PUCE Virtual

CLASE
3

Arquitectura de computadoras

Gestión de memoria y virtualización

QS WORLD UNIVERSITY RANKINGS

Educación de calidad

INTRODUCCIÓN DE LA CLASE

GRADO / POSGRADO / TECNOLOGÍAS

Estudia a tu tiempo
son interrupciones

EDUCACIÓN EN LÍNEA DE CALIDAD

Estimado estudiante, en esta semana profundizaremos en dos componentes fundamentales de la arquitectura del computador que tienen un impacto directo en la seguridad: la gestión de memoria y el proceso de arranque del sistema. Analizaremos cómo la memoria física y virtual, junto con mecanismos como paginación y segmentación, no solo optimizan el rendimiento, sino que también pueden convertirse en vectores de ataque cuando existen fallas como buffer overflow o corrupción de memoria.

Además, estudiaremos el firmware (BIOS y UEFI) y el proceso de inicio del sistema, entendiendo que el arranque constituye la primera etapa de la cadena de confianza del computador. Al finalizar esta clase, habrás comprendido cómo CPU, memoria, almacenamiento y firmware interactúan y cómo esa interacción puede derivar tanto en protección como en vulnerabilidades (Hennessy et al., 2019).

RDA 2: Comprender la arquitectura de la CPU, la memoria, el almacenamiento y los dispositivos de entrada y salida, así como la interacción entre el hardware y el software de una computadora.

Reto 2.

5. GESTIÓN DE MEMORIA Y VIRTUALIZACIÓN

La gestión de memoria es un componente esencial de la arquitectura del computador, ya que regula el almacenamiento, organización y protección de datos e instrucciones, permitiendo la ejecución simultánea de múltiples procesos sin interferencias. Este proceso combina mecanismos de hardware, como la Unidad de Gestión de Memoria (MMU), con políticas del sistema operativo para garantizar aislamiento y control eficiente de los recursos (Hennessy et al., 2019).

La virtualización amplía este modelo de abstracción al permitir no solo el aislamiento de procesos mediante memoria virtual, sino también la ejecución aislada de sistemas operativos completos sobre un mismo hardware físico. Así, la arquitectura moderna incorpora múltiples niveles de traducción y separación que optimizan el rendimiento y refuerzan la seguridad, constituyendo una base estructural frente a vulnerabilidades como la corrupción de memoria (Stallings, 2019).

