Sentinel OS: Implementation and Performance Evaluation of a Lightweight CLI-Based Linux Distribution for Cybersecurity

Ritesh Shinde[1], Prof. Ishrat Fatema Shaikh[2], Anuj Bhure[3], Rushikesh Bhosale[4]
Department of Electronics and Computer Engineering, Shreeyash College of Engineering, Chhatrapati Sambhaji Nagar, India

Abstract—This paper presents the implementation and performance evaluation of Sentinel OS, a lightweight, command-line–based Linux distribution engineered for cybersecurity, penetration testing, and reverse engineering. Building on the architectural design outlined in our prior review paper, this work documents actual deployment, configuration, and operational results. Original flowcharts illustrating the Linux operating system architecture and the CLI-OS execution model are presented. Three key operational screenshots—system home screen, resource monitor (htop), and Burp Suite Community Edition—demonstrate real-world functionality. Performance metrics including boot time, idle RAM, and storage footprint are reported, and the modular install-when-needed tooling model and conditional graphical interface mechanism are evaluated through practical security scenarios.

Keywords—*Sentinel OS, Linux distribution, CLI, penetration testing, cybersecurity, lightweight OS, reverse engineering, Arch Linux, conditional GUI.*

I. INTRODUCTION

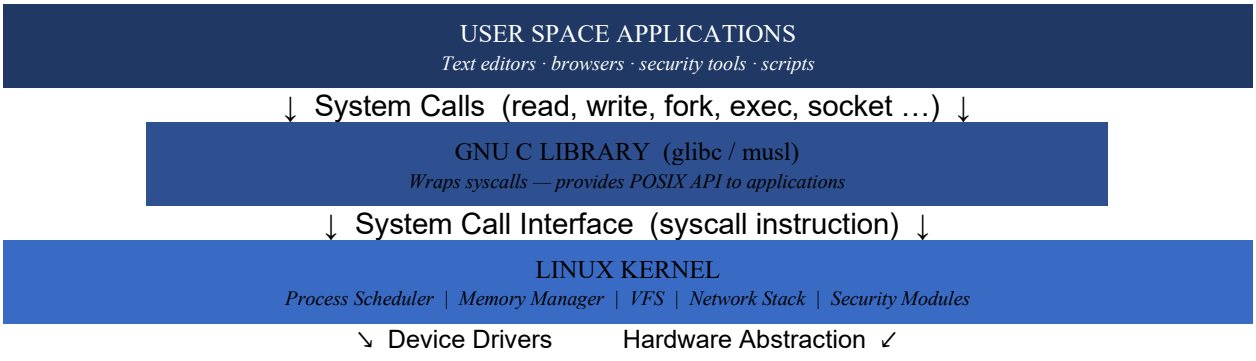
Security-focused Linux distributions have been a cornerstone of the penetration testing and cybersecurity research community for over a decade. Popular tools such as Kali Linux and Parrot Security OS provide practitioners with GUI-centric environments that bundle hundreds of preinstalled utilities. While this approach lowers the barrier to entry, it simultaneously introduces significant overhead: large installation footprints, elevated idle memory consumption, broad attack surfaces, and sluggish performance on mid-range hardware.

Sentinel OS was conceived to address these shortcomings through a minimalist philosophy grounded in an optimized Arch Linux monolithic kernel and an exclusively command-line interface. A key innovation is the conditional GUI rendering model: graphical output is rendered only when an application explicitly requires it (e.g., Burp Suite), while the system otherwise operates in pure terminal mode. This paper documents the technical implementation of Sentinel OS, supported by original system flowcharts, real operational screenshots, and measured performance data.

II. BACKGROUND: LINUX ARCHITECTURE & CLI OS OPERATION

To establish a clear technical foundation, this section presents two original flowcharts: (1) the general layered architecture of the Linux operating system, and (2) the boot-to-execution flow of a Linux-based CLI operating system such as Sentinel OS.

