# ThoxShield: Bloat-Capped Counter-REA Defense for Local-First AI Artifacts

A zero-dependency Rust tool and an agent-team loop harness that give independent developers and local-first AI products the highest-value protections against autonomous reverse-engineering agents — canary attribution, metadata-nesting breakers, and origin binding — at hundreds of bytes per artifact instead of megabytes, with every claim in this paper tied to a measurement.

THOX.ai Research · Authors: Craig Ross, Chief Executive Officer, and Tommy Xaypanya, CTO and Co-Founder (corresponding author), THOX.ai LLC · ORCID iD: 0009-0000-4425-9565 (Xaypanya)

ABSTRACT

Autonomous reverse-engineering agents (REAs) — LLM-driven pipelines that unpack, index, and summarize shipped artifacts — have collapsed the cost of cloning unprotected software: a model file, a WebAssembly module, or an application bundle can be inventoried and reconstructed for cents. Counter-REA tools answer by poisoning, exhausting, and entangling artifacts, but their strongest settings exact a heavy toll on the defender: in our measurements, a current open-source counter-REA tool at its default intensity grows a 93-byte JavaScript fixture to 133,951 bytes (1,440×) and an 8-byte WebAssembly fixture to 3,873,169 bytes. For local-first products — applications and models that ship to user devices, where download size, storage, and update cost are user-facing — that toll is often unacceptable. We present ThoxShield, a bloat-capped subset of counter-REA defense implemented as a single zero-dependency Rust binary (486,808 bytes). ThoxShield keeps the mechanisms whose analyzer cost is high and whose byte cost is low — self-identifying canary metadata, metadata-nesting breakers aimed at REA's published analyzer budgets, and a SHA-256 origin binding — and refuses, by construction, to add more than a configurable cap (default: the larger of 4 KiB or 5%) to any artifact. On reference fixtures it adds 736 bytes to a GGUF model, 578–619 bytes to a Safetensors model (fixture-dependent), and 303 bytes to a WASM module; and in an independent virtual test lab (Section 8) a production 380 MB GGUF with 290 tensors grew by 736 bytes and re-loaded in the reference GGUF reader with every tensor byte stream identical. Because modern releases are produced by teams of agents, ThoxShield also ships a lightweight loop harness in which five generic roles — coordinator, auditor, hardener, verifier, reporter — run the defense as an auditable loop with a hard separation between producing a shield and verifying it. One disclosure belongs next to every effectiveness statement in this paper, so we make it here as well: the lite shields have not yet been run against a live REA pipeline; the analyzer-budget evidence behind them is upstream-empirical (BLA, against REA) and, in our own lab, simulated and labeled as such. We release ThoxShield as open source under the MIT License so that the community whose software is exposed to REAs can apply the same protections, and we close with the limits of the approach, stated plainly.

CONTENTS

1. Introduction
2. Background: the counter-REA mechanism set
3. Threat model
4. The bloat problem, measured
5. Design: defense under a byte budget
6. The agent-team loop harness
7. Evaluation
8. Independent validation: the virtual test lab
9. What ThoxShield deliberately does not do
10. Limitations and future work
11. Conclusion
References
Artifact availability

Your AI. Your Data. Your Rules.™ · THOX®, ThoxOS™, MeshStack™ and ThoxWork™ are marks of THOX.ai LLC.

| +736 B | +303 B | 0 | 10.1 ms |
|---|---|---|---|
| growth on a production 380 MB GGUF, 290 tensors, reader-verified | bytes added to a WASM module (31 B fixture) | external dependencies in the shipped binary | team loop to verified shields on the fixture tree (one loop) |
| MEASURED | MEASURED | MEASURED | MEASURED |

## 1 Introduction

Local-first AI changes who ships valuable artifacts. Models, inference components, and application logic that once stayed on a server now sit on user devices, in public model repositories, and inside downloadable applications — exactly where an autonomous reverse-engineering agent can read them at machine speed. The defensive question is no longer whether an artifact can be inspected; it is what the inspection costs the inspector, and what raising that cost costs the defender.