5.1 Memoria física, memoria virtual, paginación y segmentación

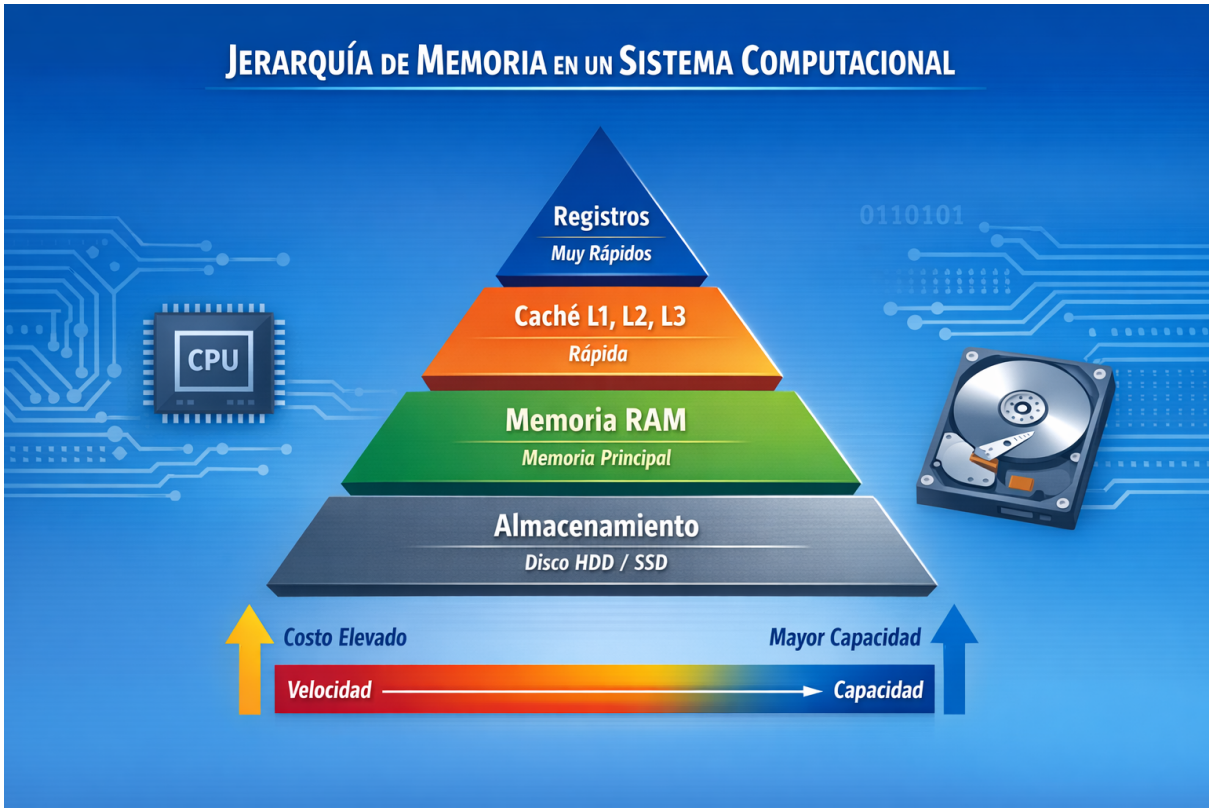
Memoria física y jerarquización

La memoria física corresponde a la RAM instalada en el equipo y constituye el espacio donde el procesador almacena temporalmente datos e instrucciones en ejecución. Es un recurso limitado y compartido por todos los procesos activos, integrado dentro de una jerarquía que abarca registros, memoria caché, memoria principal y almacenamiento secundario (Hennessy et al., 2019).

Su administración adecuada es fundamental para evitar fragmentación, desperdicio o sobrescritura de datos entre procesos, situaciones que pueden generar fallos e inestabilidad del sistema (Stallings, 2019). Por ello, la memoria física debe gestionarse estratégicamente, ya que una administración ineficiente afecta tanto el rendimiento como la seguridad. Este principio se ilustra en la Figura 1 mediante la jerarquía de memoria, organizada en niveles desde registros y caché (mayor velocidad y costo) hasta RAM y almacenamiento (mayor capacidad), evidenciando el equilibrio entre velocidad, costo y capacidad.