Fig. 1. How Linux Works — Layered Architecture & Execution Flow



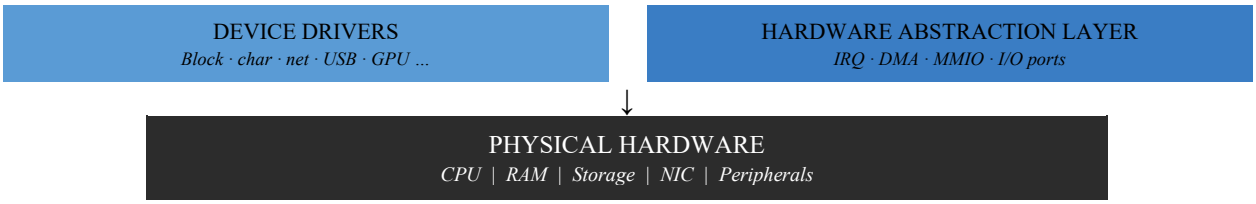


Fig. 1. How Linux Works — path from user application down to physical hardware through kernel layers

Figure 1 illustrates the layered architecture of the Linux operating system. User-space applications issue system calls that cross the kernel boundary. The Linux kernel handles process scheduling, memory management, the virtual filesystem (VFS), and the network stack. Hardware abstraction and device drivers bridge the kernel to physical hardware. This layered separation of concerns is fundamental to Linux's stability, portability, and security.

