UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

Facultad de Ingeniería

Sistemas Operativos

Documentación. Aislamiento de aplicaciones y sandboxing: mecanismos de seguridad y control de ejecución en sistemas operativos modernos.

Profesor: Dr. Gunnar Eyal Wolf Iszaevich

Grupo: 8

Integrantes:

- Alvarez Salgado Eduardo Antonio (321335630)
- Morales Castillo Arumy Lizeth (321082626)

Fecha de presentación: 18 de noviembre de 2025

Semestre: 2026-1

Índice de contenido.

1. Introducción

En los sistemas operativos modernos, la ejecución segura de aplicaciones es un desafío permanente debido al aumento de amenazas, vulnerabilidades y software no confiable. Para enfrentar este problema, los mecanismos de aislamiento y sandboxing se han consolidado como estrategias fundamentales, ya que permiten que cada aplicación opere dentro de un entorno controlado donde sus acciones están limitadas y supervisadas.

Es por ello que, a lo largo del documento se analizarán los fundamentos teóricos del sandboxing, las técnicas de aislamiento utilizadas por los sistemas operativos actuales, su arquitectura interna y diversos casos reales que muestran cómo plataformas como Windows, Linux, macOS, Android o los navegadores web han adoptado estos mecanismos. El propósito es ofrecer una visión clara y actualizada del papel que desempeña el sandboxing en la seguridad contemporánea.

2. Fundamentos teóricos

2.1. Historia y definición de sandboxing

El término sandboxing proviene de la metáfora de la “caja de arena”, el cual es un espacio controlado para que los niños puedan jugar sin enfrentar peligros. Dicho concepto fue adoptado por la informática para referirnos a ejecutar código potencialmente inseguro en un entorno aislado y así proteger el resto del sistema con riesgos controlados.

Sus fundamentos conceptuales aparecen con la llegada de Java en 1996, en donde se incorporó un modelo formal de sandboxing cuyo propósito era proporcionar un entorno restringido para ejecutar código no confiable obtenido de la red. La esencia del modelo de sandbox radica en que el código local tiene acceso completo a los recursos vitales del sistema (como el sistema de archivos), mientras que el código remoto descargado (un applet) no es confiable y solo puede acceder a los recursos limitados dentro del sandbox. [1]

2.2 Definición y propósito del aislamiento de aplicaciones

Una aplicación puede contener numerosas funcionalidades, dependencias, bibliotecas y posibles vulnerabilidades, es por esto que la finalidad del aislamiento de aplicaciones es contener el riesgo inherente que representa una aplicación

compleja, limitando así el impacto que pudiese llegar a tener ante cualquier riesgo que se presente. Este enfoque se basa en tres ideas estructurales: [2]

1. **Separación entre procesos y memoria:** Cada aplicación obtiene un espacio de direcciones independiente y protegido, lo que impide accesos ilegítimos a datos o estructuras internas de otros procesos.
2. **Limitación del alcance de una vulneración:** Si una aplicación es comprometida, la arquitectura del sistema debe garantizar que solo afecte a su propio entorno y no al sistema operativo o a otras aplicaciones.
3. **Supervisión del comportamiento:** El sistema monitorea continuamente las operaciones sensibles como el acceso a archivos, red, syscalls e IPC para contener cualquier acción fuera de lo esperado. [3]

2.3 Seguridad por diseño

La seguridad por diseño supone que los mecanismos de protección deben integrarse desde el inicio del desarrollo del sistema operativo y no añadirse posteriormente como medidas reactivas. En este paradigma, el aislamiento de aplicaciones y el sandboxing no son funcionalidades accesorias, sino componentes esenciales de la arquitectura del sistema.

