

SentinelOS: Uma Distribuição Linux Baseada em Gentoo Focada em Segurança Defensiva e Mitigação de Superfície de Ataque

Igor Paim¹, Flávia Maristela²

¹ Salvador – BA – Brasil

igorpaimdeoliveira@gmail.com

flaviamsn@ifba.edu.br

Abstract. *The evolution of digital surveillance demands desktop systems built with a strict defensive security perspective. This paper presents SentinelOS, a Gentoo-based Linux distribution designed to reduce the attack surface. By replacing legacy components with minimalist alternatives and employing a custom lockdown kernel, MAC (Mandatory Access Control) and Containers, the system creates a resilient anti-surveillance environment.*

Resumo. *A evolução da vigilância digital exige sistemas operacionais construídos sob uma ótica estrita de segurança defensiva. Este trabalho apresenta o SentinelOS, uma distribuição Linux baseada em Gentoo projetada para reduzir a superfície de ataque. Ao substituir componentes legados por alternativas minimalistas e empregar um kernel customizado, MAC (Mandatory Access Control) e Containers, o sistema cria um ambiente antivigilância resiliente.*

1. Introdução

A evolução da vigilância digital e do controle da informação transformou a segurança cibernética em uma questão fundamental de integridade e liberdade, afetando especialmente indivíduos em zonas de ameaça extrema [Greenwald 2014, Schneier 2015]. Sistemas operacionais de desktop convencionais (que executam em computadores desktop ou notebooks) frequentemente priorizam a conveniência com foco no usuário e a compatibilidade universal [Anderson 2020]. Consequentemente, tais arquiteturas carregam densas pilhas de software, muitas vezes legadas, e excesso de serviços operando em segundo plano.

Esse cenário expande a superfície de ataque estrutural do sistema [Manadhata and Wing 2011]. De fato, uma superfície de ataque ampla facilita a ação de malwares e a exploração de vulnerabilidades de dia zero (*zero-days*)¹. Diante desse panorama, o SentinelOS propõe uma estação de trabalho *Gentoo Linux* rigorosamente minimalista. Alicerçado no princípio do privilégio mínimo [Saltzer and Schroeder 1975] e no isolamento proativo, o projeto consolida o uso da *musl libc*, do protocolo Wayland e do Controle de Acesso Obrigatório (MAC) AppArmor para confinar as aplicações e atenuar o impacto de CVEs (Common Vulnerabilities and Exposures).

¹O termo “dia zero” é usado para descrever vulnerabilidades desconhecidas pelos desenvolvedores. O “zero” indica que a equipe criadora do software tem exatos “zero dias” para criar e aplicar uma correção (o patch de segurança) antes que a falha seja descoberta e explorada por criminosos.

2. Aspectos de Segurança em Sistemas Operacionais

A segurança global de uma estação de trabalho é indissociável da integridade do sistema operacional subjacente, uma vez que todas as aplicações, interpretadores e camadas de abstração superiores dependem diretamente da confiabilidade dessa base de confiança fundamental [Reiher 2023]. Caso o sistema operacional possua falhas arquitetônicas ou superfícies de ataque excessivas, qualquer esforço de segurança empregado de forma isolada no nível da aplicação torna-se inócuo [Reiher 2023]. O cenário assemelha-se à analogia clássica de construir uma estrutura de engenharia sólida sobre uma base de areia movediça: independentemente do rigor aplicado na construção superior, o comprometimento da base resulta frequentemente na falha do todo [Reiher 2023]. Sendo o principal responsável pelo controle de processos, isolamento de memória e intermediação de chamadas de sistema (*syscalls*), o sistema operacional configura o perímetro crítico de segurança do ambiente computacional [Reiher 2023].

Paralelo a isso, a fundamentação teórica para o desenvolvimento de sistemas seguros baseia-se tradicionalmente nos princípios estabelecidos por Saltzer e Schroeder (1975), que preconizam diretrizes fundamentais como a economia de mecanismos e o **Princípio do Menor Privilégio** [Reiher 2023]. A economia de mecanismos estipula que o *design* do sistema deve ser o mais simples e enxuto possível, uma vez que bases de código extensas e complexas impedem a auditoria completa e ampliam de forma exponencial a probabilidade de erros lógicos ocultos [Reiher 2023]. Em sistemas operacionais comerciais e distribuições de uso geral convencionais, esses princípios de minimalismo estrutural são frequentemente negligenciados em prol da compatibilidade universal de *hardware* e da conveniência imediata do usuário.

Vale ressaltar que análises empíricas e estudos estatísticos exploratórios baseados em dados reais de falhas em sistemas operacionais revelam que a maior incidência de colapsos e vulnerabilidades não se origina do núcleo básico isolado do *kernel*, mas sim de forma predominante na camada de serviços do sistema operacional e de pilhas de *software* utilitárias no espaço do usuário [Antunes and Júnior 2014]. Evidências empíricas demonstram que serviços redundantes a operar em segundo plano, muitas vezes mal configurados, constituem a principal causa de instabilidade e exposição de sistemas operacionais de *commodity* [Antunes and Júnior 2014]. Embora falhas críticas de *kernel* apresentem maior prevalência relativa em servidores corporativos de alta intensidade de processamento, o espaço de usuário e as suas dependências associadas representam o elo mais vulnerável e o vetor de entrada primário em estações de trabalho de uso pessoal [Antunes and Júnior 2014].

Portanto, a manutenção prolongada de componentes de *software* legados e bibliotecas dinâmicas complexas expande consideravelmente a superfície de ataque estrutural, facilitando a exploração de vulnerabilidades de dia zero (*zero-days*) e o escalonamento local de privilégios. Ademais, para atingir um nível ótimo de proteção contra vigilância ativa e exfiltração de dados, faz-se necessária a adoção da **Defesa em Profundidade** (*Defense in Depth*). Logo, configura-se uma mudança de paradigma arquitetural guiada pela **Redução da Superfície de Ataque**: mitigar riscos não apenas de forma reativa através da aplicação contínua de correções (*patches*) para falhas específicas (CVEs), mas sim atuar proativamente na mitigação de classes inteiras de fraquezas de *design* (CWEs) por meio do endurecimento (*hardening*) do sistema.

Sintetizando, o **isolamento de Processos**, a **minimização** contínua não só do espaço de usuário mas de toda **base de código** e a imposição do **princípio do menor privilégio**, constituem, portanto, as contramedidas estruturais fundamentais em sistemas operacionais que visam segurança defensiva, contrapondo algumas decisões arquitetônicas comerciais, como visto na tabela 1.

Tabela 1. Impacto Arquitetural na Confiabilidade e Segurança do Sistema

Vetor de Risco	Sistemas Convencionais	Paradigma <i>hardened</i>
Abstração do Espaço	Alta densidade de serviços locais	Confinamento por perfis e sandboxing
Base de Código Comum	glibc complexa e retrocompatível	musl libc simplificada e focada em POSIX
Mecanismo de Controle	Controle de Acesso Discrecionário	Controle de Acesso Obrigatório (MAC)

3. Trabalhos Relacionados

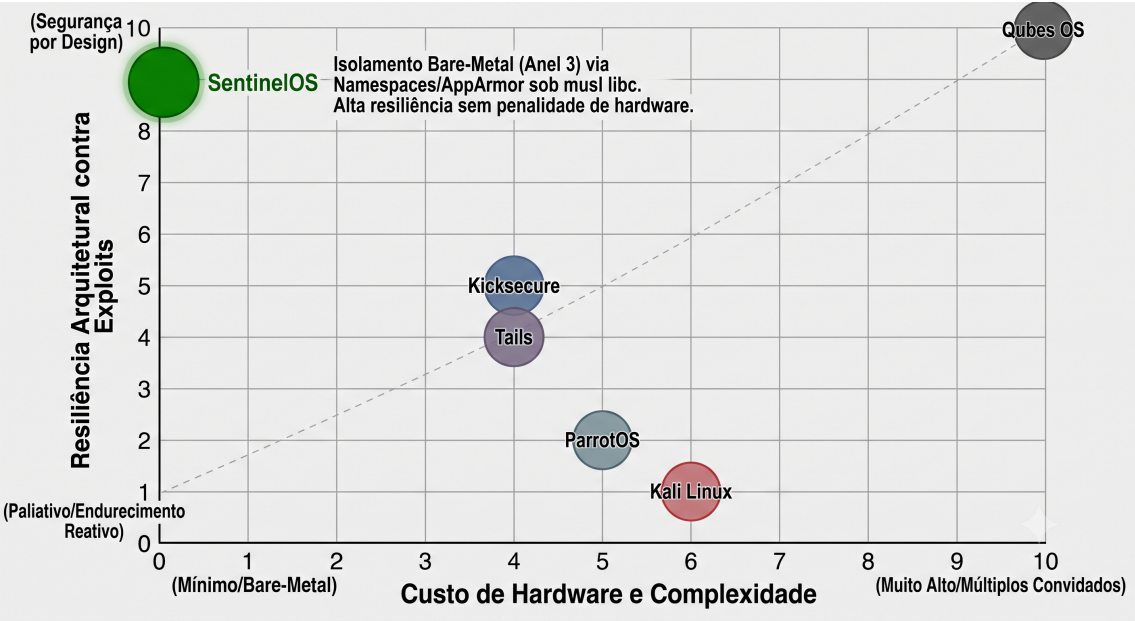


Figura 1. O gráfico retrata um comparativo entre os trabalhos mencionados. O eixo y representa a eficiência defensiva do sistema e o eixo x representa a complexidade e a quantidade de recursos de hardware necessários para executar o sistema

Fonte: Gerado por Inteligência Artificial (Nano Banana 2), baseado em instruções textuais estruturadas pelo autor Igor Paim (2026).

O desenvolvimento de sistemas operacionais orientados à segurança possui um histórico consolidado na literatura e na indústria, porém com abordagens que frequentemente divergem do paradigma de defesa em profundidade focado em minimalismo estrutural. Para evidenciar as contribuições arquiteturais do SentinelOS no contexto de segurança defensiva, é necessário contrastá-lo com as principais distribuições de segurança e privacidade disponíveis no mercado.

3.1. Segurança Ofensiva: Kali Linux e Parrot OS

Sistemas como o Kali Linux e o Parrot OS são amplamente reconhecidos como o padrão da indústria para testes de intrusão (*pentest*) e auditoria de redes. Contudo, seu foco arquitetural é a *segurança ofensiva* (embora o Parrot possua uma versão voltada para anonimato, a *Home Edition*, que se assemelha ao **Kicksecure** abordado na seção 3.4). Para acomodar e garantir a execução de centenas de ferramentas de ataque e análise, essas distribuições dependem de *kernels* modulares altamente permissivos, bibliotecas genéricas (como a `glibc`) e ambientes gráficos convencionais, muitas vezes acoplados a servidores X11. Essa necessidade de compatibilidade irrestrita resulta em uma superfície de ataque ampla. O SentinelOS, em contrapartida, atua no espectro da *segurança defensiva*. Ao adotar um *kernel* monolítico com bloqueio de estado (*Lockdown*), a biblioteca estrita `musl libc` e o isolamento gráfico do protocolo Wayland, o SentinelOS reduz proativamente as abstrações e serviços redundantes que servem como vetores de comprometimento de privilégios em sistemas genéricos.

3.2. Isolamento por Hipervisor: Qubes OS

O Qubes OS é uma referência em segurança de *desktop*, empregando o conceito de *security by compartmentalization* (segurança por compartimentação). A arquitetura utiliza o hipervisor Xen para isolar aplicações, domínios de rede e armazenamento em máquinas virtuais (VMs) rigorosamente independentes. Embora ofereça um confinamento de altíssimo nível, o Qubes OS exige vastos recursos de *hardware* (CPU e memória) e impõe uma alta sobrecarga de processamento, além de carregar a complexidade inerente de gerenciar múltiplos sistemas operacionais hóspedes simultaneamente. O SentinelOS atinge um nível de contenção comparável — mitigando a movimentação lateral e o acesso não autorizado a dados sensíveis — através de um modelo consideravelmente mais enxuto: o isolamento no espaço do usuário (*Ring 3*) via *Namespaces* e *Cgroups* (através da ferramenta desenvolvida para o sistema, o *Podman Env Builder*, abordado na seção 4.3.3), aliado ao Controle de Acesso Obrigatório (*AppArmor*). Esta abordagem garante um confinamento estrito operando de forma nativa (*bare-metal*), diminuindo a latência e a complexidade estrutural de um hipervisor completo.

3.3. Privacidade Amnésica: Tails

O Tails (*The Amnesic Incognito Live System*) é projetado para privacidade extrema e evasão de censura, executando inteiramente a partir de uma mídia removível (*Live USB*) e roteando o tráfego da máquina de forma compulsória pela rede Tor. Seu diferencial principal é o caráter amnésico: não deixar rastros físicos no dispositivo de armazenamento local após o desligamento. No entanto, o Tails é fundamentado na base da distribuição Debian, herdando toda a sua cadeia de código histórico e complexo (`glibc`, `systemd`, núcleo modular). Caso um invasor consiga explorar uma vulnerabilidade de *zero-day* no navegador durante a sessão ativa, a arquitetura subjacente do Tails oferece pouca resistência estrutural para impedir o escalonamento local de privilégios ou a execução remota de código (RCE). O SentinelOS, embora também implemente um rigoroso roteamento transparente via Tor (por meio da ferramenta desenvolvida para o sistema *torswitch*, abordada na seção 4.4.3), supera essa limitação ao focar na integridade sistêmica e na restrição de execução contínua. Ele blinda o espaço do usuário e o *kernel* contra a exploração de falhas em tempo real, mitigando a ameaça antes mesmo do desligamento da máquina.

3.4. Endurecimento de Base Convencional: Kicksecure

O Kicksecure aproxima-se dos objetivos do SentinelOS ao focar em segurança defensiva e privacidade sem exigir virtualização de hipervisor. O projeto aplica configurações rigorosas de *hardening* sobre a distribuição Debian, incluindo proteções em nível de *kernel* contra vazamentos de memória e endurecimento de serviços. O limite desta abordagem, contudo, é a dependência irreversível de uma arquitetura legada. O Kicksecure atua de forma paliativa sobre um ecossistema projetado para conveniência e universalidade. O SentinelOS mostra sua vantagem arquitetural por ser construído sob o princípio de "Segurança por Design" desde a etapa de compilação: substituindo a própria base de código (utilizando o ecossistema Gentoo e *musl*), reduzindo *daemons*² complexos e opacos (substituindo o *systemd* em favor do minimalismo do OpenRC, por exemplo) e removendo nativamente o suporte a módulos carregáveis no *kernel*. Em suma, enquanto sistemas baseados em *hardening* convencional tentam blindar componentes vulneráveis, o SentinelOS mitiga as classes de fraquezas arquiteturais (CWEs) que tornam tais blindagens necessárias.