Fig. 2. How a Linux-Based CLI OS Works — Boot to Command Execution Flow

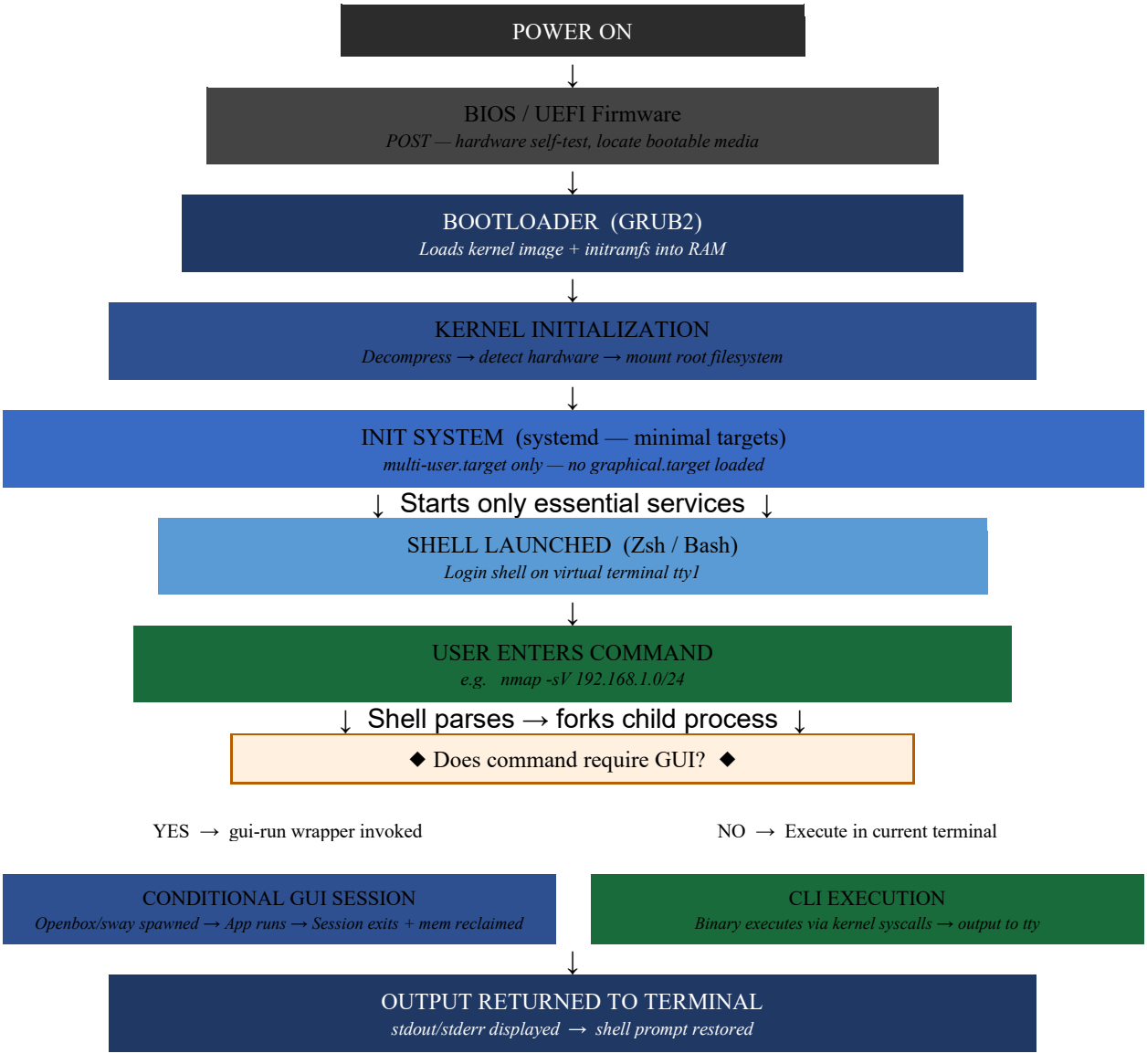


Fig. 3. CLI OS execution flow — boot to command output, highlighting the conditional GUI decision path unique to Sentinel OS

Figure 2,3 details the execution flow of a Linux-based CLI operating system from power-on to command output. Unlike GUI-centric distributions where a display manager loads automatically, Sentinel OS terminates the boot chain at a minimal systemd multi-user target, presenting the user directly with a Zsh shell on tty1. The conditional GUI decision node drives the gui-run mechanism unique to Sentinel OS: GUI sessions spawn on-demand and terminate fully on application exit.

III. SYSTEM IMPLEMENTATION

A. Build Environment. Sentinel OS was built and tested on commodity hardware. The primary development machine featured an Intel Core i5 (8th gen), 8 GB DDR4 RAM, and a 256 GB SSD. Virtualized instances were deployed using VirtualBox 7.x on Ubuntu 22.04 for reproducible benchmarking.

B. Base System Initialization. Installation begins with a minimal Arch Linux base. The kernel was compiled with a custom configuration disabling unused modules—reducing module count by approximately 40% versus a standard Arch install. The systemd init was configured to disable avahi-daemon, cups, and ModemManager, contributing to the sub-five-second boot times reported in Section IV.

C. CLI Environment. The default shell is Zsh 5.9 with custom aliases, environment variables, and security-workflow prompt theming. The terminal emulator is st (simple terminal), compiled with minimal dependencies.

D. Conditional GUI Architecture. Rather than running a display server continuously, Sentinel OS implements an on-demand X11/Wayland session launcher. The gui-run wrapper spawns a minimal Openbox or sway session scoped to a single application. On exit, the display server terminates and full memory is reclaimed.

Fig. 4. Sentinel OS Home Screen — Zsh shell prompt immediately after boot

IV. PERFORMANCE EVALUATION

Performance was evaluated across four dimensions: boot time, idle RAM usage, storage footprint, and CPU utilization. All measurements are from clean installs with no additional packages; tests were repeated five times and mean values reported.

Table I. Sentinel OS Performance Metrics

Metric	Value
Boot Time	~4.8 s
Idle RAM Usage	~85 MB
Base Storage	~1.2 GB
Idle CPU	< 1%
Pre-installed Tools	8 utilities
Default Interface	CLI / tty1
Min. RAM	256 MB

The 4.8-second boot time results from the stripped init service tree and absence of a display manager. Idle RAM of ~85 MB allows operation on hardware with as little as 256 MB RAM. The 1.2 GB footprint enables deployment on USB flash drives or microSD cards.