Counter-REA research, most completely the open-source BLA project (Blind LLM Agents), answers with three mechanism families: poison (decoys and canaries that mislead summarization), exhaust (payloads that trip analyzer resource limits), and entangle (integrity bindings that punish stripping). BLA is a substantial system: 45,357 lines of TypeScript with a 19,012-line test suite (693 tests passing in our run), a CLI and MCP server, and format shields for more than nineteen artifact types. We adopt its framing and credit it throughout; our contribution is different in kind: a measurement of what its strongest settings cost the defender, a re-implementation of its highest-value-per-byte mechanisms under an explicit byte budget, and a team-loop operating model for running those mechanisms in an agent-built release pipeline.

We claim exactly three things, and no more. (i) Defender-side bloat as a hard, measured design constraint for model artifacts. Scoring obfuscating transformations on a cost axis is 1997 prior art (Collberg, Thomborson and Low [7]); what is new here is applying an enforced byte cap to model-artifact metadata shields and measuring under it. (ii) Producer ≠ verifier separation applied to release shielding: agent loop harnesses are established practice [12], and obfuscation mechanisms are classified prior art [7][9]; the contribution is the application and the verification discipline, not the loop or the mechanisms. (iii) Claims discipline on modeled multipliers, stated as method in Section 4. That obfuscation already blunts LLM decompilers is demonstrated prior art [10], not a finding of this paper.

## 2 Background: the counter-REA mechanism set

Three results published by BLA anchor the design; we attribute them inline because all three come from that one source, not from field consensus. First, the analyzer BLA targets — REA, an open-source MCP server and CLI [2] — enforces hard

resource budgets that BLA names directly: `MODEL_JSON_NESTING` (Safetensors JSON nesting beyond 32 levels) and `MODEL_METADATA_NESTING` (GGUF metadata nesting beyond 32 array frames). These are REA's analyzer budgets, published by BLA, not properties of the formats and not an industry norm: the GGUF specification [4] defines no nesting limit, and the Safetensors format's documented anti-DoS limit is a 100 MB header cap, with `__metadata__` restricted to string-to-string values [5]. In BLA's empirical table, an artifact that crosses one of REA's budgets is downgraded from complete to limited examination or rejected outright. Second, the model-format trips are byte-cheap: in BLA's own empirical table, a GGUF file grows from 24 bytes to about 7.7 KB and a Safetensors file from 128 bytes to about 10.1 KB to trip their respective limits. Third, the exhaustion mechanisms for code artifacts are byte-expensive by design — BLA's stated rationale is that they work by flooding the analyzer's context, and the flood is stored in the artifact. A bloat-capped tool is therefore not a weaker version of the same tool; it is a selection of the mechanisms on the cheap side of that asymmetry.

The analyzer class is real and capable within limits, and the threat model in this paper is anchored to those published capability results rather than to REA alone: GPT-4 is proficient at understanding and explaining decompiled code, though weaker at detailed security analysis [11]; and in the LLM4Decompile study, both the LLM approach and Ghidra failed to effectively decompile obfuscated code [10]. REA [2] remains the one concrete, verified analyzer our breakers are designed against.

## 3 Threat model

**Protected assets.** The assets are the artifacts a local-first vendor ships — model files (GGUF, Safetensors), WebAssembly components, and application bundles — the development investment they embody, and the integrity of the release record (manifests and receipts) that describes them. The artifacts themselves sit on user devices and in public repositories, readable by anyone with the file; the confidentiality of their contents is not an asset this paper claims to protect.

**Adversary class A: the budget-limited automated REA.** An LLM-driven pipeline [2] that unpacks, indexes, and summarizes artifacts at machine speed and at scale, enforcing published resource budgets (32 nesting levels for GGUF metadata and Safetensors JSON, per BLA's empirical work). Its goal is a complete inventory and reconstruction of many artifacts at minimal cost per artifact. This is the adversary the lite shields are designed to tax: the nesting breakers trip its budgets, and the canaries pollute its summaries with self-identifying markers.

