

# MatCom Guard



En este vasto reino digital, los virus y amenazas informáticas son como plagas y ejércitos invasores que buscan corromper tus tierras y saquear tus recursos. MatCom Guard es tu muralla y tu guardia real, un sistema diseñado para vigilar y proteger tu reino (máquina virtual) basado en UNIX de cualquier intruso o actividad sospechosa.

Este proyecto evaluará tus habilidades en Sistemas Operativos, donde deberás demostrar dominio en la gestión de procesos, el control de territorios (archivos y dispositivos), y la defensa de puertos estratégicos (redes). Tu misión será construir una fortaleza impenetrable que detecte y neutralice cualquier amenaza antes de que cause estragos (fecha de entrega del proyecto: domingo, 22 de junio de 2025, 11:59 pm).

## Requisitos Funcionales

### 1. Detección y Escaneo de Dispositivos Conectados (Patrullas Fronterizas)



- Descripción: Los dispositivos USB son como mercaderes o viajeros que cruzan las fronteras de tu reino. Algunos pueden traer consigo enfermedades o espías ocultos.
- Funcionalidad:
  - Monitorear periódicamente el directorio de montaje (las puertas de entrada del reino) para detectar nuevos dispositivos conectados.
  - Realizar un escaneo recursivo del sistema de archivos del dispositivo para identificar archivos agregados, eliminados o modificados, buscando "traiciones" o "plagas".
  - Emitir alertas en tiempo real si se detectan cambios sospechosos en el sistema de archivos, indicando posibles infiltraciones.

**Cambios sospechosos en el sistema de archivos** se definen como modificaciones no autorizadas en atributos, contenido o estructura de archivos en dispositivos USB montados, que puedan indicar malware, intrusiones o comportamientos anómalos.

Ejemplos concretos (en UNIX/Linux):

1. Crecimiento inusual de tamaño: Un archivo .doc pasa de 2KB a 500MB sin intervención del usuario.

2. Replicación de archivos: Aparecen copias idénticas de un archivo con nombres aleatorios (ej: invoice.pdf → invoice\_copy\_xyz.pdf).
3. Cambio de extensión o permisos: Un archivo .txt se renombra a .exe o sus permisos cambian a 777 sin explicación.
4. Atributos modificados: Cambios en timestamps (última modificación) o ownership (ej: root → nobody).

Método de detección:

- Comparación con un baseline inicial (hash SHA-256 de archivos al momento del montaje).
- Alertar si se detectan cambios en más del X% de archivos (umbral configurable, ej: 10%).

## 2. Monitoreo Constante del Uso de Recursos de Procesos e Hilos (Guardias del Tesoro Real)



- Descripción: Al igual que los tributos y recursos del reino, la CPU y la memoria deben ser vigiladas cuidadosamente. Los procesos descontrolados son como ladrones que roban el oro del tesoro real.
- Funcionalidad:
  - Leer información de procesos desde /proc para obtener datos como ID de proceso (PID), nombre del proceso, uso de CPU y memoria.
  - Comparar el uso de recursos entre iteraciones para detectar picos inusuales, como "ladrones hambrientos".
  - Emitir alertas si algún proceso supera umbrales predefinidos para uso de CPU o memoria, señalando posibles traidores o espías.

**Pico inusual de CPU/memoria** se define como un proceso que excede umbrales predefinidos durante más de Y segundos (ej: CPU > 80% por 10s).

Fórmula de detección:

$$\text{alerta} = (\text{uso\_CPU} > \text{UMBRAL\_CPU}) \vee (\text{uso\_RAM} > \text{UMBRAL\_RAM})$$

- Valores por defecto: UMBRAL\_CPU = 70%, UMBRAL\_RAM = 50% (ajustables vía archivo de configuración /etc/matcomguard.conf).
- Ejemplo: Si firefox consume 90% de CPU por 15s, se genera alerta.

Excepciones:

- Procesos en lista blanca (ej: compiler, gnome-shell).

### 3. Escaneo de Puertos Locales (Defensores de las Murallas)

- Descripción: Los puertos abiertos son como puertas en las murallas del castillo. Si no están bien custodiadas, pueden ser explotadas por ejércitos enemigos para infiltrarse.
- Funcionalidad:
  - Escanear un rango de puertos (por ejemplo, 1-1024) utilizando sockets TCP para identificar puertos abiertos.
  - Asociar los puertos abiertos con servicios comunes (por ejemplo, puerto 22 para SSH, puerto 80 para HTTP), buscando anomalías o "puertas secretas".
  - Generar un informe con los puertos abiertos y sus servicios asociados, destacando puertos "potencialmente comprometidos".

#### Detección basada en conexión TCP (socket activo)

Método de detección simplificado:

1. Iterar sobre un rango de puertos (ej: 1-1024) intentando establecer una conexión TCP.
2. Si el puerto responde, se considera abierto ( posible servicio).
3. Si falla la conexión, se considera cerrado/filtrado.