Figura 1

Jerarquía de memoria en un sistema computacional



Fuente: Milton Román

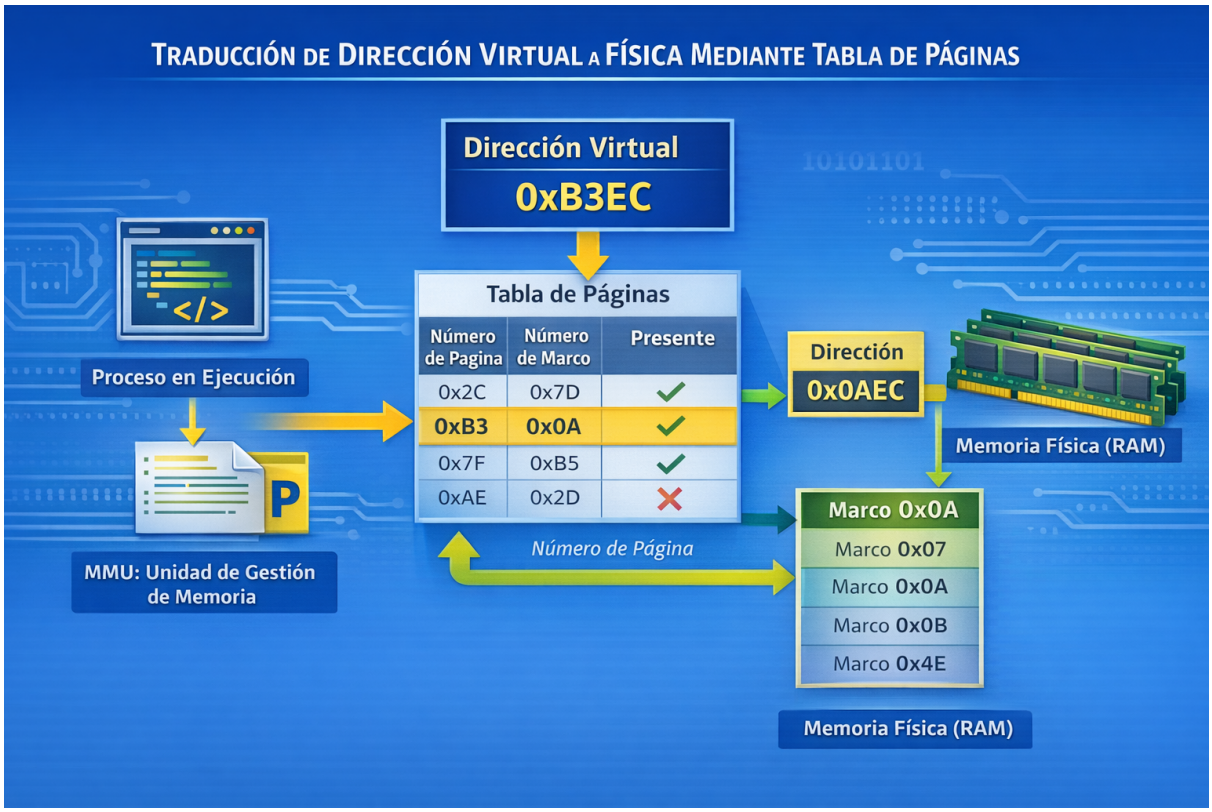
Memoria virtual

La memoria virtual es un mecanismo que otorga a cada proceso un espacio de direcciones lógico independiente de la memoria física real, permitiendo que los programas operen como si dispusieran de mayor capacidad continua de memoria. Este funcionamiento se basa en tablas de páginas administradas por la MMU, que traducen direcciones virtuales en direcciones físicas en tiempo real, garantizando el aislamiento entre procesos (Hennessy et al., 2019).

Desde la perspectiva de seguridad, esta separación impide que un programa acceda directamente a la memoria de otro. Además, posibilita que parte de la información se almacene temporalmente en disco (swap) cuando la RAM es insuficiente, mientras la CPU realiza la traducción de direcciones mediante la MMU. El mecanismo se ejemplifica en la Figura 2 con la conversión de una dirección virtual en una dirección física a través de la tabla de páginas, ilustrando el mapeo entre espacio lógico y físico.

Figura 2

Traducción de dirección virtual a física mediante tablas de páginas



Fuente: Milton Román

Para reforzar el conocimiento sobre memoria virtual, ingresa al siguiente enlace.

- **Título**: Arquitectura de Computadores II - Memoria virtual
- **Descripción**: Explicación sobre el concepto de memoria virtual
- **Enlace:** https://www.youtube.com/watch?v=trgih9B0JAk

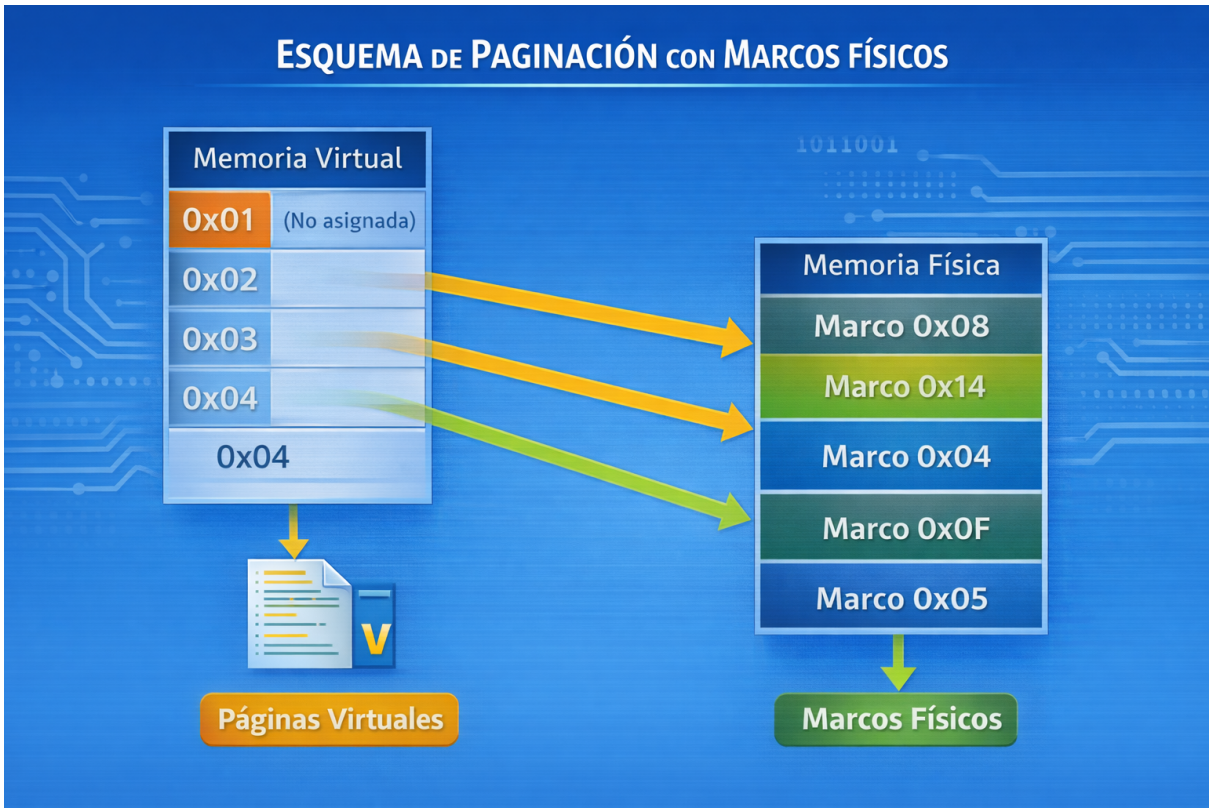
Paginación

La paginación es una técnica de gestión de memoria que divide el espacio virtual en páginas y el espacio físico en marcos de tamaño fijo, estableciendo una correspondencia entre ambos mediante una tabla administrada por el sistema. Cuando un programa solicita memoria, se asignan marcos físicos a sus páginas virtuales, permitiendo una asignación dinámica eficiente.

