TECHNICAL PAPER

Arkheionx: A Local Review Map for DeFi Smart-Contract Repositories

Value paths, assumptions, and missing tests before human review.

Yudistira Putra

PROJECT	Arkheionx
VERSION	v4.0.0
DATE	June 7, 2026
WEBSITE	https://arkheionx.dev
REPOSITORY	https://github.com/Yudis-bit/DeFi-Exploit-PoCs

ABSTRACT

Arkheionx is a local, static review map for DeFi smart-contract repositories. From a Solidity or Foundry checkout it produces a deterministic map of where value enters, moves, and exits; the trust assumptions that guard each path; the value-sensitive functions that have no matching test; and a suggested local proof direction for each gap. It runs entirely on local source: no RPC, no live-chain calls, no private keys, no exploit automation. Arkheionx does not confirm vulnerabilities, assign final severity, or prove that a protocol is safe, and it is not a substitute for a professional audit. Its purpose is narrower, and useful: to make the review surface explicit and repeatable before a human spends attention on it.

Local and static only. No RPC, no live-chain calls, no exploit automation. Human review required.

1. Abstract

Arkheionx is a local, static **review map** for DeFi smart-contract repositories. From a Solidity or Foundry checkout it produces a deterministic map of where value enters, moves, and exits; the trust assumptions that guard each path; the value-sensitive functions that have no matching test; and a suggested local proof direction for each gap.

It runs entirely on local source files. It does not make RPC or live-chain calls, it does not request private keys or secrets, and it does not automate exploits. Arkheionx does not confirm vulnerabilities, does not assign final severity, and does not prove that a protocol is safe. It is not a substitute for a professional audit.

Its purpose is narrower, and we argue useful: to make the review surface explicit and repeatable **before** a human spends attention on it. This paper states the problem, the design thesis, the v4.0.0 system, and its limits, using real output from the bundled multi-contract demo fixture. A human reviewer always makes the security decision.

2. Motivation

A passing test suite proves one thing: that the tests someone wrote pass. It says nothing about the tests nobody wrote. In DeFi, where a single unguarded value path can move user funds, the more important question is often the inverse of the green checkmark:

> *Which value paths were never tested at all?*

Answering that question is review work, not test work. Before a reviewer can judge a protocol, they reconstruct a mental model of it: which contracts hold or move value, who is allowed to call what, which external calls and accounting steps matter, what each path silently assumes, and which of those paths the existing tests never exercise. Today that model is rebuilt by hand, informally, and differently by each reviewer, every time the code changes.

Foundry is excellent at the job it is built for: compiling contracts and running the tests you write against them. Arkheionx is not a competitor to it, and not a replacement for it. It complements Foundry by helping surface the part Foundry was never meant to answer -- *what the tests do not yet cover* -- so that the reconstruction step is explicit, repeatable, and reviewable. The same applies to linters and static analyzers: they answer "what looks suspicious here?"; Arkheionx answers "where can value move, what guards it, and what did the tests skip?" Both questions are worth asking.

3. Design Thesis

The thesis behind Arkheionx is a single sentence:

> *Security review should begin with a map of value movement and assumptions, not only a list of warnings.*

A flat list of warnings is useful, but it is organized around the tool's suspicions. A review map is organized around the protocol's value. The distinction matters because most real review effort is spent answering "where should I even look first, and what is this path trusting?" -- and that is exactly the context a warning list leaves implicit.

Four commitments follow from the thesis:

- **Review context over verdicts.** The output is material a reviewer reads, not a conclusion they are asked to accept. Arkheionx ranks an *inspection order*, never a severity.
- **Deterministic local output.** The same source produces the same map. Output is plain, inspectable, and diffable, so it can be regenerated and compared across revisions without hidden network state.

- **Value-sensitive paths first.** Functions that move value, hold privilege, or make external calls are surfaced ahead of inert code.
- **Human decision support.** Every artifact ends at a person. Arkheionx makes the question sharper; it does not answer it.

4. System Overview

Arkheionx is a pipeline. Each stage narrows a whole repository down to the few places where a reviewer's attention is worth the most, and hands the decision to a human at the end.

```
repository
    |
    v
review-map         parse contracts, functions, and value signals
    |
    v
value paths        where value enters, moves, and exits
    |
    v
assumptions        the trust conditions that guard each path
    |
    v
test gaps          value-sensitive functions with no matching test
    |
    v
proof direction    a local Foundry scaffold to try by hand
    |
    v
human review       a person decides impact, exploitability, severity
```

- **Discovery / review-map.** Read the local Solidity (and Foundry test) sources, identify contracts and functions, and tag value-relevant signals such as `value-in`, `value-out`, `external-call`, and `privileged`.
- **Value paths.** Connect entries to exits into named paths (for example `deposit -> withdraw`).
- **Assumptions.** Attach the trust conditions each path depends on -- oracle freshness, share proportionality, no reentrancy, standard token behavior, and bounded admin power.
- **Test gaps.** Compare the value-sensitive functions against what the observed tests exercise, and list what is missing.
- **Proof direction.** For each gap, emit a local proof outline a human can scaffold with Foundry.
- **Human review.** A person reads the map and decides. Nothing downstream is automated.

