TAREA INVESTIGACIÓN

Hugo García

Índice

Sistemas anti-cheat a nivel de kernel: funcionamiento técnico y riesgos

Qué significa anti-cheat a nivel de kernel

Anti-cheat a nivel de kernel significa que el sistema anti-cheat de un juego o aplicación opera dentro del núcleo del sistema operativo, es decir, con más privilegios del software de tu ordenador.

Técnicas comunes que usan

Controladores (Drivers) de Kernel

Instalan un driver que se ejecuta con el máximo privilegio. Desde ahí pueden tanto ver como controlar operaciones que los cheats podrían usar para ocultarse.

Escaneo de drivers cargados

Detectar si hay drivers maliciosos, no firmados, con nombres sospechosos, o con comportamiento extraño. Esto sirve para encontrar cheats que operan también en kernel.

Verificación de integridad de memoria / ejecución de código

- Verificar que las secciones de código del juego o de drivers oficiales no han sido modificadas.
- Detectar inyección de código, trampas que parchean en memoria.
- Hashes o firmas, detectar trampas que hagan “manual mapping” de módulos o manipulen memoria sin pasar por las rutas normales.

Monitoreo de creación de procesos / hilos / callbacks del sistema

- Vigilar cuando un nuevo proceso o hilo se crea, para ver si está injectando código malicioso.
- Usar “object callbacks” del kernel para saber cuando se crean handles, se cargan imagenes (EXE, DLL) o drivers.

Hooking de llamadas del sistema / syscalls

- Interceptar llamadas que podrían usarse para trampas (lectura/escritura de memoria, mapeos de memoria, acceso a drivers, etc.).
- A veces se hacen hooks en las tablas de sistema (SSDT, IDT) para monitorear operaciones.
  Estos hooks permiten detectar comportamientos anómalos antes de que el cheat lo ejecute libremente.

Uso de Secure Boot / certificación firmware / arranque seguro

Asegurarse de que el entorno del sistema esté íntegro desde el arranque, para

dificultar que se carguen drivers maliciosos al arranque del sistema que podrían evadir detección.

Protección continua / actualización dinámica
Los cheats evolucionan: los anti-cheat necesitan actualizar sus firmas, heurísticas, reglas de detección, modelos de comportamiento. También pueden usar telemetría para detectar patrones inusuales.

Análisis heurístico / detección de comportamiento anómalo
No solo buscar firmas explícitas, sino observar cómo se comportan los programas: por ejemplo, accesos a memoria que no deberían, lecturas escritas extrañas, latencias inusuales, etc..

Bloqueo de APIs / interfaces que puedan ser usadas por cheats
Por ejemplo, prevenir que ciertos mecanismos de depuración, lectura de memoria de otros procesos, escritura en драйvers externos, etc., sean usados sin autorización. También se pueden impedir que herramientas de depuración funcionen mientras el juego está activo.

Restricciones de hardware

- Verificar que el hardware (firmware, BIOS/UEFI) no esté comprometido.
- Usar Trusted Platform Module (TPM) o características similares para asegurar que ciertos componentes, como arranque seguro, estén habilitados.
- Bloquear cheats que usan acceso DMA externo, si se pueden detectar.

Ejemplos concretos

BattlEye

- Tiene un driver kernel llamado *BEDaisy.sys*.
- Usa *callbacks* y filtros (minifilters) para detectar drivers maliciosos, módulos cargados, creación de procesos etc
- Detecta hooking de funciones del kernel, módulos ocultos, manipulaciones

Riot Vanguard

- Se carga al arranque del sistema, incluso si el juego no está abierto; su driver se ejecuta en el kernel.
- Rastrea/controla drivers que estén cargados, bloquea aquellos que considera peligrosos, opera en “ring 0”.

Easy Anti-Cheat (EAC)

- Tiene componentes de kernel para integridad de drivers, detección de modificaciones en memoria, uso de identificadores de hardware (HWID).
- Se mezcla modo usuario + kernel para balancear detección vs compatibilida

Beneficios

Detección más profunda

Prevención ante el inicio de cheats

Integridad del juego mejorada

Monitoreo en tiempo real

Cobertura frente a cheats avanzados

Mejor experiencia de juego

Reducción del coste de mantenimiento

Riesgos y costes

Vulnerabilidades de seguridad graves

Privacidad perjudicada

Inestabilidad del sistema

Falsos positivos y problemas para el usuario general

Coste de explotación si un tercero compromete el anti-cheat

Practicas recomendadas

- Que el driver/kernel solo haga lo mínimo necesario: limitar su ámbito, funcionalidades, recursos.
- Evitar dar permisos que no sean estrictamente imprescindibles.
- Rechazar drivers no firmados o con certificados comprometidos.
- Mantener una lista negra (blacklist) de drivers vulnerables conocidos.
- Usar mecanismos de arranque seguro para asegurar que sólo software firmado/caracterizado pueda cargar temprano.
- Usar componentes de confianza de hardware, como TPM o enclaves seguros, si el SO lo permite.
- Revisar la memoria, los módulos cargados, los drivers, en tiempo real o de forma muy frecuente, no solo al iniciar el juego.

