



**UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO**

**FACULTAD DE INGENIERÍA**

**Semestre en Curso: 2026-1**

**Sistemas Operativos**

**Grupo: 8**

**Profesor: Gunnar Eyal Wolf Iszaevich**

**Exposición: ZeroTrust a Nivel de Sistema Operativo**

**Integrantes:**

Escobar Díaz Victor Manuel

Hernández Rubio Josué

# Contenido

1. Descripción	3
2. Introducción	3
3. Fundamentos Arquitectónicos	3
3.1 Principios Operativos	3
3.2 Relevancia en la Arquitectura del SO	4
4. Implementación en Componentes del SO	4
4.1 Procesos y Control de Acceso	4
4.2 Gestión de Memoria y Aislamiento	5
4.3 Sincronización e IPC Seguro	5
4.4 Sistema de Archivos con Verificación Continua	6
4.5 Hardening del Kernel	6
5. Casos de Estudio Técnicos	6
5.1 SELinux (Security-Enhanced Linux)	6
5.2 Windows Integrity Mechanism	7
6. Métricas de Evaluación	7
7. Conclusión	7
8. Referencias Bibliográficas	7

# ZeroTrust a Nivel de Sistema Operativo

## 1. Descripción

El modelo de seguridad Zero Trust representa un cambio paradigmático en la arquitectura de sistemas seguros, trasladando el principio de "nunca confíes, siempre verifica"

*"Never trust, always verify."*

— John Kindervag, creador del modelo Zero Trust (Forrester Research, 2010)

desde el nivel de red hacia el núcleo mismo del sistema operativo. En esta presentación analizamos la implementación de Zero Trust mediante mecanismos fundamentales del SO como gestión de procesos, control de acceso a memoria, sistemas de archivos y llamadas al sistema, demostrando cómo algunos de los conceptos centrales de la materia permiten construir sistemas resilientes contra amenazas modernas como escalamiento de privilegios y explotación de vulnerabilidades del kernel.

## 2. Introducción

La evidencia empírica demuestra que los modelos de seguridad perimetral tradicionales resultan insuficientes frente a vectores de ataque sofisticados que operan a nivel del sistema operativo. Explotación de race conditions, corrupción de memoria, inyección de código en espacio de kernel y abusos de mecanismos IPC (Comunicación entre Procesos) evidencian la necesidad de un modelo que extienda la verificación continua hasta la capa más fundamental del sistema.

Zero Trust emerge como framework que redefine los límites de confianza, estableciendo que todo componente del sistema debe considerarse potencialmente comprometido. La relevancia para Sistemas Operativos radica en que el kernel constituye el árbitro último de todos los accesos a recursos, posicionándolo como la entidad ideal para implementar políticas Zero Trust mediante mecanismos ya existentes en la teoría de los SOs.

## 3. Fundamentos Arquitectónicos

### 3.1 Principios Operativos

La filosofía Zero Trust se sustenta en tres axiomas fundamentales:

#### Verificación Contextual Continua

Cada operación, desde una llamada al sistema hasta un acceso a memoria, debe validarse considerando identidad del proceso, estado del sistema, comportamiento histórico y metadatos de seguridad. Esto trasciende la autenticación inicial para convertirse en un proceso recurrente durante todo el ciclo de vida del proceso.

### **Privilegio Mínimo Estricto**

Los procesos reciben únicamente los permisos estrictamente necesarios para su función específica en un momento dado, revocándose inmediatamente cuando dejan de ser requeridos. Esto aplica tanto a recursos de memoria como a dispositivos, archivos y capacidades del sistema.

### **Presunción de Compromiso**

El sistema opera bajo el supuesto de que componentes aparentemente legítimos pueden estar comprometidos, implementando mecanismos de detección y contención que prevengan la propagación lateral de amenazas.

*“The principle of least privilege is a fundamental design consideration in the protection of information in computer systems.”*

— Saltzer & Schroeder (1975)

## **3.2 Relevancia en la Arquitectura del SO**

Los sistemas operativos se relacionan con Zero Trust debido a su control monolítico sobre:

- Gestión de Tablas de Páginas y protección de memoria virtual
- Implementación de System Calls como interfaz privilegiada
- Scheduling de Procesos y transiciones user/kernel mode
- Control de Dispositivos mediante drivers del kernel
- Sistemas de Archivos y políticas de acceso persistente

Estas capacidades convierten al SO en el punto de aplicación ideal para políticas Zero Trust, permitiendo implementar verificaciones a nivel de instrucción de máquina.

Aunque sabemos que algunos de sus principios distan de lo que busca Zero Trust y lo que buscan algunos SO, como permitirle al usuario acceder a múltiples recursos con una sola autenticación.

## 4. Implementación en Componentes del SO

### 4.1 Procesos y Control de Acceso

La teoría clásica de procesos establece que cada entidad de ejecución posee un Process Control Block (PCB) que define su estado, contexto y privilegios. Zero Trust extiende este modelo mediante:

- Validación Continua de Credenciales: Cada acceso a recurso requiere revalidación de identidad, incluso para procesos previamente autenticados. Esto mitiga ataques de token reuse y session hijacking.
- Context-Aware Permission Checking: Las decisiones de acceso consideran no solo UID/GID tradicionales, sino también comportamiento reciente, parentesco de procesos y patrones de uso de recursos.

Ejemplo Técnico:

Un proceso de navegador que solicita acceso a `/etc/shadow` genera una alerta de seguridad inmediata, incluso si se ejecuta con privilegios de usuario no privilegiado, debido a la incongruencia contextual entre su función declarada y el recurso solicitado.

