auto Parts

Protección: Akamai Bot Manager v2 | Target: oreillyauto.com

Fecha: Febrero 2026 | Tipo: Pentesting Autorizado

1. Resumen Ejecutivo

Se realizó un análisis exhaustivo de las protecciones anti-bot implementadas en `oreillyauto.com`. El sistema utiliza Akamai Bot Manager v2, una de las soluciones anti-bot más sofisticadas del mercado.

Enfoque	Resultado	Tasa de Éxito
Bypass con navegador real (Playwright/Puppeteer)	VIABLE	~95%
Bypass sin navegador (Python/Node.js puro)	NO VIABLE	N/A

2. Arquitectura de Protección Identificada

2.1 Capas de Defensa

Se identificaron tres capas principales de protección que trabajan en conjunto:

Capa	Componentes
Capa 1: Edge (CDN)	TLS Fingerprinting (JA3/JA4), HTTP/2 Fingerprinting (AKAMAI_H2), Rate Limiting por IP, Geolocalización
Capa 2: JS Challenge	Script principal: `IzBCtKGXYB.js` (~512 KB ofuscado), Script secundario: `cpPG0UHDtu.js`, Objeto global: `window.bmak`, 16+ funciones de fingerprinting, Telemetría de interacción humana
Capa 3: Backend	Machine Learning para análisis comportamental, Correlación de fingerprints con BD, Análisis estadístico de te

2.2 Endpoints Identificados

Endpoint	Propósito
`/Cfpd0nbIV/.../IzBCtKGXYB`	Sensor script principal (POST sensor_data)

Endpoint	Propósito
`/akam/13/pixel_{{ID}}`	Pixel de telemetría (POST fingerprint data)
`/akam/13/{{ID}}`	Script secundario de fingerprinting

3. Sistema de Cookies

3.1 Cookie _abck (Principal)

Formato: `{{HASH_32_HEX}}~{{STATUS}}~{{TIMESTAMP}}~{{DATA}}~{{SIGNATURE}}`

Status	Significado	Acceso
0	Validado	Permitido
-1	No validado	Bloqueado/Challenge
-2	Sospechoso	Rate limited

3.2 Cookie bm_sz

Session binding y verificación de integridad.

Formato: `{{HASH}}~{{YYMMDDHHMMSS}}~{{BROWSER_ID}}~{{DATA}}`

3.3 Otras Cookies

Cookie	Propósito
ak_bmsc	Session identifier
bm_sv	Visit tracking
bm_mi	Machine ID
RT	Response time

4. Análisis del JavaScript de Akamai

4.1 Script Principal: IzBCtKGXYB.js

Característica	Valor
Tamaño	512.6 KB
Líneas	~15,000+ (una sola línea ofuscada)
Ofuscación	Control flow flattening + string encoding + dead code injection

Técnicas de ofuscación identificadas:

1. Control Flow Flattening: Uso de switch-case con variables dinámicas que determinan el flujo de ejecución en runtime, haciendo imposible el análisis estático.
2. String Encoding: Strings almacenados en arrays y decodificados en runtime mediante funciones wrapper.
3. Function Wrapping: Funciones envueltas en capas de indirección usando apply() con argumentos dinámicos.
4. Dead Code Injection: ~40% del código son funciones que nunca se ejecutan.

4.2 Desobfuscación Parcial Lograda

Se logró desobfuscar parcialmente el script e identificar las categorías de detección:

Categoría	Código	Descripción
Navigator	nav	Propiedades del navegador
Device	dp	Propiedades del dispositivo
Screen	sr	Dimensiones de pantalla
Canvas	cv	Canvas fingerprint (SHA-1)
Browser	br	Tipo de navegador
Bot Detection	z	Detección de bots (CRÍTICO)
Battery	bt	Battery API
Chrome Check	crc	Verificación window.chrome
Token	t	SHA-1 token de sesión
Session Hash	u	Hash de sesión

4.3 Razones por las que es Imposible Desobfuscar Completamente

Dead Code Injection: ~40% del código son funciones que nunca se ejecutan, complicando el análisis.

Dynamic Function Generation: Funciones creadas en runtime con eval.
Anti-Tampering: Checksums internos que detectan modificaciones al script.
Versión-Específico: El script cambia cada ~24-48 horas.