Esta medida se utiliza tanto en el desarrollo de software como en la creación de dispositivos, servicios en la nube e incluso infraestructuras tecnológicas. Su principal idea es que la prevención siempre será más efectiva y económica que la corrección una vez el daño ya se ha producido. [4]

Este enfoque garantiza que los controles de seguridad estén presentes en cada capa del sistema operativo. Por ejemplo, tecnologías como SELinux, AppArmor, el sandbox de macOS o los permisos declarativos de Android fueron concebidas desde su diseño para restringir las acciones de cada aplicación y evitar que ejecutables no autorizados obtengan un acceso indebido a la información del sistema. Al integrar la seguridad desde el origen, se construyen sistemas más resistentes contra técnicas modernas de explotación, tales como escalamiento de privilegios, ejecución arbitraria de código o bypass de controles de acceso.

2.4. Tipos de sandbox y enfoques de aislamiento

El sandboxing no corresponde a una sola técnica, sino a una familia de mecanismos que emplean distintos enfoques según las necesidades de seguridad, rendimiento y

flexibilidad del sistema. Aunque las plataformas modernas combinan varios de ellos, es posible clasificarlos en grupos que representan las estrategias predominantes en la industria actual. Entre los tipos de sandbox se encuentran:

- **Sandbox basados en virtualización y microVMs:** ofrecen el aislamiento más fuerte mediante máquinas virtuales ligeras controladas por hipervisores.
- **Sandbox basados en contenedores:** emplean namespaces (procesos, red, etc) para aislar la vista del sistema y cgroups para limitar recursos como CPU, RAM y E/S.
- **Sandbox basados en filtrado de llamadas al sistema:** limitan las syscalls permitidas para reducir la superficie de ataque. [2]
- **Sandbox basado en la nube:** Los usuarios pueden probar aplicaciones o archivos de forma aislada y segura sin depender del hardware físico, lo que protege los equipos locales de las consecuencias del malware. [5]

Cada uno de estos enfoques ofrece niveles distintos de protección y rendimiento, y su elección depende de factores como la necesidad de confiabilidad, la importancia del rendimiento, el nivel de aislamiento requerido y el tipo de aplicaciones a ejecutar.

3. Arquitectura y funcionamiento del sandbox en los sistemas operativos

3.1 Conceptos esenciales

1. **Kernel:** Es el encargado de manejar las llamadas del sistema, que son solicitudes de programas y procesos para acceder a los recursos del sistema incluyendo el procesador, la memoria y los periféricos. [6]
2. **namespaces:** Son una abstracción del kernel que permite ofrecer a un proceso una visión personalizada y limitada de ciertos componentes del sistema, como el árbol de procesos, la pila de red, los puntos de montaje o los identificadores de usuario. Cada namespace crea una instancia “privada” de estos recursos, de modo que un proceso aislado no puede ver ni interactuar con el resto del sistema fuera de su namespace. [7]
3. **cgroups**: Los control groups o cgroups permiten regular recursos tales como CPU, memoria, ancho de banda etc, determinando cuántos de estos recursos puede consumir un conjunto de procesos, garantizando que un proceso aislado no pueda degradar ni monopolizar recursos críticos. [8]

3.2. Arquitectura general del sandbox

La arquitectura de un sandbox moderno se basa en crear un espacio seguro en el que una aplicación se ejecuta bajo restricciones estrictas. Para lograrlo, el sistema operativo establece un **entorno aislado** que puede implementarse mediante contenedores, máquinas virtuales ligeras o espacios restringidos dentro del propio sistema. En este entorno, el proceso opera con recursos limitados y bajo la supervisión de un **monitor** encargado de verificar sus acciones.

