The AI Trust Agent, Closing the Installer Exploit Gap

The AI Trust Agent: Closing the Installer Exploit Gap

Problem

Traditional trust (code signatures, PKI) verifies **who signed the installer**, not **what the installer actually does**. Build pipelines get popped, adware “options” sneak in, and post-install scripts quietly plant telemetry, updaters, drivers, and scheduled tasks. No mainstream stack performs **intent validation** of the installer’s script/logic *before* the user clicks “Accept.”

Core Idea

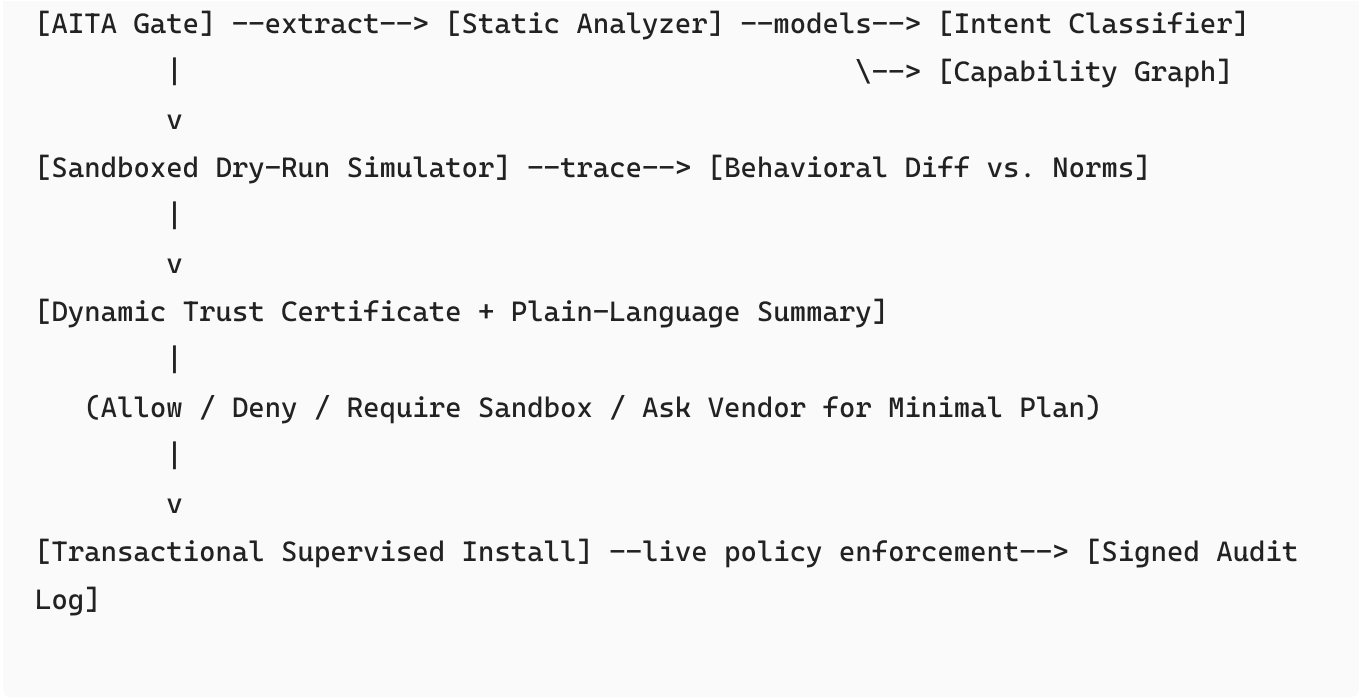
AI-OS introduces an **AI Trust Agent (AITA)** that intercepts installers pre-execution, **decompiles / expands their scripts**, simulates the full install plan in a sandbox, and issues a **Dynamic Trust Certificate (DTC)**: a signed, human-readable, and machine-verifiable assessment of intent, privilege use, network behavior, persistence hooks, and deviation from expected norms for that software category.

Goals

- **Pre-consent clarity**: show users what will happen *before* anything happens.
- **Contextual trust**: “Is this behavior normal for a video player?” not just “Is this signed?”
- **Transactional installs**: atomic, supervised, and fully reversible.
- **Policyable**: orgs can codify “never allow kernel drivers from note-taking apps,” etc.