3.5. A Base Estrutural: Gentoo vs. Distribuições Binárias (Debian e Arch Linux)

A escolha da distribuição base representa um fator determinante para a integridade arquitetural do projeto. Distribuições amplamente adotadas, como Debian e Arch Linux, operam primariamente sob o modelo de empacotamento de binários pré-compilados. O Debian prioriza a estabilidade e a compatibilidade universal, enquanto o Arch Linux adota um modelo de atualizações contínuas (*rolling release*) com foco em simplicidade de uso. Contudo, ambas dependem estruturalmente da biblioteca *glibc* e do gerenciador de serviços *systemd*, características que inserem um volume expressivo de código legado e tendem a ampliar a superfície de ataque basal.

Adicionalmente, o ecossistema de pacotes binários dificulta a aplicação sistêmica de mitigações de compilação (como a *Hardened Toolchain*) e torna a transição estrutural para bibliotecas minimalistas um processo propenso a instabilidades. O SentinelOS, por sua vez, fundamenta-se no Gentoo Linux, uma metadistribuição baseada na compilação do código-fonte. Essa arquitetura viabiliza a construção de todo o espaço do usuário sob diretrizes de segurança customizadas, permitindo a integração nativa da *musl libc* e do inicializador OpenRC. Dessa forma, o ecossistema Gentoo fornece o suporte técnico necessário para aplicar o princípio da economia de mecanismos desde a etapa de construção, viabilizando ajustes profundos que apresentariam complexidade considerável caso fossem adaptados sobre bases de binários genéricos.

4. SentinelOS: O Paradigma da Defesa em Profundidade

A arquitetura do SentinelOS é estruturada com base no princípio militar e computacional da *Defesa em Profundidade* (Defense in Depth) [Schneider 1999]. Este paradigma assume que, invariavelmente, defesas de fronteira falharão ou vulnerabilidades *zero-day* serão exploradas. Portanto, o sistema operacional não confia em uma única barreira; em vez disso, emprega múltiplas camadas de segurança autônomas e sobrepostas [Anderson 2020]. Se um invasor comprometer a primeira camada, ele imediatamente

²*Daemon é um programa que executa em plano de fundo sem necessitar de interação do usuário*

esbarrará em restrições de uma camada subjacente, contendo o raio de explosão (*blast radius*) do ataque [Newman 2015].

O SentinelOS operacionaliza essa teoria através de quatro macro-camadas de isolamento: (a) Kernel Monolítico e Lockdown, (b) Espaço Minimalista de Usuário, (c) Isolamento de Processos e Controle de Acesso Obrigatório (MAC) e (d) Perímetro de Rede e Anonimato Sistêmico. Cada uma destas camadas será detalhada nas seções subsequentes.

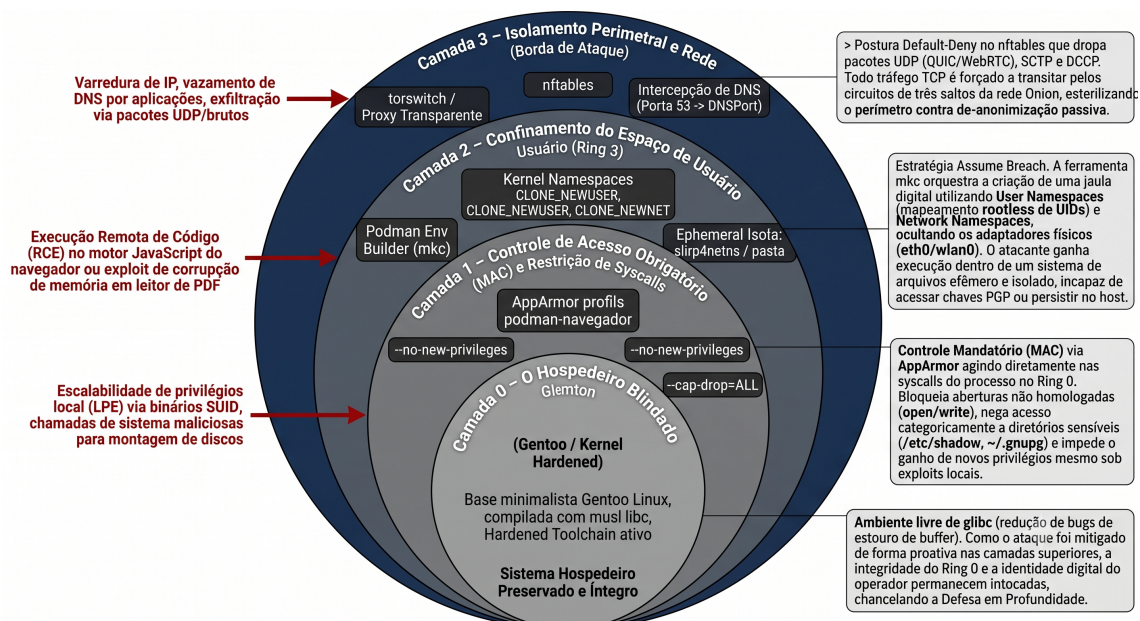


Figura 2. Diagrama de Defesa em Profundidade do SentinelOS: Mecanismo de mitigação em cascata onde ataques perimetrais de rede são retidos na Camada 3 via nftables, tentativas de execução de código ficam retidas nos namespaces da Camada 2 via mkc, e ganchos de escalabilidade de privilégios são interceptados pela camada 1 e pelo Ring 0 pelo AppArmor.

Fonte: Gerado por Inteligência Artificial (Nano Banana 2), baseado em instruções textuais estruturadas pelo autor Igor Paim (2026).

4.1. Camada 0: Kernel endurecido

A base de confiança do SentinelOS reside no espaço do processador (Ring 0), onde opera o kernel Linux. Para compreender a inovação do projeto, é necessário analisar a arquitetura de kernels de distribuições genéricas ou mainstream (como Ubuntu, Fedora e Debian). Estas adotam, por necessidade de compatibilidade universal com variados hardwares, kernels modulares que utilizam Loadable Kernel Modules (LKM) [Bovet and Cesati 2005]. Em outras palavras, as distribuições genéricas compilam o kernel com suporte a milhares de drivers via módulos dinâmicos, carregados sob demanda ou através de um initramfs [Silberschatz et al. 2014]. Embora prático para o usuário final, esse modelo apresenta pontos de falha críticos sob a ótica da segurança, tais como:

- **Injeção de Código em Tempo de Execução:** O suporte a LKMs permite que o usuário *root* insira código diretamente no espaço de memória do kernel. Se um

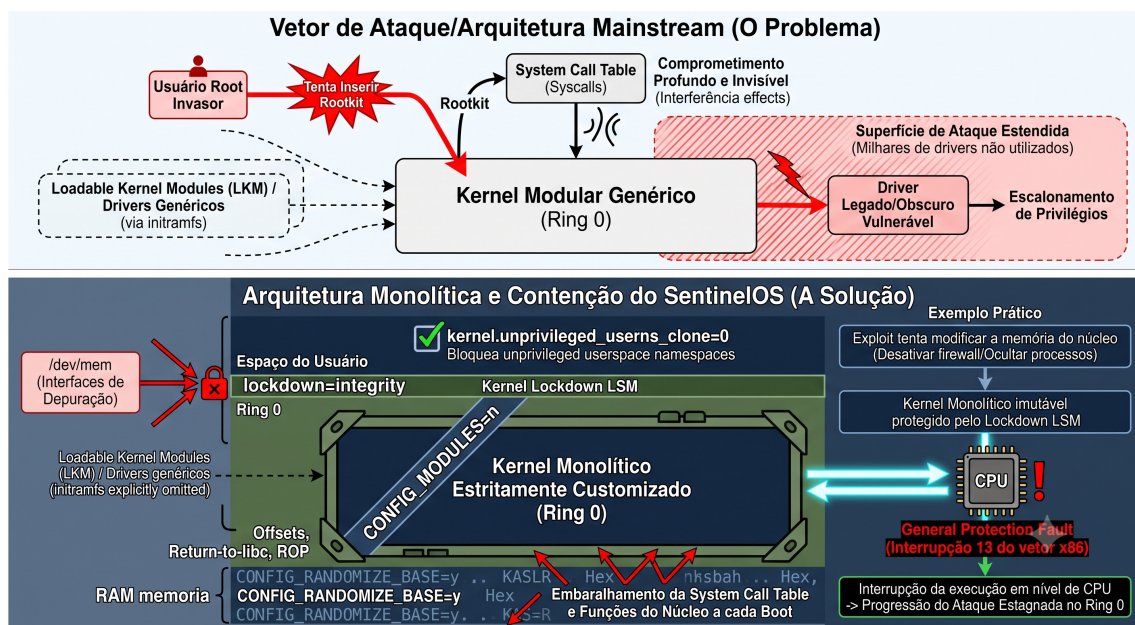


Figura 3. Mecanismos de Proteção e Contenção da Camada 0 do SentinelOS: Diagrama comparativo entre a mutabilidade modular das distribuições *mainstream* e a rigidez monolítica imposta pelas diretivas `CONFIG_MODULES=n`, `lockdown=integrity` e KASLR, culminando no isolamento de falhas por exceção de *hardware* (*General Protection Fault*).

Fonte: Gerado por Inteligência Artificial (Nano Banana 2), baseado em instruções textuais estruturadas pelo autor Igor Paim (2026).

invasor obtém privilégios de superusuário, ele pode carregar um *rootkit*³ de nível de kernel que intercepta chamadas de sistema (*syscalls*), tornando o comprometimento profundo e invisível às ferramentas de monitoramento operando no espaço do usuário [Edge 2012].

- **Superfície de Ataque Estendida:** A métrica de superfície de ataque aumenta consideravelmente com a presença de milhares de drivers não utilizados [Manadhata and Wing 2011]. Vulnerabilidades em drivers de hardware obscuros (ex: adaptadores de rede legados ou sistemas de arquivos antigos) podem ser usadas para escalonamento de privilégios, mesmo que o hardware físico não esteja presente na máquina [Anderson 2020]. A Abordagem do SentinelOS rompe com o modelo modular genérico em favor de uma arquitetura estritamente monolítica e customizada.
- **Mecanismo de Compilação Estática:** Durante a fase de construção, fundamentada no repositório Gentoo, o suporte a módulos é desativado (`CONFIG_MODULES=n`). Apenas os drivers estritamente necessários para o hardware alvo são compilados de forma embutida no binário do kernel. Isso dispensa o uso de *initramfs* genéricos, removendo mais um vetor clássico de modificação persistente e manipulação de *boot*.
- **Kernel Lockdown LSM:** O sistema ativa o Linux Security Module (LSM) de

³Rootkit é uma classe de software malicioso (malware) projetada para obter e manter acesso de administrador (nível root) a um sistema operacional, mascarando sua própria presença e a de outros malwares

Lockdown no modo de integridade (`lockdown=integrity`). Este mecanismo fortalece a barreira entre o espaço do usuário e o kernel, impedindo que o próprio usuário *root* modifique a memória do kernel via interfaces de depuração como `/dev/mem` ou altere parâmetros críticos do processador, efetivamente impondo o princípio do privilégio mínimo até mesmo ao administrador do sistema [Saltzer and Schroeder 1975].

- **Restrição de Clones de Namespaces Unprivileged:** O SentinelOS altera os parâmetros de tempo de execução do núcleo para desativar a clonagem de *User Namespaces* por processos não *root* (`sysctl -w kernel.unprivileged_usersns_clone=0`).
- **Randomização do Espaço de Endereçamento do Núcleo (KASLR):** O SentinelOS impõe a ativação mandatória do mecanismo *Kernel Address Space Layout Randomization* (`CONFIG_RANDOMIZE_BASE=y`). Esta diretiva de compilação instrui o *bootloader* a embaralhar e aleatorizar a posição onde a imagem descompactada do *kernel* é mapeada na memória RAM física a cada inicialização do sistema. Ao remover a previsibilidade estática de endereços de funções do *Ring 0* e estruturas críticas como a *System Call Table*, o KASLR dificulta o cálculo matemático de deslocamentos (*offsets*) exigido por técnicas avançadas de exploração baseadas em reaproveitamento de código, tais como *Return-to-libc* e ROP (*Return-Oriented Programming*), mitigando nativamente o impacto de falhas latentes de corrupção de memória.
- **Exemplo Prático e Contenção:** Em um sistema convencional, um *exploit* - um programa, trecho de código ou sequência de comandos criado especificamente para tirar proveito de uma falha ou vulnerabilidade de segurança em um sistema de software, hardware ou rede - que explore uma falha de corrupção de memória poderia tentar carregar um módulo malicioso para desativar o firewall (`iptables/nftables`) ou ocultar processos maliciosos da lista de execução, por exemplo. No SentinelOS, como o kernel é imutável em tempo de execução e não possui código nativo para o carregamento de módulos, essa tentativa resulta em uma exceção de proteção de hardware (General Protection Fault) ⁴, sem suporte a módulos (Lockdown LSM) o processador identifica qualquer tentativa de modificação da memória do núcleo como uma violação de acesso ao *Ring 0*, interrompendo a execução em nível de CPU. Essa abordagem garante a eficácia das políticas de segurança enforceáveis [Schneider 1999], estagnando a progressão do ataque no *Ring 0*, mesmo que o perímetro da aplicação tenha sido rompido.

4.2. Camada 1: Espaço do Usuário Enxuto (*Minimal Userland*)

A redução significativa da base de código auditável é alcançada substituindo os padrões da indústria por implementações minimalistas e estritamente focadas em POSIX - *Portable Operating System Interface*, conjunto de normas técnicas definidas pelo IEEE junto com o *The Open Group* que estabelece uma interface padronizada entre o sistema operacional e as aplicações. Esta camada aplica o princípio da economia de mecanismos, onde a simplicidade estrutural reduz a probabilidade de erros lógicos não detectados. A

⁴General Protection Fault (GPF), ou Falha de Proteção Geral, é uma exceção de hardware (especificamente a interrupção 13 do vetor x86). O processador a emite quando um programa tenta executar uma operação de memória proibida. Isso protege o sistema operacional e outros processos contra corrupção de dados ou travamentos.

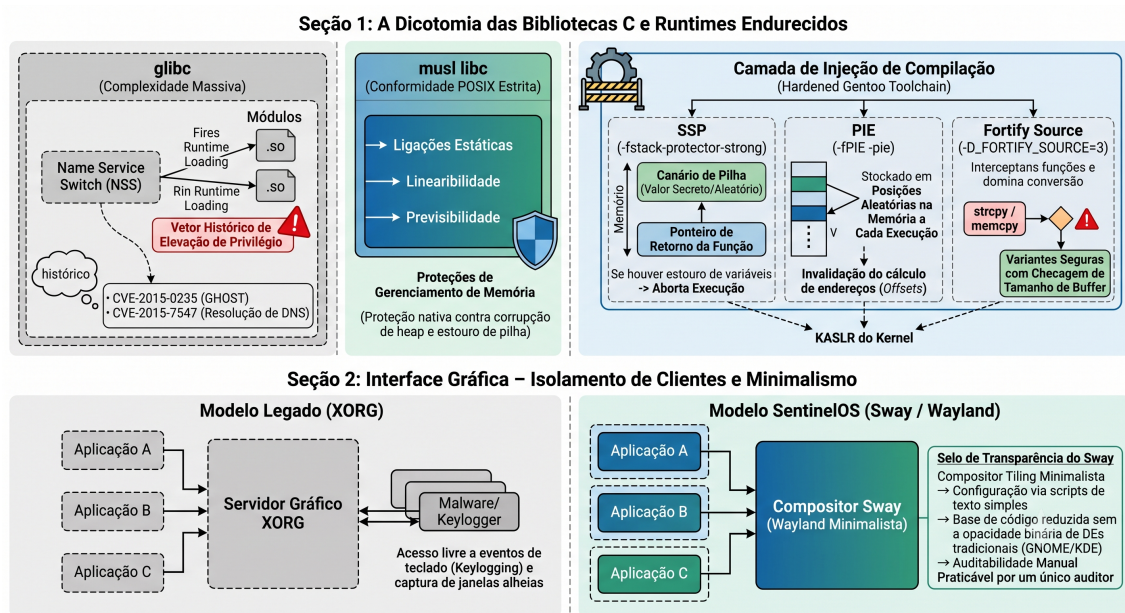


Figura 4. Mecanismos de Proteção e Economia de Mecanismos da Camada 1: Estrutura comparativa demonstrando a mitigação de vulnerabilidades via substituição da *glibc* pela *musl libc* sob a *Hardened Toolchain* (SSP, PIE e *Fortify Source*) e o isolamento gráfico de clientes imposto pela arquitetura minimalista do compositor *Sway/Wayland*.