Puertos "potencialmente comprometidos" (sin servicios esperados):

- Criterios de alerta:
  - Puerto abierto sin servicio común asociado (ej: 31337, 6667).
  - Puerto alto (>1024) sin justificación (ej: 8080 podría ser legítimo, 4444 sospechoso).

Ejemplo de detección:

bash

```
./matcomguard --scan-ports 1-1024
```

Salida:

```
[ALERTA] Puerto 31337/tcp abierto (possible backdoor).
```

```
[OK] Puerto 22/tcp (SSH) abierto (esperado).
```

#### 4. Interfaz Gráfica (El Gran Salón del Trono)

- Descripción: Implementar una interfaz gráfica para visualizar los resultados del sistema, como el Gran Salón del Trono donde el monarca toma decisiones estratégicas.
- Funcionalidad:
  - Mostrar en tiempo real los dispositivos conectados, los procesos monitoreados y los puertos abiertos, como un mapa interactivo del reino.
  - Permitir al usuario interactuar con el sistema para iniciar escaneos o generar reportes, como un consejo de guerra.
  - Herramientas Recomendadas: Bibliotecas como GTK+.

Crear una aplicación de consola con 4 entradas:

1. Escanear sistema de archivos
2. Escanear memoria
3. Escanear puertos
4. Escanear todo

En todos los casos al terminar se debe Imprimir reporte en consola o exportar a formato pdf.

#### Requisitos No Funcionales

##### 1. Plataforma de Pruebas (Campo de Entrenamiento)

- El sistema debe ser probado en una máquina virtual con un sistema operativo basado en UNIX, simulando un campo de entrenamiento para preparar a tus tropas.

##### 2. Lenguaje de Programación (Armadura del Caballero)

- El proyecto debe ser desarrollado en C para aprovechar las bibliotecas estándar y las capacidades de bajo nivel del lenguaje, asegurando que el sistema sea robusto como un caballero medieval.

#### Conclusión

MatCom Guard es tu legado como señor feudal de este reino digital. Construye una fortaleza impenetrable, entrena a tus guardias y defiende tus tierras de las plagas y ejércitos invasores. Tu sabiduría y habilidad decidirán el destino de tu reino.

¡Que la protección de tu reino comience!  

Para probar **MatCom Guard** de manera sistemática, se deben diseñar **casos de prueba** que cubran todos los requisitos funcionales (RF1, RF2, RF3) y escenarios. Por ejemplo:

---

### 1. Casos de Prueba para RF1 (Detección de Cambios en Dispositivos USB)

**Objetivo:** Verificar que el sistema detecta archivos modificados, creados o eliminados en un USB.

**Pruebas a realizar:**

Caso	Acción	Resultado Esperado
<b>Inserción USB</b>	Montar un USB con archivos conocidos.	El sistema registra el dispositivo y establece un baseline (hashes de archivos).
<b>Archivo modificado</b>	Cambiar el contenido de un .txt en el USB.	Alerta: "Cambio detectado en /mnt/usb/archivo.txt (hash modificado)".
<b>Archivo nuevo</b>	Crear un malware.exe en el USB.	Alerta: "Archivo sospechoso añadido: /mnt/usb/malware.exe".
<b>Archivo eliminado</b>	Borrar un archivo legítimo.	Alerta: "Archivo eliminado: /mnt/usb/documento.pdf".
<b>Atributos cambiados</b>	chmod 777 a un archivo.	Alerta: "Permisos modificados en /mnt/usb/script.sh (ahora 777)".

## 2. Casos de Prueba para RF2 (Monitoreo de Procesos)

**Objetivo:** Validar que se detectan picos anómalos de CPU/RAM.

**Pruebas a realizar:**

Caso	Acción	Resultado Esperado
Proceso legítimo	Ejecutar gcc (alto uso de CPU).	No hay alerta si está en lista blanca.
Proceso malicioso	Ejecutar un script con while true; do :; done.	Alerta: "Proceso 'bash' usa 99% CPU (umbral: 70%)".
Fuga de memoria	Ejecutar un programa que consuma RAM (tail /dev/zero).	Alerta: "Proceso 'tail' usa 90% RAM (umbral: 50%)".
Proceso fantasma	Matar un proceso y verificar limpieza.	El sistema ya no lo monitorea.

## 3. Casos de Prueba para RF3 (Escaneo de Puertos)

**Objetivo:** Confirmar que se detectan puertos abiertos sospechosos.

**Pruebas a realizar:**

Caso	Acción	Resultado Esperado
Puerto legítimo	Abrir SSH (sudo systemctl start sshd).	Reporte: "Puerto 22/tcp (SSH) abierto  ".
Puerto sospechoso	Abrir puerto 31337 con nc -l 31337 &.	Alerta: "Puerto 31337/tcp abierto (no estándar)  ".

Caso	Acción	Resultado Esperado
<b>Puerto alto inesperado</b>	Abrir puerto 4444 con Python: python3 -m http.server 4444.	Alerta: "Servidor HTTP en puerto no estándar 4444/tcp".
<b>Timeout en puertos cerrados</b>	Escanear puerto 9999 (cerrado).	Reporte: "Puerto 9999/tcp cerrado  ".