High-Level Flow

```
[User launches installer]
        |
        v
```



Dynamic Trust Certificate (DTC)

- **Identity**: hashes, signers, timestamp, supply-chain attestations (SLSA/C2PA/SBOM if present).
- **Intent Summary** (plain language): files, services, drivers, scheduled tasks, startup entries, registry/launchd/systemd changes, network endpoints contacted, telemetry channels.
- **Privilege Map**: requested elevation, kernel hooks, device access, protected paths.
- **Risk Score + Rationale**: outliers vs. category norms, suspicious calls, opaque downloads.
- **Policy Verdict**: allow / deny / sandbox-only / require vendor-minimal plan.
- **Repro Plan Hash**: reproducible “minimal install” patch if AI prunes unnecessary steps.

Engineering Deep Dive: Hooking & Supervising Installers

0) Threat Model Cliff Notes

- **Supply-chain compromised but signed**: malicious script logic passes PKI.
- **Over-privileged installers**: bury persistence, drivers, telemetry.
- **Time-of-check/time-of-use**: installer fetches payloads after consent.
- **Evasion**: anti-VM, environment checks, encrypted scripts, JIT downloaders.

We assume attackers know we're analyzing; we design to **contain + explain**, not just detect.

---

1) Interception ("AITA Gate")

Windows

- **File type handlers**: hijack `.msi`, `.msix`, `.exe` (NSIS/Inno/Custom), `.appx` via AI-OS shell shims.
- **Windows Installer API**: `MsiOpenPackage`, `MsiEnumFeatures`, `MsiRecordReadString` to parse MSI tables (InstallExecuteSequence, CustomAction).
- **MSIX/AppX**: use OPC readers / `IAppxPackageReader` to enumerate manifest, capabilities.
- **Electron/Squirrel**: unwrap `.nupkg`, parse embedded `RELEASES`, inspect delta packages & `Update.exe` behavior.
- **Node/Python/Rust pkg installers**: intercept when called from official bootstrappers; parse embedded scripts or sidecar JSON/JS.

macOS

- **.pkg / .mpkg**: `pkgutil --expand` to extract BOMs, payloads, and `Scripts` (preinstall/postinstall).
- **Notarization check**: read ticket, but do not trust as security—just metadata.
- **EndpointSecurity + AMFI hooks** during live supervise (later section).

Linux

- **.deb**: `ar x` → `control.tar.*` / `data.tar.*`; parse `preinst`, `postinst`, `prerm`, `postrm`.
- **.rpm**: `rpm2cpio` → extract; parse `%pre`, `%post`, `%trigger*`.
- **Script interpreters**: sh/dash/bash/python/perl/lua; detect and extract inline heredocs, obfuscated blobs.

Universal

- **Heuristics** to classify installer framework (MSI/NSIS/Inno/Squirrel/pkg/rpm/deb/custom).
- **Network quine**: override resolver/proxy in sandbox so any URL fetch hits a **controlled mirror** (records request without exfiltration).

---

2) Static Analysis Pipeline

1. **Unpack/Decompress** all layers: nested archives, signed containers, script payloads.
2. **De-obfuscate**:
   - String tables, XOR/RC4 patterns, `eval()` chains, compressed JS/Lua chunks.
   - Macro expansion (where applicable), bytecode decompilation (NSIS, V8 snapshot heuristics).
3. **AST & IR Generation**:
   - Build per-language AST (bash/posix, PowerShell, JS, Python).
   - Normalize to a **Unified Install IR (UIIR)** with ops: `FS_WRITE`, `REG_SET`, `SERVICE_CREATE`, `KEXT_INSTALL`, `SYSTEMD_ENABLE`, `LAUNCHD_PLIST`, `SCHEDULE_TASK`, `NET_CONNECT`, `DOWNLOAD`, `DRIVER_INSTALL`, `BPF_ATTACH`, etc.
4. **Capability Graph**:
   - Nodes: resources (files, keys, services, interfaces, domains).
   - Edges: actions with required privilege and persistence class.
5. **Policy Pre-Check**:
   - Hard blocks (e.g., `DRIVER_INSTALL` for category “note-app”).
   - “Why needed?” prompts embedded into DTC if policy threshold crossed.

---

3) Behavioral Modeling & “Intent” Check

- **Category classifier** (LLM + heuristics) labels app: “video player”, “IDE”, “driver pack”.
- Match UIIR against **Norm Profiles** per category:
  - Expected: codecs write to `Program Files`, optional shell extension, auto-update (ask).
  - Outliers: kernel driver, credential provider, outbound constant telemetry for offline tool.
- **Intent consistency**: compare vendor claim text (from installer strings/website) vs. observed plan. Mismatch raises risk.

---

4) Sandboxed Dry-Run (Pre-consent)

Run the installer **unmodified** but **redirected**:

- **Filesystem**: overlayfs (Linux), union filter driver (Windows), APFS snapshots (macOS).
- **Registry** (Windows): hive virtualization per process.