**Adversary class B: the stripper-aware analyst.** An analyst (human-guided or agent-guided) who knows the shields are present and works from the format specifications. Section 8 measures this adversary directly: a roughly 60-line spec-only stripper recovers the originals in under a millisecond. Against class B the shields provide no resistance; what survives is attribution, if the canaries are left in place, and the origin binding — which, stated plainly, also serves this adversary as a recovery oracle and a provenance fingerprint (Section 8).

**Adversary class C: the determined human.** An interactive analyst with time and tooling. Theory and survey evidence cap what any client-side shield can do here: virtual-black-box obfuscation is impossible in general [8], and interactive analysis defeats many obfuscations [9]. No claim in this paper is made against class C.

**Defender assumptions.** The defender owns the artifacts being shielded (ThoxShield's policy is own-artifacts-only); shields are applied before release, inside the release pipeline; verification is performed by a process independent of the one that produced the shield (Section 6); and the analyzer budgets the breakers target are as published by BLA's empirical work against REA — if those budgets change, the breakers' visible +3 margin (Section 10) is the first thing to re-check.

**Out of scope.** Secrecy of artifact contents; tamper resistance; runtime adversaries on the device (including debugger deception, excluded by policy in Section 9); prompt-injection layers aimed at the analyzing agent (excluded in Section 9); and the protection of real secrets, which belong on trusted infrastructure, not inside shipped artifacts (Section 10).

## 4 The bloat problem, measured

We built the upstream tool from source (Node v24.22.1), ran its full test suite (121 files, 693 passed, 4 skipped, 152.8 s), and hardened a fixture directory at its default bankrupt intensity. Table 1 records the growth. We also record a claims-discipline finding: the tool's showcase benchmark reports aggregate token amplification of 639.5× in its README and 612.5× in its state document, and printed 1067.3× in our run. These are modeled estimates from a token estimator, not measurements of an analyzer process — BLA itself labels them "modeled," under a README benchmark section headed "Internal Estimates" — and their disagreement across versions is precisely why this paper reports no amplification figure as a result. We also state a gap our literature survey found rather than papering over it: no independent, non-BLA measurement of agent cost amplification against model artifacts was found; the only figures in this space are BLA's, split as above into an empirical REA-outcome table and modeled estimates.

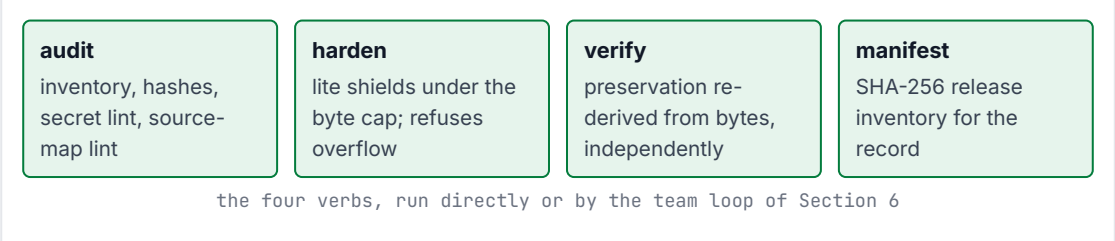
| Fixture | Original | After upstream harden (bankrupt) | Growth |
|---|---|---|---|
| JavaScript | 93 B | 133,951 B | 1,440× |
| Python | 44 B | 39,795 B | 904× |
| GGUF model | 104 B | 7,584 B | 73× |
| WebAssembly | 8 B | 3,873,169 B | 484,146× |

Table 1. Measured defender-side cost of full-intensity counter-REA hardening on small fixtures (evidence class: MEASURED). Small fixtures exaggerate ratios; the absolute bytes are the load-bearing figures — a multi-megabyte addition to a WASM component is a product decision, not a detail.

