ANATOMIST

SECURITY

Vyper

Security Assessment

April 19th, 2025 — Prepared by Anatomist Security

Celix Lin celix@anatomi.st

Kevin Wang kevinwang@anatomi.st

Table of Contents

1 —— Severity Level

CRITICAL	**Vulnerabilities enabling direct theft or irrecoverable financial loss.** • Direct loss of funds • Misconfigured authorization or access controls
HIGH	**Vulnerabilities causing significant financial or operational damage, but are more difficult to exploit.** • Loss of funds dependent on specific victim interactions • Exploitation requiring high capital relative to potential profit
MEDIUM	**Vulnerabilities that cause a recoverable DoS or extra fees/time.** • Exceeding Computational Limits • Partial data corruption that doesn't result in unrecoverable loss
LOW	**Issues with low impact or requiring specific conditions.** • Design oversights that do not threaten core operations • Minor race conditions unlikely to cause serious harm
INFO	**Opportunities for improvement with no immediate threat, typically addressing best practices or clarity.** • Aligning with coding standards or project conventions • Simplifying code to improve readability and maintainability

2 —— Scope

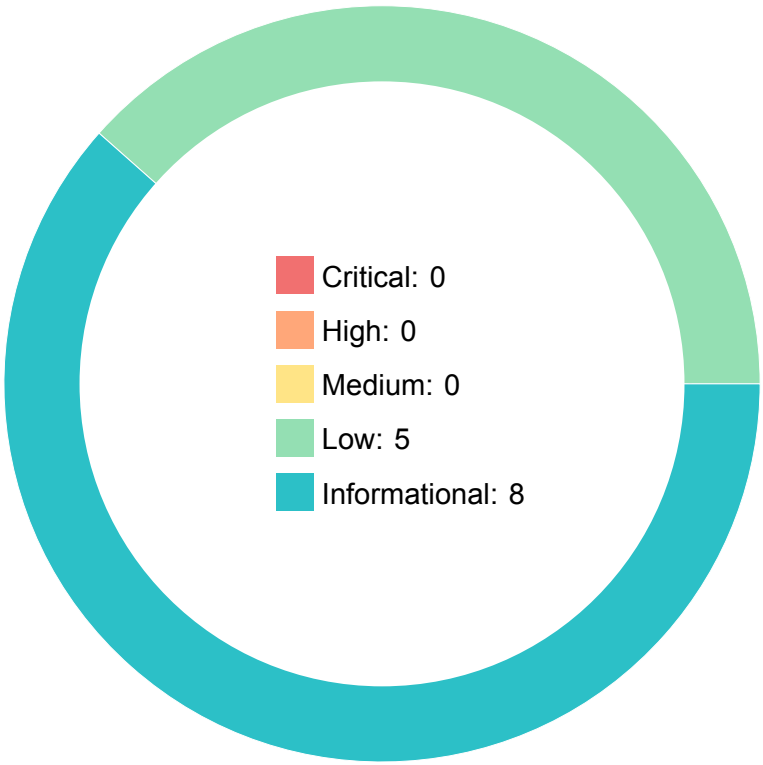
This assessment covered Vyper features and fixes listed below:

- **feat[lang]: nonreentrancy by default** — `PR#4563`
- **feat[lang]: add raw_create() builtin** — `61d259a`
- **feat[lang]: @raw_return decorator** — `PR#4568`
- **feat[lang]: enable bitwise ops for bytesM types** — `d5677b1`
- **fix[codegen]: fix overcopying of bytes in make_setter** — `9486a41`
- **fix[lang]: extend as_wei_value to all numeric types** — `94cf162`
- **feat[lang]: bubble up create revertdata** — `cef84e8`
- **ban calling nonreentrant functions from other nonreentrant functions** — `1f994f5`
- **fix[codegen]: interleaved effects eval for some builtins** — `fc2473c`
- **refactor[lang]: refactor decorator parsing** — `eb2efa3`
- **fix[ux]: fold keccak and sha256 of constant hexbytes** — `6ed37b8`
- **fix[lang]: filter oob array access during folding** — `2d515d3`
- **fix[lang]: disallow blockhash in pure functions** — `79001f3`
- **refactor[parser]: refactor pragma parsing** — `6a00171`
- **refactor[parser]: put settings on Module AST node** — `c05c1b9`
- **feat[lang]!: move sqrt to stdlib** — `1591a12`
- **fix[lang]: block modules in structs** — `8fc1d64`
- **fix[ux]: typechecking for loop annotation of list variable** — `ded0394`
- **fix[lang]!: forbid calling __default__** — `4b32292`
- **fix[parser]: block value assignment in for targets** — `57b9854`
- **fix[lang]: only reserve builtins at the top level** — `8d2fd72`
- **refactor[stdlib]: refactor math.sqrt implementation** — `PR#4575`

