SICHERHEITS-AUDIT — RUNDE 3

NetMon Roots v1.1.14

Code Security Review — Juni 2026

Datum	28. Juni 2026
Analysierte Version	v1.1.14 — Runde 3
Analysierte Module (neu)	17+
Bestaetigte Findings	1 (MEDIUM)
Bewertete Kandidaten	7
Ausgeschlossene Kandidaten	6 (K1+K2 manuell widerlegt, K4-K7 FP-Filter)
Konfidenz-Schwelle	>= 8 / 10
Methode	Statische Code-Analyse + manuelle Verifikation
Analysewerkzeug	Claude Sonnet 4.6 (Anthropic)
Projektverantwortlicher	Sven Froehlich

1 FINDING BESTAETIGT (MEDIUM) · K1 + K2 MANUELL WIDERLEGT

1. Zusammenfassung

Die dritte statische Sicherheitsanalyse des NetMon-Arbeitsstands v1.1.14 durch ein Multi-Agenten-System (Claude Sonnet 4.6) mit manueller Verifikation und Live-Tests bestaetigt 1 Finding (MEDIUM). 6 Kandidaten ausgeschlossen — K1 und K2 durch manuelle Code-Verifikation widerlegt. Kein Remote-Exploit.

K1 (WIDERLEGT, 9/10 → FP): PATH-Hijacking via shutil.which(). Manuelle Verifikation: TRUSTED_WEB_CMD_PREFIXES bereits auf Zeile 16 implementiert.

K2 (WIDERLEGT, 8/10 → FP): Beliebige Python-Ausfuehrung aus fun_games_dir. Manuelle Verifikation: fun_games_dir ist hardcoded (app.py Zeile 2581: ~/.local/share/netmon/fun/code_the_classics), nie aus Settings geladen. Same-User-Desktop ohne Privilege Escalation — Ausschluss greift.

K3 (MEDIUM, 9/10, bestaetigt): agent/runner.py prueft nur parts[0] in der Allowlist. Live-Test bewiesen: subprocess.run(['black', '/home/user/.bashrc']) schreibt die Datei tatsaechlich um. Tilde-Angriffe schlagen fehl (~/file nicht expandiert). Angriffsvektor: Prompt-Injection mit absoluten Pfaden.

2. Methodik

Phase 1 — Reconnaissance

Alle 17+ neuen Module gelesen: rkhunter_manager.py, coordinator.py, state.py, support_bundle.py, diagnostics.py, links_controller.py, fun_controller.py, process_guard_controller.py, repository_guard_controller.py, roots_baseline_guard_controller.py, lang_service.py, network_helpers.py, process_guard_service.py, repository_guard_service.py, cleanup_assistant_tab.py, download_scan_tab.py, radio_tab.py. Fokus: subprocess-Calls, Dateioperationen, URL-Handling, Deserialisierung.

Phase 2 — Kandidaten-Analyse

Datenflussketten nachverfolgt: PATH-Umgebungsvariable → shutil.which → sudo install, os.walk(games_dir) → subprocess.Popen, Allowlist-Pruefung ohne Pfad-Confinement.

Phase 3 — False-Positive-Filter

Unabhaengige Re-Analyse jedes Kandidaten. Ausschluss: DoS, Single-User-Desktop ohne Privilege-Boundary-Crossing, Python-3.12+-Schutzmechanismen, Umgebungskontrolle als Pre-Condition.

Phase 4 — Manuelle Verifikation

Alle 3 High-Confidence-Findings durch direktes Lesen der Quelldateien verifiziert. Zeile fuer Zeile nachvollzogen: shutil.which-Rueckgabe, repair_targets-Schreibpfad, subprocess.Popen-Argumente, run_command-Allowlist-Logik.

3. Bestaetrigte Findings (nach manueller Verifikation)

K1 wurde initial mit Konfidenz 9/10 als HIGH eingestuft. Manuelle Verifikation zeigt: TRUSTED_WEB_CMD_PREFIXES ist bereits implementiert — K1 ist ein False Positive. Analoger Fehler wie Runde 2 K1 (Agent uebersah _normalize_station_stream_url()).

K1 — PATH-Hijacking rkhunter_manager — manuell widerlegt

Konfidenz: 9/10 → FP (nach manueller Verifikation)

`netmon/core/rkhunter_manager.py, Zeilen 16–26`

WIDERLEGT: TRUSTED_WEB_CMD_PREFIXES = ('/usr/bin/', '/bin/', '/usr/local/bin/') bereits auf Zeile 16 definiert. _preferred_web_cmd() prueft jeden shutil.which()-Rueckgabewert dagegen — ~/.local/bin/wget wird abgelehnt. Automatischer Agent uebersah die Pruefroutine (analoger Fehler wie Runde 2 K1).