## 5 Design: defense under a byte budget

ThoxShield is a single Rust binary with zero external crates (SHA-256 and the JSON parser are implemented in-tree), organized as four verbs and a harness. **Audit**

walks a release tree, detects formats by magic bytes, hashes every artifact, and lints for plaintext secrets and shipped source maps. **Harden** applies light shields to the three formats whose trips are byte-cheap: GGUF (canary key-values plus a 35-deep nested ARRAY metadata entry, original key-values preserved verbatim and in order), Safetensors (canary `__metadata__` strings — the format allows string-to-string metadata only [5] — plus a nesting breaker carried on the first tensor descriptor, tensor payload untouched), and WASM (an inert custom section carrying a canary and the original file's SHA-256; the original bytes remain a strict prefix). The WASM shield's payload preservation rests on the specification itself: for custom sections, §5.5.3 of the WebAssembly Core Specification provides that errors in that data "must not invalidate the module" [6]. **Verify** re-derives the preservation properties from the bytes, not from the hardener's report. **Manifest** emits the SHA-256 inventory a release record needs. The bloat cap is enforced inside harden: an artifact whose shield would exceed the cap is refused, not trimmed silently. Canaries are synthetic and self-identifying; the enclave-manifest canary states in its own text that no enclave is present, because a privacy-first tool must not lie to the humans who later audit the artifact.




the four verbs, run directly or by the team loop of Section 6

## 6 The agent-team loop harness

Agent-built pipelines need the defense to run the way the pipeline runs: as a team, in a loop, with receipts. The harness (`thoxshield team`) defines five generic roles — coordinator, auditor, hardener, verifier, reporter — and no personas, frameworks, or model calls: each role is one of the CLI verbs run as a separate process, coordinated through a workorder and a state file that any agent can resume. The loop runs audit → lint gate → harden → verify → report, hardening from the original inputs on every iteration, and stops on exactly three conditions: all artifacts verified, a blocking lint finding (a plaintext secret is not a shielding problem and must stop a release), or exhaustion of the iteration budget, reported as failure. Iterative LM-agent tool-use loops are mature prior art (SWE-agent [12]); this harness claims no novelty as a loop. Its contribution is the application: producer ≠ verifier separation as a release gate for shields. The verifier's verdict, never the hardener's, closes the loop. The same protocol is published in prose (`harness/WORKORDER.md`) so an external agent — a coding assistant, a CI system, a human — can run the identical loop without the harness binary.




Figure 1. The team loop. The verifier is a separate process from the hardener; its verdict, never the hardener's report, closes the loop.

## 7 Evaluation

All figures are from `exp/results/summary.json`, the single evidence source. The ThoxShield fixture, preservation, and loop figures are re-derived from this paper's artifact pack by `exp/run_experiments.py`; the upstream figures and the 232-file audit timing are recorded measurements from the same session, with their commands noted in `summary.json`. On the reference fixtures, hardening added 736 B (GGUF, current fixture; the v0.1.0 fixture measured 746 B of inserted metadata before the alignment fix of Section 8), 578–619 B (Safetensors, fixture-dependent), and 303 B (WASM) — two to four orders of magnitude below the full-intensity figures of Table 1 on comparable fixtures. Preservation was checked independently of the tool's own verifier: a Python parser confirmed GGUF key preservation, tensor-count stability, byte-identical post-metadata content, and a nesting depth of 35; a Python JSON parse confirmed Safetensors header validity, byte-identical tensor payload, and a topology nesting depth of 36; Node's `WebAssembly.validate` accepted the WASM module before and after, and the hardened module compiles with its exports unchanged. The audit verb inventoried a 232-file tree in 480.4 ms. For production scale: hardening the 380 MB production GGUF of Section 8 took 4.7 s wall time (lab item HRD-02, Section 8). GGUF growth is 746 B of inserted metadata adjusted by alignment re-padding (Section 8), an observed band of ±31 B; Safetensors additions are fixture-dependent (578–619 B observed) because the header is rewritten. On byte-identical fixtures, the same-fixture comparison is direct: a 1,328 B spec-correct GGUF grows to 8,875 B under upstream bankrupt and 2,096 B under ThoxShield lite; a 36 B WASM module grows to 6,880,103 B under upstream bankrupt and 339 B under ThoxShield lite. The team loop reached verified shields on a clean three-artifact tree in one loop (10.1 ms in the reference run; 10–50 ms across lab runs), and, on a tree seeded with a planted flag marker and a shipped source map, stopped before hardening, named both artifacts, wrote its report, and exited non-zero — the gate behaving as designed. The tool's own unit tests cover SHA-256 against published vectors and JSON round-tripping.

