STATE MIND

Storage layout

Review report

29-01-2024 - 08-03-2024

Table of contents

1. Project Brief

Title	Description
Client	Vyper
Project name	Storage layout
Timeline	29-01-2024 - 08-03-2024

Short Overview

The Vyper team has requested Statemind to review the storage layout of all released versions of the Vyper language.

Vyper is a security-oriented, pythonic smart-contract programming language that targets the Ethereum Virtual Machine (EVM). Storage layout plays a key role in contract storage variable utilizations and the correctness of reentrancy guarding.

Project Scope

The review covered the following files and releases:

- utils.py
- keccak256_helper.py
- common.py
- external_function.py
- utils.py
- primitives.py
- base.py
- module.py
- compile_ir.py
- shortcuts.py
- global_context.py
- utils.py
- nodes.py
- user.py
- function.py
- data_locations.py
- data_positions.py
- return_.py
- context.py
- internal_function.py
- identifiers.py
- subscriptable.py
- bytestrings.py
- utils.py
- base.py

The Statemind reviewed storage layouts from the latest version v0.3.10 to v0.1.0-beta.16.

Please note, that although the listed project scope is big, we mostly targeted storage-related functionality. Due to the improvements in the Vyper language the scope changes with versions. All scope changes follow the layout changes in the following releases:

- https://github.com/vyperlang/vyper/compare/v0.3.9...v0.3.10
- https://github.com/vyperlang/vyper/compare/v0.3.8...v0.3.9
- https://github.com/vyperlang/vyper/compare/v0.3.7...v0.3.8
- https://github.com/vyperlang/vyper/compare/v0.3.6...v0.3.7
- https://github.com/vyperlang/vyper/compare/v0.3.5...v0.3.6
- https://github.com/vyperlang/vyper/compare/v0.3.4...v0.3.5
- https://github.com/vyperlang/vyper/compare/v0.3.3...v0.3.4
- https://github.com/vyperlang/vyper/compare/v0.3.2...v0.3.3
- https://github.com/vyperlang/vyper/compare/v0.3.1...v0.3.2
- https://github.com/vyperlang/vyper/compare/v0.3.0...v0.3.1
- https://github.com/vyperlang/vyper/compare/v0.2.16...v0.3.0
- https://github.com/vyperlang/vyper/compare/v0.2.15...v0.2.16
- https://github.com/vyperlang/vyper/compare/v0.2.14...v0.2.15
- https://github.com/vyperlang/vyper/compare/v0.2.13...v0.2.14
- https://github.com/vyperlang/vyper/compare/v0.2.12...v0.2.13
- https://github.com/vyperlang/vyper/compare/v0.2.11...v0.2.12
- https://github.com/vyperlang/vyper/compare/v0.2.10...v0.2.11
- https://github.com/vyperlang/vyper/compare/v0.2.9...v0.2.10
- https://github.com/vyperlang/vyper/compare/v0.2.8...v0.2.9
- https://github.com/vyperlang/vyper/compare/v0.2.7...v0.2.8
- https://github.com/vyperlang/vyper/compare/v0.2.6...v0.2.7
- https://github.com/vyperlang/vyper/compare/v0.2.5...v0.2.6
- https://github.com/vyperlang/vyper/compare/v0.2.4...v0.2.5
- https://github.com/vyperlang/vyper/compare/v0.2.3...v0.2.4
- https://github.com/vyperlang/vyper/compare/v0.2.2...v0.2.3
- https://github.com/vyperlang/vyper/compare/v0.2.1...v0.2.2
- https://github.com/vyperlang/vyper/compare/v0.1.0-beta.17...v0.2.1
- https://github.com/vyperlang/vyper/compare/v0.1.0-beta.16...v0.1.0-beta.17

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	0 (0 fixed, 0 acknowledged)
Informational	4 (2 fixed, 2 acknowledged)
Total	4 (2 fixed, 2 acknowledged)

4. Conclusion

During the review, we uncovered a total of 4 issues:

4 informational severity issues (2 fixed, 2 acknowledged).

All acknowledged issues are planned to be fixed.

5. Findings report

INFORMATIONAL-01	Possible rounding errors in gas estimations	Acknowledged

Description

Usage of **math.ceil** operation is not recommended due to floating point errors. Most use cases are implemented using **ceil32**, except the following line in **keccak256_helper.py**. **add_gas_estimate=_gas_bound(ceil(to_hash.typ.maxlen / 32))**.

Recommendation

We recommend replacing ceil with the existing **ceil32**. **add_gas_estimate=_gas_bound(ceil32(to_hash.typ.maxlen) // 32)**.

INFORMATIONAL-02	Missing security advisory	Acknowledged

Description

The Vyper compiler below version **0.3.0** did not respect reentrancy keys on **__default__** functions. The bug was acknowledged and fixed as part of the 0.3.0 release but was not denoted in the security advisory.

Recommendation

We recommend including this issue in the Vyper security advisory.

INFORMATIONAL-03	Incorrect memory deallocation for large arrays in versions v0.2.11 - v0.3.7	Fixed at 489348

Description

The root cause of this bug is the same as in the **incorrect storage layout for contracts containing large arrays** (the rounding error by **math.ceil** usage).
This bug was introduced in another fix of the memory deallocation issue caused by incorrect size calculation.
The **VariableRecord**'s **size** property returns the word size of the variable by applying **math.ceil**.

```
def size(self):
    if hasattr(self.typ, "size_in_bytes"):
        return math.ceil(self.typ.size_in_bytes / 32)
    return math.ceil(self.typ.memory_bytes_required / 32)
```

Meanwhile, the **Context** uses the **VariableRecored**, invoking the size property to deallocate the memory.

```
for name, var in released:
    self.memory_allocator.deallocate_memory(var.pos, var.size * 32)
    del self.vars[name]
```

It is important to note that this issue is not exploitable in a regular environment due to the large values required to trigger this bug.

Recommendation

We recommend avoiding floating point operations.

INFORMATIONAL-04	Unused variable	Fixed at d8d98c

Description

The **_prune_unreachable_code** has an unused variable **instr**.

```
instr = assembly[i]
if isinstance(instr, list):
    instr = assembly[i][-1]
```

Recommendation

We recommend removing unused variables or utilizing them as expected.

STATE
MIND