Según Stallings (2019), este mecanismo reduce la fragmentación externa y constituye la base de la protección de memoria en sistemas modernos, ya que cada página puede configurarse con permisos específicos de lectura, escritura y ejecución. Esto refuerza la seguridad al impedir accesos indebidos, por ejemplo, mediante el uso del bit NX (No-eXecute). La Figura 3 muestra cómo páginas virtuales pueden mapearse a marcos físicos no contiguos, demostrando que la continuidad lógica no requiere continuidad física.

Figura 3

Paginación de memoria con marcos físicos



Fuente: Milton Román

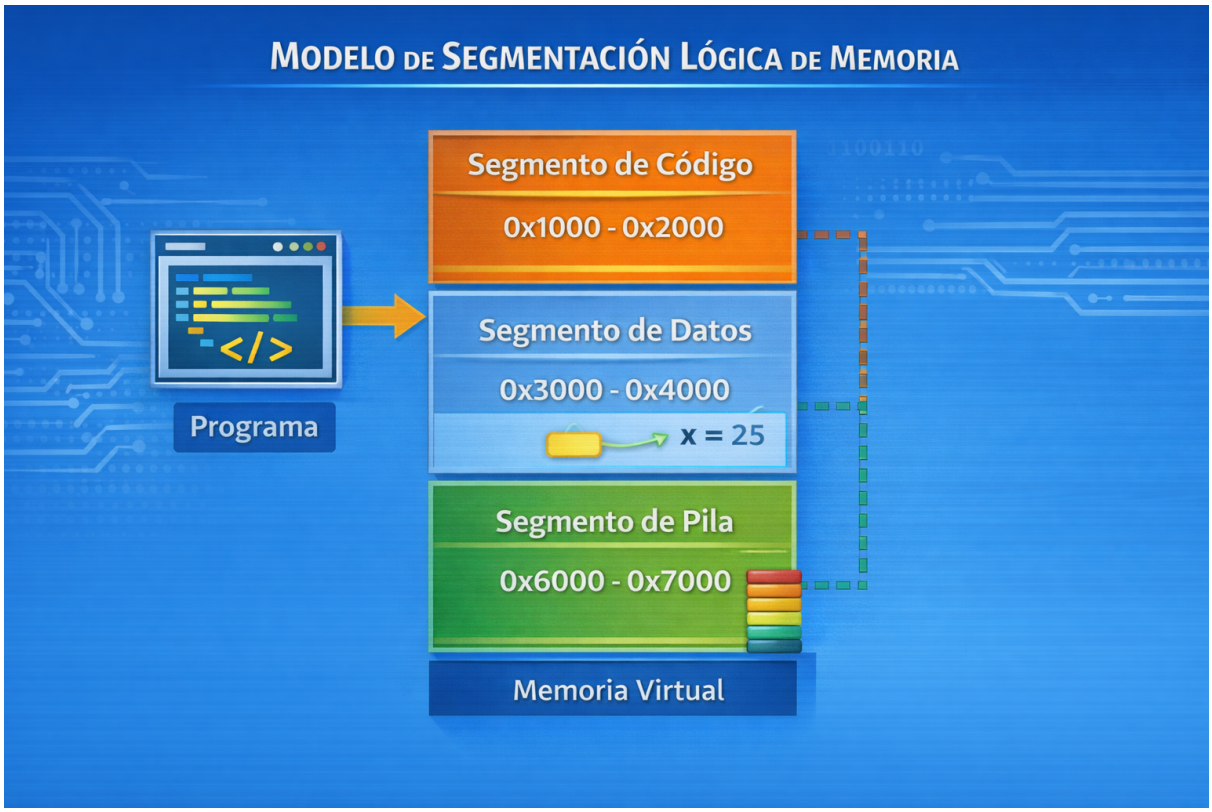
Segmentación

La segmentación es una técnica de gestión de memoria que divide el espacio en bloques lógicos de tamaño variable, como código, datos y pila, reflejando la estructura semántica del programa. A diferencia de la paginación, que realiza una división uniforme, la segmentación organiza la memoria según la función de cada parte del software (Martínez Amador, 2012).

Este modelo permite aplicar permisos diferenciados por segmento, por ejemplo, marcar el código como ejecutable pero no modificable, fortaleciendo la seguridad. No obstante, una validación incorrecta de los límites puede provocar corrupción de memoria. La Figura 4 muestra cómo cada segmento ocupa un rango específico dentro del espacio de direcciones virtual, evidenciando la organización funcional del programa.