- **Services/Drivers**: simulate service manager; stub driver load calls (return success, log).
- **Network**: force DNS→sandbox proxy; record SNI, URLs, cert chains; **return canned bytes** so logic proceeds.
- **Schedulers**: virtualize Task Scheduler/crontab; record entries, do not persist.
- **Kernel/Elevations**: intercept `ShellExecute("runas")`, `NT*` calls, `seteuid`, `AuthorizationExecuteWithPrivileges` to map requested elevations.

Outputs:

- **Full Install Plan** (ordered ops + diffs).
- **Secret fetches** (downloaded payload hashes).
- **Persistence points** (autoruns/services/drivers).
- **Telemetry endpoints**.

---

5) Dynamic Trust Certificate (DTC) & UX

- **Plain English** (example):
  - "Will create 142 files in `C:\Program Files\AcmeEditor\`, add a background updater (auto-start), and contact `updates.acme.com` every 6 hours. No drivers. Requests admin to modify system fonts."
- **Controls**:
  - `[Install as-is]` `[Install without updater]` `[Sandboxed only]` `[Block]`
- **For enterprises**:
  - Non-interactive policy mode: thresholds map to automatic decisions.

---

6) Transactional, Supervised Install (Post-consent)

When the user/org allows (possibly with pruning), AI-OS executes the plan **atomically**:

- **Snapshot**: VSS (Windows), APFS snapshot (macOS), LVM/Btrfs/ZFS (Linux).
- **Live Mediation**:
  - **Windows**: ETW + minifilter FS driver + registry callback + WFP (network) to allow/deny per op in real time.
  - **Linux**: eBPF LSM hooks (openat, setxattr, bpf_map_update), fanotify, seccomp-notify for brokered syscalls, netfilter.

    - **macOS**: EndpointSecurity framework (ES events), NetworkExtension, AMFI/SMAppService checks.
- **Policy enforcement**:
    - Strip “extras” if user toggled off (updater, toolbars).
    - Block unexpected new endpoints discovered during live run.
- **Rollback**: if anything diverges from the approved plan, abort + revert snapshot.
- **Attestation**: sign the final audit with device key; bind DTC → actual result hash.

---

7) Adversarial Tactics & Countermeasures

- **Encrypted second-stage payloads**: detect staged decryptors; force decryption in sandbox; record keys via API interception.
- **Anti-VM / environment checks**: provide *real* machine fingerprints via broker to keep behavior honest while maintaining containment.
- **JIT codegen** (V8/JS, .NET, JITed PowerShell): instrument loader APIs; capture IR at JIT boundaries.
- **Time bombs / user-action gated steps**: automate UI actions in dry-run with recorded user intent tokens.
- **Self-delete / self-hide**: FS overlay prevents real deletion; all writes land in diff layer.

---

8) Minimal Install Plans (“Surgical Mode”)

AITA can propose a **pruned plan**:

- Remove updaters; keep manual update via AI-OS repo.
- Replace opaque downloads with **mirrored, hashed artifacts**.
- Block system-wide hooks unless app category justifies them.
- Output: **Reproducible Minimal Plan** (YAML/JSON), signed; can be re-applied later non-interactively.

Schema sketch (YAML):

```
package: com.acme.editor
version: 5.2.1
hash: sha256:...
allow:
```

```
  - FS_WRITE: ["%ProgramFiles%/AcmeEditor/**"]
  - REG_SET: ["HKCU\\Software\\AcmeEditor\\**"]
deny:
  - SERVICE_CREATE
  - SCHEDULE_TASK
  - DRIVER_INSTALL
network:
  allow_domains: ["updates.acme.com"]
  frequency_hours: 168
post_install_tests:
  - launch: "AcmeEditor.exe --version"
  - verify_files: ["AcmeEditor.exe", "lib/core.dll"]
```

---

9) Developer/Vendor Path (Carrot + Stick)

- **Carrot**: Vendors ship a **Machine-Readable Install Manifest (MRIM)** that AITA can verify → green path (faster install, better rating).
- **Stick**: If behavior deviates from MRIM or category norms, DTC flags and org policies auto-block.
- **Supply-chain**: accept SBOMs, SLSA provenance, and **repro-build attestations**; include them in DTC.

---

10) Privacy & Governance

- All analysis runs **locally** by default.
- If cloud assist is enabled, only **hashed features and redacted tokens** leave the device; never raw scripts or user data.
- Users can export DTC + Audit as evidence for vendor disputes or compliance.