Fonte: Gerado por Inteligência Artificial (Nano Banana 2), baseado em instruções textuais estruturadas pelo autor Igor Paim (2026).

biblioteca padrão (*standard library*) e a interface gráfica do sistema são abordadas nas subseções seguintes.

4.2.1. A Dicotomia das Bibliotecas C: *glibc* vs *musl libc*

A *glibc*, o padrão em distribuições *mainstream*, possui uma arquitetura que visa compatibilidade histórica e funcionalidades extensas, o que resulta em uma base de código vasta e complexa. Por outro lado, o SentinelOS opta pela *musl libc* devido a diferenciais críticos de segurança:

- **Superfície de Ataque e Código Limpo:** A *musl* é projetada para ser simples e legível, facilitando a auditoria manual. Ao contrário da *glibc*, que utiliza mecanismos como o *Name Service Switch* (NSS) para carregar módulos dinâmicos (.so) em tempo de execução — um vetor histórico de vulnerabilidades de elevação de privilégio, a *musl* foca em ligações estáticas e previsibilidade [Felipe 2023].
- **Gerenciamento de Memória:** A *musl* implementa proteções nativas contra corrupção de *heap* e estouro de pilha de maneira mais restritiva que a *glibc*, mitigando classes de falhas exploradas por ataques de *buffer overflow* [Felipe 2023].
- **Mecanismos de Proteção de Compilação (*Hardened gentoo Toolchain*):** Todo o espaço de usuário é compilado via GCC utilizando flags rigorosas de mitigação de exploits de memória, incluindo:

- **SSP (*Stack Smashing Protection* - `-fstack-protector-strong`):** Injeta canários de pilha (`stack canary`), um valor secreto e aleatório colocado na memória do computador para detectar ataques de Buffer Overflow, antes dos ponteiros de retorno de funções para abortar execuções em caso de estouro de variáveis locais.
- **PIE (*Position Independent Executable* - `-fPIE -pie`):** Força o binário a ser carregado em posições aleatórias da memória a cada execução, trabalhando em sinergia com o KASLR do kernel (mencionado na seção 4.1) para invalidar o cálculo fixo de endereços por atacantes.
- **Fortify Source (`-D_FORTIFY_SOURCE=3`):** Substitui chamadas perigosas de strings (como `strcpy` e `memcpy`) por variantes seguras que checam o tamanho dos *buffers* em tempo de compilação e execução.
- **Resiliência a CVEs:** A complexidade arquitetônica e a retenção de código legado na `glibc` resultaram, historicamente, em falhas críticas de segurança, a exemplo da vulnerabilidade GHOST (CVE-2015-0235) e do comprometimento via resolução de DNS (CVE-2015-7547). O SentinelOS, por adotar a `musl libc`, mitiga estruturalmente essas superfícies de ataque. Ao focar no minimalismo e na conformidade estrita com os padrões POSIX, o sistema restringe vetores de exploração enraizados em implementações vulneráveis não presentes em sua base.

4.2.2. Interface Gráfica: Sway (Wayland) em Detrimento do XORG

A transição para o protocolo Wayland, através do compositor Sway, resolve falhas estruturais de segurança presentes no servidor XORG há décadas.

- **Isolamento de Clientes:** No XORG, qualquer aplicação conectada ao servidor gráfico pode capturar todos os eventos de teclado (*keylogging*) e o conteúdo de outras janelas sem permissão explícita. O Wayland isola cada cliente, garantindo que o compositor controle estritamente o que cada aplicação pode acessar [Høgsberg 2012].
- **Auditabilidade vs. Complexidade de DEs:** Embora ambientes como KDE Plasma e GNOME já suportem Wayland, eles trazem consigo milhões de linhas de código e dependências em binários opacos que tornam a auditoria manual impraticável.
- **A Vantagem do Sway:** O SentinelOS adota o Sway por ser um compositor *tiling* minimalista. Sua configuração é baseada em *scripts* de texto simples e sua base de código reduzida permite que um único auditor verifique a integridade do ambiente gráfico. Ao evitar a opacidade de binários complexos e interfaces redundantes de *Desktop Environments* (DEs) tradicionais, o projeto garante transparência total sobre o que cada processo executa na interface do usuário.

4.3. Camada 2: Isolamento de Processos e Controle de Acesso Obrigatório (MAC)

Enquanto as camadas anteriores focam na integridade do núcleo e na redução da base de código, a Camada 2 estabelece a política de confinamento de aplicações. O SentinelOS assume que qualquer software que interaja com a rede é um vetor de risco em potencial.

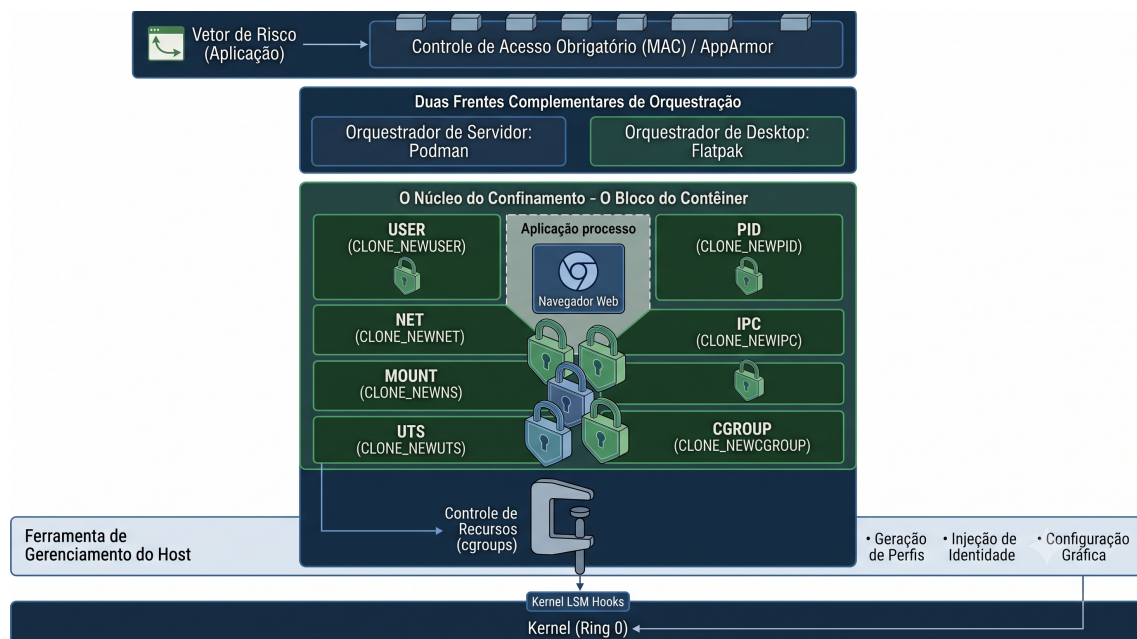


Figura 5. Mecanismos de Confinamento e Controle Mandatário da Camada 2: Representação estrutural do ecossistema de isolamento do SentinelOS, destacando a ação conjunta dos 7 *namespaces* do kernel, o controle quantitativo de *cgroups*, a imposição perimetral de caminhos do AppArmor via ganchos LSM, e a automação declarativa de perfis de segurança gerenciada pela ferramenta proprietária.

Fonte: Gerado por Inteligência Artificial (Nano Banana 2), baseado em instruções textuais estruturadas pelo autor Igor Paim (2026).

4.3.1. Conceitos de MAC e o Papel do AppArmor

O Controle de Acesso Obrigatório (MAC) difere do modelo tradicional de permissões do Linux, o controle de acesso discricionário (Discretionary Access Control ou DAC), por não permitir que o proprietário de um arquivo altere suas permissões de forma arbitrária [Stallings and Brown 2017]. No SentinelOS, a política de segurança é centralizada e imposta pelo kernel, independentemente da vontade do usuário através do **LSM**⁵ (*Linux Security Modules*).

Existem algumas Tecnologias que implementam o **LSM**, o SELinux (*Security-Enhanced Linux*), baseado em rotulos (*labels*), e o AppArmor, baseado em caminhos (*path-based*) são as mais proeminentes. O SentinelOS utiliza o AppArmor como mecanismo de MAC primário devido à sua arquitetura simples baseada em caminhos de arquivos.

- **Vantagens:** Sua principal vantagem em relação ao SELinux (que utiliza *labels*) é a facilidade de auditoria e criação de perfis humanos-legíveis [Bauer 2006]. Ele permite criar "jaulas" cirúrgicas para aplicações específicas, como o navegador ou

⁵O LSM atua como uma interface de "ganchos" (hooks) dentro do kernel. Sempre que um processo tenta realizar uma chamada de sistema crítica (como abrir um arquivo, executar um binário ou abrir um socket de rede), o kernel primeiro verifica as permissões normais do usuário e, em seguida, consulta o módulo de segurança ativo (como o AppArmor ou SELinux) para tomar a decisão final de bloqueio ou permissão.

clientes de VPN.

- **Desvantagens:** Em comparação ao SELinux, o AppArmor possui uma granularidade ligeiramente menor em objetos que não são arquivos (como estruturas complexas de rede), porém, para um ambiente de desktop focado em antivi-gilância, a simplicidade de auditoria do AppArmor reduz a probabilidade de erros de configuração que deixariam o sistema vulnerável.
- **Mecanismo:** O AppArmor retira a herança de privilégios, impedindo que um processo comprometido acesse pastas como `~/ .ssh` ou `~/ .gnupg`, mesmo que o processo tenha sido iniciado pelo usuário dono desses arquivos.

4.3.2. Virtualização de Aplicações: A Engenharia de Namespaces, CGroups, Pod-man e Flatpak

O isolamento moderno de processos no ecossistema Linux não repousa sobre hipervi-sores de máquina virtual tradicionais, mas sim sobre primitivos nativos implementados diretamente no núcleo do sistema operacional (Ring 0). O SentinelOS converge esses mecanismos para estabelecer um ambiente de confinamento de alta performance e baixa sobrecarga (*overhead*). As duas ferramentas basilares fornecidas pelo *Kernel* para este fim são os *Control Groups* (*cgroups*) e os *Namespaces*.

Enquanto os *cgroups* atuam como um medidor e limitador de recursos quantitativos do sistema — restringindo o teto percentual de uso do processa-dor (CPU), cotas de memória RAM e largura de banda de E/S de disco (*I/O*) [Linux man-pages project - cgroups] —, os **Namespaces** operam na dimensão qualita-tiva, alterando a percepção que um processo possui sobre os recursos globais do sistema operacional [Linux man-pages project - namespaces]. Em suma, um *namespace* envolve um processo em uma abstração virtualizada, fazendo-o crer que é o único nó ativo na máquina.

Para estruturar o confinamento estrito de aplicações como navegadores web e uti-litários de terceiros na Camada 2, o SentinelOS apoia-se nativamente em sete tipos fun-damentais de *Namespaces*:

- **User Namespace (CLONE_NEWUSER):** É o pilar central da segurança *rootless*. Ele permite a virtualização de IDs de usuário e de grupo (UIDs e GIDs). Na prática, um processo pode possuir privilégios de superusuário (UID 0 ou *root*) dentro do seu escopo isolado, mas quando suas chamadas de sistema (*syscalls*) to-cam o *Kernel* do hospedeiro (*host*), esse ID é traduzido via tabelas de mapeamento para um UID arbitrário de usuário comum e sem privilégios (ex: UID 100050). Isso neutraliza ataques de escape de contêiner (*container escape*) baseados em identidade.
- **Network Namespace (CLONE_NEWNET):** Virtualiza por completo a pilha de rede, fornecendo ao processo suas próprias tabelas de roteamento, regras de fi-rewall independentes, barramento de *loopback* privado e interfaces de rede virtu-ais.
- **Mount Namespace (CLONE_NEWNS):** Isola os pontos de montagem vistos pelo processo. O contêiner recebe uma árvore de diretórios exclusiva (*rootfs*) desvincu-lada do *host*. Modificações, montagens ou desmontagens efetuadas pela aplicação enclausurada não afetam o sistema de arquivos persistente da máquina hospedeira.

- **PID Namespace (`CLONE_NEWPID`):** Isola a árvore de identificadores de processos. A aplicação disparada assume o papel de processo inicial (`PID 1`), mapeando seus filhos de forma sequencial interna. O processo torna-se incapaz de enxergar, enviar sinais (`kill`) ou interagir com os processos em execução no sistema operacional hospedeiro.
- **IPC Namespace (`CLONE_NEWIPC`):** Isola os recursos de Comunicação Inter-Processos, incluindo filas de mensagens POSIX, segmentos de memória compartilhada, semáforos etc. Impede que um *malware* em execução na aplicação intercepte buffers de memória de processos legítimos do sistema via IPC.
- **UTS Namespace (`CLONE_NEWUTS`):** Permite a definição de um nome de host (*hostname*) e domínio específicos para o contêiner, mascarando o nome de rede real do SentinelOS.
- **Cgroup Namespace (`CLONE_NEWCGROUP`):** Virtualiza a visão do processo sobre os caminhos e subestruturas dos próprios *control groups*, impedindo que a aplicação containerizada descubra ou altere seus próprios limites de recursos de *hardware*.

Com base nesses primitivos, o SentinelOS rege a execução de softwares adotando duas frentes complementares de orquestração:

- **Orquestração via Podman (*Engine Native*):** O sistema opta pelo Podman em detrimento da arquitetura convencional do Docker devido a rupturas estruturais de *design*. O Docker depende de um *daemon* central permanente (`dockerd`) operando com privilégios de `UID 0` no *host* para gerenciar o ciclo de vida dos contêineres, constituindo um vetor clássico de alvo para escalonamento local de privilégios. O Podman opera de forma *daemonless*, instanciando os *namespaces* diretamente através do gerenciador de processos padrão do sistema operacional. Por ser *rootless* por padrão [Hat 2023], o SentinelOS garante que se o motor do contêiner ou o runtime sofrerem um comprometimento crítico, o invasor permanecerá restrito ao perímetro de segurança de um usuário de baixa autoridade no *host*.
- **Isolamento via Flatpak e Bubblewrap:** Para softwares complexos de interface gráfica e alta densidade de dependências (como o Firefox), o SentinelOS delega a gerência ao ecossistema Flatpak. O Flatpak atua como uma camada de abstração que invoca o utilitário de baixo nível *bubblewrap*. O *bubblewrap* utiliza os mesmos ganchos de *namespaces* do kernel Linux para erguer caixas de areia (*sandboxes*) estritas, viabilizando o controle granular e declarativo de permissões periféricas — barrando o acesso direto a barramentos de áudio (ALSA), aceleração de hardware bruta e forçando a conformidade com diretórios efêmeros.