Figura 4

Segmentación lógica de memoria



Fuente: Milton Neptalí Román

5.2 Vulnerabilidades como Buffer Overflow y Memory Corruption

Las vulnerabilidades de memoria son amenazas críticas porque pueden permitir ejecución arbitraria de código, escalamiento de privilegios o el control total del sistema (Stallings, 2019). Están directamente relacionadas con la forma en que la arquitectura gestiona la memoria física y virtual (Hennessy et al., 2019).

Por ello, la protección de memoria es una función esencial del hardware moderno, ya que, sin mecanismos adecuados de aislamiento, los errores de programación pueden comprometer la estabilidad y la seguridad del sistema (Stallings, 2019).

1. **Buffer Overflow (Desbordamiento de búfer)**

Un buffer overflow ocurre cuando se escriben más datos de los que un espacio de memoria puede contener, sobrescribiendo posiciones adyacentes, generalmente por falta de validación de entradas o controles de límites.

Si el desbordamiento modifica la dirección de retorno en la pila, el procesador puede ejecutar código malicioso al finalizar la función, ya que el CPU simplemente sigue la dirección indicada por el contador de programa sin distinguir si fue alterada intencionalmente (Hennessy et al., 2019).

2. **Tipos de Buffer Overflow**

a) **Stack Overflow (Desbordamiento en la pila)**

Ocurre en la pila de ejecución, donde se almacenan:

- Variables locales
- Parámetros de funciones
- Dirección de retorno

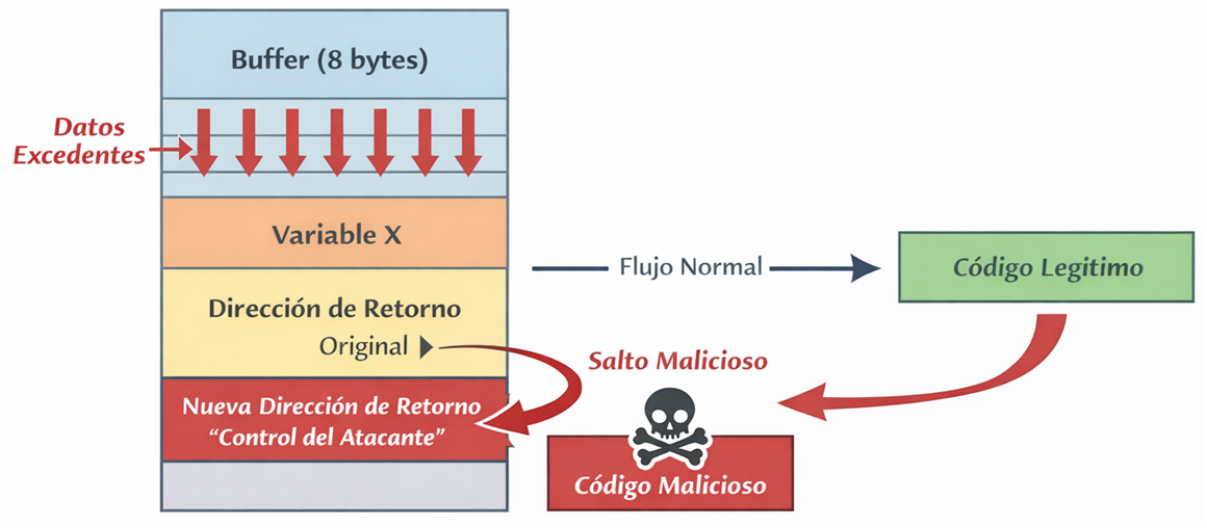
Es el tipo más clásico y ampliamente explotado.

Cuando un programa sobreescribe memoria adyacente, puede alterar instrucciones críticas y permitir ejecución de código malicioso.

La Figura 5 ilustra cómo un error de control de límites en memoria puede comprometer el flujo de ejecución del sistema y convertirse en una vulnerabilidad crítica de seguridad.

Figura 5

Representación de desbordamiento de pila.



Fuente: Milton Román (Adaptado de Stallings (2019)

b) Heap Overflow (Desbordamiento en el montón)

Ocurre en memoria dinámica (heap). Puede alterar:

- Punteros
- Estructuras dinámicas
- Objetos

Este tipo es más complejo, pero asimismo peligroso, especialmente en aplicaciones modernas orientadas a objetos (Stallings, 2019).

3. Corrupción de Memoria (Memory Corruption)

La corrupción de memoria es un término amplio que abarca cualquier modificación indebida del contenido de memoria, no solo desbordamientos. Puede incluir uso de punteros no inicializados, doble liberación de memoria, acceso a memoria liberada y escrituras fuera de límites.