5. Sistema de Fingerprinting

5.1 Categorías de Fingerprint Recolectadas

A) Navigator Properties (nav)

Incluye: `userAgent`, `appName`, `hardwareConcurrency`, `platform`, `language`, `plugins`, `webdriver` (crítico para detección de Selenium), `maxTouchPoints`.

B) Device Properties (dp)

Verifica APIs disponibles: `indexedDB`, `addEventListener`, `MutationObserver`, `Int8Array`, `compatMode`.

C) Screen/Window (sr)

Recopila: `innerWidth/Height`, `outerWidth/Height`, `screen.availWidth/Height`, `colorDepth`, `pixelDepth`.

D) Canvas Fingerprint (cv)

Genera un hash SHA-1 de un canvas renderizado con operaciones específicas de dibujo (texto, rectángulos, gradientes). Cada entorno produce un hash único.

E) Battery API (bt)

Recopila estado de batería: `charging`, `chargingTime`, `dischargingTime`, `level`.

F) Chrome Object Check (crc)

Verifica la existencia y estructura de `window.chrome` incluyendo `app.isInstalled`, `InstallState`, `RunningState`.

G) Bot Detection (z) — CRÍTICO

Busca propiedades conocidas de herramientas de automatización:

Propiedad	Herramienta Detectada
$cdc_	ChromeDriver
$wdc_	WebDriver
__webdriver_unwrapped	Selenium
__driver_evaluate	WebDriver
__webdriver_evaluate	Driver
__selenium_evaluate	Selenium
__fxdriver_unwrapped	Selenium IDE Recorder
__driver_unwrapped	Selenium
_Phantom / callPhantom	PhantomJS

5.2 Timing Profiles

El sistema mide tiempos de ejecución de cada categoría de fingerprinting. Tiempos artificiales (distribución uniforme) son detectados como bot.

Operación	Tiempo (ms)	Notas
Canvas (cv)	39	Más variable — importante para detección
Bot Detection z1	10	Fase principal
Bot Detection z2	2	Verificación
Bot Detection z3	1	Check final
Total main	1317	Tiempo total de ejecución
Compute	54	Tiempo de cálculo
Network send	658	Envío de telemetría

6. Binding Criptográfico de Sesión

6.1 Variable bazadebezolkohpepadr

Se inyecta en el HTML como: `<script>window.bazadebezolkohpepadr = "590870621";</script>`

Valor único por sesión (cambia cada request). Se usa para generar tokens criptográficos que vinculan el sensor_data a la sesión específica.

6.2 Token t (SHA-1)

Fórmula: `t = SHA1(bazadebezolkohpepadr.toString())`

Ejemplo: bazadebezolkohpepadr = 590870621 → t = `8b34e3434766c476bc6c80f5f7d0e6372dbed5a4`

6.3 Token u (Session Hash)

Valor fijo por dominio/configuración: `u = "38b42109eb1e5fd8bb615624556e1098"` (MD5)

6.4 URL del Pixel

Fórmula: `pixel_id = (833 XOR bazadebezolkohpepadr).toString(16)`

URL: `/akam/13/pixel_ + pixel_id`

7. Estructura del sensor_data

7.1 Formato Bot Manager v2

Formato: `VERSION;FLAG1;FLAG2;FLAG3;BM_SZ;HASH;COUNTERS;ENCODED_PAYLOAD`

Campo	Pos.	Descripción
VERSION	0	Versión del protocolo (3)
FLAG1	1	Estado inicialización (0)
FLAG2	2	Tipo de eventos (1=mouse, 2=keyboard)
FLAG3	3	Reservado (0)
BM_SZ	4	Valor de cookie bm_sz
HASH	5	Base64(SHA-256(fingerprint_data))
COUNTERS	6	mouse,clicks,keydown,keyup,touch,scroll
PAYLOAD	7+	Telemetría codificada

7.2 Ejemplo Real Capturado

```
3;0;1;0;3294261;1QzFe4zz+mlT0a7XlxiEHFqz5JZVysvW1wbM/oUtLQ0=;58,6,0,0,16,0;[ENCODED]
```

7.3 Diferencias: Real vs Generado

Métrica	Real (Chrome)	Generado (JSDOM)
Longitud	1613-4936 chars	4430 chars
Flags	0;1;0	1;2;0
Counters	58,6,0,0,16,0	44,7,0,1,6,0
Frecuencia char "	264	358
Entropía	4.2 bits	3.8 bits

