Comprehensive Prompt for Windsurf AI: PAASS Arch Linux Installation Script Analysis & Enhancement

Context

I'm developing **PAASS (Privacy-Augmented Arch Security System)**, a fork of Luke Smith's LARBS (Luke's Auto-Rice Bootstrapping Scripts), designed as a next-generation, privacy-centric Arch Linux platform for cybersecurity research. I have an existing installation script (`arch-secure-deploy.sh`) that needs comprehensive analysis, refactoring, and enhancement to achieve 99.99% reliability.

Reference Documentation: Please analyze the attached "Arch Linux Security Research Platform.docx" (or the comprehensive technical specification document provided) which contains detailed requirements for all six foundational pillars of this platform.

Current Setup

Existing Resources

- **Script Location:** `Y:\mega\projects\personal\PAASS\static\arch-secure-deploy.sh` (Windows path)
- **Repository Structure:**
  - PAASS (my fork) ← upstream: `LukeSmithxyz/LARBS`
  - `voidbari` (my dotfiles fork) ← upstream: `LukeSmithxyz/voidrice`
- **Testing Environments Available:**
  - VS Code with Windsurf integration
  - Arch Linux WSL2
  - Arch Linux VirtualBox VM
- **Target Hardware:** Lenovo ThinkPad P1 Gen 5 (Type 21DC)

Current Script Features (Preserve These)

The existing `arch-secure-deploy.sh` already implements:

- BTRFS subvolume structure: `@`, `@home`, `@var`, `@snapshots`, `@log`, `@var/cache`
- LUKS2 encryption with single passphrase for root and home
- Default partition sizes: 50GB root, remainder for home
- Interactive configuration prompts
- Comprehensive logging and error handling
- State persistence for recovery
- Weekly BTRFS snapshot automation
- Default variables for hostnames, usernames, volume names

CRITICAL: Do NOT change these default values without asking for my explicit review first.

Primary Objectives

1. Script Analysis & Refactoring

Please analyze my existing `arch-secure-deploy.sh` script and:

A. Identify Code Quality Issues

- Redundant code patterns (e.g., repetitive mount commands)
- Incomplete function implementations (e.g., `validatepassphrasestrength`)
- Error handling gaps
- Missing edge case coverage

B. Propose Specific Refactorings

For each issue found, provide:

- **Current code snippet** (exact lines from my script)
- **Problem description** (why it needs improvement)
- **Proposed refactored code** (complete, production-ready replacement)
- **Benefits** (reliability improvement, maintainability, etc.)

Example format:

CURRENT (Lines 450-455):

mount -o subvol=@,compress=zstd:3,noatime /dev/mapper/rootcrypt /mnt/root

PROBLEM: Repeated mount options across multiple subvolumes, violates DRY

PROPOSED REFACTOR:

```
mount_btrfs_subvol() {
local subvol="$1"
local mountpoint="$2"
# ... (complete implementation)
}
```

C. Enhance Error Handling

- Add retry logic where appropriate
- Improve failure recovery mechanisms
- Ensure graceful degradation for non-critical operations
- Add comprehensive pre-flight checks

D. Add Missing Functionality

Based on the reference document, identify and implement:

- Enhanced passphrase strength validation (regex-based complexity checks)
- GRUB installation fallback mechanisms (multiple methods)
- Network connectivity resilience (multiple test hosts)
- TPM2 enrollment for LUKS auto-unlock
- AppArmor/SELinux integration
- Advanced firewall configuration

2. Repository Management Strategy

A. Git Workflow Implementation

Create a complete workflow for managing forks:

Required Scripts:

1. **sync-upstream.sh** - Merge upstream LARBS/voidrice changes
2. **GitHub Actions workflow** - Automated weekly upstream sync
3. **track-changes.sh** - Document modifications in PAASS-CHANGES.md

Workflow Requirements:

- Preserve my customizations during upstream merges
- Handle merge conflicts intelligently
- Create pull requests when conflicts occur
- Maintain separate branches: paass-main, voidbari-main

B. Configuration Backup Strategy

Design a system to:

- Store all customizations in cloud (GitHub)
- Enable one-command restoration on any machine
- Version control for progs.csv, sysctl configs, iptables rules
- Automated backup of LUKS headers, SSH keys, GPG keys

3. Testing Environment Recommendation

Analyze these options and recommend the BEST for automated testing WITHOUT my intervention:

Environment	Pros	Cons
Arch WSL2	Fast, Windows integration	No UEFI, limited systemd
VirtualBox VM	Full UEFI, real partitioning	Slower, more overhead
VS Code + Windsurf	Direct code execution	???