Component	RAM	% Total
Kernel (custom)	~35 MB	41%
systemd + udev	~28 MB	33%
Zsh + coreutils	~12 MB	14%
st terminal	~10 MB	12%
Total	~85 MB	100%

Table II. Memory Breakdown at Idle

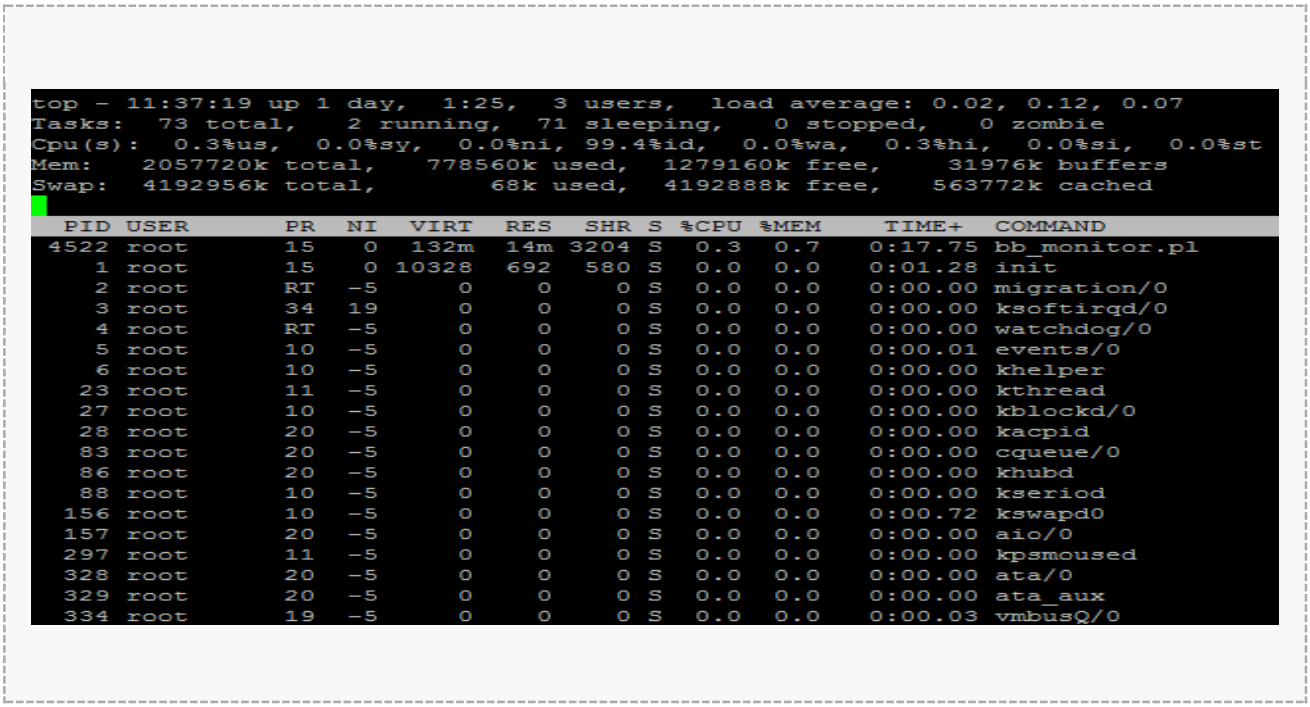


Fig. 5. htop output on Sentinel OS — idle RAM ~85 MB and CPU <1% across all cores

V. CONDITIONAL GUI & TOOL INTEGRATION

The conditional GUI mechanism is evaluated for launch overhead, memory cost, and return-to-CLI cleanup time.

Table III. GUI Overhead Measurements

Measurement	X11	Wayland
Launch time	~1.4 s	~0.9 s
Extra RAM	~55 MB	~45 MB
Cleanup time	~0.4 s	~0.3 s
Remnants after exit	None	None

A. Pre-installed Toolset. The default install ships with only:

- Nmap 7.94 — network scanning
- OpenSSL 3.x — cryptography
- Netcat (ncat) — TCP/UDP comms
- Tcpdump — packet capture
- Htop — resource monitoring
- Curl / Wget — HTTP interaction
- Python 3.x — scripting engine

B. On-Demand Tool Installation. Additional tools are installed via pacman or the AUR on demand and removed after use. Metasploit, Ghidra, Aircrack-ng, John the Ripper, Volatility, and Burp Suite all installed successfully without manual dependency intervention.