- Verificar integridad de los archivos en disco y del código cargado, hashes, firmas.
- No depender únicamente de firmas; usar heurísticas para detectar comportamientos inusuales (e.g. accesos a memoria extraños, patrones de input anómalos).
- Analítica post-partida para detectar trampas que quizás no se captaron en tiempo real.
- Transparencia: informar a los usuarios qué datos se recopilan, cómo, cuándo, y para qué se usan.
- Limitar recolección de datos solo a lo estrictamente necesario. Evitar acceso indiscriminado al sistema.
- Permitir auditoría / revisión del código de seguridad cuando sea posible (o auditoría externa).
- Programación defensiva: manejo adecuado de errores, validaciones, evitar condiciones de carrera.
- Buenas prácticas de desarrollo seguro: revisión de código, pruebas de fuzzing, testing en diferentes configuraciones de hardware/software.
- Actualización continua: parchear vulnerabilidades tan pronto se descubran.
- Probar en configuraciones variadas (diferentes versiones de SO, hardware, drivers de terceros) para evitar conflictos, cuelgues, BSOD.
- Evitar bloquear funcionalidad legítima ni causar comportamiento inesperado al usuario final.
- No confiar únicamente en el anti-cheat en kernel: tener también medidas en espacio de usuario, validaciones del servidor, chequeos de integridad, lógica del juego segura.
- Implementar servidor autoritativo para partes críticas del juego (daño, físicas, etc.).
- Permitir desinstalar o desactivar el componente kernel de forma clara y segura, respetando políticas del SO.
- Que las actualizaciones o cambios críticos requieran consentimiento cuando corresponda.
- Cumplir leyes de protección de datos (por ejemplo GDPR en Europa) si recopilas datos de los usuarios.
- Revisar impacto en privacidad y realizar análisis de riesgo.

- Mantener registros adecuados (logs) para transparencia y en caso de disputas.

Informe técnico: fallo de CrowdStrike (19 julio 2024)

Línea temporal

- **04:09 UTC, 19 julio 2024** — CrowdStrike desplegó una actualización de contenido (Channel File 291) que contenía nuevas Template Instances relacionadas con visibilidad de IPC (named pipes). Muchos sensores en Windows recibieron esa actualización y comenzaron a evaluar las nuevas reglas.
- **Minutos posteriores** — Al evaluar eventos IPC, el Content Interpreter intentó acceder a un campo (21º) que no existía en el array proporcionado por el sensor (solo había 20), lo que produjo un acceso fuera de límites y fallo en modo kernel (invalid page fault / crash). Esto produjo BSOD/bootloops en sistemas con el sensor activo.
- **En paralelo** — Los incidentes se propagaron rápidamente a infraestructuras cloud (VMs en Azure y Google Compute Engine) y a clientes empresariales que usaban Falcon, causando reinicios masivos y amplios fallos operativos. CISA emitió alerta.
- **Remediación inicial** — CrowdStrike y Microsoft coordinaron mitigaciones; CrowdStrike publicó procedimientos para eliminar manualmente el .sys problemático o arrancar en Modo Seguro/WinRE para limpiar el sensor. Posteriormente se desplegaron correcciones en el sensor y parches al proceso de validació

Causa

CrowdStrike lanzó una actualización de configuración para su software *Falcon Sensor* en Windows la cual contenía un error lógico. En concreto, el sensor esperaba un cierto número de campos de entrada pero recibió uno de más, provocándose una lectura de memoria fuera de los límites.

Debido a que el sensor opera en un nivel kernel con altos privilegios, ese error causó que muchos sistemas Windows se bloquearan (pantalla azul, BSOD), entrasen en bucle de arranque, o entraran en modo recuperación.

Impacto

El fallo global de CrowdStrike en julio de 2024 tuvo un impacto significativo a nivel mundial, afectando a aproximadamente 8,5 millones de dispositivos con Windows. Aunque esto representó menos del 1% de todos los sistemas Windows, la magnitud de la interrupción fue histórica debido a la criticidad de los servicios afectados.

El incidente interrumpió operaciones en sectores como aviación, salud, finanzas, administración pública y telecomunicaciones.

Departamentos como el de Justicia, Energía, Educación y Defensa de EE. UU., así como agencias estatales de vehículos motorizados, experimentaron interrupciones. Incluso se declararon emergencias en ciudades como Portland, Oregón.

Aeropuertos, sistemas de control de tráfico aéreo y plataformas en la nube como Microsoft Azure y Google Compute Engine reportaron fallos masivos.

Soluciones aplicadas por CrowdStrike

A las 05:27 UTC del 19 de julio, CrowdStrike revirtió la actualización problemática, Channel File 291, que había causado el fallo en los sistemas Windows.

En lugar de realizar despliegues globales simultáneos, CrowdStrike permitió a los clientes elegir cuándo aplicar las actualizaciones para minimizar riesgos.

Se implementaron controles adicionales en el sistema de validación interna para evitar que errores similares se filtraran en futuras actualizaciones.