Según Ibarra Mayorga (2009), la arquitectura moderna ha incorporado mecanismos para mitigar estos errores, debido a que históricamente han generado vulnerabilidades críticas.

4. Relación con la arquitectura de memoria

Estas vulnerabilidades están directamente relacionadas con los conceptos vistos en 5.1:

- Memoria física: El desbordamiento ocurre físicamente en RAM, sobrescribiendo datos contiguos (Hennessy et al., 2019).
- Memoria virtual: La memoria virtual aísla procesos entre sí, pero no evita que un proceso dañe su propio espacio interno (Stallings, 2019).
- Paginación: Permite establecer bits de protección NX (No eXecute), Permisos R/W/X. Estos mecanismos mitigan explotación, pero no eliminan el error original (Stallings, 2019).
- Segmentación: Permite separar regiones lógicas (código, datos, pila) y asignar permisos diferenciados (Ibarra Mayorga, 2009).

5. Mecanismos de mitigación arquitectónicos

La industria ha incorporado múltiples mecanismos para reducir el impacto de estas vulnerabilidades:

a) **NX Bit (No Execute):** Evita ejecutar código en regiones marcadas como datos (Stallings, 2019).
b) **ASLR (Address Space Layout Randomization):** Aleatoriza ubicaciones de memoria para dificultar predicción de direcciones (Hennessy et al., 2019).
c) **Stack Canaries:** Inserta valores de verificación antes de la dirección de retorno.
d) **DEP (Data Execution Prevention):** Impide ejecución en regiones no autorizadas.

Stallings (2019) destaca que estos mecanismos dependen del soporte conjunto de hardware y sistema operativo.

6. Escenario integrado de explotación

Flujo simplificado de un ataque:

1. Programa vulnerable recibe entrada excesiva.
2. Se sobrescribe la dirección de retorno.
3. El atacante introduce dirección hacia código malicioso.
4. El CPU ejecuta instrucciones desde esa dirección.
5. Se obtiene ejecución arbitraria.

Si existen mecanismos de protección:

- El ataque puede bloquearse.
- Se genera excepción.
- El sistema operativo interviene.

Si no existen, se compromete el sistema.

7. Impacto en ciberseguridad

Las vulnerabilidades de memoria constituyen una parte significativa de los CVEs históricos, se emplean en ataques de día cero, permiten escalamiento de privilegios y facilitan la persistencia de malware, por lo que la gestión de memoria es un componente estructural de la seguridad del sistema (Stallings, 2019).

Problemas como buffer overflow y memory corruption derivan de cómo la arquitectura organiza y protege la memoria: la memoria física es donde ocurre el error, la memoria virtual aísla procesos sin evitar fallos internos, la paginación introduce permisos de mitigación y la segmentación aplica protección diferenciada. Así, la gestión de memoria no solo influye en el rendimiento, sino también en la superficie de ataque y la capacidad defensiva del sistema (Hennessy et al., 2019).

UNIDAD 6. FIRMWARE Y PROCESO DE ARRANQUE

El firmware y el proceso de arranque conforman la primera capa operativa del sistema, ejecutándose antes del sistema operativo y definiendo el nivel inicial de seguridad (Stallings, 2019). Tras un reset, el procesador inicia la ejecución desde una dirección predefinida donde se encuentra el firmware, dando comienzo a la cadena de arranque (Hennessy et al., 2019).

El firmware es software de bajo nivel almacenado en memoria no volátil que controla el hardware y opera con máximo privilegio. Aunque históricamente se implementó mediante BIOS, actualmente ha sido reemplazado por UEFI, que ofrece mayor modularidad y mecanismos de seguridad avanzados (UEFI, 2021).

6.1 BIOS y UEFI

BIOS (Basic Input/Output System)

El BIOS fue el firmware predominante en arquitecturas x86 tradicionales. Opera en modo real de 16 bits, con acceso limitado al primer megabyte de memoria y utiliza el esquema de particiones MBR. Su arranque es secuencial, poco flexible y carece de mecanismos criptográficos nativos. Durante el inicio ejecuta el POST, detecta dispositivos y carga el sector de arranque del disco (Stallings, 2019).

No obstante, presenta limitaciones importantes: no soporta discos mayores a 2 TB, no verifica criptográficamente el sistema operativo, posee una arquitectura monolítica difícil de extender y es vulnerable a ataques tipo bootkit. Desde la perspectiva de seguridad, su ausencia de mecanismos robustos de validación facilitó históricamente la persistencia de malware en el sector de arranque.

UEFI (Unified Extensible Firmware Interface)

UEFI reemplaza al BIOS tradicional e introduce una arquitectura moderna, modular y extensible (UEFI, 2021).