Deliverable: Provide a complete automated testing script for the recommended environment that:

- Creates test VM/environment
- Runs `arch-secure-deploy.sh` in dry-run mode
- Validates script logic without destructive operations
- Reports any errors or warnings

4. Security Enhancements (Per Reference Document)

Implement or enhance these features based on the comprehensive specification:

A. Cryptographic Hardening

- LUKS2 with Argon2id KDF (memory-hard)
- TPM2 integration for auto-unlock with PCR binding
- LUKS header backup automation
- Key management protocols

B. Kernel Hardening

- Comprehensive sysctl configuration (60+ parameters from ANSSI, KSPP, CIS)
- Kernel command-line hardening (ASLR, PTI, Spectre mitigations)
- AppArmor profiles for critical applications
- Signed kernel module enforcement

C. Network Privacy Stack

- Tor configuration with entry/exit node preferences
- I2P daemon setup and integration
- VPN chaining support (OpenVPN + WireGuard)
- iptables default-deny firewall
- DNS leak prevention (DNSCrypt-proxy, systemd-resolved)
- MAC address randomization

D. Ephemeral Storage

- tmpfs RAM-only workspace script
- ZRAM encrypted swap (instead of disk swap)
- Secure file deletion utilities (shred, srm)
- eCryptfs encrypted vault setup

E. BTRFS Optimization

- Multi-tier snapshot strategy (hourly/daily/weekly)
- Intelligent cleanup with retention policies
- Compression benchmarking and tuning
- Subvolume security hardening (mount options)

5. POSIX Process Management Module

Based on the reference document's Python script example (`/usr/local/bin/process-manager.py`):

Requirements:

- Demonstrate fork/exec for process creation
- IPC via POSIX pipes (parent-child communication)
- Signal handling (SIGINT, SIGTERM) for graceful shutdown
- Error handling for all system calls
- Resource monitoring integration
- Production-ready code quality

Deliverable: Complete, tested Python script with:

- Comprehensive docstrings
- Type hints
- Unit tests (if possible in this context)
- Integration with systemd service

6. Documentation Requirements

For every change or addition, provide:

Inline Documentation

==========

==========

SECTION: Partition Layout Configuration

==========

==========

Purpose: Create GPT partition table with:

- 1GB EFI System Partition (FAT32)

- 50GB Root partition (LUKS2 → BTRFS)

- Remainder: Home partition (LUKS2 → BTRFS)

Security Note: Using LUKS2 with Argon2id KDF for resistance against

GPU-accelerated attacks. Single passphrase for UX.

==========

==========

Change Log Format

For PAASS-CHANGES.md:

[2025-11-26] - Enhanced Script v2.0

Added

- TPM2 auto-unlock enrollment in Phase 3
- Multi-fallback GRUB installation (3 methods)
- AppArmor profile enforcement in Phase 11

Changed

- Refactored mount functions to use DRY principle (Lines 450-520)
- Enhanced passphrase validation with entropy estimation

Fixed

- GRUB installation on NVRAM-disabled systems
- Race condition in snapshot cleanup logic

Specific Questions to Address

Critical Decisions Needed

1. **Snapshot Retention Strategy:**
   - Current: 12 weekly snapshots
   - Proposed: Multi-tier (24 hourly, 7 daily, 12 weekly)?
   - **Ask me:** Which strategy to implement?
2. **Kernel Hardening Trade-offs:**
   - `nosmt=force` disables hyperthreading (20-30% performance loss)
   - Stronger security vs. usability
   - **Ask me:** Enable full mitigations or performance mode?
3. **Default Software Selection:**
   - Current `progs.csv` includes basic tools
   - Reference doc suggests extensive security toolkit
   - **Ask me:** Full toolkit install or minimal base + manual additions?
4. **TPM2 Auto-Unlock:**
   - Requires Secure Boot + custom key enrollment
   - Complex but eliminates passphrase typing
   - **Ask me:** Implement TPM2 or stick with passphrase-only?

Success Criteria

The enhanced script MUST:

Reliability

- ✅ Handle all error conditions gracefully
- ✅ Provide clear error messages with recovery steps
- ✅ Support resumption after interruption (state file)
- ✅ Validate all user inputs with regex
- ✅ Include pre-flight checks (disk space, network, tools)

Security

- ✅ Implement all 6 foundational pillars from reference doc
- ✅ Use security best practices (no hardcoded credentials)
- ✅ Follow principle of least privilege
- ✅ Enable security features by default (fail-secure)

Maintainability

- ✅ DRY principle (no code duplication)
- ✅ Modular functions (single responsibility)
- ✅ Comprehensive inline documentation
- ✅ Consistent naming conventions
- ✅ Version-controlled configuration

Usability

- ✅ Clear prompts with examples
- ✅ Sane defaults (50GB root, UTC timezone, etc.)
- ✅ Progress indicators during long operations
- ✅ Summary at end with next steps