O reflexo prático dessa arquitetura integrada manifesta-se diretamente na blindagem da pilha de comunicações. O contêiner do navegador roda mapeando o `UID` do usuário interno para um `UID` não privilegiado no *host*. Estruturalmente, o orquestrador delega o gerenciamento de rede ao utilitário `slirp4netns` (ou `pasta`), criando um *Network Namespace* (`CLONE_NEWNET`) isolado. Esse mecanismo instancia uma interface virtual efêmera (tipicamente `tap0`) restrita ao ambiente interno da aplicação. Como o processo do contêiner carece de privilégios para interrogar ou modificar os adaptadores físicos do hospedeiro (`eth0` ou `wlan0`), o endereço IP real da máquina permanece

oculto e inacessível a nível de sistema de arquivos, neutralizando ganchos de vazamento de metadados de rede por aplicações corrompidas.

4.3.3. Ferramenta de Gerenciamento Customizada

Uma das contribuições centrais deste projeto é a ferramenta proprietária de gerenciamento de contêineres construída sobre a interface do Podman. Diferente de soluções voltadas à distribuição estática de aplicações, a ferramenta atua como um orquestrador focado em liberdade de desenvolvimento e controle granular de isolamento.

- **Agregação ao Projeto e Isolamento Dinâmico:** A ferramenta abstrai a complexidade arquitetônica ao gerar dinamicamente arquivos `Dockerfile` e `scripts` de compilação em tempo de execução. Uma característica arquitetural crítica é a injeção automática do mapeamento de identificadores do usuário do *host* (`USER_UID` e `USER_GID`) para o interior da imagem. Isso mitiga os conflitos clássicos de Controle de Acesso Discrecional (DAC) ao montar volumes locais, permitindo que múltiplos gestores de pacotes (como `apt`, `pacman` ou `apk`) operem de forma transparente sem comprometer a integridade do sistema hospedeiro.
- **Mecanismo de Ação e Restrição por Perfis:** O sistema implementa *wrappers* programáticos em Python e gera entradas de atalho (`.desktop`) dinâmicas, integrando o contêiner de forma imperceptível à interface do usuário. A imposição de políticas de segurança ocorre por meio de um subsistema de perfis em disco (`.profile`). Ao instanciar um contêiner, o *script* intercepta a execução e injeta blocos de *flags* nativas predefinidas (como `--cap-drop`, `--no-new-privileges` ou opções do `seccomp`). Ademais, o mecanismo gerencia a exposição da superfície de ataque gráfica, restringindo o acesso ao *socket* do servidor Wayland (*flag* `--gui`) estritamente aos ambientes que demandam interface visual.

4.4. Camada 3: Perímetro de Rede e Anonimato Sistêmico

A última fronteira de defesa do SentinelOS atua sobre a comunicação, operando sob a premissa de que o ambiente de rede é hostil e sujeito à vigilância ativa ou censura estatal [Schneier 2015]. Esta camada é composta por um firewall restritivo e um mecanismo de roteamento por cebola (*onion routing*) que abrange todo o sistema.

4.4.1. Filtragem com `nftables` e Política *Default-Deny*

O SentinelOS utiliza o `nftables` em substituição ao legado `iptables`, aproveitando uma sintaxe mais limpa e maior eficiência no processamento de pacotes diretamente no *kernel* [Russell 2014].

- **Mecanismo *Stateful* e Inspeção de Entrada:** O firewall opera com inspeção de estado (*stateful inspection*), adotando uma postura de *default-deny* (negação padrão) estrita para as cadeias de entrada (*input*) e encaminhamento (*forward*). Embora o tráfego de saída (*output*) seja permitido por padrão para garantir a usabilidade da rede, pacotes entrantes só são aceitos se pertencerem a conexões previamente estabelecidas pelo próprio *host* (através da regra `ct state`

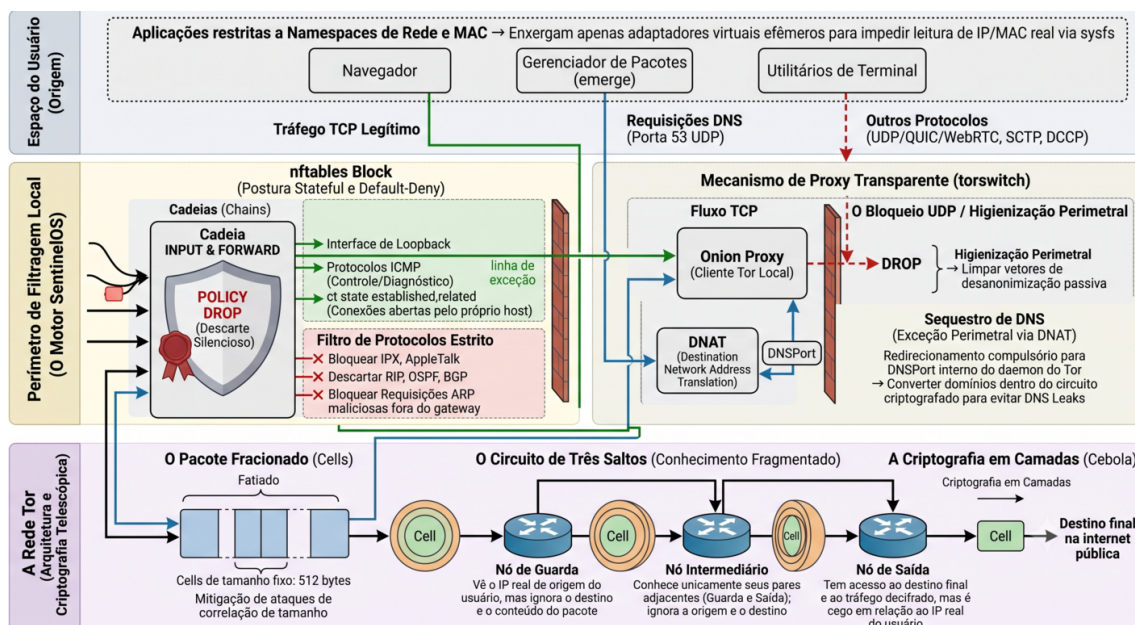


Figura 6. Arquitetura de Rede e Anonimização da Camada 3: Fluxograma técnico demonstrando a filtragem perimetral via `nftables`, o sequestro de DNS via DNAT para a `DNSPort`, o descarte de pacotes UDP/SCTP/DCCP pelo proxy transparente (*torswitch*), e o encapsulamento do tráfego TCP em células de 512 bytes sob criptografia telescópica na rede Tor.

Fonte: Gerado por Inteligência Artificial (Nano Banana 2), baseado em instruções textuais estruturadas pelo autor Igor Paim (2026).

established, related), com exceções apenas para a interface de *loopback* e protocolos de diagnóstico (ICMP).

- **Filtragem Estrita de Protocolos e Resiliência Perimetral:** O SentinelOS adota uma postura de confinamento de tráfego severa, limitando a pilha de comunicação ativa estritamente aos protocolos IPv4, IPv6 e ao tráfego de controle e diagnóstico ICMP. Através de diretivas estruturadas no `nftables`, ganchos de protocolos exóticos ou legados (como IPX e AppleTalk), bem como anúncios de roteamento dinâmico não solicitados (como RIP, OSPF e BGP), são bloqueados e descartados (*policy drop*) na cadeia *input*. Adicionalmente, ganchos de resolução perimetral redundantes na camada de enlace (como requisições ARP maliciosas fora do escopo do gateway) também são filtrados. Essa restrição explícita neutraliza vetores de reconhecimento passivo, ataques de injeção de cabeçalhos malformados no núcleo do sistema operacional e tentativas de envenenamento de tráfego local (*Spoofing*), convertendo a estação de trabalho em um nó estruturalmente isolado em redes potencialmente hostis.

4.4.2. Roteamento Tor: Arquitetura e Criptografia em Camadas

Para assegurar a evasão contra técnicas de vigilância e análise de tráfego, o sistema integra o roteamento através da rede Tor (*The Onion Router*). A arquitetura constrói circuitos virtuais de forma telescópica, onde o *Onion Proxy* (cliente local) negocia chaves

criptográficas independentes com três nós sequenciais: o nó de guarda (*guard relay*), o intermediário (*middle relay*) e o de saída (*exit relay*) [Dingledine et al. 2004]. Para mitigar ataques baseados em correlação de tamanho, todo o tráfego é fatiado e encapsulado em *cells* (células) de tamanho fixo padrão de 512 bytes.

- **Criptografia Telescópica e *Perfect Forward Secrecy* (PFS):** O cliente encapsula os dados em três camadas sobrepostas de criptografia simétrica (AES), cujas chaves são previamente estabelecidas via protocolo Diffie-Hellman com cada roteador do circuito. À medida que a célula transita, cada nó decifra e remove (“descasca”) exclusivamente a camada de criptografia endereçada a ele. O uso de chaves efêmeras neste processo garante o *Perfect Forward Secrecy*, impedindo que um eventual comprometimento futuro de um nó permita a decifragem do tráfego interceptado no passado.
- **Desacoplamento de Estado e Identidade:** A topologia de três saltos impõe um modelo rigoroso de conhecimento fragmentado. O nó de guarda reconhece o endereço IP de origem do usuário, mas ignora a natureza e o destino do pacote; o nó intermediário conhece unicamente seus pares adjacentes; e o nó de saída tem acesso ao tráfego decifrado e ao destino, mas é cego em relação à identidade do emissor.
- **Resiliência contra Análise de Tráfego:** Esse isolamento arquitetônico fragmenta o vetor de ataque. Como nenhum ponto da rede detém simultaneamente o endereço IP real e o destino final, tentativas de desanonimização e *fingerprinting* de rede exigem a observação e correlação estatística simultânea nas bordas de entrada e saída, um cenário computacionalmente custoso e inviável para adversários passivos locais.

4.4.3. Superioridade do Proxy Transparente e Prevenção de Vazamentos

Diferente de distribuições convencionais que dependem exclusivamente de aplicações isoladas como o *Tor Browser*, o SentinelOS implementa um **Proxy Transparente** em nível de sistema operacional através da ferramenta proxy customizada criada para o sistema, chamada *torswitch*, que, ao ser invocada no terminal, roteia todo o tráfego da máquina pela rede Tor, sendo capaz de desfazer o processo quando chamada novamente. Esta abordagem resolve a falha crítica de “vazamento de aplicação”, onde processos fora do navegador expõem inadvertidamente o endereço IP real do usuário.

- **Confinamento de Protocolo e *Default-Deny*:** Reconhecendo que a rede Tor transporta nativamente apenas fluxos orientados a conexão via TCP [Dingledine et al. 2004], o SentinelOS adota uma postura de negação padrão (*default-deny*) para toda a extensão da camada de transporte e protocolos adjacentes. Com exceção do tráfego DNS controlado (abordado no item abaixo), toda a pilha UDP (utilizada por protocolos modernos de baixa latência como QUIC e WebRTC), bem como protocolos de transporte alternativos como SCTP (*Stream Control Transmission Protocol*) e DCCP (*Datagram Congestion Control Protocol*), são descartados (*drop*) pelo *nftables*. Essa higienização perimetral mitiga na base vetores de desanonimização passiva que comumente burlam *proxies* de nível de aplicação.

- **Sequestro de DNS (*DNS Hijacking*) como Exceção Perimetral:** Para viabilizar a resolução de nomes sem a exposição de metadados, o firewall abre uma única exceção estrita na pilha UDP: ele intercepta requisições direcionadas à porta 53 e as redireciona via *Destination Network Address Translation* (DNAT) para o serviço `DNSPort` interno do daemon do Tor. Este mecanismo garante que a conversão de domínios em endereços IP ocorra exclusivamente dentro dos circuitos criptografados da rede *onion*, neutralizando riscos de vazamento de DNS (*DNS Leaks*) que poderiam revelar a localização geográfica ou o perfil de navegação do usuário ao provedor de acesso (ISP).
- **Blindagem Sistêmica e Mitigação de Exfiltração:** Enquanto soluções convencionais baseadas em navegadores protegem apenas o tráfego HTTP/HTTPS local, o proxy transparente do SentinelOS força a totalidade do tráfego TCP legítimo do espaço do usuário — incluindo o gerenciamento de pacotes via `emerge`, utilitários de terminal e processos em segundo plano — a transitar obrigatoriamente pelos circuitos de três saltos, garantindo auditoria e anonimato irrestritos a nível de sistema operacional.
- **Análise de Escopo e Vetores de Contenção Adicionais:** É imperativo destacar que, embora o `nftables` impossibilite o vazamento de pacotes brutos direcionados à rede pública fora dos túneis criptográficos, a de-anonimização do operador ainda poderia ocorrer via exfiltração de metadados internos em nível de aplicação. Um processo malicioso interativo poderia interrogar interfaces de *hardware* locais (`sysfs`) ou tabelas de roteamento para ler diretamente strings contendo o endereço IP real ou o endereço MAC físico do hospedeiro, transmitindo-os de forma textual por dentro do canal TCP legítimo do Tor. Portanto, a integridade da Camada 3 é intrinsecamente dependente do isolamento de *namespaces* de rede e do Controle de Acesso Obrigatório (MAC) implementados na Camada 2, garantindo que aplicações expostas enxerguem apenas adaptadores de rede virtuais e efêmeros.

5. Análise de Vulnerabilidades e Mitigação Proativa

Para compreender a eficácia macroarquitetural do SentinelOS, é necessário utilizar os padrões globais de catalogação de falhas de segurança: o *Common Vulnerabilities and Exposures* (CVE) e o *Common Weakness Enumeration* (CWE). Enquanto o CVE identifica instâncias específicas e datadas de falhas em softwares, o CWE cataloga os tipos de fraquezas de *design*, implementação ou arquitetura que permitem a existência dessas falhas.

A arquitetura em camadas do SentinelOS não visa apenas mitigar CVEs individuais de forma reativa através de remendos de código (*patch management*); a meta central é a invalidação preventiva de classes inteiras de fraquezas estruturais por meio do endurecimento do sistema (*system hardening*). A Tabela 2 resume o contraste conceitual entre a abordagem das distribuições convencionais e a solução proposta.

5.1. Mitigação Detalhada por Camada e Engenharia de Software

Abaixo, dissecam-se a engenharia de segurança de cada uma das quatro macro-camadas do SentinelOS, suportada por casos de estudo e mapeamento explícito de exploits de mercado.