All security assessment and analysis was conducted between *April 14, 2025* and *April 18, 2025*, using the specified commit hashes as reference points for code stability. Code modifications or commits beyond this time frame were excluded from the scope of this audit.

3 —— Summary

Overall, we identified 13 findings. These findings are categorized into vulnerabilities and informational suggestions. Vulnerabilities present immediate security risks and should be remediated with high priority. Informational recommendations, while not posing immediate threats to system integrity, address potential security weaknesses that could lead to vulnerabilities if left unaddressed in future development cycles.



Vulnerabilities	Severity	Status
Incomplete Variable Annotation Checks	LOW	UNRESOLVED
Panic In `_complex_make_setter`	LOW	UNRESOLVED
Interleaved `raw_create` Argument Evaluation Order	LOW	UNRESOLVED
Invalid Global Variable Names	LOW	UNRESOLVED
Missing Comma in `RESERVED_KEYWORDS` Definition	LOW	UNRESOLVED

Informational Recommendations
`bytesM` Incapable of Constant Folding
Incorrect `archive / solc_json` Output
Invalid `modules` Type Allowed in Event Definition
Incorrect Mutability for Builtin Functions
Redundant Decorators
Inconsistent Behavior on Duplicate Import
Code Enhancements
Documentation Improvements

4 —— Vulnerabilities

4.1 Incomplete Variable Annotation Checks

Overall Severity:	LOW	Status:	UNRESOLVED

Description

Invalid variable annotations such as `public(public(TYPE))` are accepted by the compiler due to missing duplication checks for the first two variable traits.

```
# vyper/ast/nodes.py#L1408
class VariableDecl(VyperNode):
    ...
    def __init__(self, *args, **kwargs):
        ...
        # unwrap reentrant and public. they can be in any order
        for _ in range(2):
            func_id = self.annotation.get("func.id")
            if func_id not in ("public", "reentrant"):
                break
            _check_args(self.annotation, func_id)
            setattr(self, f"is_{func_id}", True)
            # unwrap one layer
            self.annotation = self.annotation.args[0]
        ...
```

Additionally, the grammar definition in the experimental parser is less restrictive than the actual implementation. For example, `immutable(transient(TYPE))` is allowed in the experimental parser but should not be allowed according to the actual implementation.

```
# vyper/ast/grammar.lark#L46
variable: NAME ":" type
// NOTE: Temporary until decorators used
variable_annotation: ("public" | "reentrant" | "immutable" | "
   transient") "(" (variable_annotation | type) ")"
variable_def: NAME ":" (variable_annotation | type)
```

Remediation

Enforce a strict ordering of variable traits, which may both enhance contract readability and allow the compiler to check against invalid code more easily.

4.2 Panic In `_complex_make_setter`

Overall Severity:	LOW	Status:	UNRESOLVED

Description

`_complex_make_setter` assumes `lhs` is not abi-encoded provided that the conditions in the following snippet hold. However, these assumptions are not necessarily valid as shown in the POC. This results in an unhandled assertion failure and compiler panic.

```
# vyper/codegen/core.py#L1120
def _complex_make_setter(left, right, hi=None):
    ...
    if left.is_pointer and right.is_pointer and simple_encoding
       and not has_dynamic_data:
        # both left and right are pointers, see if we want to
           batch copy
        # instead of unrolling the loop.
        assert left.encoding == Encoding.VYPER
        ...
```

PoC

```
interface FooBar:
    def test() -> (uint256, uint256): payable

@deploy
def __init__(ext: FooBar):
    x: uint256 = 2
    a: (uint256, uint256) = (x, x)
    #fails to compile
    b: (uint256, uint256) = extcall ext.test(
       default_return_value = a)
```

Remediation

Additional code should be added to handle abi-encoded `lhs`.