A reader can interpret any review-map run from six recurring fields:

Output field	Meaning	Example
Value path	How value enters, moves, and exits	`Vault: deposit -> withdraw`
Assumption	A trust condition a reviewer must verify	Oracle price is fresh and trusted
Test gap	A value-sensitive function with no matching test	`Vault.withdraw [high; exit]`
`Source: file:line`	Parsed location of the function to open	`src/Vault.sol:63`
Proof direction	A local Foundry scaffold to try by hand	`proof-vault-withdraw`
Human review note	The reviewer decides impact and severity	Human review required

5. Value Paths

A **value path** is a static review path along which assets, shares, accounting state, or privileged control may affect user value. It is a place to look, not a finding.

In the bundled demo fixture, Arkheionx surfaces three value paths. The output below is real, abridged from `arkheionx value-paths examples/vault-strategy-oracle-fixture`:

```
Summary
  Total value paths: 3
  Priority buckets: high 3, medium 0, low 0
  Value exits: 3
  With assumptions: 3
  Test references: 0

Top Value Paths
  1. Strategy: invest -> divest          [high; HEURISTIC; coverage none]
  2. Vault: deposit -> emergencyWithdraw  [high; HEURISTIC; coverage none]
  3. Vault: deposit -> withdraw           [high; HEURISTIC; coverage none]
```

Each path records its entry and exit, the conditions worth checking (caller entitlement, reentrancy ordering, access-control guards), and the assumptions it leans on. `coverage none` and `Test references: 0` are the signal a reviewer cares about here: value leaves the system along these paths, and the observed tests do not exercise the exits. The label `HIGH` is **review order**, not severity, and `HEURISTIC` is a reminder that these are statically inferred, not confirmed behaviors.

6. Assumptions

Every value path trusts something. Arkheionx makes those trusts explicit as **assumptions** -- review prompts a human should verify, never claims that an assumption is violated.

From `arkheionx assumptions examples/vault-strategy-oracle-fixture`, the fixture yields seven assumptions across six categories:

```
Summary
  Total assumptions: 7
  Categories: access-control 1, accounting 2, math 1, oracle 1,
              reentrancy 1, token 1
  With related test gaps: 5

Top Assumptions
  1. Admin/owner role is trusted and bounded   [access-control; unverified]
  2. Token and price decimals are normalized   [math; unverified]
  3. Oracle price is fresh and trusted         [oracle; unverified]
  4. External calls cannot re-enter unsafe state [reentrancy; unverified]
  5. Tokens behave like standard ERC20         [token; unverified]
```

Each assumption is marked `unverified` and lists the functions it touches and the test gaps related to it. The point is not that any of these is wrong. The point is that they are the load-bearing beliefs of the protocol, and a reviewer should decide, deliberately, whether each one holds.

7. Test Gaps

A **test gap** is a value-sensitive function for which Arkheionx observed no matching local test. Gaps are produced by comparing the value-relevant functions discovered in the source against the functions the observed tests

exercise. A gap does not mean a bug exists. It means: *inspect this path, or write a local test for it.*

The demo fixture has a green-looking test suite and five gaps. The output below is real, from `arkheionx test-gap-map examples/vault-strategy-oracle-fixture`:

```
Summary
  Total test gaps: 5
  Priority buckets: high 5, medium 0, low 0
  Evidence-linked: 0
  With proof suggestions: 5

Top Test Gaps
  1. PriceOracle.setPrice [high; admin]   Source: src/PriceOracle.sol:27
  2. Strategy.divest      [high; exit]    Source: src/Strategy.sol:44
  3. Vault.emergencyWithdraw [high; exit] Source: src/Vault.sol:72
  4. Vault.setOracle      [high; admin]   Source: src/Vault.sol:103
  5. Vault.withdraw       [high; exit]    Source: src/Vault.sol:63
```

Two fields deserve an honest distinction, because they are easy to conflate:

- **`Source: <file>:<line>`** is the parsed location of the function, taken directly from the source. It is present so a reviewer can open the exact line. These references are real: `src/Vault.sol:63` is `withdraw`, `:72` is `emergencyWithdraw`, `:103` is `setOracle`.
- **`Evidence-linked: 0`** counts something different: gaps linked to *generated* proof or trace artifacts. It is zero until you actually run a local proof or trace. A parsed source location is not the same as executed evidence, and Arkheionx reports them separately rather than blurring the two.

