nullsec Kernel Configuration Guide

Hardening, networking, and forensic capture for security research platforms

Lateralus Language

bad-antics · April 2026 · nullsec / Lateralus Language Research

ABSTRACT This guide documents the kernel configuration choices made for nullsec. The configuration balances security hardening (to protect the platform itself) with forensic visibility (to capture traffic and system calls for security research). We document every non-default kernel option, the rationale for each choice, and the trade-offs between hardening and capability. The guide applies to both the nullsec Linux build (x86-64) and the Lateralus OS kernel (RISC-V).

1. Hardening vs Capability Trade-Off

A security research platform faces a unique tension: it must be hardened against external attack (because researchers often run it on networks with adversarial traffic) while simultaneously providing deep visibility into its own operation (for analyzing malware, monitoring network captures, and performing forensic analysis).

The nullsec kernel configuration resolves this by separating hardening (kernel-level protections) from capability (user-space tooling access). Hardening options that would prevent security research (e.g., restricting raw socket access) are kept disabled in research mode and enabled in lab mode.

2. Memory Hardening

The following memory hardening options are enabled in both modes:

```
CONFIG_RANDOMIZE_BASE=y          # KASLR: randomize kernel load address
CONFIG_RANDOMIZE_MEMORY=y        # ASLR for kernel memory regions
CONFIG_PAGE_TABLE_ISOLATION=y    # KPTI: mitigate Meltdown
CONFIG_HARDENED_USERCOPY=y       # validate usercopy bounds
CONFIG_SLAB_FREELIST_RANDOM=y    # randomize slab freelist order
CONFIG_SLAB_FREELIST_HARDENED=y  # detect double-free in slab
CONFIG_INIT_STACK_ALL_ZERO=y     # zero-initialize stack variables
CONFIG_STACKPROTECTOR_STRONG=y   # stack canaries on all functions
```

These options provide defense-in-depth against kernel exploits that target memory corruption vulnerabilities. They have negligible performance impact (<2% on typical workloads).

3. Network Visibility Options

For network security research, the kernel must allow raw socket access and promiscuous mode without dropping privileges. The following options are enabled in research mode only:

```
CONFIG_PACKET=y                  # raw packet sockets (tcpdump)
CONFIG_TUN=y                     # TUN/TAP for VPN and proxies
CONFIG_NETFILTER=y               # packet filtering (iptables)
CONFIG_NETFILTER_NETLINK=y       # nftables support
CONFIG_IP_ADVANCED_ROUTER=y      # policy routing
CONFIG_NET_IPIP=y                # IP-in-IP tunneling
CONFIG_NET_IPGRE=y               # GRE tunneling
```

In lab mode (the isolated research environment), these options remain enabled but the nullsec capability system restricts them to specific user accounts with the net_research capability.

4. Syscall Auditing and eBPF

All system calls are audited via the kernel audit subsystem. The nullsec audit configuration records: process creation and destruction, network connections, file opens and closes, and privilege escalation attempts.

```
CONFIG_AUDIT=y                  # enable audit subsystem
CONFIG_AUDITSYSCALL=y           # per-syscall audit records
CONFIG_BPF=y                    # Berkeley Packet Filter
CONFIG_BPF_SYSCALL=y            # BPF syscall for eBPF programs
CONFIG_BPF_JIT=y                # JIT-compile BPF programs
CONFIG_HAVE_EBPF_JIT=y          # architecture supports eBPF JIT
```

eBPF programs provide a safe way to add custom telemetry without loading a kernel module. nullsec ships several eBPF programs for common research tasks: network traffic sampling, process execution tracing, and file system activity monitoring.

5. Forensic Capture Configuration

For forensic analysis, the kernel is configured with persistent memory for crash dumps and a write-protected audit log partition:

```
CONFIG_CRASH_DUMP=y             # kdump: capture kernel crash dumps
CONFIG_PROC_VMCORE=y            # /proc/vmcore for crash analysis
CONFIG_KEXEC=y                  # kexec for crash kernel loading
CONFIG_KEXEC_FILE=y             # load crash kernel from file
CONFIG_MAGIC_SYSRQ=y            # SysRq key for emergency access
CONFIG_MAGIC_SYSRQ_SERIAL=y     # SysRq over serial port
```

The audit log is stored on a separate partition with the dm-integrity device mapper to detect offline tampering. The partition key is stored in the TPM, ensuring the audit log is unreadable without physical access to the TPM chip.

6. Disabled Options and Rationale

The following options are explicitly disabled in nullsec to reduce attack surface:

```
# CONFIG_MODULES is not set           # no loadable kernel modules
# CONFIG_KEXEC_SIG_FORCE is not set   # allow unsigned kexec for research
# CONFIG_BLUETOOTH is not set          # no Bluetooth (unused, reduces attack surface)
# CONFIG_FIREWIRE is not set           # no FireWire DMA (security risk)
# CONFIG_THUNDERBOLT is not set        # no Thunderbolt DMA attacks
```

The most significant disabled option is CONFIG_MODULES: disabling loadable kernel modules eliminates an entire class of privilege escalation attacks. All required drivers are compiled into the kernel. This limits hardware compatibility but improves security.

7. Lateralus OS Equivalent Configuration

The Lateralus OS kernel (used in the RISC-V nullsec variant) provides equivalent capabilities via its capability-based security model rather than Linux's sysctl/capabilities approach:

```
// Lateralus OS security configuration (capability table)
const RESEARCH_CAPS: [Capability] = [
    Capability::NetRaw,       // raw socket access
    Capability::NetPromiscuous, // promiscuous mode
    Capability::AuditRead,    // read audit log
```

```
    Capability::CrashDump,    // access crash dumps
    Capability::EbpfLoad,     // load eBPF programs
];
```

The Lateralus OS model is more fine-grained than Linux's capability system: capabilities are scoped to specific resources (e.g., NetRaw(interface: "eth0")) rather than system-wide.

8. Build and Verification

The nullsec kernel is built reproducibly using the Lateralus build system. The kernel configuration is version-controlled in the nullsec repository; any change to the configuration requires a review approval from a nullsec core maintainer.

```
# Build the nullsec kernel
ltl build --config nullsec-kernel.config linux-kernel

# Verify the binary matches the published hash
sha256sum bzImage
# d8f3a2... (matches nullsec/releases/latest/SHA256SUMS)
```

The SHA-256 of each kernel image is published with each nullsec release and should be verified before booting any nullsec image.