4.3 Interleaved `raw_create` Argument Evaluation Order

Overall Severity:	LOW	Status:	UNRESOLVED

Description

`raw_create` does not always evaluate arguments from left to right. Although the document states builtin arguments currently do not follow any specific evaluation order, there has been a constant effort on fixing this. We therefore recommend enforcing left-to-right evaluation order for newly added builtin to avoid adding to the problem.

```
# vyper/builtins/functions.py#L1706
class RawCreate(_CreateBase):
    ...
    def _build_create_IR(self, expr, args, context, value, salt,
        revert_on_failure):
        ...
        with scope_multi((initcode, value, salt), ("initcode", "
           value", "salt")) as (
            b1,
            (initcode, value, salt),
        ):
            bytecode_len = get_bytearray_length(initcode)
            with bytecode_len.cache_when_complex("initcode_len")
                as (b2, bytecode_len):
                maxlen = initcode.typ.maxlen
                ret.append(copy_bytes(buf, bytes_data_ptr(
                   initcode), bytecode_len, maxlen))
                ...
```

PoC

```
a: uint256

@deploy
def __init__():
    initcode: Bytes[100] = b'a'
    res: address = raw_create(initcode, self.test1(), value=self
        .test2())

@internal
def test1() -> uint256:
    self.a = 1
    return 1

@internal
def test2() -> uint256:
    self.a = 2
    return 2
```

Remediation

Evaluate all arguments from left to right. This can be done by caching arguments in the memory, similar to how `initcode` is handled.

4.4 Invalid Global Variable Names

Overall Severity:	LOW	Status:	UNRESOLVED

Description

The compiler currently allows declaring global variables `__default__` and `__init__`. This is especially problematic when the variable `__default__` is declared as `public`, since its getter function will now be treated as the `fallback` function by the compiler.

PoC

```
# poc.vy, `vyper -f ir poc.vy`

__init__: uint256
__default__: public(uint256)
```

Remediation

Disallow usage of both `__init__` and `__default__` as variable names.

4.5 Missing Comma in `RESERVED_KEYWORDS` Definition

Overall Severity:	LOW	Status:	UNRESOLVED

Description

The list of reserved keywords lacks a comma after `flag`. This results in ”flag” and the following string being merged into a single entry and fails to block improper usage of reserved keywords.

```
# vyper/ast/identifiers.py#L72
# Cannot be used for variable or member naming
RESERVED_KEYWORDS = _PYTHON_RESERVED_KEYWORDS | {
    # decorators
    "public",
    "external",
    ...
    "event",
    "enum",
    "flag"    # <- missing comma
    ...
```

Remediation

Add a comma after ”flag”.

5 —— Informational Recommendations

5.1 `bytesM` Incapable of Constant Folding

Description

Due to `BytesM` being stored as a '0x'-prefixed string in the `vy_ast.Hex` node, constant folding pass in semantic analysis is unable to process it.

```
# vyper/ast/parse.py#L487
def visit_Num(self, node):
    ...
    value = node.node_source_code

    # deduce non base-10 types based on prefix
    if value.lower()[:2] == "0x":
        if len(value) % 2:
            raise SyntaxException(
                "Hex notation requires an even number of digits"
                ,
                self._source_code,
                node.lineno,
                node.col_offset,
            )
        node.ast_type = "Hex"
        node.value = value
```

