STATE MIND

Vyper compiler modules

Review report

11-03-2024 - 23-05-2024

Table of contents

1. Project Brief

Title	Description
Client	Vyper
Project name	Vyper compiler modules
Timeline	11-03-2024 - 23-05-2024
Initial commit	6fb750af3972160f871f62bb49a44827acf06423
Final commit	eb011367cc769d62a084deff62153825e626f87a

Short Overview

The Vyper team has requested Statemind to review the module system for the upcoming v0.4.0 release of the Vyper language.

Vyper is a security-oriented, pythonic smart-contract programming language that targets the Ethereum Virtual Machine (EVM). The Vyper v0.4.0 has overhauled the import system and undergone major refactoring. This update enables developers to create modular components for easy contract integration.

During the module review, the Statemind team emphasized the examination of storage layout and code generation. Storage layout plays a key role in contract storage variable utilizations and the correctness of reentrancy guarding. Codegen is largely responsible for translating the generated AST into the intermediate representation, which later becomes bytecode.

Project Scope

The review covered the following files and releases:

- environment.py
- user.py
- function.py
- utils.py
- import_graph.py
- data_positions.py
- base.py
- phases.py
- return_.py
- ir_node.py
- core.py
- _convert.py
- data_locations.py
- subscriptable.py
- bytestrings.py
- module.py
- global_.py
- constant_folding.py
- compile_ir.py
- input_bundle.py
- module.py
- external_call.py
- context.py
- utils.py
- module.py
- base.py
- local.py
- getters.py
- common.py
- address_space.py
- stmt.py
- memory_allocator.py
- expr.py
- functions.py

The review covered the first beta version, v0.4.0b1, until the fourth release candidate, v0.4.0rc4.

The following pull requests were in scope:

- https://github.com/vyperlang/vyper/pull/3663
- https://github.com/vyperlang/vyper/pull/3729
- https://github.com/vyperlang/vyper/pull/3769
- https://github.com/vyperlang/vyper/pull/3786
- https://github.com/vyperlang/vyper/pull/3927
- https://github.com/vyperlang/vyper/pull/3924
- https://github.com/vyperlang/vyper/pull/3949
- https://github.com/vyperlang/vyper/pull/3971
- https://github.com/vyperlang/vyper/pull/4007
- https://github.com/vyperlang/vyper/pull/3655
- https://github.com/vyperlang/vyper/pull/3738
- https://github.com/vyperlang/vyper/pull/3817
- https://github.com/vyperlang/vyper/pull/3818
- https://github.com/vyperlang/vyper/pull/3832
- https://github.com/vyperlang/vyper/pull/3454
- https://github.com/vyperlang/vyper/pull/3844
- https://github.com/vyperlang/vyper/pull/3941
- https://github.com/vyperlang/vyper/pull/3925
- https://github.com/vyperlang/vyper/pull/3789

2. Finding Severity breakdown

All vulnerabilities discovered during the review are classified based on their potential severity and have the following classification:

Severity	Description
Critical	Bugs leading to assets theft, fund access locking, or any other loss of funds to be transferred to any party.
High	Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.
Medium	Bugs that can break the intended contract logic or expose it to DoS attacks, but do not cause direct loss of funds.
Informational	Bugs that do not have a significant immediate impact and could be easily fixed.

Based on the feedback received from the Customer regarding the list of findings discovered by the Contractor, they are assigned the following statuses:

Status	Description
Fixed	Recommended fixes have been made to the project code and no longer affect its security.
Acknowledged	The Customer is aware of the finding. Recommendations for the finding are planned to be resolved in the future.

3. Summary of findings

Severity	# of Findings
Critical	0 (0 fixed, 0 acknowledged)
High	0 (0 fixed, 0 acknowledged)
Medium	4 (4 fixed, 0 acknowledged)
Informational	10 (3 fixed, 7 acknowledged)
Total	14 (7 fixed, 7 acknowledged)

4. Conclusion

During the review, we uncovered a total of 14 issues:

- 4 medium severity issues (all fixed);
- 10 informational severity issues (3 fixed, 7 acknowledged).

All acknowledged issues are planned to be fixed. Proposed bytecode optimizations will be possible with the release of a new backend Venom (currently experimental).