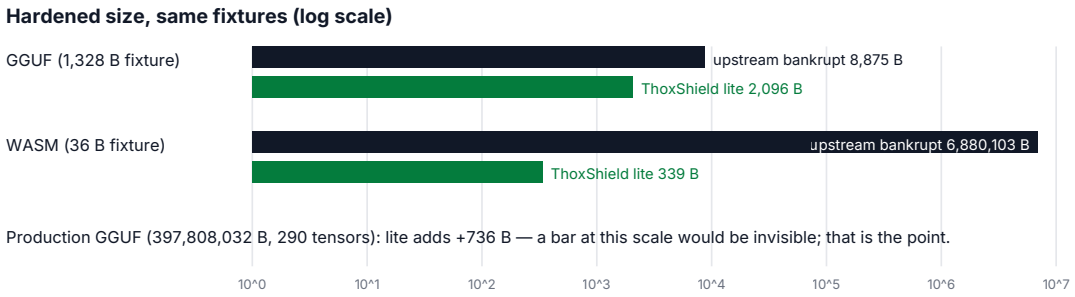



Figure 2. Same-fixture hardened sizes (lab item COMP-01, MEASURED) and the production-scale lite delta. Upstream's full-intensity settings grow the WASM fixture by five orders of magnitude; the lite shields stay in hundreds of bytes.

> PRODUCT IMPLICATIONS
>
> For a local-first vendor, the practical consequence is that counter-REA defense can move from a specialist, per-release decision to a default pipeline step: audit in CI on every candidate, light shields on every published model and WASM component, independent verification in the release record. The cost line moves from megabytes and a Node toolchain to hundreds of bytes and one static binary — small enough to protect community and open-source releases with the same pipeline used for commercial ones.

## 8 Independent validation: the virtual test lab

The v1.0.0 edition of this paper rested on the authors' own parsers. For v1.1.0 the THOX Research Team reviewed an independently produced test lab and its claims (the v1.1.x editions carry that rebuilt evidence forward; v1.1.2's changes are presentational and packaging — the template rebuild, the threat-model consolidation of Section 3, and the reproducibility scoping of Section 7 — not new measurements); the review found eight defects in that lab's evidence (invalid GGUF fixtures without tensor-info records, a hardcoded pass, a simulated analyzer that compared depths instead of enforcing its budget, self-verification labeled as independence, and a bundled binary that matched no recorded build). We rebuilt the lab to the review's standard: spec-correct fixtures, computed passes only, a budget enforced during traversal, self-checks labeled as self-checks, and the canonical binary's SHA-256 recorded in the evidence pack. The rebuilt lab runs twelve evidence items; ThoxShield's own `verify` is labeled SELF-CHECK throughout, and independence is claimed only where the reference implementations provide it: gguf-py, the safetensors library, and Node. The lab code ships in this pack (`exp/vlab.py`).