While several operations such as xor may be optimized further down the pipeline in the IR optimization phase, some other operations such as shifts are not, resulting in missed optimization opportunities.

```
# vyper/ir/optimizer.py#L50
arith = {
    "add": (operator.add, "+", UNSIGNED),
    "sub": (operator.sub, "-", UNSIGNED),
    "mul": (operator.mul, "*", UNSIGNED),
    "div": (evm_div, "/", UNSIGNED),
    "sdiv": (evm_div, "/", SIGNED),
    "mod": (evm_mod, "%", UNSIGNED),
    "smod": (evm_mod, "%", SIGNED),
    "exp": (evm_pow, "**", UNSIGNED),
    "eq": (operator.eq, "==", UNSIGNED),
    "ne": (operator.ne, "!=", UNSIGNED),
    "lt": (operator.lt, "<", UNSIGNED),
    "le": (operator.le, "<=", UNSIGNED),
    "gt": (operator.gt, ">", UNSIGNED),
    "ge": (operator.ge, ">=", UNSIGNED),
    "slt": (operator.lt, "<", SIGNED),
    "sle": (operator.le, "<=", SIGNED),
    "sgt": (operator.gt, ">", SIGNED),
    "sge": (operator.ge, ">=", SIGNED),
    "or": (operator.or_, "|", UNSIGNED),
    "and": (operator.and_, "&", UNSIGNED),
    "xor": (operator.xor, "^", UNSIGNED),
}

def _optimize(node: IRnode, parent: Optional[IRnode]) -> Tuple[
   bool, IRnode]:
    value = node.value
    ...
    if value in arith:
        parent_op = parent.value if parent is not None else None

        res = _optimize_binop(value, argz, annotation, parent_op
           )
        ...
```

PoC

```
# poc.vy : `vyper -f ir poc.vy` / `vyper -O none -f ir poc.vy`
@deploy
def __init__():
    # a can't be constant folded since its value is kept as a
       string
    a: bytes32 = 0
       x0000000000000000000000000000000000000000000000000000000000000000
        << 2
    b: uint256 = 2 << 2
    # c also can't be constant folded, but if ir optimization is
        enabled, it'll be optimised downstream
    c: bytes1 = 0x01 ^ 0x02
    d: uint256 = 1 ^ 2
```

Remediation

Consider storing bytesM internally as an integer, or add support for bytesM type in constant folding pass.

5.2 Incorrect `archive / solc_json` Output

Description

Using a fixed level of 0 for filtering built-in modules prevents the inclusion of user-defined `math` modules in the `archive / solc_json` output.

```
# vyper/compiler/output_bundle.py#L59
@cached_property
def compiler_inputs(self) -> dict[str, CompilerInput]:
    inputs: list[CompilerInput] = [
        t.compiler_input for t in self._imports if not
           _is_builtin(0, t.qualified_module_name)
    ]
    inputs.append(self.compiler_data.file_input)

    sources = {}
    for c in inputs:
        path = safe_relpath(c.resolved_path)
        # note: there should be a 1:1 correspondence between
        # resolved_path and source_id, but for clarity use
           resolved_path
        # since it corresponds more directly to search path
           semantics.
            sources[_anonymize(path)] = c

    return sources
```

Additionally, when the output format is set to `archive / solc_json`, the compiler panics when the compiled code imports modules using relative paths referencing parent directory.

```
# vyper/compiler/output_bundle.py#L118
@cached_property
def used_search_paths(self) -> list[str]:
    ...
    for c in self.compiler_inputs.values():
        ...
        for sp in reversed(search_paths):
            if c.resolved_path.is_relative_to(sp):
                ...

        # this shouldn't happen unless a file escapes its
           package,
        # *or* if we have a bug
        if not ok:
            raise CompilerPanic(f"Invalid path: {c.resolved_path
               }")
    ...
```

PoC 1

```
# poc.vy, `vyper -f solc_json poc.vy`
from . import math

@deploy
def __init__():
    pass
```

```
# math.vy
def test():
    pass
```

PoC 2

```
# cwd/poc.vy, `vyper -f solc_json poc.vy` / `vyper poc.vy`
from .. import x

@deploy
def __init__():
    pass
```

```
# x.vy
def test():
    pass
```

Remediation

Implement proper builtin file detection that does not depend on a pseudo level, and raise a proper error when searching outside of current directory.