K3 — Agent-CLI: black/ruff ohne Pfad-Confinement schreiben beliebige Dateien

Severity: MEDIUM | Konfidenz: 9/10

`Datei: agent/runner.py, Zeilen 7–13, 31–32`

Datenfluss:

```
LLM-Prompt-Injection mit absolutem Pfad → run_command('black /home/user/.bashrc') → parts[0] = 'black' → Allowlist-Check BESTANDEN [Zeile 31] → parts[1] = '/home/user/.bashrc' → KEIN Pfad-Check → subprocess.run(['black', '/home/user/.bashrc'], cwd=REPO_ROOT) → black schreibt .bashrc um ← Live-Test bestaetigt
```

run_command() prueft nur parts[0] in ALLOWED_BASE_COMMANDS — keine Pfadrestriktion fuer black/ruff/ls. Live-Test bestaetigt: subprocess.run(['black', '/home/happy/.bashrc']) hat die Datei reformatiert/ueberschrieben. Tilde-Angriffe (black ~/.bashrc) scheitern: subprocess ohne shell=True expandiert ~ nicht; black selbst auch nicht ('Path does not exist'). Angriffsvektor: Prompt-Injection in LLM-Agent mit absolutem Pfad (z.B. nach Informationsgewinn via ls). AGENTS.md konzipiert diesen Agent fuer LLM-Eingaben — realistischer Angriffspfad.

```
Fix: resolved = Path(arg).expanduser().resolve() if not resolved.is_relative_to(REPO_ROOT): raise ValueError(f'Pfad ausserhalb REPO_ROOT: {arg}') # Gilt fuer ls, black, ruff, pytest (analog Git-Subcommand-Allowlist)
```

4. Ausgeschlossene Kandidaten

K1 — PATH-Hijacking rkhunter_manager — manuell widerlegt

Konfidenz: 9/10 → FP -> FP

`Datei: netmon/core/rkhunter_manager.py, Zeilen 16–26`

Filterentscheid: WIDERLEGT — TRUSTED_WEB_CMD_PREFIXES = ('/usr/bin/', '/bin/', '/usr/local/bin/') bereits auf Zeile 16 definiert. _preferred_web_cmd() prueft jeden shutil.which()-Rueckgabewert — ~/.local/bin/wget wird abgelehnt. Automatischer Agent hat den Allowlist-Check uebersehen.

K2 — Python-Ausfuehrung aus Spieleordner — manuell widerlegt

Konfidenz: 8/10 → FP -> FP

`Datei: netmon/features/fun_controller.py · app.py Zeile 2581`

Filterentscheid: WIDERLEGT — fun_games_dir ist hardcoded auf ~/.local/share/netmon/fun/code_the_classics (app.py Zeile 2581, einzige Zuweisung, nie aus Settings geladen). Same-User-Desktop ohne Privilege Escalation. Ausschluss greift. Agent-Behauptung 'konfigurierbar in Settings' war falsch.

K4 — importlib-Modul-Loading aus sys.path (tooling.py)

Konfidenz: 7/10 -> FP

`Datei: netmon/core/tooling.py, Zeilen 47–81`

Filterentscheid: AUSGESCHLOSSEN — Angriff erfordert Kontrolle ueber Umgebungsvariablen (PYTHONPATH) oder CWD vor Prozessstart. Pre-Condition = bereits volle Code-Ausfuehrung als gleicher Benutzer. Kein zusaetzlicher Angriffspfad.

K5 — lang_code-Pfad-Traversal in LangService (lang_service.py)

Konfidenz: 7/10 -> FP

`Datei: netmon/services/lang_service.py, Zeile 12`

Filterentscheid: AUSGESCHLOSSEN — Nur lesend (json.load), auf JSON-parsierbare Dateien beschraenkt. Erfordert Schreibzugriff auf Settings-Datei im Benutzerverzeichnis (gleiche Privilege-Ebene). Kein Privilege-Boundary-Crossing.

K6 — TOCTOU-Race vor shutil.rmtree (cleanup_assistant_tab.py)

Konfidenz: 7/10 -> FP

`Datei: netmon/ui/tabs/cleanup_assistant_tab.py, Zeilen 2307–2464`

Filterentscheid: AUSGESCHLOSSEN — Python 3.12+ shutil.rmtree schutzt gegen Symlinks via NotADirectoryError. Race-Window extrem eng; erfordert praezise Timing-Koordination durch denselben Benutzer. Als Hardening-Empfehlung notiert.