5. Findings report

MEDIUM-01	Storage collisions using storage override	Fixed at 96a838

Description

The **set_storage_slots_with_overrides** doesn't handle module storage allocation or respect the module's storage layout. While only the main file layout can be overridden, despite adding overriding of the module to the input file (**--storage-layout-file**), all modules will have a default storage layout:

```
slot 0 - global reentrancy
...
```

A user can unknowingly create storage collisions or two different reentrancy locks.

Recommendation

We recommend reconsidering the logic of the storage overriding functionality.

MEDIUM-02	**standard-json** import path	Fixed at 75c75c

Description

The error appears during input-json compilation and explicit passing root_folder parameter (**-p**). It seems the **root_folder** parameter should've not been considered to find the contract's sources.

The reason why the error occurs is that **InputBundle** trying to combine **root_folder** with a contract's name from **sources** section of the input JSON:

https://github.com/vyperlang/vyper/blob/6fb750af3972160f871f62bb49a44827acf06423/vyper/compiler/input_bundle.py#L96-L107

and then it tries to find the contract's sources by the path that has been just combined:

https://github.com/vyperlang/vyper/blob/6fb750af3972160f871f62bb49a44827acf06423/vyper/compiler/input_bundle.py#L203

Meanwhile, **self.input_json** is defined as

```
{
    <contract_path>: {
        "content": <contract_source_literal>
    }
}
```

contract_path is the same as it's been received from the input:

https://github.com/vyperlang/vyper/blob/6fb750af3972160f871f62bb49a44827acf06423/vyper/cli/vyper_json.py#L278

https://github.com/vyperlang/vyper/blob/6fb750af3972160f871f62bb49a44827acf06423/vyper/cli/vyper_json.py#L175

Thus, the only ways for the contracts to be found are the cases when the **root_folder** parameter either is not set(so it's set to the default value, i.e **.**) or explicitly set to value **.**

Also, the only test-case checking combination of input-JSON and **root_folder** tests the case of obviously wrong root path:

https://github.com/vyperlang/vyper/blob/6fb750af3972160f871f62bb49a44827acf06423/tests/unit/cli/vyper_json/test_compile_json.py#L219-L221

Recommendation

We recommend not considering **-p** parameter since the explicit **root_folder** setting has no point in the context of the **standard-json** compilation.

MEDIUM-03	Reentrant lock for overridden storage layout	Fixed at 96a838

Description

The **set_storage_slots_with_overrides** function allocates reentrancy lock only for **the first function**

```
variable_name = GLOBAL_NONREENTRANT_KEY

# re-entrant key was already identified
if variable_name in ret:
    continue


...
ret[variable_name] = {"type": "nonreentrant lock", "slot": reentrant_slot}
```

A contract with more than one guarded function will not compile, due to the error occurring during the **get_nonreentrant_lock** execution. **'ContractFunctionT' object has no attribute 'reentrancy_key_position'**

Recommendation

We recommend setting the reentrancy position before skipping iteration.

```
# re-entrant key was already identified
if variable_name in ret:
    reentrant_slot = ret[variable_name]["slot"]
    type_.set_reentrancy_key_position(VarOffset(reentrant_slot))
    continue
```

MEDIUM-04	HashMap index checks when the subscript is folded	Fixed at 54616d

Description

The **HashMap** index is incorrectly checked due to unfolded subscript value.

```
m: HashMap[uint256, uint256]

@external
def foo():
  self.m[0-1] = 2
```

The example above successfully compiles and executes.

Recommendation

We recommend folding the subscript value to type-check accurately.

INFORMATIONAL-01	Ambiguous compiler error	Acknowledged

Description

Compiling a contract with a JSON-imported interface with invalid contents will throw an undefined error.

test_code.vy

```
import test_import

implements: test_import

@external
def func():
   pass
```

test_import.json is an empty or invalid JSON file.

```
Error compiling: test_code.vy
AssertionError

During handling of the above exception, another exception occurred:

vyper.exceptions.CompilerPanic: unhandled exception

  contract "test_code.vy:1", line 1:0
  ---> 1 import test_import
  -------^
       2


This is an unhandled internal compiler error. Please create an issue on Github to notify the developers!
https://github.com/vyperlang/vyper/issues/new?template=bug.md
```

Recommendation

We recommend creating an error message for users.