8. Proof Direction

For each test gap, Arkheionx suggests a **proof direction**: a local outline a human can turn into a Foundry test. It is a planning artifact. It is not exploit automation, it does not execute anything, it makes no live-chain call, and it is not a bounty submission.

A proof direction has four parts -- set up state, exercise the path, assert the property, then review the result by hand. The excerpt below is real, one of five from `arkheionx proof-plan examples/vault-strategy-oracle-fixture`:

```
Vault.withdraw - Prove Vault.withdraw under value-sensitive scenarios
  Objective: Value leaving via withdraw is fully accounted and cannot
             exceed the caller's entitlement.
  Setup:     Open the repo in a local Foundry project. Deploy the
             contract with mock tokens (and a mock price feed if
             oracle-dependent).
  Action:    Exercise Vault.withdraw across the suggested scenarios
             with adversarial inputs.
  Assertions: withdrawal boundary; zero amount; full balance;
              partial balance.
  Related:   gap-vault-withdraw, asm-standard-erc20
```

The command reports `Executed proofs: 0 (planning only)`. Arkheionx writes the outline; a person writes and runs the test, reads the result, and decides what it means.

9. Command Interface

The v4.0.0 stable surface is seven commands. Two orient you; five build and read the review map.

Command	Purpose	Output
`arkheionx version`	Package and milestone metadata	Version and next step
`arkheionx doctor`	Check local environment and project layout	Environment, project, safety status
`arkheionx review-map .`	Build the local review map (first run)	Ranked inspect-first list, artifacts
`arkheionx value-paths .`	Where value enters, moves, exits	Ranked value paths with assumptions
`arkheionx assumptions .`	Trust conditions each path depends on	Grouped assumptions, related gaps
`arkheionx test-gap-map .`	Value-sensitive functions with no test	Ranked gaps with `Source: file:line`
`arkheionx proof-plan .`	Local Foundry proof-scaffold directions	Per-target objective, setup, asserts

`doctor` reports environment and safety state before any review. Its output is deliberately plain about the boundaries:

```
Status   WARN
Version  4.0.0
Python   3.12.3

Safety
LOCAL  RPC calls           disabled by default
LOCAL  Live-chain actions  disabled
LOCAL  Private keys        not requested
LOCAL  Analysis            authorized local/static only
```

The `WARN` here is honest, not an error: it reports that the current directory is not itself a Foundry project, so results are source-level rather than compiler-confirmed. Run inside a Foundry repository for compiler-confirmed context.

Beyond the stable seven, the source tree carries advanced and experimental commands (for example `scan`, `test-plan`, `search`, and the workbench verbs). They are not the canonical first run and are not central to this paper.

10. Demo Fixture

The repository ships a small multi-contract fixture at `examples/vault-strategy-oracle-fixture` so the workflow can be run and verified end to end. It is local/static demo code only: not production code, not a deployable recommendation, and not an exploit target. There is no planted bug. The lesson comes from leaving real value paths untested, not from hiding a flaw.

```
   asset: MockToken (ERC20-like), moved along every edge below
        |
        v
   deposit ---> [ Vault ] --- rebalance ---> [ Strategy ]
        ^           |                            |
        |           | withdraw                   | harvest
        |           | emergencyWithdraw          v
      caller        v                       [ PriceOracle ]
                value exits                 admin: setPrice
                                            Vault.setOracle repoints it
```

- **Vault** -- the share vault. `deposit` pulls value in; `withdraw` and the privileged `emergencyWithdraw` push value out.
- **Strategy** -- `invest` pulls value in from the vault; `divest` returns it; `harvest` marks gains using the oracle price.
- **PriceOracle** -- an admin-set price source; `getPrice` guards accounting.
- **MockToken** -- the ERC20-like asset moved between the contracts above.

The test, `test/Vault.t.sol`, covers the `deposit` entry path and a price sanity check. It deliberately leaves the value **exits** (`withdraw`, `emergencyWithdraw`, `divest`) and the **admin** setters (`setOracle`, `setPrice`) untested. Running the canonical first command surfaces exactly that gap. The output below is real, abridged from `arkheionx review-map examples/vault-strategy-oracle-fixture`:

```
OK    Inspect repository  4 source files, 1 test files
OK    Map review surface  3 contracts, 14 functions, 3 value paths, 5 test gaps
OK    Write artifacts     11 files

Inspect first
1  HIGH   Strategy.divest          Signals  external-call, value-out
2  HIGH   Vault.emergencyWithdraw  Signals  external-call, privileged, value-out
3  HIGH   Vault.withdraw           Signals  external-call, value-out
```

A green test run, three untested value exits, and a ranked place to start. That is the whole idea, on more than one contract.