Tabela 2. Mapeamento de Fraquezas (CWE) vs. Estratégia de Defesa do SentinelOS

CWE (Fraqueza Estrutural)	Abordagem Mainstream (Commodity)	Abordagem Proativa (SentinelOS)
CWE-250: Execução com privilégios desnecessários (<i>Execution with Unnecessary Privileges</i>)	Processos e <i>daemons</i> secundários operando com UID 0 (<i>root</i>) para facilitar a compatibilidade.	Isolamento compulsório via <i>Podman Env Builder (rootless)</i> e confinamento MAC via AppArmor.
CWE-119: Restrições impróprias de operações dentro de buffers de memória (<i>Improper Restriction of Operations within the Bounds of a Memory Buffer</i>)	Dependência da <i>glibc</i> , cuja densidade de código legado e otimizações complexas expandem bugs de memória.	Adoção da <i>musl libc</i> , reescrita minimalista sob conformidade POSIX, e aplicação de <i>Hardened Toolchain</i> .
CWE-494: <i>Download of Code Without Integrity Check</i>	Distribuição de binários opacos pré-compilados e acoplamento a cadeias de suprimentos extensas.	Compilação total a partir do código-fonte verificado (<i>Gentoo Portage</i>) com auditoria de <i>hashes</i> .
CWE-200: <i>Exposure of Sensitive Information to an Unauthorized Actor</i>	Protocolo gráfico X11 com barramento global compartilhado e firewalls permissivos de rede local.	Arquitetura cliente-servidor estrita do Wayland e política perimetral <i>default-deny</i> baseada em <i>nftables</i> .

5.1.1. Camada 0: Hardening do Núcleo e Blindagem do Ring 0

A Camada 0 foca no endurecimento estrutural do kernel Linux, atacando diretamente vulnerabilidades de Escalonamento Local de Privilégios (LPE) e injeção hostil de código em modo núcleo. Distribuições *mainstream* utilizam kernels modulares e genéricos para garantir que qualquer periférico funcione sem configuração prévia. Essa conveniência introduz o CWE-440 (Expected Behavior Violation) e o CWE-267 (Privilege Defined Key Policy), pois expõe milhões de linhas de código de drivers não auditados à exploração do espaço do usuário. O SentinelOS desarticula essa superfície de ataque aplicando as blindagens severas no código-fonte do núcleo citadas na seção 4.1

Caso de Estudo CVE-2024-1086 (Netfilter LPE): Esta vulnerabilidade de alta gravidade explorava uma falha do tipo *Use-After-Free* (UAF) no subsistema Netfilter do kernel Linux. O *exploit* comercial baseava-se na capacidade de um usuário comum invocar a chamada de sistema `unprivileged_usersn_clone` para criar um namespace de rede isolado, obtendo ganchos administrativos virtuais para injetar tabelas malformadas e corromper o *heap* do Ring 0. No SentinelOS, este ataque falha na etapa de reconhecimento inicial: a restrição do `sysctl`, mostrada na seção 4.1, barra a criação do namespace, impedindo o acionamento da lógica vulnerável do Netfilter.

5.1.2. Camada 1: Espaço de Usuário Minimalista e Endurecimento da Toolchain

A Camada 1 gerencia as bibliotecas base que intermediam as chamadas de sistema (*syscalls*). O ecossistema *mainstream* repousa sobre a monocultura da *glibc* e do gerenciador monolítico *systemd*. Essa arquitetura expande significativamente o CWE-119 (Memory Buffer Overflow) devido à necessidade de manter retrocompatibilidade com códigos da década de 1990 e à inclusão de mecanismos complexos como o *Name Service Switch* (NSS), que carrega dinamicamente módulos de tradução de nomes (*.so*) em tempo de execução. O SentinelOS redefine o espaço de usuário através do ecossistema Gentoo Hardened, abordado na seção 4.2

Casos de Estudo de Imunidade Estrutural:

- **CVE-2015-0235 (Vetor *GHOST*):** Um estouro de *buffer* baseado em *heap* na função de resolução de nomes `gethostbyname()` da *glibc*. No SentinelOS, a PoC deste exploit (seção 6.6.2) apresenta eficácia reduzida, pois a reimplementação desta função na *musl libc* valida rigorosamente os limites de memória e descarta o código vulnerável herdado da engenharia da GNU.
- **CVE-2023-4911 (Vetor *Looney Tunables*):** Esta falha permitia LPE imediata manipulando a variável de ambiente `GLIBC_TUNABLES` para estourar o indexador de memória do vinculador dinâmico (`ld.so`). O SentinelOS possui bloqueio estrutural passivo contra esta ameaça, dado que o carregador da *musl* não implementa o recurso proprietário de sintonia de alocação (*tunables*) da GNU, ignorando nativamente a variável hostil, como visto na seção 6.6.1.
- **CVE-2024-3094 (Vetor *XZ Utils Backdoor*):** Um ataque sofisticado à cadeia de suprimentos onde um código oculto inserido na biblioteca *liblzma* interceptava chaves criptográficas no servidor SSH (`sshd`). O funcionamento do *payload* malicioso dependia de uma "monocultura de alvos": exigia especificamente ganchos indiretos da *glibc* (*IFUNCs*) e o acoplamento do `sshd` com o mecanismo de ativação de sockets do *systemd*. Ao quebrar essa monocultura através da combinação *musl* + *OpenRC*, o SentinelOS desarticulou o funcionamento do *backdoor*, tornando a ameaça inoperante sem necessitar de vacinas de *software*, como provado na seção 6.6.3.

5.1.3. Camada 2: Contenção de Execução Remota via *Podman Env Builder*

A Camada 2 foca no isolamento das aplicações do usuário (Ring 3), assumindo a premissa de comprometimento iminente, paradigma conhecido na literatura de segurança como *Assume Breach* [Rose et al. 2020]. Em sistemas operacionais convencionais, navegadores web e leitores de PDF rodam diretamente sobre o host e possuem acesso irrestrito de leitura ao diretório home do usuário (**CWE-200**). Caso um atacante explore um zero-day de Execução Remota de Código (RCE) no motor JavaScript do navegador, ele obtém acesso imediato a arquivos de configuração locais, históricos de navegação e chaves PGP privadas.

Para neutralizar a progressão lateral decorrente de uma RCE, o SentinelOS introduz o *Podman Env Builder* (`mkc`). Trata-se de uma ferramenta orquestradora proprietária desenvolvida em Python que automatiza a containerização modular de aplicações do espaço de usuário.

A engenharia de contenção do `mkc` opera sob as seguintes premissas técnicas, dado o cenário de ataque abordado na seção 6.3:

- **Virtualização *Rootless*:** O contêiner é inicializado sem ganchos de superusuário no *host*. O mapeamento de UIDs garante que o ID do processo invasor dentro do contêiner seja traduzido para um UID sem privilégios no sistema hospedeiro, impedindo a quebra de isolamento (*container escape*).
- **Supressão de Escalamento Extra-Contêiner:** O orquestrador injeta compulsoriamente a flag `--no-new-privileges`. Essa diretiva instrui o kernel a bloquear ganchos de transição de contexto, impedindo que processos filhos herdem bits *SUID* ou ganhem novas capacidades (*capabilities*), mitigando a falha CWE-250.
- **Separação Estrita de Namespaces e Confinamento MAC:** O sistema de arquivos do host é montado de forma isolada, impedindo que a aplicação conteinerizada enxergue diretórios confidenciais (como `~/.local/` ou `~/.gnupg/`). Complementarmente, um perfil rígido do AppArmor (*Mandatory Access Control* - *MAC*) monitora o fluxo de chamadas de sistema da aplicação, bloqueando chamadas não homologadas de rede e escrita.

Se o navegador for invadido, o atacante permanece "enjaulado" em um sistema de arquivos efêmero e virtualizado, incapaz de roubar a identidade digital do operador ou modificar o estado persistente da máquina.

5.1.4. Camada 3: Rede Oculta e Controle de Fluxo Stateful

A Camada 3 isola o SentinelOS contra ataques de reconhecimento ativo, mapeamento de portas (*port scanning*), varreduras perimetrais e ataques de injeção local, vulnerabilidades mapeadas sob o CWE-200 e CWE-294 (Authentication Bypass by Catching).

O perímetro de rede é regido por uma postura de Negação Padrão (*Default-Deny*) estruturada através do subsistema `nftables`, associada ao roteamento compulsório via rede Tor de forma transparente, como evidenciado na seção 4.4.

Caso de Estudo CVE-2024-47176 (Exploração Remota do CUPS): Esta cadeia de vulnerabilidades permitia a execução de comandos remotos sem autenticação. O vetor de ataque baseava-se em um atacante enviando um pacote UDP hostil contendo uma URL de impressora maliciosa para a porta 631 do alvo, forçando o serviço CUPS em segundo plano a processar a requisição e baixar um *script* de execução automática. No SentinelOS, este ataque é mitigado por dupla barreira independente: primeiro, o pacote UDP invasor é descartado silenciosamente na interface física pelo firewall perimetral; segundo, devido ao princípio de economia de mecanismos, a imagem mínima do sistema não possui pilhas ou *daemons* de impressão instalados, eliminando o vetor bi-direcional por completo, como mostrado na poc da seção 6.8.

6. Validação e cenários de teste

6.1. Atualizações de segurança

Tendo em vista as constantes correções e novas versões de software que mitigam vulnerabilidades conhecidas, faz-se necessário manter o espaço do usuário e o do kernel atualizados. Para isso, é possível utilizar o ecossistema Gentoo e a ferramenta de automação

make visando garantir a manutenção do SentinelOS. O espaço do usuário pode ser atualizado por meio do `portage`, o gerenciador de pacotes do Gentoo, enquanto o `make` é empregado para recompilar o kernel. Este último processo preserva as configurações de segurança previamente estabelecidas quando o alvo `olddefconfig` é utilizado.

6.2. Cenários de ataque

Na tabela a seguir são demonstrados alguns cenários de ataques e como o SentinelOS reage a eles. Posteriormente os cenários que merecem destaque são detalhados em suas próprias subseções.

Tabela 3. Matriz de Cenários de Ataque e Resiliência Arquitetural do SentinelOS

Cenário de Ataque	Mitigação Ativa do SentinelOS	Resultado Final
1. Execução Remota de Código (RCE) via Navegador O atacante explora uma vulnerabilidade <i>zero-day</i> no navegador web para executar código malicioso e tentar comprometer o sistema operacional subjacente.	O navegador é executado isoladamente por meio do orquestrador <i>Podman Env Builder</i> (<code>mkc</code>). A injeção da <i>flag --no-new-privileges</i> impede o escalonamento, enquanto o <i>AppArmor</i> e os <i>Namespaces</i> bloqueiam o acesso ao sistema de arquivos do <i>host</i> (ex: <code>/home</code> e <code>/etc</code>).	Ataque Contido. O atacante obtém êxito inicial em invadir o processo do navegador, mas fica "enjaulado" no contêiner <i>rootless</i> . Não consegue acessar arquivos pessoais ou comprometer o <i>host</i> .
2. Varredura de Portas e Exploração Direta (Port Scanning) O atacante realiza uma varredura de portas (<i>port scanning</i>) na rede para mapear e explorar serviços vulneráveis em execução no <i>host</i> .	O <i>firewall nftables</i> é configurado com uma postura <i>default-deny</i> estrita na cadeia <i>input</i> . Apenas conexões estabelecidas pelo próprio usuário (<i>stateful inspection</i>) ou pacotes ICMP de diagnóstico são aceitos.	Bloqueio. O atacante consegue descobrir que a máquina existe (via <i>Ping</i>), mas todas as portas TCP/UDP aparecem como filtradas. O ataque morre na camada de rede.
Continua na próxima página...		

Tabela 3 – Continuação da página anterior

Cenário de Ataque	Mitigação Ativa do SentinelOS	Resultado Final
3. Captura de Tela e Keylogging por Malware Local O atacante executa um <i>script</i> malicioso localmente para monitorar as teclas digitadas globalmente (<i>keylogging</i>) e capturar a tela, a fim de roubar credenciais e informações sensíveis.	A adoção exclusiva do protocolo gráfico Wayland elimina o <i>socket</i> global inerente ao servidor X11 (Xorg). A comunicação gráfica ocorre cliente-a-servidor, isolando eventos de <i>input/output</i> de forma nativa.	Bloqueio. O <i>malware</i> não consegue interceptar <i>inputs</i> de outras janelas e obtém apenas uma tela preta caso tente capturar o ambiente gráfico fora do seu escopo.
4. Injeção de Rootkit no Kernel (LKM) O atacante utiliza privilégios de administrador previamente obtidos para injetar um módulo de <i>kernel</i> malicioso (LKM), com o objetivo de ocultar a sua presença e garantir persistência no sistema.	O <i>kernel</i> do SentinelOS é compilado de forma monolítica (sem suporte a Loadable Kernel Modules - LKM) e com a imposição do Lockdown LSM no modo integridade, restringindo acessos ao <i>/dev/mem</i>.	Bloqueio. Mesmo com acesso <i>root</i>, o invasor é impedido pelo próprio <i>kernel</i> de alterar sua memória em tempo de execução ou injetar módulos externos.
5. Exploração de Vulnerabilidades Legadas (Ex: GHOST) O atacante executa um binário malicioso para explorar falhas de <i>buffer overflow</i> em funções desatualizadas (como a <i>gethostbyname</i>), a fim de obter execução arbitrária de código.	O sistema repousa sobre a biblioteca base musl libc, que foi reescrita com foco em minimalismo e conformidade POSIX, descartando o código histórico e complexo mantido por retrocompatibilidade.	Bloqueio Estrutural. O ataque falha instantaneamente por inexistência do vetor de ataque. O código vulnerável da <i>glibc</i> não existe no sistema base.
<i>Continua na próxima página...</i>		

Tabela 3 – Continuação da página anterior

Cenário de Ataque	Mitigação Ativa do SentinelOS	Resultado Final
6. Exfiltração de Dados com Conexão C2 (<i>Reverse Shell</i>) e anonimização O atacante utiliza um <i>malware</i> alojado em segundo plano para estabelecer uma conexão reversa (<i>Reverse Shell</i>) com um servidor de Comando e Controle (C2), visando exfiltrar dados.	O <i>firewall</i> permite tráfego de saída (<i>output accept</i>) para usabilidade, logo a conexão passa. Contudo, todo o tráfego da rede é roteado compulsoriamente através dos túneis de 3 saltos da rede Tor.	Êxito Parcial. O <i>malware</i> consegue contatar o C2, mas a origem física do sistema (<i>IP real</i> do usuário) permanece oculta do servidor malicioso devido à criptografia telescópica do Tor.
7. Comprometimento Físico de Firmware (<i>Evil Maid Attack</i>) O atacante aproveita o acesso físico momentâneo à máquina para injetar um <i>rootkit</i> diretamente no <i>firmware</i> (chip SPI/UEFI), com o intuito de comprometer o sistema antes da inicialização do SO.	O SentinelOS foca estritamente nas camadas de anéis de proteção superiores (Ring 0 e Ring 3), aplicando <i>hardening</i> no <i>kernel</i> e espaço do usuário, mas não gerencia a integridade do <i>hardware</i> em si.	Êxito do Atacante. O ataque ocorre no "Ring -1". Como o <i>firmware</i> carrega antes do sistema operacional, todas as barreiras do SentinelOS e do <i>kernel</i> são invisíveis ao <i>hardware</i> comprometido.
8. Engenharia Social (Entrega voluntária de privilégios) O atacante emprega técnicas de engenharia social para persuadir o usuário a fornecer a senha de superusuário e executar um <i>script</i> fora do <i>sandbox</i>, a fim de subverter as defesas do sistema.	Embora políticas estritas de <i>AppArmor</i> possam limitar ações catastróficas, o consentimento do superusuário contorna diretamente o modelo MAC e desfaz o Controle de Acesso Discrecional (DAC).	Êxito do Atacante. As defesas arquiteturais colapsam. O SentinelOS mitiga falhas de <i>software</i>, mas não pode impedir decisões ativamente maliciosas autorizadas pelo administrador legítimo da máquina.
Continua na próxima página...		