### 4.2 Gestión de Memoria y Aislamiento

Los mecanismos de memoria virtual, fundamentales en cualquier SO moderno, se convierten en herramientas críticas para Zero Trust:

- W<sup>X</sup> (Write XOR Execute) Estricto: Ninguna página de memoria puede ser simultáneamente escribible y ejecutable, previniendo inyección de código malicioso.
- ASLR (Address Space Layout Randomization) Mejorado: Randomización no solo en carga inicial, sino durante ejecución para dificultar ataques de memory corruption.
- Memory Tagging Extensions: Uso de hardware moderno (ARM MTE, SPARC ADI) para detectar accesos a memoria fuera de bounds.
- Análisis de Comportamiento de Heap/Stack: Monitoreo continuo de patrones de asignación y liberación de memoria para detectar heap spraying o stack overflow attempts.

*“Address space layout randomization (ASLR) is a technique that can help prevent the exploitation of memory corruption vulnerabilities.”*

— PaX Team (2001), pioneros en ASLR.

### 4.3 Sincronización e IPC Seguro

Los mecanismos de comunicación inter-proceso, esenciales para sistemas modernos, representan un vector de ataque crítico que Zero Trust aborda mediante:

- Autenticación en Memoria Compartida: Validación de identidad del proceso antes de permitir ataques a segmentos de memoria compartida.
- Mensaje de verificación de integridad: Firmado digital de mensajes IPC para prevenir tampering o spoofing.
- Recursos de Cuotas Estrictas: Límites en uso de semáforos, mutexes y otros primitivos para prevenir denial-of-service. (DoS)

Ejemplo de Implementación:

```
// Pseudocódigo para IPC con verificación Zero Trust
int zero_trust_msg_send(pid_t dest, struct message *msg) {
    if (!process_authenticated(dest)) {
        audit_event("IPC_TO_UNVERIFIED_PROCESS", current_pid(), dest);
        return -EPERM;
    }

    if (!context_validates_ipc(current_pid(), dest, msg->type)) {
        audit_event("IPC_VIOLATION_CONTEXT", current_pid(), dest);
        return -EACCES;
    }

    return traditional_msg_send(dest, msg);
}
```

### 4.5 Hardening del Kernel

El núcleo del SO se fortalece mediante técnicas que aprovechan conceptos fundamentales:

- System Call Filtering: Uso de mecanismos como seccomp-bpf para restringir syscalls disponibles por proceso.
- Kernel Module Signing: Requerimiento de firma criptográfica para todos los módulos cargados dinámicamente.

- Privilege Escalation Monitoring: Auditoría continua de intentos de elevación de privilegios mediante setuid o capacidades.
- Control Flow Integrity: Protección contra code reuse attacks mediante verificación de flujo de ejecución.

## **5. Casos de Estudio Técnicos**

### **5.1 SELinux (Security-Enhanced Linux)**

SELinux implementa Mandatory Access Control (MAC) mediante:

- Type Enforcement: Cada proceso y objeto recibe un tipo de seguridad, con políticas que definen interacciones permitidas.
- Role-Based Access Control: Asignación de privilegios basada en roles funcionales más que en identidad de usuario.
- Multi-Level Security: Soporte para clasificaciones jerárquicas tipo Bell-LaPadula.

### **5.2 Windows Integrity Mechanism**

Windows implementa controles similares mediante:

- Mandatory Integrity Control: Procesos y objetos reciben niveles de integridad (Low, Medium, High, System).
- User Account Control (UAC): Elevación consciente de privilegios con consentimiento explícito del usuario.
- Virtualization-Based Security: Uso de hypervisor para aislar procesos críticos del kernel principal.

## **6. Métricas de Evaluación**

La efectividad de implementaciones Zero Trust se mide mediante:

- Reducción en Successful Privilege Escalation
- Tiempo de Detección de Compromisos
- Overhead de Performance en System Calls
- False Positive Rate en Behavioral Analysis

## 7. Conclusión

Zero Trust constituye una evolución natural de los principios fundamentales de Sistemas Operativos, llevando conceptos como aislamiento de procesos, control de acceso y gestión de recursos hacia su expresión más rigurosa. La implementación exitosa requiere integración profunda con mecanismos del kernel, transformando al SO de plataforma pasiva a componente activo en la defensa de sistemas.

Los desafíos restantes, particularmente en balance entre seguridad y usabilidad, representan áreas fértiles para investigación futura, posicionando a Zero Trust como framework de seguridad predominante para la próxima generación de sistemas operativos.

*“Security is not a product, but a process.”*

— Bruce Schneier (2000)

## 8. Referencias Bibliográficas

1. Saltzer, J. H., & Schroeder, M. D. (1975). The Protection of Information in Computer Systems. Proceedings of the IEEE.
2. Loscocco, P. A., & Smalley, S. D. (2001). Integrating Flexible Support for Security Policies into the Linux Operating System. NSA Technical Report.
3. Microsoft Corporation. (2021). Windows Zero Trust Implementation Guide. Microsoft Security Documentation.
4. The Linux Foundation. (2023). Kernel Self-Protection Project Technical Documentation. kernel.org.
5. NIST Special Publication 800-207. (2020). Zero Trust Architecture. U.S. Department of Commerce.
6. Karger, P. A., & Schell, R. R. (1974). Multics Security Evaluation: Vulnerability Analysis. USAF Electronic Systems Division.
7. McKusick, M. K., Neville-Neil, G. V., & Watson, R. N. M. (2014). The Design and Implementation of the FreeBSD Operating System. Addison-Wesley.