11. Safety Model

The boundaries are not an afterthought; they are part of the design. Arkheionx operates only on authorized local source.

- Local and static only. No RPC. No live-chain calls.
- No exploit automation. No transaction broadcasting. No auto-submit.
- No private keys, seed phrases, or secrets are requested or required.
- No severity assignment. Priority is review order, not severity.
- Not an audit, and not a substitute for one.
- A human reviewer is required.

Arkheionx does	Arkheionx does not
Read local Solidity / Foundry source	Make RPC or live-chain calls
Map value paths, assumptions, test gaps	Confirm or exploit vulnerabilities
Show `Source: file:line` for each gap	Assign final severity

Arkheionx does	Arkheionx does not
Suggest local proof directions	Execute proofs or broadcast transactions
Produce deterministic local artifacts	Request private keys or secrets
Support a human reviewer's decision	Replace a professional audit or human review

Run Arkheionx only on repositories you are authorized to review. Do not treat generated artifacts as standalone evidence of exploitability, safety, or impact.

12. Use Cases

Pre-audit readiness. A builder runs Arkheionx before an external review to find untested value paths, document the assumptions each path depends on, write the missing local tests, and hand a reviewer a clearer surface. The goal is to spend the audit on judgment, not on reconstruction.

Bug bounty triage. A researcher uses the ranked map to choose where to look first, turns each test gap into a manual hypothesis, and validates it independently with their own local tests. Arkheionx output is a starting point, never a submission: do not submit it as a vulnerability by itself.

Reviewer onboarding. A reviewer new to a codebase uses the map to understand value flow, roles, and assumptions quickly, building review context before the deep read instead of during it.

13. Limitations

Arkheionx is deliberately modest about what it knows.

- It is static heuristic analysis, not execution. Signals are inferred from source, not proven.
- Cross-contract value flow is surfaced today as per-contract paths; deep end-to-end tracing across contracts is limited.
- Fixture output demonstrates the workflow; it is not real-protocol proof.
- It does not understand every test semantically; a present test is not always a meaningful one, and absence of a test is not proof of a defect.
- It offers no guarantee of completeness. Output can be noisy and can miss things.
- Human validation is required for every result.

14. V4 Implementation Status

v4.0.0 makes the local review-map workflow the stable, supported public surface. As reported by `arkheionx version`, the package version is `4.0.0`, the current milestone is `v4.0.0`, the next is `v4.1.0`, and the latest separately tagged stable release remains `v3.1.0` (which the source installers and the GitHub Action pin to). Concretely, v4.0.0 includes:

- The stable review-map workflow and its focused views (`review-map`, `value-paths`, `assumptions`, `test-gap-map`, `proof-plan`) plus `version` and `doctor`.
- A packaged build (wheel and sdist) and a command-complete CLI reference.
- The bundled multi-contract demo fixture used throughout this paper.
- `Source: <file>:<line>` references on every resolvable test gap.
- A public documentation site and project docs.

This is a stable workflow surface, not a claim of completeness. The broader "protocol security control plane" remains planned direction, not a shipped runtime.

15. Roadmap

Near-term direction, without dates and without promises:

- An authorized real-protocol case study, to move beyond fixtures.
- Deeper cross-contract value-flow tracing.
- Richer evidence links between gaps and locally generated proof or trace artifacts.
- Higher-quality bug bounty hypotheses from the gap and assumption model.
- More review-oriented output and visualization.

16. Conclusion

Arkheionx does not prove that a protocol is safe, does not confirm vulnerabilities, and does not replace a human auditor. What it does is smaller and concrete: it turns a repository into a local, deterministic map of value-sensitive paths, the assumptions that guard them, and the tests that were never written -- and then it stops, and hands that map to a person. The reviewer still decides where value can move, what is actually at risk, and what it is worth. The aim of this work is only to make that decision better informed, and to make the first hour of review repeatable.

Appendix A: Reproducing this paper's output

Every command excerpt above is real engine output, abridged for length and locked by the project's tests. To reproduce it from a local checkout:

```
python3 -m pip install -e .
arkheionx version
arkheionx doctor
arkheionx review-map   examples/vault-strategy-oracle-fixture
arkheionx value-paths  examples/vault-strategy-oracle-fixture
arkheionx assumptions  examples/vault-strategy-oracle-fixture
arkheionx test-gap-map examples/vault-strategy-oracle-fixture
arkheionx proof-plan   examples/vault-strategy-oracle-fixture
```

`review-map` exits with code 1 by design when it has review guidance to report; this is documented behavior, not a crash.

Appendix B: Further reading

- V4 stable scope
- CLI reference
- Review map model
- What Arkheionx is not
- Public surface
- Bug bounty workflow
- Pre-audit workflow
- Demo fixture
- Project README