Tabela 3 – Continuação da página anterior

Cenário de Ataque	Mitigação Ativa do SentinelOS	Resultado Final
9. Backdoor na Compressão XZ Utils (CVE-2024-3094) O atacante injeta uma <i>backdoor</i> na biblioteca de compressão <code>liblzma</code> (XZ Utils) para interceptar autenticações SSH e obter Execução Remota de Código (RCE).	O <i>payload</i> malicioso foi desenhado para explorar a monocultura de <i>software</i>: dependia intrinsecamente de extensões exclusivas da <code>glibc</code> (<i>IFUNCs</i>) e da interligação do <code>sshd</code> com o <code>systemd</code>. O SentinelOS utiliza <code>musl libc</code> e descarta dependências acopladas.	Bloqueio Estrutural. O ataque falha por incompatibilidade arquitetural. A diversidade do sistema (<code>musl</code>) e o minimalismo neutralizam passivamente ameaças complexas criadas para a maioria dos sistemas operacionais de <i>commodity</i>.
10. Escalonamento via Looney Tunables (CVE-2023-4911) O atacante manipula a variável de ambiente <code>GLIBC_TUNABLES</code> para acionar um <i>buffer overflow</i> no carregador dinâmico (<code>ld.so</code>), com o objetivo de escalar privilégios localmente para superusuário.	A falha reside na análise de variáveis do vinculador dinâmico estritamente presente no ecossistema da <code>glibc</code>. A abstração do espaço de usuário no SentinelOS foi inteiramente migrada e recompilada contra a <code>musl libc</code>.	Bloqueio Estrutural. Ataque anulado por inexistência do vetor. Demonstra o valor prático da redução da superfície de ataque através da adoção de código mais limpo, enxuto e rigoroso aos padrões POSIX.
11. Exploração Remota do Sistema de Impressão (CVE-2024-47176) O atacante envia pacotes UDP maliciosos (porta 631) para o serviço CUPS com o objetivo de registrar impressoras falsas e forçar a execução de comandos remotos na máquina alvo.	O princípio de economia de mecanismos previne a execução de <i>daemons</i> redundantes (<i>bloatware</i>) em segundo plano. Além disso, a arquitetura de negação padrão (<i>default-deny</i>) na cadeia <i>input</i> do <i>firewall</i> detém tráfego de entrada não solicitado.	Bloqueio. Dupla mitigação: o tráfego UDP malicioso é ativamente bloqueado pelo <code>nftables</code> e, mesmo que a rede local fosse confiável, a ausência de pilhas de serviços de impressão ativas previne o processamento.
<i>Continua na próxima página...</i>		

Tabela 3 – Continuação da página anterior

Cenário de Ataque	Mitigação Ativa do SentinelOS	Resultado Final
12. LPE no Sub-sistema Netfilter do Kernel (CVE-2024-1086) O atacante utiliza um <i>malware</i> alojado no espaço de usuário para explorar uma falha <i>Use-After-Free</i> (UAF) nas tabelas de roteamento do <i>kernel</i> , a fim de escalar privilégios (LPE) para <i>root</i> .	O ataque requer que o processo malicioso manipule configurações de rede. Quando contido pelo <i>Podman Env Builder</i> , a ausência da <i>capability</i> <code>CAP_NET_ADMIN</code> (privilégio mínimo) e o confinamento por espaços de nomes (<i>namespaces</i>) restringem este acesso. Quando executado diretamente sobre o host a restrição de clones de <i>namespaces</i> unprivileged, detalhada na seção 4.1 barra o ataque.	Ataque Contido. O ataque falha mesmo sendo executado diretamente sobre o host por conta do kernel endurecido do SentinelOS

Para validar a robustez da matriz apresentada na Tabela 3, o sistema foi submetido a testes controlados em ambiente de laboratório isolado. Utilizou-se uma máquina atacante executando ferramentas de auditoria padrão de mercado, com o objetivo de confrontar cada uma das quatro macro-camadas do SentinelOS e registrar empiricamente as respostas defensivas do núcleo e do espaço do usuário.

6.3. Contenção de Execução Remota de Código (RCE) via Navegador Web

O primeiro cenário simulou o comprometimento completo do navegador web do sistema operacional através de um ataque do tipo *Drive-by Download* ⁶ ou exploração de uma vulnerabilidade *zero-day* de corrupção de memória no motor JavaScript. O objetivo do teste foi avaliar a capacidade do SentinelOS de conter a progressão do ataque dentro do domínio isolado da aplicação, impedindo que o invasor acesse o sistema de arquivos do hospedeiro (*host*) ou altere os privilégios do processo.

- **Código de Simulação do Vetor de Ataque (`poc_rce.sh`):** Para emular o ganho de controle do processo do navegador por um atacante remoto sem a necessidade de instanciar um *exploit* comercial complexo, desenvolveu-se um *script* que é injetado diretamente no escopo de execução da aplicação. O *script* tenta ler arquivos sensíveis do usuário e modificar permissões do sistema:

```
#!/bin/sh
echo "[+] RCE executado com sucesso no contexto do processo
."
echo "[*] Tentando ler arquivos confidenciais do host..."
cat /home/user/.gnupg/secring.gpg 2>/dev/null || echo "[-]
Falha ao ler chaves PGP."
cat /etc/shadow 2>/dev/null || echo "[-] Falha ao ler /etc/
shadow."
```

⁶Drive-by Download ocorre quando um código malicioso é baixado e instalado no seu dispositivo de forma automática e silenciosa. A infecção acontece simplesmente quando você visita um site comprometido ou malicioso, sem que você precise clicar em nenhum botão de download ou aprovar a instalação.

```
echo "[*] Tentando escalar privilegios via bit SUID..."
chmod +s /bin/sh 2>/dev/null || echo "[-] Falha ao aplicar
bit SUID."
```

- **Procedimento e Comando Executado:** No SentinelOS, a execução do navegador web pode operar sob diferentes camadas de abstração conforme a arquitetura desejada pelo operador, alternando entre o empacotamento *Flatpak* ou o encapsulamento nativo via ecossistema *Podman Env Builder* (`mkc`) sob restrições estritas de privilégio mínimo.

Para mitigar a complexidade operacional e reduzir a necessidade de digitação manual de extensas cadeias de parâmetros diretamente no terminal a cada execução, a CLI disponibiliza a diretiva `profile`. Este comando abstrai a infraestrutura como código local, viabilizando o agrupamento declarativo de diretivas de segurança que podem ser injetadas opcionalmente tanto na etapa de compilação (`build`) quanto na de inicialização (`run`) do contêiner. O provisionamento da matriz de segurança do navegador isolado é instanciado conforme a sintaxe abaixo:

```
mkc profile -n navegador_isolado -- "--security-opt
apparmor=podman-navegador --no-new-privileges --cap-drop
=ALL --net=slirp4netns"
```

Uma vez persistido o perfil declarativo em disco, o processo de compilação da imagem customizada e estruturação das camadas do sistema de arquivos é disparado invocando o módulo de construção, especificando os requisitos gráficos (`--gui`) e atrelando as restrições previamente salvas:

```
mkc build --gui -n navegador_isolado --profile
navegador_isolado
```

Finalizado o protocolo de instalação, o ciclo de execução do ambiente torna-se transparente para o operador do sistema. O contêiner pode ser inicializado de forma determinística chamando o atalho gerado diretamente no terminal ou invocando-o visualmente através do *wmenu* (o menu dinâmico de aplicações focado no ecossistema Wayland integrado ao SentinelOS).

- **Análise Anatômica das Diretivas de Defesa:** A contenção deste cenário baseia-se na sobreposição e coordenação de três mecanismos de isolamento configurados na Camada 2, cujos parâmetros são processados dinamicamente pelo orquestrador do SentinelOS:
 - **Virtualização *Rootless*:** O contêiner do navegador roda mapeando o UID do usuário interno para um UID não privilegiado na *host*. Mesmo que o atacante vire *root* dentro do contêiner, ele permanece sendo um usuário comum sem direitos no sistema real.
 - **Flag `--no-new-privileges`:** Esta diretiva, injetada pelo orquestrador do SentinelOS, impede que o processo ou qualquer um de seus filhos ganhe novos privilégios através da execução de binários com bit *SUID* ou *SGID* ativo.
 - **Perfil Customizado do AppArmor (`--security-opt apparmor=podman-navegador`):** Esta flag instrui o motor do

Podman a abdicar de seu perfil de confinamento genérico e carregar um domínio de segurança restritivo e dedicado, instanciado no subsistema do *Kernel*. O perfil atua aplicando o Controle de Acesso Obrigatório (*Mandatory Access Control* - *MAC*) sobre o processo. Na prática, ele monitora em tempo real as chamadas de sistema (*syscalls*) disparadas pelo navegador e veda o acesso a diretórios externos ao escopo da aplicação. Para implementar e validar esse confinamento, o ciclo de criação e ativação do perfil obedece aos seguintes passos estruturais no SentinelOS:

1. **Escrita da Declaração de Políticas:** Cria-se um arquivo de texto puro em `/etc/apparmor.d/podman-navegador` definindo explicitamente as permissões do binário. A sintaxe utiliza regras declarativas baseadas em permissões explícitas de leitura (*r*), escrita (*w*) e execução (*ix*), associadas ao bloqueio padrão de montagens (*mount*) e capacidades administrativas (*capability*), conforme o modelo simplificado abaixo:

```
# /etc/apparmor.d/podman-navegador
profile podman-navegador flags=(
    attach_disconnected,mediate_deleted) {
    # Inclui definicoes base do sistema
    #include <abstractions/base>

    # Nega categoricamente acesso a pastas
    criticas de chaves e dados
    deny /home/user/.gnupg/** rwlkm,
    deny /etc/shadow r,

    # Permite apenas leitura e escrita no buffer
    efemero do container
    /home/user/.local/share/containers/** rwk,
}
```

2. **Compilação e Carregamento no Núcleo:** Com o arquivo de política estruturado, o script de inicialização do sistema operacional compila a regra declarativa em bytecode de baixo nível e a injeta diretamente na memória viva do *Kernel* Linux através do utilitário oficial de gerenciamento do AppArmor:

```
apparmor_parser -r -W /etc/apparmor.d/podman-
navegador
```

3. **Vinculação em Tempo de Execução:** Uma vez que o perfil reside na tabela de segurança ativa do Ring 0, a flag passada pela CLI `mkc` associa o identificador de segurança (*security context*) do processo do contêiner ao nome do perfil carregado, forçando o isolamento definitivo avaliado no *dmesg*.
- **Saída Esperada (Ambiente Vulnerável):** Em uma distribuição *mainstream* convencional onde o navegador é executado diretamente no host sem perfis de confinamento MAC ativo, o *exploit* RCE obtém acesso irrestrito ao diretório `/home` do usuário. O atacante consegue clonar chaves criptográficas PGP, ler históricos de

navegação, roubar tokens de sessão e, caso utilize uma vulnerabilidade LPE adjacente, consegue alterar arquivos estruturais em `/etc` para garantir persistência definitiva no sistema.

- **Saída Obtida no SentinelOS:** A execução do comando de intrusão resultou no bloqueio imediato e controlado de todas as ações hostis, exibindo o seguinte log no terminal do auditor:

```
[+] RCE executado com sucesso no contexto do processo.  
[*] Tentando ler arquivos confidenciais do host...  
[-] Falha ao ler chaves PGP.  
[-] Falha ao ler /etc/shadow.  
[*] Tentando escalar privilegios via bit SUID...  
[-] Falha ao aplicar bit SUID.
```

A inspeção dos logs de auditoria do sistema (`dmesg`) registrou os alertas de negação gerados ativamente pelo AppArmor:

```
apparmor="DENIED" operation="open" profile="podman-  
navegador"  
name="/home/user/.gnupg/secring.gpg" requested_mask="r"  
denying_mask="r"
```

O ensaio valida que, embora o atacante tenha obtido o *Êxito Inicial* em corromper a memória do navegador (Camada 2), o impacto do ataque foi mitigado de forma integral. O invasor permaneceu “enjaulado” em um ambiente de arquivos virtualizado e efêmero, incapaz de exfiltrar segredos do operador ou de comprometer a integridade do sistema operacional hospedeiro, cancelando a eficácia da abordagem de isolamento proativo do SentinelOS.

- **Consequências Práticas do Confinamento em Contêiner:** A materialização desse ensaio consolida os impactos arquiteturais decorrentes da execução do navegador web em um ambiente efêmero controlado pela ferramenta `mkc`, resumidos em três frentes:

1. **Negação de Persistência Hostil:** Como o sistema de arquivos exposto ao navegador é uma camada sobreposta e isolada pelo Podman, quaisquer binários maliciosos baixados, *scripts* de inicialização injetados ou modificações em arquivos de perfil (como o `.bashrc` interno do contêiner) são descartados assim que a instância é encerrada. O atacante perde a capacidade de estabelecer persistência de longo prazo na máquina hospedeira.
2. **Esterilização de Dados de Aplicação:** Caso o invasor consiga comprometer o processo do navegador e tente inspecionar a memória local ou os arquivos adjacentes para extrair credenciais, ele encontrará apenas o escopo estéril e virtualizado do contêiner. Arquivos de configuração globais do hospedeiro, chaves criptográficas privadas e dados de outras aplicações tornam-se invisíveis e inacessíveis, mitigando a exfiltração de identidades digitais.
3. **Invalidação de Ataques de Quebra de Escopo (*Container Escape*):** A coordenação mandatária entre a execução *rootless* (mapeamento de UIDs) e o perfil estrito do AppArmor garante que, mesmo se o *exploit* RCE obti-

ver sucesso em comprometer o motor de renderização da aplicação, o invasor estará sujeito às restrições do Controle de Acesso Obrigatório (MAC) impostas diretamente pelo núcleo do sistema operacional (Ring 0). A tentativa de disparar ganchos de execução administrativa extra-contêiner resulta em falha imediata por ausência de privilégios ou barramento de chamadas de sistema (*syscalls*), neutralizando a progressão lateral do ataque.

6.4. Validação Experimental da Camada 3: Resiliência contra Varredura de Portas e Reconhecimento Ativo

O segundo cenário experimental teve como objetivo avaliar a eficácia do perímetro de rede do SentinelOS quando submetido a ataques de reconhecimento ativo e mapeamento de superfície. O modelo de ameaça assume que um atacante posicionado na mesma rede local (LAN) tentará identificar portas de comunicação abertas e serviços ativos no hospedeiro para, subsequentemente, planejar uma exploração direcionada.