Deliverables Requested

Phase 1: Analysis Report

1. **Script Audit Report** (Markdown format)
   - Issues found with severity ratings (Critical/High/Medium/Low)
   - Proposed refactorings with code snippets
   - Risk assessment for each change
2. **Testing Environment Recommendation**
   - Comparison matrix (WSL vs VirtualBox vs other)
   - Automated testing script for recommended environment
   - CI/CD integration plan (GitHub Actions)

Phase 2: Enhanced Script

1. **`arch-secure-deploy-v2.sh`**
   - All refactorings applied
   - New features from reference doc
   - 100% error handling coverage
   - Comprehensive logging
2. **Supporting Scripts**
   - `sync-upstream.sh` (upstream merge automation)
   - `test-in-virtualbox.sh` (automated VM testing)
   - `system-health.sh` (monitoring dashboard)
   - `integrity-check.sh` (AIDE integration)
3. **Configuration Files**
   - `/etc/sysctl.d/99-hardening.conf` (60+ kernel parameters)
   - `/etc/iptables/iptables.rules` (default-deny firewall)
   - `/etc/tor/torrc` (privacy-optimized Tor config)
   - `/etc/apparmor.d/*` (custom profiles)

Phase 3: Documentation

1. **PAASS-CHANGES.md** (change tracking)
2. **INSTALLATION.md** (step-by-step guide)
3. **RECOVERY.md** (emergency procedures)
4. **MAINTENANCE.md** (weekly/monthly/quarterly tasks)

Phase 4: Repository Setup

1. **GitHub Actions Workflows**
   - .github/workflows/sync-upstream.yml
   - .github/workflows/test-script.yml
   - .github/workflows/security-audit.yml
2. **Directory Structure**

```
PAASS/
├── larbs.sh (modified LARBS installer)
├── static/
│   ├── arch-secure-deploy-v2.sh ← MAIN SCRIPT
│   ├── progs.csv (customized package list)
│   └── configs/ (all config files)
├── scripts/ (helper scripts)
├── docs/ (comprehensive documentation)
└── .github/ (CI/CD workflows)
```

Constraints & Preferences

MUST Preserve

- Existing default variables (hostname: devta, user: patel, etc.)
- BTRFS subvolume names (@, @home, @var, @snapshots, @log)
- Partition size defaults (50GB root, remainder home)
- Single LUKS passphrase UX (don't force separate keys)

MUST Ask Before Changing

- Snapshot retention count (currently 12)
- Root partition size (currently 50GB)
- Performance vs. security trade-offs (SMT disable, etc.)
- Software package selections beyond base system

Preferred Tools

- **Shell:** zsh (primary), bash (compatibility)
- **Editor:** neovim (with nvim-lsp)
- **AUR Helper:** paru (preferred over yay)
- **Bootloader:** GRUB (with Secure Boot support)
- **Init System:** systemd
- **Firewall:** iptables (with nftables migration path)

Coding Standards

- Follow ShellCheck recommendations
- Use `set -euo pipefail` in all bash scripts
- Prefer `[[` over `[` for conditionals
- Use `$(command)` over backticks
- Quote all variables: `"$variable"`
- Use readonly for constants: `readonly CONSTANT=value`

Example Interaction Flow

You (Windsurf AI):

> I've analyzed your script. Found 15 issues:
> **CRITICAL (3):**
> 1. Line 450: `validatepassphrasestrength()` is empty stub
> 2. Line 780: GRUB install has no fallback
> 3. Line 1200: No TPM2 enrollment
>
> **HIGH (5):** ...
> Shall I proceed with refactoring all CRITICAL issues first?

Me:

> Yes, start with CRITICAL issues. For #3 (TPM2), ask me first about Secure Boot requirements.

You:

> Refactored validatepassphrasestrength():
>
> ```
> [complete implementation]
> ```
>
> For TPM2 enrollment: Do you have Secure Boot enabled with custom keys, or should I design for both scenarios?

Me:

> Secure Boot enabled, but no custom keys yet. Provide instructions for key enrollment.

[Iteration continues...]

Final Notes

- **Priority:** Reliability > Features. The script must NEVER break.
- **Testing:** Test EVERY change in VirtualBox before recommending.
- **Communication:** Ask clarifying questions when requirements conflict.
- **Incremental:** Deliver in phases (analysis → refactor → enhance → test).
- **Documentation:** Over-document rather than under-document.

When you're ready to proceed, start with:

1. Confirm you have access to `arch-secure-deploy.sh` (attached file)
2. Confirm you have access to reference documentation
3. Present the Script Audit Report (Phase 1, Deliverable 1)

Thank you for your meticulous attention to detail and expert-level Linux systems knowledge!