**The lab earned its keep immediately: it caught a critical defect in ThoxShield v0.1.0.** The first hardener copied the post-metadata bytes verbatim, which shifts a GGUF file's tensor data section off its alignment boundary (`general.alignment`, default 32); reference readers locate tensor data on that boundary, so on a production 380

MB GGUF all 290 tensors read misaligned through gguf-py. The v1.0.0 checks could not see this because they compared raw tail bytes, not reader-visible tensors. ThoxShield v0.1.1 re-pads after the tensor-info region, and its `verify` now checks tensor-info identity, data-section identity, and data-start alignment. Post-fix, the production GGUF grows by 736 B and gguf-py loads it with every tensor byte stream identical; hardening it twice is byte-identical; a 1.7 MB vocab GGUF grows by 777 B. The hardened Safetensors loads in the reference library with all tensors numerically identical. The simulated analyzer, now enforcing REA's 32-level budget (`MODEL_METADATA_NESTING`) during the walk, aborts on the nesting breaker at depth 33 (SIMULATED; a live REA run remains future work). Depth figures in this paper use stated roots: GGUF breaker depth is 35 nested ARRAY frames; the Safetensors header measures 38 by whole-header JSON depth (root = 1) and 36 for the topology value alone — the two conventions behind the inconsistent figures in earlier drafts.

**Strip test.** The lab also measures removal: a roughly 60-line stripper written from the format specifications, with no ThoxShield code, recovers the GGUF and WASM originals byte-exactly and the Safetensors payload exactly, in under a millisecond per artifact. We report this as a headline limitation, not a footnote: the lite shields are tripwires for budget-limited automated analyzers and carriers of attribution canaries. They are not, and are not sold as, tamper resistance. The origin binding cuts both ways: the embedded SHA-256 of the original tells a stripper exactly when a recovery is correct, and lets anyone match a shielded file against public unshielded models — the attribution mechanism and the fingerprint are the same bytes.

## 9 What ThoxShield deliberately does not do

Three exclusions are policy, not backlog. (1) **No prompt-injection layers.** Some counter-REA systems plant instructions aimed at the analyzing agent, including imitations of legal and regulatory notices. The mechanism is a known attack class, not a gray area: retrieved content blurs the data/instruction boundary for LLM-integrated applications (indirect prompt injection [13]). A transparency-first vendor does not ship fake regulatory text inside user artifacts. (2) **No runtime deception of users.** Shields that detect a debugger and serve synthetic telemetry deceive the artifact's owner as readily as its adversary; they belong, if anywhere, in sealed training challenges, not in shipped products. (3) **No exhaustion flooding.** Flooding is the mechanism whose cost lands on the user's disk, bandwidth, and battery; the cap exists so it cannot be enabled by accident. Each exclusion narrows the tool; we consider the narrowing a feature and record it so adopters can make their own judgment with open eyes. Underlying all three: ThoxShield claims cost and attribution, never secrecy — a ceiling set by theory, not modesty, since virtual-black-box obfuscation is impossible in general [8] and the standard survey concludes that interactive analysis defeats many obfuscations, so protection by obfuscation "remains highly challenging" [9].

## 10 Limitations and future work

First, the lite shields' nesting breakers are aimed at analyzer limits that upstream work verified empirically against a live REA pipeline; a live REA run against ThoxShield's own output has not yet been performed and is the first item of future work, together with a sealed-vs-unsealed exposure study. Production-scale integrity evidence is no longer future work: Section 8 covers a 380 MB production model. Second, coverage is intentionally narrow: JavaScript, Python, SQLite, and archive formats are audit-only in this tool, and teams needing AST-level weaving should use the upstream system; likewise, only the GGUF evidence is production-sourced — the Safetensors and WASM results rest on spec-correct fixtures, not real-world artifacts of those formats. Third, manifests are unsigned in this version; Ed25519 signing of the manifest is designed and scheduled. Fourth, removal is cheap and now measured: the strip test of Section 8 recovers originals in under a millisecond, so any deployment must assume the shields can be removed by an analyst who knows they are present; what they tax is the unmodified, budget-limited automated pass, plus they carry attribution if left in place. The breaker's margin over REA's published analyzer budget (+3 levels) is visible in every shipped artifact, so analyzer authors can raise their budgets at zero cost; open-sourcing accepts that bounded half-life deliberately. Fifth, the tool loads artifacts fully in memory: measured peak RSS is about twice the artifact size (1,026 MiB on a 512 MiB GGUF across audit, harden, and verify), so multi-GB models need matching RAM until a streaming implementation lands. Sixth, no client-side shield, including this one, creates secrecy against a determined human analyst [8][9]; ThoxShield buys cost and attribution, and real secrets belong on trusted infrastructure.