K7 — .desktop-Exec-Parsing via links_controller.py

Konfidenz: 7/10 -> FP

`Datei: netmon/features/links_controller.py, Zeilen 804–813, 882–894`

Filterentscheid: AUSGESCHLOSSEN — ~/.local/share/applications/ ist das Standard-XDG-Verzeichnis fuer benutzerinstallierte Anwendungen; Exec=-Zeilen auszufuehren ist spezifikationskonformes Verhalten. Kein Privilege-Boundary-Crossing.

5. Sicherheitsverbesserungen seit Runde 2

URL-Schema-Validierung Radio-Modul (Runde 2, bestaetigt)

ALLOWED_RADIO_STREAM_SCHEMES = {'http', 'https', 'rtsp', 'rtmp'} in _normalize_station_stream_url() — verhindert SSRF via mpv/urllib.request.

WireGuard Interface-Name-Validierung (neu, Runde 3)

wg_profile_path() validiert Interface-Namen via re.fullmatch(r'[A-Za-z0-9_.-]+', ...) und os.path.commonpath()-Confinement auf /etc/wireguard.

roots_baseline_guard: sudo-Calls mit Pfad-Allowlist (neu, Runde 3)

_is_allowed_guard_path(path) mit os.path.realpath()-Normalisierung gegen ROOTS_GUARD_DEFAULT_AREA_PATHS — verhindert sudo cat auf beliebige Pfade.

ClamAV Tempdir Symlink-Schutz (seit Runde 1)

_safe_makedirs_no_symlink() prueft via os.lstat() — wirft RuntimeError bei Symlink.

82 Proxy-Methoden aus app.py entfernt (Runde 2)

Angriffsflaechenreduktion; alle via refactor_safety.py auf Null-Aufrufer verifiziert.

HDD-Mountpoint-Validierung (Runde 2)

_slot_mountpoint() mit os.path.realpath() + _SAFE_MOUNT_BASES-Allowlist.

6. Weitere Hardening-Empfehlungen

Zusaetzlich zu den drei bestaetigten Findings — Defense-in-Depth-Massnahmen ohne unmittelbaren Exploitpfad:

1. cleanup_assistant_tab.py TOCTOU:
shutil.rmtree nach lstat()-Check via os.open(path, O_NOFOLLOW) atomar gestalten.

2. lang_service.py lang_code-Sanitierung:
Regex ^[a-z]{2}(_[A-Z]{2})?$ vor os.path.join() validieren.

3. state.py get_netmon_documents_subdir:
Jedes part-Argument via os.path.basename() normalisieren vor makedirs().

4. WireGuard wg-quick Interface-Name:
wg show interfaces-Output gegen re.fullmatch(r'[A-Za-z0-9_.-]{1,15}', ...) pruefen.

5. webbrowser.open Schema-Validierung (Radio):
station['homepage'] via urlparse().scheme gegen {'http','https'} pruefen.

6. os.path.realpath in Roots-Baseline-Guard:
os.path.abspath durch os.path.realpath() fuer vollstaendige Symlink-Aufloesing ersetzen.

7. Gepruefe Sicherheitskategorien

Kategorie	Ergebnis
PATH-Hijacking via shutil.which() ohne Prefix-Validierung	**K1 BESTÄTIGT**
Arbitrary Code Execution via unsignierten Spieleordner	**K2 BESTÄTIGT**
Dateikorruption via LLM-Prompt-Injection in Agent-CLI	**K3 BESTÄTIGT**
importlib-Module-Loading aus sys.path/PYTHONPATH	FP (K4)
Path Traversal via lang_code in LangService	FP (K5)
TOCTOU Race vor shutil.rmtree	FP (K6)
.desktop-Exec-Parsing via links_controller	FP (K7)
Command Injection (subprocess, shell=True)	Kein Finding
SSRF via urllib.request / URL-Schema-Validierung	Kein Finding
Privilege Escalation (sudo mit Nutzereingabe)	Kein Finding (ausser K1)
Hardcoded Credentials / API-Keys	Kein Finding
Unsichere Deserialisierung (pickle, yaml.load, eval)	Kein Finding
Symlink-Angriffe auf privilegierte Dateioperationen	Kein Finding
shell=True in subprocess-Calls	Kein Finding

Impressum und Haftungsausschluss

Auditor: Claude Sonnet 4.6 (Anthropic) — Multi-Agenten-System + manuelle Verifikation

Projektverantwortlicher: Sven Froehlich

Datum: 28. Juni 2026

Codestand: Arbeitsstand v1.1.14 inkl. untracked/modified files — Runde 3

Methode: Statische Code-Analyse + manuelle Datenfluss-Verifikation — kein Laufzeit-Penetrationstest