Sus principales características técnicas son:

- Soporte para 32 y 64 bits.
- Capacidad de direccionamiento ampliado de memoria.
- Uso del esquema de particiones GPT (GUID Partition Table).
- Interfaz modular con drivers propios.
- Soporte de red y sistemas de archivos.
- Arquitectura basada en servicios y aplicaciones pre-OS.

A diferencia del BIOS, UEFI funciona como un pequeño entorno operativo previo al sistema principal. Puede cargar ejecutables EFI desde una partición especial (EFI System Partition).

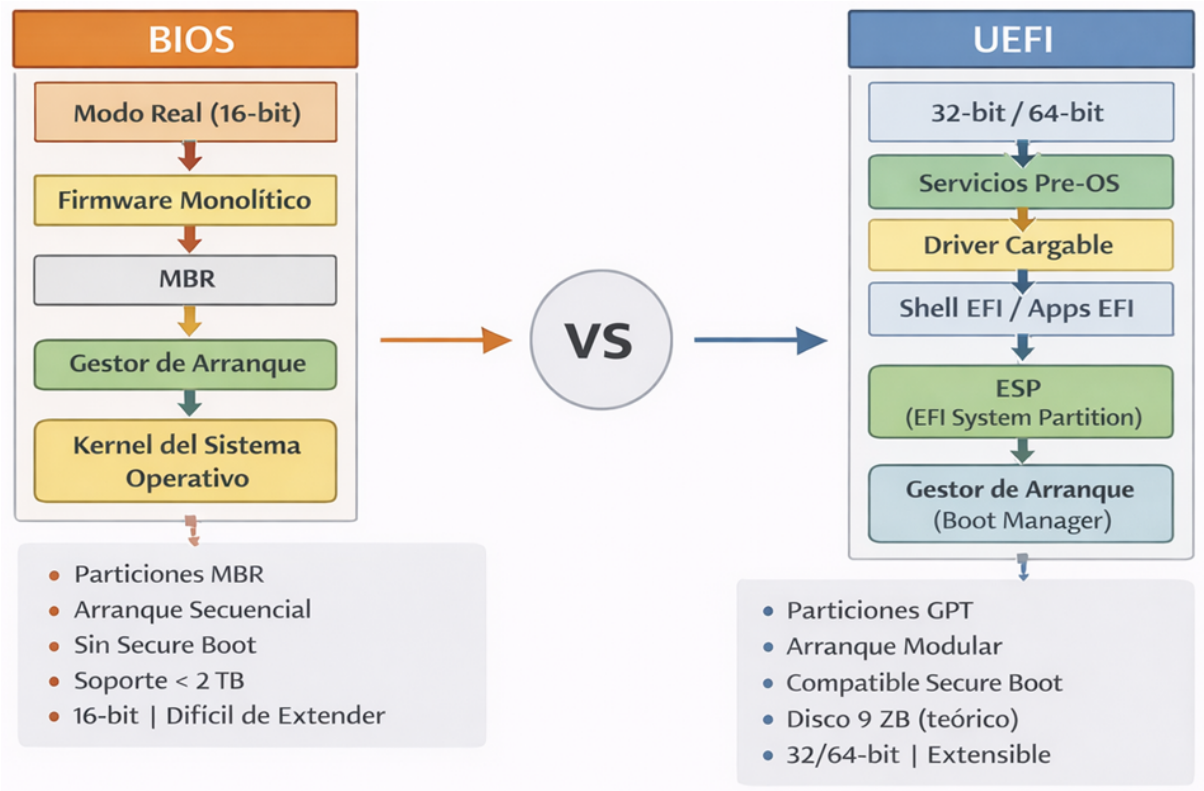
Diferencias técnicas BIOS vs UEFI

Característica	BIOS	UEFI
Modo de operación	16 bits (modo real)	32/64 bits
Esquema de partición	MBR	GPT
Límite de disco	2 TB	> 9 ZB (teórico)
Seguridad	Sin validación criptográfica	Secure Boot
Extensibilidad	Limitada	Modular
Interfaz	Texto básico	Gráfica y programable

Desde la perspectiva de seguridad, la principal diferencia es que UEFI incorpora Secure Boot, un mecanismo que verifica la firma digital del gestor de arranque antes de ejecutarlo, estableciendo una cadena de confianza que reduce la ejecución de código no autorizado (UEFI, 2021).

Figura 6

Comparación estructural BIOS vs UEFI



Fuente: Milton Román

6.2 Proceso de inicio del sistema y su importancia en la seguridad del computador

El proceso de inicio del sistema es la secuencia mediante la cual el computador pasa de un estado sin ejecución a la carga completa del sistema operativo, constituyendo el primer eslabón de la cadena de confianza, ya que el código inicial determina qué componentes controlarán el hardware (Stallings, 2019).