## 11 Conclusion

Counter-REA defense does not have to be heavy to be real. Measured carefully, its most efficient mechanisms cost hundreds of bytes; capped honestly, they fit the products and communities that need them most; and run as a verified team loop, they become a release habit rather than a specialist event. ThoxShield is our attempt at that habit, released openly so others can adopt, audit, and improve it. Your AI. Your Data. Your Rules.™

## References

[1] BLA Contributors. BLA: Blind LLM Agents (bla-agents v3.2.4). Open-source counter-REA toolkit, MIT License. https://github.com/qybaihe/bla

[2] Morluto. REA: Reverse Engineer Anything. Open-source MCP server and CLI (npm: rea-agents), MIT License. https://github.com/morluto/rea

[3] National Institute of Standards and Technology. Secure Hash Standard (SHS), FIPS PUB 180-4, 2012 (reissued 2015; the reissue changed only the Applicability Clause).

[4] ggml-org. GGUF Format Specification, docs/gguf.md, format version 3. https://github.com/ggml-org/ggml/blob/master/docs/gguf.md

[5] Hugging Face. Safetensors format specification. https://github.com/huggingface/safetensors (documented limits: 100 MB header cap; `__metadata__` is a string-to-string map only)

[6] W3C WebAssembly Community Group. WebAssembly Core Specification, §5.5.3 (Custom Sections): errors in custom-section data "must not invalidate the module."

[7] Collberg, C., Thomborson, C., and Low, D. A Taxonomy of Obfuscating Transformations. Technical Report 148, Department of Computer Science, University of Auckland, 1997. https://hdl.handle.net/2292/3491

[8] Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S., and Yang, K. On the (Im)possibility of Obfuscating Programs. CRYPTO 2001; journal version: J. ACM 59(2), Article 6, 2012. DOI 10.1145/2160158.2160159

[9] Schrittwieser, S., Katzenbeisser, S., Kinder, J., Merzdovnik, G., and Weippl, E. Protecting Software through Obfuscation: Can It Keep Pace with Progress in Code Analysis? ACM Computing Surveys 49(1), 1–37, 2016. DOI 10.1145/2886012

[10] Tan, H., Luo, Q., Li, J., and Zhang, Y. LLM4Decompile: Decompiling Binary Code with Large Language Models. arXiv:2403.05286, 2024.

[11] Pordanesh, S. and Tan, B. Exploring the Efficacy of Large Language Models (GPT-4) in Binary Reverse Engineering. arXiv:2406.06637, 2024.

[12] Yang, J., Jimenez, C., Wettig, A., Lieret, K., Yao, S., Narasimhan, K., and Press, O. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. arXiv:2405.15793, 2024 (NeurIPS 2024).

[13] Greshake, K., Abdelnabi, S., Mishra, S., Endres, C., Holz, T., and Fritz, M. Not What You've Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection. arXiv:2302.12173, 2023 (AISec '23).

## Artifact availability

Ross, C. and Xaypanya, T. ThoxShield: Bloat-Capped Counter-REA Defense for Local-First AI Artifacts, v1.1.2 — experiment code and evidence ship in the artifact pack: `exp/run_experiments.py` re-derives the ThoxShield fixture, preservation, and loop figures from the pack (a thoxshield binary is the one prerequisite); `exp/vlab.py` is the virtual test lab of Section 8; `exp/results/summary.json` is the single evidence source for every number in this paper. The ThoxShield tool (v0.1.2) is open source under the MIT License; this paper is licensed CC BY 4.0.