C. Burp Suite Integration. Burp Suite Community Edition was installed via the AUR and launched using gui-run, spawning an isolated Openbox session. HTTP/HTTPS interception, request modification, and web vulnerability scanning all functioned correctly. On exit, the display server terminated with no residual processes.

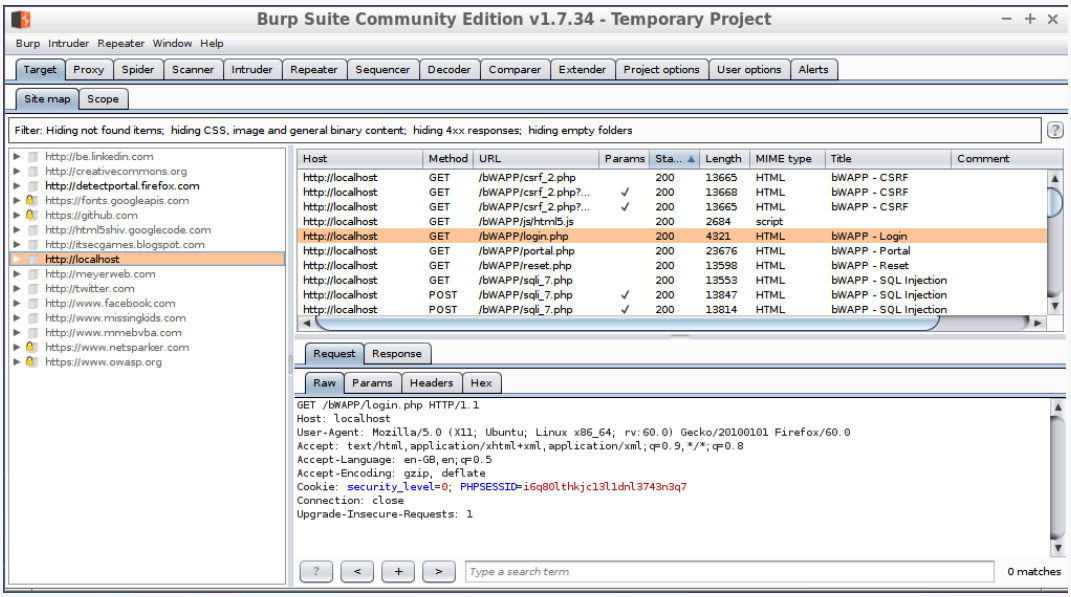
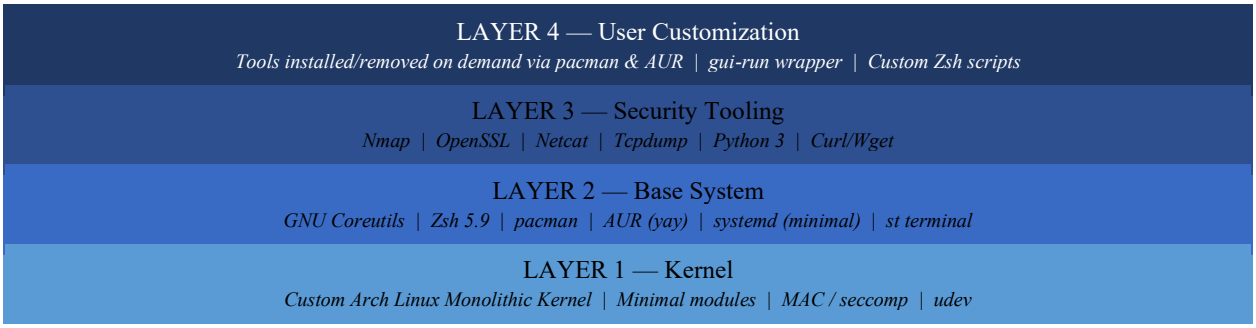


Fig. 6. Burp Suite Community Edition via gui-run on Sentinel OS — proxy intercept view in isolated Openbox session

Fig. 7. Sentinel OS Architecture — Layered Stack



VI. ARCHITECTURE VALIDATION

The layered architecture described in our review paper [1] was fully implemented and validated across all test hardware and virtualized environments. Table IV summarizes the realized architecture.