El componente central de la arquitectura es el **mecanismo de control**, responsable de interceptar las llamadas al sistema y compararlas con las **políticas definidas** para determinar si deben permitirse o bloquearse. Dichas políticas especifican los recursos accesibles, tales como archivos, red o dispositivos, y el tipo de interacciones permitidas. La interfaz del kernel actúa como garante del cumplimiento de estas reglas, impidiendo que el proceso acceda a recursos que no le pertenecen o que intente escapar del entorno aislado. Gracias a esta estructura modular, los sandboxes logran contener comportamientos inesperados y mitigar los riesgos asociados con la ejecución de software no confiable o malicioso. [9]

3.3. Aislamiento de memoria y recursos

Uno de los pilares fundamentales del sandboxing es el aislamiento de memoria y recursos, ya que la separación estricta entre procesos evita interferencias y fugas de información. El sistema operativo asigna a cada proceso un espacio de direcciones privado mediante memoria virtual protegida, de modo que ninguna aplicación pueda leer o modificar datos ajenos sin autorización. La unidad de gestión de memoria (MMU) y los mecanismos de protección del kernel garantizan que cada proceso sólo acceda a su propio espacio.

Además de la memoria, el sandbox controla otros recursos críticos. A través de namespaces es posible ofrecer a cada aplicación una versión aislada del sistema, incluyendo procesos, redes y puntos de montaje, de modo que solo tenga visibilidad del entorno que se le asigna. Por su parte, los cgroups limitan el uso de CPU, RAM, almacenamiento y otros recursos para evitar abusos y ataques de denegación de servicio. El acceso al sistema de archivos se restringe mediante rutas específicas o directorios virtualizados, lo que dificulta fugas de información y protege la integridad del sistema. Gracias a estas medidas, incluso si una aplicación presenta un

comportamiento malicioso, sus efectos quedan confinados y no representan una amenaza para el resto del sistema.

3.4. Control de permisos y políticas de acceso

El éxito del sandboxing depende en gran medida del control de permisos y de las políticas de acceso que determinan las capacidades del proceso. Estas políticas conforman un marco que regula el comportamiento de la aplicación, estableciendo explícitamente qué operaciones puede realizar.

El sandbox se encarga de verificar cada acción a través de la interceptación de llamadas al sistema. Cuando un proceso intenta ejecutar una operación no autorizada, el sistema la bloquea inmediatamente o la ejecuta bajo un entorno seguro alternativo. Este mecanismo evita que aplicaciones comprometidas puedan escalar privilegios, acceder a información sensible o realizar acciones peligrosas, fortaleciendo así la postura de seguridad general del sistema. [10]

4. Casos reales e implementaciones destacadas de sandboxing

4.1. Windows Sandbox y AppContainer

En el ecosistema de Microsoft, el sandboxing se implementa principalmente mediante AppContainer y Windows Sandbox. El primero proporciona un entorno restringido para aplicaciones modernas, aplicando permisos declarativos que limitan su acceso a archivos, red y configuraciones internas. Windows Sandbox, en cambio, emplea virtualización ligera basada en Hyper-V para crear un entorno temporal completamente aislado en el que se puede ejecutar software desconocido sin riesgo para el sistema principal. Todo lo realizado dentro de este entorno desaparece tras su cierre, convirtiéndolo en una herramienta eficaz para analizar software potencialmente inseguro. [11]

4.2. macOS y iOS Sandbox

Apple utiliza un modelo de sandbox obligatorio en iOS y opcional en macOS. En ambos casos, las aplicaciones se ejecutan en contenedores con permisos mínimos definidos mediante entitlements. El sistema incorpora firma de código obligatoria y controles estrictos que impiden que una aplicación acceda a recursos no autorizados. La combinación del sandbox con la revisión de aplicaciones en la App

Store permite un enfoque preventivo que reduce significativamente el riesgo de ejecución de software malicioso. [2]

4.3. Linux: SELinux, AppArmor y seccomp

Linux ofrece una amplia variedad de mecanismos de sandboxing. SELinux implementa un modelo de control de acceso obligatorio altamente detallado basado en etiquetas, que define las interacciones precisas entre procesos y recursos. AppArmor proporciona un mecanismo similar pero con perfiles más simples y flexibles, lo que facilita su adopción en sistemas de usuario y servidores. Por su parte, seccomp permite restringir las llamadas al sistema que puede ejecutar un proceso, reduciendo de forma drástica la superficie de ataque y protegiendo al kernel frente a comportamientos peligrosos. [2]