Para a execução do ensaio, configurou-se um ambiente de laboratório isolado contendo uma estação atacante executando a ferramenta de auditoria *Nmap* e o SentinelOS como o alvo da intrusão. O procedimento experimental seguiu as etapas descritas a seguir:

1. **Varredura Conclusiva de Portas (*Port Scanning*):** A partir do vetor de ataque, disparou-se uma varredura do tipo *SYN Stealth Scan*⁷ abrangendo a totalidade das portas de transporte possíveis (65.535 portas TCP), utilizando a flag de supressão de requisições de eco (`-Pn`) para mascarar a assinatura inicial do escaneamento. O comando operacionalizou-se através da sintaxe: `nmap -sS -p--Pn [IP_ALVO]`.
2. **Monitoramento Passivo de Descarte:** Paralelamente, no SentinelOS, utilizou-se o utilitário `tcpdump` e a leitura do *buffer* de mensagens do *kernel* (`dmesg`) para inspecionar a chegada dos pacotes IP na interface de rede e analisar o comportamento da pilha de protocolos do sistema operacional.

Os resultados empíricos demonstraram a robustez da política *default-deny* estruturada no `nftables`. No terminal da máquina atacante, a totalidade das 65.535 portas TCP foi categorizada pelo *Nmap* sob o estado estrito de *filtered* (filtradas). Na perspectiva do SentinelOS, a análise dos pacotes confirmou que os *frames* síncronos (*SYN packets*) enviados pelo agressor não receberam nenhuma resposta de retorno (*SYN-ACK* ou *RST*).

Em vez de rejeitar ativamente a conexão — o que confirmaria a existência do host ativo e forneceria metadados sobre a pilha de rede —, o `nftables` instruiu o *kernel* a descartar silenciosamente (*drop*) os pacotes na cadeia *input*. Esse comportamento atenua varreduras externas à estação de trabalho, mitigando o mapeamento perimetral e interrompendo a progressão do ataque antes que qualquer falha de nível de aplicação possa ser exposta.

⁷SYN Stealth Scan é uma forma de mapeamento de portas (port scan) usado para descobrir quais serviços estão ativos em um servidor. Ele é conhecido como um scan "semiaberto" porque envia pacotes de requisição, mas nunca conclui o aperto de mãos (handshake) do TCP.

6.5. Validação Experimental da Camada 1: Isolamento de Interface Gráfica contra *Keylogging* e Captura de Tela

O terceiro cenário de teste buscou validar a eficácia do isolamento de processos no domínio da interface gráfica do SentinelOS. O modelo de ameaça avaliado compreende um cenário de pós-infecção, onde um agente malicioso, já em execução local no espaço do usuário (*Ring 3*), tenta realizar a interceptação global de eventos de teclado (*keylogging*) e a captura não autorizada da tela (*screen scraping*) para exfiltrar credenciais, chaves criptográficas ou dados sensíveis digitados em outras janelas legítimas.

Para a reprodução experimental do vetor de ataque, utilizou-se o ambiente gráfico padrão do SentinelOS baseado no compositor *Sway* (protocolo *Wayland*). O ensaio foi dividido em duas frentes de ataque simultâneas para simular ferramentas de monitoramento clássicas:

1. **Simulação de Interceptação de Entrada (*Input Sniffing*):** Executou-se um *script* que tentou invocar a ferramenta legada *xinput* para escutar eventos de barramento gráfico.
2. **Simulação de Captura de Tela Externa (*Screen Scraping*):** Forçou-se a execução de um utilitário de captura baseado no servidor X11 legado (através da camada de compatibilidade *XWayland*), disparando o comando de captura de janela raiz: `import -window root prova_conceito.png`.

Os resultados experimentais demonstraram o colapso dos vetores de ataque devido às restrições arquiteturais da Camada 1. No primeiro ensaio, a ferramenta *xinput* falhou ao tentar registrar um gancho (*hook*) global de eventos, pois o *Wayland* não possui um barramento de entrada compartilhado entre clientes.

No segundo ensaio, o utilitário baseado em X11 falhou em capturar o contexto gráfico das aplicações adjacentes (como o terminal ou gerenciador de senhas). Devido à arquitetura cliente-servidor estrita do *Wayland*, onde o compositor renderiza cada janela em um *buffer* de memória isolado e privado, o processo invasor obteve apenas o retorno de uma imagem opaca (*blank screen*) ou um erro de violação de contexto.

Diferente do ecossistema *Xorg*, onde qualquer aplicação possui acesso irrestrito ao mapa de bits da tela e ao fluxo global de teclas, o SentinelOS provou garantir o isolamento eficaz da interface. Esse resultado valida empiricamente que, mesmo sob a execução de um código hostil local, as credenciais e o ambiente visual do usuário permanecem confinados e inacessíveis a processos fora de seu domínio explícito.

6.6. Validação Experimental da Camada 1: Imunidade Estrutural contra Vetores de Ataque da *glibc* e do *systemd*

Os cenários 5, 9 e 10 estão diretamente relacionados à superfície de ataque estendida da biblioteca padrão C e do sistema de inicialização (*init*) das distribuições *mainstream*. Os ensaios experimentais avaliaram o impacto prático da diversidade de *software* e da redução da superfície de ataque obtidas pela substituição da biblioteca padrão do projeto GNU (*glibc*) pela implementação minimalista *musl libc*, além da adoção de um sistema de *init* mais enxuto, o *OpenRC*, em detrimento do *systemd*.

O modelo de ameaça compreende ataques baseados na exploração de vulnerabilidades críticas históricas e contemporâneas que visam a Execução Remota de Código

(RCE) ou o Escalonamento Local de Privilégios (LPE) para o nível de superusuário (*root*). Embora tais falhas já tenham sido corrigidas analiticamente por meio de atualizações em sistemas convencionais, o propósito destes ensaios é demonstrar a resiliência arquitetural do ecossistema do SentinelOS. Para isso, realizou-se uma validação baseada em testes de incompatibilidade de vetores e análise de vinculação dinâmica, detalhados a seguir.

6.6.1. Ensaio 1: Escalonamento Local via *Looney Tunables* (CVE-2023-4911)

O ataque baseado na falha *Looney Tunables* explora um estouro de *buffer* baseado em *heap* no carregador dinâmico (*ld.so*) da *glibc* quando este processa a variável de ambiente *GLIBC_TUNABLES*. O objetivo do teste foi verificar se o vinculador do SentinelOS processaria ou colapsaria diante de argumentos maliciosamente formatados.

- **Procedimento e Comando Executado:** A partir de uma sessão de usuário comum não privilegiado, disparou-se uma tentativa de estouro do indexador do carregador através do comando:

```
env -i GLIBC_TUNABLES="glibc.malloc.mxfast=shmmx" /bin/sh -c "whoami"
```

Adicionalmente, inspecionou-se a árvore de dependências binárias dos utilitários centrais utilizando o comando:

```
ldd /bin/sh
```

- **Análise do Vektor de Teste:** O comando executa o interpretador em um ambiente controlado para isolar o comportamento do vinculador:
 - **env -i:** O utilitário *env* com a flag *-i* (*ignore-environment*) limpa completamente todas as variáveis de ambiente herdadas da sessão do usuário (como *\$PATH* e *\$USER*), garantindo que o programa subsequente inicie em um estado limpo.
 - **GLIBC_TUNABLES:** Uma extensão proprietária da *glibc* projetada para permitir o ajuste fino de parâmetros da biblioteca em tempo de execução, sem necessidade de recompilação.
 - **"glibc.malloc.mxfast=shmmx":** O argumento tenta sintonizar o tamanho máximo dos blocos de alocação rápida de memória (*fastbins*). A introdução proposital de caracteres textuais misturados ao símbolo de igual (=) foi projetada para engatilhar um erro de lógica no código de varredura (*parsing*) do carregador dinâmico do Linux.
 - **/bin/sh -c "whoami":** Invoca o interpretador de comandos padrão para executar a string de verificação de identidade logo após a inicialização do ambiente.
- **Saída Esperada (Ambiente Vulnerável):** Em um sistema *mainstream* afetado que utilize a *glibc*, o vinculador dinâmico falha ao calcular o tamanho da string contendo o caractere =, realizando uma cópia de dados para um espaço de memória menor do que o necessário. Este estouro de *buffer* corrompe os ponteiros de privilégio do processo. Como o carregador gerencia binários com bits de permissão elevada ativos (*SUID*), a exploração resulta em uma falha de segmentação (*Segmentation Fault*) ou na geração de um *shell* interativo com UID 0 (*root*), permitindo o controle total da máquina.

- **Saída Obtida no SentinelOS:** O interpretador executou o comando normalmente e retornou o nome do usuário comum, ignorando por completo a variável injetada. A saída do comando `ldd /bin/sh` exibiu o seguinte mapeamento estrutural:

```
/lib/ld-musl-x86_64.so.1 (0x7f8a3c...)
libc.musl-x86_64.so.1 => /lib/ld-musl-x86_64.so.1 (0
x7f8a3c...)
```

Como o SentinelOS substituiu a biblioteca do ecossistema GNU, o vinculador dinâmico nativo (`ld-musl`) desconhece completamente o recurso de *tunables* e ignora a variável de ambiente passivamente. O ataque foi inteiramente neutralizado na raiz por inexistência do vetor de ataque no código-fonte, sem a necessidade de intervenção ativa de softwares de detecção.

6.6.2. Ensaio 2: Exploração de Vulnerabilidades Legadas via Vetor *GHOST* (CVE-2015-0235)

A vulnerabilidade histórica *GHOST* envolve um estouro de *buffer* baseado em *heap* na função `_nss_hostname_digits_dots()`, acionada pelas funções de resolução de nomes `gethostbyname()` e `gethostbyname2()` da `glibc`. O objetivo deste teste foi avaliar o comportamento do SentinelOS ao executar um binário que invoca deliberadamente este vetor de estouro.

- **Código da Prova de Conceito (`poc_ghost.c`):** O programa em C foi estruturado para alocar um argumento e passá-lo diretamente para a função de resolução de rede, simulando o gatilho exato do *exploit*:

```
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define BUFFER_SIZE 1024

int main(void) {
    struct hostent *result;
    int herr;
    // Aloca um buffer preenchido com caracteres numericos e
    pontos
    char *dummy_host = malloc(BUFFER_SIZE);
    if (!dummy_host) return 1;

    memset(dummy_host, '0', BUFFER_SIZE - 1);
    dummy_host[BUFFER_SIZE - 1] = '\\0';

    // Dispara a funcao vulneravel da glibc
    result = gethostbyname(dummy_host);

    if (result == NULL) {
        printf("Rejeicao segura ou host nao encontrado.\\n");
    }
}
```

```

    } else {
        printf("Comportamento inesperado/vulneravel.\n");
    }

    free(dummy_host);
    return 0;
}

```

- **Procedimento e Comando Executado:** Compilou-se o código-fonte acima utilizando o compilador nativo do sistema e executou-se o binário resultante no ambiente do SentinelOS:

```

gcc -o poc_ghost poc_ghost.c
./poc_ghost

```

- **Análise Anatômica do Vetor de Teste:** A PoC explora uma falha no cálculo de tamanho necessário para o alinhamento de memória interna:
 - **malloc(BUFFER_SIZE):** Aloca um espaço controlado no *heap* para armazenar a string hostil.
 - **memset(dummy_host, '0', ...):** Preenche o vetor com caracteres numéricos simulando um endereço IP artificialmente gigantesco. O gatilho específico da falha exige que o argumento comece com um dígito e termine com um ponto, contornando filtros superficiais.
 - **gethostbyname():** Função obsoleta responsável por converter nomes de domínio em endereços IP. Na *glibc*, ela aciona internamente o mecanismo modular do NSS.
- **Saída Esperada (Ambiente Vulnerável):** Em sistemas obsoletos que utilizam versões vulneráveis da *glibc*, a função interna falha ao calcular o tamanho do *buffer* de destino para a resolução dos dígitos. O tamanho real da string estoura os limites do espaço reservado no *heap*, sobrescrevendo metadados de alocação adjacentes. Isso resulta no desvio do fluxo de execução do binário (comunicação de estado *vulnerable*) ou no encerramento forçado do programa com um erro de falha de segmentação (*Segmentation Fault*).
- **Saída Obtida no SentinelOS:** O processo foi compilado normalmente e, ao ser executado, imprimiu imediatamente a mensagem em tela:

```

Rejeicao segura ou host nao encontrado.

```

A *musl libc* foi desenvolvida com foco em segurança estrutural e conformidade rigorosa aos padrões POSIX modernos, omitindo completamente a arquitetura modular complexa do *Name Service Switch* (NSS) que dava origem à função interna defeituosa da *glibc*. Como a reimplementação da função *gethostbyname()* na *musl* valida estritamente os limites de memória antes de processar strings, a carga maliciosa foi tratada de forma estéril e segura, provando o bloqueio estrutural por diversidade de código.

6.6.3. Ensaio 3: Intercepção de Cadeia de Suprimentos via XZ Utils (CVE-2024-3094)

O *backdoor* inserido na biblioteca de compressão `liblzma` (XZ Utils) foi projetado para interceptar o fluxo de autenticação do servidor SSH (`sshd`). O código malicioso dependia de funcionalidades específicas do vinculador dinâmico da `glibc` (*IFUNCs*) e do acoplamento do `sshd` com o `systemd` através do mecanismo de *socket activation*. O teste buscou auditar a capacidade de persistência desse ecossistema hostil no SentinelOS.

- **Procedimento e Comando Executado:** Executou-se uma varredura de símbolos no binário do servidor SSH e nas bibliotecas de transporte para rastrear funções de carregamento indireto e dependências de serviços, utilizando os comandos:

```
readelf -W symbols /usr/sbin/sshd | grep -i "
    gnu_indirect_function"
rc-status --all | grep -i "systemd"
```

- **Saída Esperada (Ambiente Vulnerável):** Em distribuições *mainstream* afetadas, o vinculador dinâmico resolve as funções indiretas (*IFUNCs*) injetando o gancho malicioso em tempo de execução, enquanto o `systemd` inicializa o serviço expondo o socket modificado para a rede externa.
- **Saída Obtida no SentinelOS:** O comando `readelf` retornou uma saída vazia, confirmando que a `musl libc` não possui suporte ou código nativo para gerenciar *IFUNCs*. Paralelamente, o comando `rc-status` confirmou que o gerenciador de inicialização ativo é o OpenRC, contendo apenas *scripts* de texto simples legíveis e sem qualquer vínculo com serviços monolíticos:

```
Dynamic Services
  openrc-settingsd ... [ started ]
Runlevel: default
  sshd ... [ started ]
```

Como o mecanismo do *backdoor* exige a monocultura do ecossistema `glibc` + `systemd`, a arquitetura minimalista e distinta do SentinelOS desarticulou o ataque passivamente por incompatibilidade estrutural.