---

11) Performance Strategy

- Parallel unpack + heuristics by file type.
- IR generation is cached by hash; common installers become near-instant.

- Dry-run uses fast overlay; typical <1–3× installer runtime.
- Live supervise only when needed; trusted green-path vendors skip heavy hooks.

---

12) Implementation Notes by Installer Family

- **MSI**: parse `InstallUISequence`, `InstallExecuteSequence`, `CustomAction` (DLL/EXE/script). Simulate `StandardActions` (CostFinalize, InstallValidate). Flag `Deferred` CAs running as `LocalSystem`.
- **NSIS**: decompile script (`!include` resolution), inspect `ExecShell`, `WriteRegStr`, `WriteUninstaller`, `nsExec`. Watch for `inetc::get` and opaque downloads.
- **Inno Setup**: parse `[Run]`, `[Tasks]`, `[Registry]`, Pascal Script sections. Identify `PrivilegesRequired=admin` implications.
- **Squirrel/Electron**: examine `Update.exe` verbs, `RELEASES`, delta packages, app.asar content; scan JS install hooks for post-install network calls.
- **macOS pkg**: inspect `Distribution` XML, `Scripts` (pre/post), `launchd` plists, `AuthorizationExecuteWithPrivileges` use.
- **.deb/.rpm**: parse maintainer scripts; detect `curl|wget|bash` chains; ensure systemd units aren't auto-enabled without reason.

---

13) Policy Examples (Enterprise)

- "IDE & Editors: deny DRIVER_INSTALL, deny SERVICE_CREATE, allow SCHEDULE_TASK if frequency ≥ 24h and user consents."
- "Finance dept: installers must be MRIM-signed; telemetry blocked unless whitelisted domain & 30-day frequency."
- "Zero-trust endpoints: only Minimal Plans allowed; no network during install."

---

14) Developer API (for Vendors)

Vendors can pre-negotiate an AITA-friendly plan:

```
{
  "package": "com.acme.editor",
  "version": "5.2.1",
```

```
  "capabilities": ["fs_write", "shell_extension"],
  "persistence": [],
  "network": [{"domain": "updates.acme.com", "purpose": "update-check",
"interval_hours": 168}],
  "sbom_ref": "sha256:...",
  "provenance": "slsa.level3",
  "signature": "cose_sign1(...)"
}
```

If runtime deviates → instant yellow/red flag.

---

15) What Makes This Hard (and Why We'll Win)

- Installers are messy, heterogeneous, sometimes intentionally opaque.
- We skip the brittle signature-only model and treat installs like **surgery with a flight recorder**: simulate first, mediate live, rollback if weird.
- The **human-readable intent** is the unlock. Once users see "this wants a kernel driver for a notes app," the market self-corrects.

Optional Appendix: Minimal Pseudocode (Conceptual)

```
def analyze_installer(path):
    blob = unpack_all_layers(path)
    meta = extract_identity(blob)
    lang_units = deobfuscate_and_parse(blob)  # JS/PS/Bash/etc -> ASTs
    uiir = asts_to_uiir(lang_units)           # Unified Install IR
    cat = classify_category(uiir, meta, blob.strings)
    norms = load_norm_profile(cat)

    dry = sandbox_dry_run(blob)               # record ops, endpoints,
privilege asks
    plan = merge_uiir_and_trace(uiir, dry.trace)

    risks = score(plan, norms, policies)
    dtc = build_dtc(meta, plan, risks, cat)
```

```
    user_choice = present_summary_and_controls(dtc)
    if user_choice.approved:
        result = supervised_transaction_install(blob, user_choice.plan)
        audit = sign_audit(result, dtc)
        return audit
    else:
        return {"status": "blocked", "dtc": dtc}
```

AI Trust Agent — README Summary

The Problem:
Traditional trust certificates only confirm *who signed the installer*, not *what the installer actually does*. Supply chain compromises, hidden telemetry, and privilege escalation sneak into "trusted" packages every day.

The Solution:
The **AI Trust Agent (AITA)** for AI-OS scans installer scripts *before* they run. It simulates the full install in a sandbox, summarizes behavior in plain language, and issues a **Dynamic Trust Certificate (DTC)** that validates **intent, privilege use, and hidden persistence** — not just signatures.

Key Features:

- Pre-install clarity: know exactly what the installer will change.
- Contextual trust: is this normal behavior for this app type?
- Transactional installs: atomic, supervised, fully reversible.
- Minimal install plans: strip out hidden updaters, telemetry, or bloat.

This closes the **installer exploit gap** that no AV or PKI solves today. Signed ≠ Safe. AI-OS makes intent the new root of trust.