INFORMATIONAL-02	Incorrect NamespaceCollision error message	Fixed at 214a35

Description

The error message in the **validate_assignment** missing an **f**.

```
if prev_decl is None:
    msg += " as a {prev}"
```

For example, the contract:

```
var: uint256

flag var:
    a
```

Will throw an error **vyper.exceptions.NamespaceCollision: 'var' has already been declared as a {prev}**.

Recommendation

We recommend fixing an f-string.

```
if prev_decl is None:
    msg += f" as a {prev}"
```

INFORMATIONAL-03	Pure allows reading storage slots of modules	Fixed at 20432c

Description

The **_validate_pure_access** validates access via checking against **MUTABLE_ENVIRONMENT_VARS** & **CONSTANT_ENVIRONMENT_VARS**.

The following contract will be successfully compiled.

test.vy

```
import test_2

initializes: test_2

@external
@pure
def func() -> uint256:
    return test_2.var
```

test_2.vy

```
var: uint256
```

Recommendation

We recommend adding to the function **_validate_pure_access** check against all existing module names.

INFORMATIONAL-04	Incorrect **InitializerException** hint message	Acknowledged

Description

The **_validate_global_initializes_constraint** is called only for the main file.

```
found_module = module_t.find_module_info(u)
if found_module is not None:
    hint = f"add `initializes: {found_module.alias}`to the top level of "
    hint += "your main contract"
else:
    # CMC 2024-02-06 is this actually reachable?
    hint = f"ensure {module_t}`is imported in your main contract!"
```

Here it uses **module_t** to hint to users, which essentially tells them to import the main file to the main file.

It can be triggered via nesting the imports, e.g.:

test_err.vy

```
import test_err_1

initializes: test_err_1
```

test_err_1.vy

```
import test_err_2
import test_err_3

initializes: test_err_2[
    test_err_3 := test_err_3
]
```

test_err_2.vy

```
import test_err_3

uses: test_err_3

@external
def set_some_mod():
    a: uint256 = test_err_3.var
```

test_err_3.vy

```
var: uint256
```

Error message:

```
Error compiling: test_err.vy
vyper.exceptions.InitializerException: module `test_err_3.vy`is used but never initialized!

 (hint: ensure `test_err.vy`is imported in your main contract!)

 contract "test_err_2.vy:3", line 3:0
      2
 ---> 3 uses: test_err_3
 -------^
      4
```

Recommendation

We recommend replacing **module_t** with **u** to hint users about non-imported modules.

```
found_module = module_t.find_module_info(u)
if found_module is not None:
    hint = f"add `initializes: {found_module.alias}`to the top level of "
    hint += "your main contract"
else:
    # CMC 2024-02-06 is this actually reachable?
    hint = f"ensure `{u}`is imported in your main contract!"
```

INFORMATIONAL-05	**Improper cyclic function call check**	**Acknowledged**

Description

The **_compute_reachable_set** will recursively call itself with the modified **path** variable, checking only the **root = path[0]** variable for every **fn_t.called_functions** iteration.

```
if g == root:
    message = " -> ".join([f.name for f in path])
    raise CallViolation(f"Contract contains cyclic function call: {message}")
```

In the following example, the path list will be **[foo, bar, bar, bar, ...]**, where **root** is **foo**, falling into infinite recursion.

```
@external
def foo():
    self.bar()

@internal
def bar():
    self.bar()
```

The error message **RecursionError: maximum recursion depth exceeded while calling a Python object**

Recommendation

We recommend introducing a more precise check.

INFORMATIONAL-06 | Possible bytecode optimizations via vyper | Acknowledged

Description

1. If an internal function is used in the constructor and runtime, then the compiler will generate the same bytecode for both runtime and deployment.

This optimization would reduce deployment costs.

2. Consecutive variable assignments are not optimized by the compiler.

```
@external
def func() -> uint256:
    a: uint256 = 1337
    a = 1338
    return a
```

The compiler would generate bytecode for redundant operations.

Recommendation

We recommend implementing optimizations in the compiler.