Arquitectónicamente, comienza cuando el procesador recibe una señal de reset y el contador de programa apunta a una dirección predefinida donde reside el firmware. El CPU ejecuta automáticamente esa instrucción sin verificar su legitimidad, por lo que la integridad del firmware resulta crítica para la seguridad del sistema (Hennessy et al., 2019).

1. Fases técnicas del proceso de Inicio

En la Figura 7 observamos la secuencia del proceso de arranque del sistema, donde se representan las etapas principales del boot de manera ordenada y numerada.

Figura 7

Secuencia del proceso de arranque

Fuente: Milton Román

Al encender el equipo, el CPU entra en un estado inicial conocido, deshabilita interrupciones y comienza a ejecutar el firmware desde una dirección predefinida; si este ha sido alterado, el sistema puede iniciar con código malicioso privilegiado (Stallings, 2019).

El firmware (BIOS/UEFI) realiza el POST verificando memoria, almacenamiento y configuración; en sistemas UEFI incluye validaciones adicionales, aunque puede ser vulnerable a ataques como bootkits que se ejecutan antes del sistema operativo (UEFI, 2021).

Luego se identifica el dispositivo de arranque: en BIOS se carga el MBR y en UEFI una aplicación EFI. El bootloader localiza y carga el kernel, operando con altos privilegios, por lo que su alteración puede permitir la inserción de código malicioso.

Finalmente, el kernel inicializa la memoria virtual, configura la paginación, activa el modo protegido y establece niveles de privilegio, habilitando los mecanismos de aislamiento entre procesos y la separación entre modo usuario y modo kernel (Hennessy et al., 2019).

2. Amenazas asociadas al proceso de inicio

Bootkits

Malware que infecta el proceso de arranque para ejecutarse antes del sistema operativo.

Características:

- Persistencia elevada.
- Difícil detección.
- Operación con privilegios máximos.

Firmware Rootkits

Modifican el firmware directamente.

Impacto:

- Sobreviven a formateos.
- Pueden reinstalar malware en cada inicio.

La protección del firmware es esencial debido a este tipo de amenazas avanzadas.

3. Ejemplo Integrado de Ataque

Escenario:

1. El atacante obtiene acceso administrativo.
2. Modifica el bootloader.
3. Inserta código para desactivar ASLR.
4. El sistema arranca normalmente.
5. El usuario no detecta alteraciones.
6. El atacante obtiene persistencia total.

Este ejemplo demuestra que el arranque es un punto estratégico de control del sistema.

4. Importancia Estratégica en Ciberseguridad

El proceso de inicio es crítico porque:

- Se ejecuta con privilegio máximo.
- Determina qué código controlará el hardware.
- Establece la configuración inicial de seguridad.
- Define la superficie de ataque inicial.

Hennessy y Patterson (2019) destacan que la arquitectura del sistema confía en que el entorno inicial es legítimo; si esa confianza se rompe, toda la estructura de protección posterior pierde efectividad.

Refuerce el conocimiento sobre la BIOS y el sistema de arranque del sistema en el siguiente enlace:

- **Título del enlace:** UEFI + GPT o BIOS + MBR: funcionamiento y secuencia de arranque
- **Descripción del enlace relacionado:** Este artículo explica cómo funcionan los principales elementos del firmware (BIOS y UEFI), la secuencia típica de arranque del sistema, y cómo cada tipo de firmware carga el gestor de arranque y el sistema operativo. Incluye explicaciones paso a paso del flujo de arranque desde encendido hasta la carga del kernel del sistema operativo
- **Enlace:** UEFI + GPT o BIOS + MBR: Funcionamiento y secuencia de arranque

Educación de calidad

REFERENCIAS

GRADO / POSGRADO / TECNOLOGÍAS

- **Hennessy, Hennessy, J. L., & Patterson, D. A. (2019).** ARQUITECTURA DE COMPUTADORES: Un enfoque cuantitativo. McGrawHill. https://doi.org/1-55860-069-8
- **Ibarra Mayorga, M. F. (2009).** Evolución de las arquitecturas de las computadoras. eLibro. Obtenido de https://elibro.puce.elogim.com/es/lc/puce/titulos/28671
- **Martínez Amador, H. (2012).** Arquitectura de computadoras: basado en competencias para nivel superior. Obtenido de https://elibro.puce.elogim.com/es/lc/puce/titulos/130397
- **Stallings, W. (2019).** Computer organization and architecture : designing for performance (11ava ed.). Pearson Education. Obtenido de https://os.ecci.ucr.ac.cr/ci0114/material/Stallings/Computer-Organization-Architecture-11th.pdf
- **UEFI, F. (2021).** UEFI Specification (Interfaz de firmware unificada que reemplaza al BIOS tradicional). Obtenido de https://uefi.org/specifications?utm_source=chatgpt.com

PUCE Virtual

síguenos en:

www.pucevirtual.puce.edu.ec