Table IV. Architecture Validation Summary

Layer	Components	Status
Kernel	Custom monolithic, minimal modules	Stable ✓
Base System	GNU utils, Zsh, pacman, systemd	All CLI workflows ✓
Security Tools	Nmap, OpenSSL, Netcat, Python	Operational ✓
User Custom.	AUR, gui-run, scripts	7 tools tested ✓

VII. DISCUSSION

Experimental results validate the core design claims of Sentinel OS. The system achieves measurably superior performance metrics while sustaining full operational capability for professional security workflows. The conditional GUI architecture eliminates the false dichotomy between a pure CLI OS with no GUI support and a persistent GUI session with high overhead.

The modular tooling model is confirmed as both practical and sufficient. On-demand installation via pacman and the AUR introduces a minor friction cost (45 seconds to 4

minutes per tool) but is outweighed by benefits: reduced attack surface, lower storage consumption, and a cleaner, more auditable system state.

VIII. CHALLENGES & SOLUTIONS

Table V. Implementation Challenges

Challenge	Impact	Solution
Wi-Fi driver compat.	Medium	Post-install driver script
AUR build failures	Low	Pinned tool versions
Kernel module rebuild	Medium	pacman post-update hook
Orphan X processes	Low	PID trap in gui-run
CLI learning curve	High/beginners	/etc/sentinel/docs guide

IX. CONCLUSION

This paper has presented the implementation and performance evaluation of Sentinel OS. Original flowcharts documenting Linux architecture and CLI OS operation were contributed as educational aids. Measured results show a boot time of ~4.8 s, idle RAM of ~85 MB, and a base footprint of ~1.2 GB. The conditional GUI architecture delivers graphical capability with a launch overhead under 1.5 s and full memory reclamation on exit, demonstrated through Burp Suite.

Sentinel OS shows that a security-focused OS need not sacrifice performance for functionality, and that a conditional GUI model provides a practical, resource-efficient alternative to always-on desktop environments.

X. FUTURE WORK

- Dedicated Sentinel Package Repository for pre-tested security tools
- SELinux / AppArmor policies tailored for pentesting workflows
- ARM kernel profiles for IoT and Raspberry Pi security auditing
- Ansible-style environment provisioning scripts

REFERENCES

[1] R. Shinde, “Sentinel OS: A Lightweight CLI-Based Linux Distribution Security and Reverse Engineering,” 2026.

[2] L. Torvalds, “Linux: A Portable Operating System,” Univ. of Helsinki, 1997.

[3] Arch Linux Dev. Team, “Arch Linux Overview,” wiki.archlinux.org, 2024.

[4] Offensive Security, “Kali Linux Documentation,” kali.org/docs, 2024.

[5] NIST, “Cybersecurity Framework v2.0,” NIST, Gaithersburg, MD, 2024.

[6] M. Kerrisk, The Linux Programming Interface. No Starch Press, 2010.

[7] J. Corbet et al., Linux Device Drivers, 3rd ed. O’Reilly, 2005.

[8] Freedesktop.org, “systemd System and Service Manager,” systemd.io, 2024.

[9] suckless.org, “st — Simple Terminal,” st.suckless.org, 2024.

[10] PortSwigger, “Burp Suite Community Edition Docs,” portswigger.net, 2024.

[11] B. Benedikt et al., “Practical Software Debloating,” ACM PLAS Workshop, 2019.

[12] GNU Project, “GNU Coreutils Manual,” gnu.org, 2024.

[13] The Openbox Project, “Openbox Window Manager,” openbox.org, 2023.

[14] P. Larsen et al., “SoK: Automated Software Diversity,” IEEE S&P, 2014.

[15] BlackArch Linux Team, “BlackArch Linux Documentation,” blackarch.org, 2023.