ANASTASIA LABS

Audit Tracker

OpShin Audit

Contents

Project Name: OpShin Audit
URL: Catalyst Proposal

Project Name:	Opshin Language Audit
Audit Period:	Oct 27th,2024
Team Member(s) Assigned:	Suganya Raju, Eric Lee
Audit Start Date:	Oct 2024
Audit End Date:	

Milestone1 - Project Kickoff and Planning

The milestone1 was approved in November 2024.

Milestone2 - Language Analysis Report

As part of Milestone2 we are required to submit a detailed analysis report on the OpShin Language.

Topics to cover

1. UPLC debugging Architectural Decisions
   - How types are mapped to UPLC types
   - Static type Inferencer-> How ATI is implemented within OpShin
   - Mapping of Python -> Pluto -> UPLC
   - Storage of variables in statemonads
2. Opshin compiler pipeline
   - rewrites - how effective it is written and area of improvement
   - optimization - how effective it is written and area of improvement
   - conversion -> pluto
3. Performance and script size testing
4. Overall language constructs

Areas Covered

1. UPLC debugger through Gastronomy
2. Tested Language constructs – output of opshin eval, eval_uplc,
3. Code Coverage - using Pytest -cov tool
4. Checked build artifacts
5. Tried to replicate aiken acceptance tests in opshin(only first 20 test cases)
   - Challenge - opshin supports only a limited language constructs, its a challenge to replicate most of the aiken acceptance tests.

Hunting Critical Security vulnerabilities

The compiler pipeline consists of the following steps:

Tokenization -> AST building -> Type checking and type inference -> Code generation (Pluto)

(Pluto-to-UPLC is out of scope)

Each of these steps can potentially introduce Critical errors into the final validator output. The step which is most likely to contain such vulnerabities is the Pluto-to-UPLC conversion step, but that step isnt in this audits scope.

Tokenization step

- Not throwing an error when a bytearray literal has odd length
- Not throwing an error when mixing tabs and spaces for the indentation
- Not throwing an error when encountering inconsistent indentation

AST building

- Collecting build errors but not throwing them at the end
- Incorrect grouping of tokens, leading to unexpected behavior

Type checking

- Not throwing an error when a type is wrong

Type inference

- Infering the wrong type, leading to the wrong Pluto code being generated

Code generation

- Code omitted
- If the generated code is textual: missing tokens (e.g. missing quotes for literal strings)