INFORMATIONAL-07 | Unintended error message for bytestring convert | Acknowledged

Description

```
@external
def foo(a: Bytes[10]) -> Bytes[11]:
  return convert(a, Bytes[11])
```

The compilation of the example above will throw **vyper.exceptions.InvalidType: Value and target type are both 'Bytes[11]'** instead of **vyper.exceptions.TypeMismatch: Can't convert Bytes[10] to Bytes[11]**

The first condition in the **_cast_bytestring** function can't be reached

```
def _cast_bytestring(expr, arg, out_typ):
    # ban converting Bytes[20] to Bytes[21]
    if isinstance(arg.typ, out_typ.__class__) and arg.typ.maxlen <= out_typ.maxlen:
        _FAIL(arg.typ, out_typ, expr)
        # ^- should raise TypeMismatch(f"Can't convert {ityp} to {otyp}", source_expr)
```

Due to the check in the **Convert::infer_arg_types**:

```
if target_type.compare_type(value_type):
    raise InvalidType(f"Value and target type are both '{target_type}'", node)
```

Which is triggered by the **_BytestringT** comparison conditions.

```
if self._length:
    if not other._length:
        other.set_length(max(self._length, other._min_length))
    return self._length >= other._length
```

Recommendation

We recommend clarifying the error message.

INFORMATIONAL-08	Other code improvement issues	Acknowledged

Description

1. Extra for-loop

   In function **_runtime_reachable_functions** there is an extra for-loop:

```
for fn_t in module_t.exposed_functions:
    assert isinstance(fn_t.ast_def, vy_ast.FunctionDef)

    ret.update(fn_t.reachable_internal_functions)
    ret.add(fn_t)

# create globally unique IDs for each function
for fn_t in ret:
    id_generator.ensure_id(fn_t)
```

Operation **id_generator.ensure_id(fn_t)** could be executed within the first loop.

2. Redundant generalization in function signature

   There is no point in defining function **_ir_for_internal_function** with ***args** and ***kwargs** signature since it always calls **generate_ir_for_internal_function** which receives positional non-default arguments only.

```
def _ir_for_internal_function(func_ast, *args, **kwargs):
    return generate_ir_for_internal_function(func_ast, *args, **kwargs).func_ir
```

```
def generate_ir_for_internal_function(
    code: vy_ast.FunctionDef, module_ctx, is_ctor_context: bool
) -> InternalFuncIR
```

3. Code duplications

   Functions **ast_to_dict** and **dict_to_ast** from **vyper/ast/parse.py** are not used and are duplicates of the same functions from **vyper/ast/utils.py**.

Recommendation

We recommend considering the listed code improvements.

INFORMATIONAL-09	Compilation errors due to not folded **convert** value	Acknowledged

Description

Due to the unfolded value, the **convert** value is incorrectly type-checked.

```
@external
def foo()->bytes32:
  return convert(0-1, bytes32)
```

```
@external
def foo()->bool:
  return convert(0-1, bool)
```

Returns **vyper.exceptions.TypeMismatch: Expected uint8 but literal can only be cast as int104 or int96** instead of compiling.

```
@external
def foo()->uint256:
  return convert(0-1, uint256)
```

Returns
vyper.exceptions.StaticAssertionException: assertion found to fail at compile time. (hint: did you mean 'raise'?) [assert, [sge, -1 <0-1>, 0]]
instead of **vyper.exceptions.InvalidLiteral: Number out of range**.

Recommendation

We recommend folding the **convert** value to type-check accurately.

INFORMATIONAL-10	Attribute error due to missing reentrancy_key_position via nesting call	Fixed at 4c66c8

Description

The **uses** analysis does not account for nested **nonreentrant** functions.

main.vy

```
import test_1

@external
def foo():
  test_1.bar()
```

test_1.vy

```
@internal
def bar():
   self.baz()
   return

@nonreentrant
@internal
def baz():
   return
```

The variant with exporting an external function.

main.vy

```
import test_1

exports:test_1.bar_ext

@external
def foo():
  pass
```

test_1.vy

```
@external
def bar_ext():
   self.baz()

@nonreentrant
@internal
def baz():
   return
```

Both examples will return

AttributeError: 'ContractFunctionT' object has no attribute 'reentrancy_key_position'. Did you mean: 'set_reentrancy_key_position'?

Recommendation

We recommend additionally checking the **nonreentrant** usage for all invoked functions.

STATE
MIND