8. Formato del Pixel Data (Fingerprint POST)

8.1 Estructura URL-Encoded

El pixel POST envía todos los datos de fingerprinting en formato URL-encoded:

Parámetro	Descripción	Ejemplo
ap	WebM support	true
bt	Battery API (JSON)	{charging: true, level: 1}
cv	Canvas fingerprint (SHA-1)	554a70f0f8b2...
z	Bot check (JSON)	{a: 590870621, b: 1, c: 1}
t	SHA-1 token	8b34e343...
u	Session hash (MD5)	38b42109...
nav	Navigator (JSON)	{userAgent: "...", ...}
sr	Screen dims (JSON)	{inner: [1280,720], ...}
dp	Device props (JSON)	{indexedDB: 1, ...}
nap	Permissions string	11311144241322244122
br	Browser type	Chrome
timing	Timing profile (JSON)	{main: 1317, compute: 54}

9. Scripts Adicionales Encontrados

Script	Propósito	Tamaño
zBCpKGXYB.js	Akamai Bot Manager principal	512 KB
cpPG0UHDtu.js	Akamai secundario	~100 KB
ruxitagentjs_*.js	Dynatrace RUM (NO anti-bot)	~150 KB
gtm.js	Google Tag Manager	~80 KB
fbevents.js	Facebook Pixel	~50 KB
chunk-vendors.*.js	Vue.js dependencies	~800 KB
bv.js	Bazaarvoice (reviews)	~200 KB

10. Intentos de Bypass Realizados

10.1 Enfoque A: Simulación Pura (Python + Node.js) — FALLIDO

Ejecutar `IzBCtKGXYB.js` en JSDOM, simular eventos mouse/keyboard, generar sensor_data con `bmak.get_telemetry()`, POST al endpoint.

Causas del fallo:

- Canvas fingerprint difiere (JSDOM genera hash diferente al navegador real).
- Timing patterns artificiales (distribución uniforme vs log-normal esperada).
- WebGL no implementado en JSDOM.
- Audio fingerprint no disponible.
- Detección de propiedades faltantes en window.

10.2 Enfoque B: Replay de Capturas — FALLIDO

Capturar sensor_data de navegador real y reenviar con diferentes timestamps.

Causa: Session binding criptográfico. Cada sensor_data está vinculado a: `bazadebezolkohpepadr` específico, cookie `_abck` inicial, y timestamp de generación.

10.3 Enfoque C: Hybrid — Navegador Real — VIABLE (~95%)

Usar Playwright con `playwright-stealth` para que el navegador real maneje todo el challenge de Akamai de forma transparente.

La cookie `_abck` se valida automáticamente al ejecutar el JS en un entorno real de navegador.

11. Vulnerabilidades Identificadas

11.1 API de Precios Sin Protección Completa

El endpoint `/product/pricing-availability/v2` acepta hasta 300 productos por request con menos validación que las páginas de producto normales.

Requisitos: Cookie de sesión válida, Token CSRF, Headers correctos.

11.2 Bazaarvoice API Externa

API de reviews con key hardcodeada, sin protección de Akamai. Endpoint accesible directamente.

12. Archivos del Proyecto

12.1 Scripts de Análisis

Archivo	Propósito
akamai_bypass.py	Módulo principal de bypass
sensor_data_generator.py	Generador de sensor_data con JSON
cookie_fixer.py	Corrección de cookies
analyze_sensor_data.py	Análisis de sensor_data
capture_traffic.py	Captura de tráfico
akamai_deobfuscator.py	Intento de desofuscación

12.2 Capturas Realizadas

Se realizaron múltiples capturas de tráfico con sus correspondientes POST data y 90+ archivos JavaScript extraídos.

13. Conclusiones Técnicas

¿Por qué el bypass sin navegador es impracticable?

#	Factor	Detalle
1	Complejidad del Script	512KB de código ofuscado con anti-tampering
2	Fingerprinting Multi-Capa	16+ técnicas que requieren APIs nativas del navegador
3	Session Binding Criptográfico	Cada request vinculado a la sesión
4	ML Backend	Análisis comportamental detecta patrones artificiales
5	Actualizaciones Frecuentes	El script cambia cada 24-48 horas

Recomendación

Utilizar Playwright + playwright-stealth en modo headless para el scraping. Combinar con el endpoint de pricing ({code('/product/pricing-availability/v2')}) que acepta 300 productos por request para maximizar eficiencia y minimizar requests.