6.7. Validação Experimental da Camada 3: Anonimização e Mitigação de Exfiltração de Dados via Conexão de Comando e Controle (C2)

O sexto cenário experimental avaliou o comportamento perimetral do SentinelOS diante de uma intrusão bem-sucedida em nível de aplicação, onde um agente malicioso tenta estabelecer uma conexão reversa (*Reverse Shell*) com um servidor externo de Comando e Controle (C2) para fins de exfiltração de dados. O modelo de ameaça assume que, embora o perímetro de entrada (*input*) esteja blindado pelo `nftables`, uma aplicação autorizada (como o navegador contêinerizado) foi corrompida e disparou um fluxo de saída (*output*) em direção à internet.

Para simular este vetor de ataque de forma controlada e avaliar a persistência do isolamento, estruturou-se um ensaio utilizando um servidor externo sob controle do auditor, atuando como o receptor C2, e a estação SentinelOS. O experimento seguiu a seguinte metodologia:

1. **Instanciação do Receptor C2:** No servidor remoto externo, inicializou-se o utilitário `netcat` em modo de escuta ativa na porta TCP 4444, através do comando: `nc -lvp 4444`.
2. **Disparo da Conexão Reversa:** No ambiente do SentinelOS, executou-se um comando simulando o *payload* do *malware* para forçar o redirecionamento do interpretador de comandos (`sh`) para o IP público do servidor C2.
3. **Auditoria de Subversão de Regras (Tentativa de Desativação do Proxy):** Uma vez estabelecido o acesso interativo no terminal do receptor C2, executaram-se comandos maliciosos para tentar desativar as tabelas de roteamento do firewall e paralisar o daemon do Tor, simulando um atacante tentando burlar o proxy transparente:

```
nft flush ruleset
rc-service tor stop
```

4. **Inspeção e Rastreamento de Tráfego:** Capturaram-se os *logs* de conexão no servidor receptor para registrar o endereço IP de origem do fluxo de dados e o nível de privilégio do *shell* obtido.

Os resultados experimentais evidenciaram o funcionamento correto do Proxy Transparente e a eficácia da segregação de privilégios implementada no SentinelOS. No servidor C2 externo, a conexão reversa foi estabelecida com sucesso, permitindo o recebimento de dados (caracterizando um *Êxito Parcial* do ataque em nível de conectividade de aplicação). O comando `whoami` executado dentro do canal reverso confirmou que o atacante operava sob o contexto restrito de um usuário comum sem privilégios.

Durante a etapa de tentativa de subversão, as ações do atacante para desativar o proxy falharam. Tanto o comando `nft flush ruleset` quanto o `rc-service tor stop` retornaram a mensagem padrão do sistema:

```
Permission denied (ou Operation not permitted)
```

Como o *payload* hostil herdou os privilégios restritos do processo vulnerável do espaço do usuário (*Ring 3*), ele foi categoricamente impedido pelo mecanismo de Controle de Acesso Discrecional (DAC) do Linux de alterar estruturas de rede controladas pelo *Ring 0* (Kernel). Ademais, o perfil do *AppArmor* associado ao processo bloqueou nativamente ganchos de administração de rede (`CAP_NET_ADMIN`).

A análise do endereço IP de origem registrado no servidor C2 demonstrou que o fluxo permaneceu sendo roteado por um nó de saída (*exit node*) criptográfico pertencente à rede Tor. Esse comportamento comprova que, mesmo sob ataque de conexão reversa, as regras de DNAT no `nftables` se mantiveram intactas e invioláveis.

Dessa forma, o experimento valida que o SentinelOS neutraliza a capacidade do atacante de realizar o rastreamento geográfico, o *fingerprinting* da rede ou desativar os controles de anonimato do usuário. Mesmo sob um cenário de infecção ativa, o tráfego permanece envelopado na criptografia telescópica de três saltos, garantindo a negação de metadados cruciais devido à incapacidade do invasor de subverter as regras do firewall perimetral.

6.8. Validação Experimental da Camada 1 e Camada 3: Resiliência contra a Exploração Remota do Sistema de Impressão (CUPS)

O sexto cenário experimental teve como propósito validar a resiliência combinada do SentinelOS diante de ataques direcionados a serviços de rede locais redundantes, utilizando como caso de estudo a cadeia de vulnerabilidades críticas descoberta no sistema de impressão CUPS (Common UNIX Printing System), catalogada sob o CVE-2024-47176. O modelo de ameaça prevê um atacante posicionado na rede local que envia pacotes UDP maliciosos para a porta 631, com o objetivo de forçar o registro automatizado de uma impressora hostil e disparar a execução remota de código (RCE) no computador alvo sem a interação do usuário.

Para a verificação empírica deste vetor de ameaça, o ensaio de laboratório foi estruturado em duas etapas distintas de auditoria e ataque:

1. **Simulação de Injeção de Pacote UDP (*Packet Injection*):** A partir de uma estação agressora na mesma sub-rede, disparou-se um pacote UDP customizado contendo a sequência de comandos do *exploit* direcionada para o IP do SentinelOS na porta de escuta padrão do serviço de impressão, utilizando a ferramenta *netcat*:

```
echo "0 3  
http://malicious-printer/printers/fake nc -u -w1  
[IP_ALVO] 631.
```
2. **Auditoria Estatística de Processos e Serviços:** No SentinelOS, realizou-se uma varredura interna por meio do gerenciador de serviços do OpenRC (*rc-status*) e do inspetor de processos (*ps*) para verificar a existência de ganchos de execução vinculados ao barramento do CUPS.

Os resultados experimentais atestaram a eficácia do princípio da Economia de Mecanismos integrado à postura de filtragem do sistema, gerando um cenário de mitigação em duas camadas autônomas. Primeiramente, a auditoria de processos confirmou que o SentinelOS, por ser construído sob um perfil estritamente minimalista no Gentoo, não possui os binários ou *daemons* do CUPS instalados ou ativos em segundo plano. A ausência deste *bloatware* arquitetural elimina o vetor de ataque em nível de aplicação por pura inexistência do código vulnerável no espaço do usuário (*Userland*).

Secundariamente, mesmo em um cenário hipotético onde o serviço fosse necessário e estivesse ativo, o ataque falharia na fronteira de rede. A inspeção dos contadores do *nftables* demonstrou que o pacote UDP intruso foi descartado (*dropped*) ao atingir a interface física de rede. Como a cadeia *input* opera sob uma política estrita de *default-deny*, qualquer tráfego entrante não solicitado e que não pertença a uma conexão previamente estabelecida pelo usuário é bloqueado.

O experimento valida que a arquitetura do SentinelOS neutraliza com sucesso ameaças complexas de rede local de maneira passiva e ativa. Ao fundir a redução da superfície de ataque com um firewall perimetral estrito, o sistema impede que serviços secundários comprometam a integridade da estação de trabalho.

7. Contribuições do Projeto e Análise de Resultados

Este projeto transcende a mera configuração e o agrupamento de ferramentas pré-existentes, apresentando contribuições diretas nas áreas de engenharia de software e

segurança de sistemas. A construção do SentinelOS exigiu o desenvolvimento de soluções proprietárias para mitigar as barreiras de usabilidade e reprodutibilidade inerentes a sistemas de alta segurança restritiva.

As principais contribuições deste trabalho dividem-se em três eixos centrais de engenharia e arquitetura, descritos a seguir.

7.1. Desenvolvimento da Ferramenta de Orquestração de Contêineres

O uso isolado do motor Podman para o confinamento de aplicações de *desktop* (com interface gráfica) exige a injeção manual de *sockets* do servidor gráfico, controle minucioso de permissões e gestão complexa de *flags* do *kernel*. Isso inviabiliza a adoção contínua e segura por parte do usuário. Para solucionar este gargalo, a principal contribuição em nível de *software* deste projeto é a construção da ferramenta proprietária de gerenciamento de ambientes, o *Podman Env Builder* (mkc).

- **Mitigação de Conflitos de Controle de Acesso (DAC):** A ferramenta resolve o problema clássico de permissões na montagem de volumes locais. Através da interceptação e injeção dinâmica dos identificadores reais do hospedeiro (`USER_UID` e `USER_GID`) no `Dockerfile` em tempo de compilação, o sistema garante total integridade e transparência na manipulação de arquivos entre o contêiner e o sistema *host*.
- **Isolamento de Superfície Gráfica e Wrappers Dinâmicos:** O orquestrador isola a complexidade de configuração do protocolo Wayland. Ele permite que a interface gráfica seja ativada sob demanda e gera programaticamente *scripts* de execução (*wrappers* em Python) e atalhos (`.desktop`). Isso permite que aplicações executadas no *Ring 3* enjauladas se integrem nativamente à interface do usuário.
- **Padronização e Imposição de Restrições de Segurança:** A ferramenta implementa um subsistema de perfis em disco que assegura a injeção automatizada de diretrizes nativas de *sandboxing* (como `--cap-drop`, `--no-new-privileges` e opções estritas de `seccomp`), eliminando o erro humano na instanciação de contêineres de alta criticidade.

7.2. Automação e Reprodutibilidade da Instalação do Sistema

A construção do zero de um sistema operacional baseado no ecossistema Gentoo Linux exige a compilação do código-fonte e a aplicação manual de parâmetros rígidos, um procedimento árduo e altamente suscetível a erros estruturais que podem comprometer a segurança.

- **Garantia de Reprodutibilidade:** O projeto contribui com a arquitetura de uma ferramenta de instalação automatizada. Mesmo encontrando-se em fase de refinamento contínuo, a lógica de *scripts* de *bootstrap* prova que a pesquisa não é puramente teórica, consolidando os extensos manuais de segurança em um processo de *deploy* determinístico e auditável.
- **Padronização de Hardening:** A automação garante a distribuição invariável de configurações sensíveis, como o *kernel* customizado monolítico, a imposição do *Lockdown LSM* em modo de integridade, as regras *default-deny* do `nftables` e o arquivo de roteamento transparente `torrc`. O *script* transforma o *hardening* sistêmico em um artefato de *software* tangível.

7.3. Validação do Paradigma Antivigilância em *Desktops*

Por fim, o SentinelOS constitui, de forma global, uma prova de conceito (PoC) empírica sobre a viabilidade operacional do princípio da *Defesa em Profundidade* [Schneider 1999] aplicado a estações de trabalho pessoais.

- **Arquitetura:** O projeto mostra a possibilidade prática de orquestrar tecnologias díspares e altamente restritivas — a `musl libc`, o protocolo Wayland puro, o isolamento por *Namespaces/Cgroups* e o *AppArmor* — em uma distribuição única, sem inviabilizar completamente a usabilidade.
- **Segurança por Design:** Ao deslocar o paradigma defensivo tradicional de ”aplicação reativa de correções (*patches*)” para a ”redução da superfície de ataque”(removendo código legado, LKMs e serviços desnecessários), o projeto entrega um modelo fundacional resiliente contra técnicas de vigilância em nível de rede e ameaças do tipo *zero-day*.

7.4. Trabalhos Futuros: Testes de Estresse e Validação em Ambientes Reais

Apesar dos indícios promissores obtidos nos ensaios laboratoriais e da coesão arquitetural, o SentinelOS ainda precisa ser mais extensivamente testado. Para o amadurecimento da distribuição e sua transição para um ambiente de produção estável, propõem-se frentes de desenvolvimento futuras concentradas na auditoria técnica e na análise empírica de viabilidade operacional.

Em primeiro lugar, o sistema deve ser submetido a baterias de testes de estresse (*stress testing*) e auditorias de segurança independentes e prolongadas. Faz-se necessária a aplicação de técnicas automatizadas de injeção de falhas (*fuzzing*), especialmente sobre a ferramenta de orquestração proprietária (`mkc`) e sobre os perfis customizados do *AppArmor*. Adicionalmente, a condução de simulações de intrusão avançadas (*Red Teaming*) permitirá avaliar a resiliência das políticas de contenção do *kernel* e do roteamento via proxy transparente frente a ataques complexos, persistentes e de múltiplas etapas, frequentemente associados a Ameaças Persistentes Avançadas (APTs).

Em segundo lugar, a eficácia de um sistema operacional focado em segurança defensiva é intrinsecamente dependente da sua viabilidade prática para o operador humano. Portanto, os próximos passos do projeto devem englobar a implantação do SentinelOS em cenários cotidianos por meio de grupos focais compostos por usuários reais — incluindo indivíduos em áreas sensíveis, como pesquisadores, comunicadores e ativistas. Esta fase de testes práticos visa coletar métricas sobre a fricção operacional introduzida pela política de negação padrão (*default-deny*), o impacto (*overhead*) de processamento das múltiplas camadas de confinamento e a curva de aprendizado imposta pelas interfaces minimalistas.

A avaliação do retorno (*feedback*) desses usuários é fundamental para refinar a ergonomia da ferramenta de contêineres e garantir que as rigorosas restrições arquiteturais não inviabilizem a usabilidade diária, buscando o alinhamento ideal entre a mitigação da superfície de ataque e a fluidez do ambiente de trabalho.

Referências

Anderson, R. (2020). *Security engineering: a guide to building dependable distributed systems*. John Wiley & Sons, 3rd edition.

- Antunes, M. P. and Júnior, R. M. (2014). Um estudo empírico exploratório em confiabilidade de sistemas operacionais. In *Anais do XXXIV Congresso da Sociedade Brasileira de Computação (CSBC)*, Porto Alegre, RS, Brasil. SBC.
- Bauer, M. (2006). Paranoid penguin: An introduction to novell apparmor. *Linux Journal*.
- Dingledine, R., Mathewson, N., and Syverson, P. (2004). Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*.
- Edge, J. (2012). Linux kernel module rootkits. In *LWN.net*. Eklektix, Inc.
- Felipe, R. (2023). *Comparison of C/POSIX standard libraries*. Acesso em: Maio 2026.
- Greenwald, G. (2014). *No place to hide: Edward Snowden, the NSA, and the US surveillance state*. Metropolitan Books.
- Hat, R. (2023). Shortcomings of rootless podman.
- Høgsberg, K. (2012). The wayland display server protocol. In *Proceedings of the Linux Symposium*.
- Linux man-pages project - cgroups. cgroups(7) — linux manual page.
- Linux man-pages project - namespaces. namespaces(7) — linux manual page.
- Manadhata, P. K. and Wing, J. M. (2011). An attack surface metric. *IEEE Transactions on Software Engineering*, 37(3):371–386.
- Newman, S. (2015). *Building Microservices: Designing Fine-Grained Systems*. O’Reilly Media, Sebastopol.
- Reiher, P. (2023). Introduction to operating system security. In *Operating Systems: Three Easy Pieces*. Version 1.00. Capítulo complementar.
- Rose, S., Borchert, O., Mitchell, S., and Connelly, S. (2020). Zero trust architecture. Technical Report NIST Special Publication 800-207, National Institute of Standards and Technology (NIST), Gaithersburg, MD.
- Russell, R. (2014). Nftables: The successor to iptables. *Linux Journal*.
- Saltzer, J. H. and Schroeder, M. D. (1975). The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308.
- Schneider, F. B. (1999). Enforceable security policies. *ACM Transactions on Information and System Security (TISSEC)*, 2(1):30–50.
- Schneier, B. (2015). *Data and Goliath: The hidden battles to collect your data and control your world*. W. W. Norton & Company.
- Stallings, W. and Brown, L. (2017). *Computer Security: Principles and Practice*. Pearson.