4.4. Navegadores web: Chrome Sandbox y Firefox Content Process

Los navegadores, aunque se ejecutan dentro de un sistema operativo convencional, están expuestos al tipo de contenido más peligroso y variable, es por esto que también implementan sus propios modelos internos de sandboxing. Chrome utiliza un modelo multiproceso donde cada pestaña, extensión y plugin opera en un entorno aislado con privilegios reducidos. Firefox adopta una arquitectura similar mediante content processes, separando el motor de renderizado, la red y la interfaz gráfica para contener los efectos de posibles vulnerabilidades. Estos mecanismos han sido fundamentales para prevenir ataques basados en JavaScript, accesos fuera de contexto y explotación de motores de renderizado. [2]

4.5. Android: sandbox por usuario y UID

Mientras que un *usuario* del sistema operativo tradicional puede tener múltiples procesos y permisos, en Android cada UID se trata como un usuario independiente a nivel del kernel.

Este modelo impide el acceso directo entre aplicaciones sin la autorización explícita del usuario. Además, Android integra permisos declarativos, firma de aplicaciones y un entorno de ejecución aislado mediante la Android Runtime, lo que complementa el aislamiento proporcionado por el kernel de Linux y por SELinux en modo

enforcing. Gracias a ello, el sistema ofrece un entorno robusto para la ejecución de aplicaciones móviles. [12]

Referencias:

[1] Oracle. (2023). *Java SE Platform Security Architecture*. Recuperado de: https://docs.oracle.com/en/java/javase/21/security/java-se-platform-security-architecture.html

[2] Hostragons Global Limited. (2025, 9 marzo). Técnicas de aislamiento de procesos y sandboxing en sistemas operativos. Recuperado de: https://www.hostragons.com/es/blog/tecnicas-de-aislamiento-de-procesos-y-sandboxing-en-sistemas-operativos/

[3] Garfinkel T, Rosenblum, M. (2023). *A Virtual Machine Introspection Based Architecture for Intrusion Detection*. Recuperado de: https://suif.stanford.edu/papers/vmi-ndss03.pdf

[4] Asenjo, R. (2025). ¿Qué es Security by Design o la Seguridad desde el Diseño? LISA News. Recuperado de: https://www.lisanews.org/ciberseguridad/que-es-security-by-design-o-la-seguridad-desde-el-diseno/

[5] Startup Defense. (2025). Qué es el sandboxing. Recuperado de: https://www.startupdefense.io/es-us/blog/que-es-el-sandboxing

[6] Formación Informática. (s. f.). ¿Qué es un kernel y cómo funciona en un sistema operativo? Recuperado de: https://formacioninformatica.es/blog/que-es-un-kernel-y-como-funciona-en-un-sistema-operativo

[7] Belkin, S. (2024). Qué son los namespaces y qué podemos hacer con ellos. Linux Sin Humo. Recuperado de: https://sergiobelkin.com/posts/que-son-los-namespaces-y-que-podemos-hacer-con-ellos/

[8] Belkin, S. (2024, 25 de octubre). Qué son los cgroups y para qué sirven. Linux Sin Humo. Recuperado de: https://sergiobelkin.com/posts/que-son-los-cgroups-y-para-que-sirven/

[9] Anderson, R. (2020). *Security Engineering: A Guide to Building Dependable Distributed Systems* (3rd ed.). Wiley.

[10] Saltzer, J. H., & Schroeder, M. D. (1975). *The protection of information in computer systems*. Recuperado de: https://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15712-s05/www/readings/Saltzer75.pdf

[11] Microsoft Corporation. (2025). Espacio aislado de Windows (Windows Sandbox). Microsoft Learn. Recuperado de: https://learn.microsoft.com/es-es/windows/security/application-security/application-isolation/windows-sandbox/

[12] Android Open Source Project. (s. f.). Application Sandbox. Recuperado de: https://source.android.com/docs/security/app-sandbox?hl=es-419