5.3 Invalid `modules` Type Allowed in Event Definition

Description

The current implementation does not restrict module types to be defined within event definitions. The code below currently panics when resolving `abi_types` instead of outputting a proper error.

```
# vyper/semantics/types/base.py#L172
@property
def abi_type(self) -> ABIType:
    """
    The ABI type corresponding to this type
    """
    raise CompilerPanic("Method must be implemented by the
       inherited class")
```

PoC

```
# poc.vy, `vyper poc.vy`
import x

event E:
    f: x

@deploy
def __init__():
    pass
```

```
# x.vy
def test():
    pass
```

Remediation

Disallow usage of module type within event definition.

5.4 Incorrect Mutability for Builtin Functions

Description

With the addition of mutability information to builtin functions, we recommend assigning the `NONPAYABLE` modifier to `Send`, `SelfDestruct`, `RawLog`, `CreateMinimalProxyTo`, `RawCreate`, `CreateForwarderTo`, `CreateCopyOf`, and `CreateFromBlueprint`. Additionally, `RawCall` mutability depends on arguments, therefore cannot be directly assigned the `NONPAYABLE` modifier, we recommend either resolving its mutability based on the contract function being called, or raising warnings to ensure that user are made aware of potential misusage.

Another more security-oriented approach is to set `NONPAYABLE` as the default mutability modifier, and manually assigning `PURE` and `VIEW` modifiers to functions that do not read or modify storage. This prevents mutable functions from being inadvertently called in non-mutable contexts.

Remediation

The mutability of some builtins should be corrected to `NONPAYABLE`.

5.5 Redundant Decorators

Description

The compiler currently permits redundant decorator combinations, such as applying both `pure` and `reentrant`, or using `reentrant` on internal functions. To prevent code ambiguity and ensure clarity, it would be preferable to disallow these combinations.

PoC

```
#pragma nonreentrancy on

# pure function is always reentrant
@pure
@reentrant
@external
def test():
    pass

# internal function default is reentrant
@reentrant
def test():
    pass
```

Remediation

Implement stricter decorator checking rules to block ambiguity and redundancy in contract.

5.6 Inconsistent Behavior on Duplicate Import

Description

The exception handling differs between duplicated normal imports and duplicated built-in imports. The difference will not affect the compilation result, but we recommend raising the correct `DuplicateImport` exception in both cases.

PoC

```
$ vyper test.vy
Error compiling: test.vy
vyper.exceptions.DuplicateImport: lib imported more than once!

  contract "test.vy:1", line 1:0
  ---> 1 import lib
  -------^
       2 import lib

$ vyper test.vy
Error compiling: test.vy
vyper.exceptions.NamespaceCollision: 'math' has already been
   declared as a ModuleInfo(module_t=/path/to/math.vy, alias='
   math', ownership=<ModuleOwnership.NO_OWNERSHIP: 'no_ownership
   '>, ownership_decl=None)

  contract "test.vy:2", line 2:0
       1 import math
  ---> 2 import math
  -------^
       3
```

Remediation

Raise the same exception in both cases.

5.7 Code Enhancements

Description

We recommend a few changes to enhance code quality:

- The assert in wei calculation should be `<= 2**256 - 1` instead of `< 2**256 - 1`.

```
@process_inputs
def build_IR(self, expr, args, kwargs, context):
    value = args[0]

    denom_divisor = self.get_denomination(expr)
    with value.cache_when_complex("value") as (b1, value):
        ...
        elif value.typ == DecimalT():
            # sanity check (so we don't have to use safemul)
            assert (SizeLimits.MAXDECIMAL * denom_divisor) <
                2**256 - 1
            sub = [
                "seq",
                ["assert", ["sge", value, 0]],
                ["div", ["mul", value, denom_divisor],
                   DECIMAL_DIVISOR],
            ]
        ...
```

- Rename nonreentrant pragma to `external_nonreentrant` to clarify that it does not apply to internal functions.
- Currently, pragmas can be placed at any location within the source file. To enforce consistency and improve readability, we recommend restricting pragma placement to the very beginning of the file.

Remediation

Adopt the recommended changes.

5.8 Documentation Improvements

Description

We recommend a few changes to the documentation:

- Since `__default__` is no longer treated as a special case, the documentation should be updated accordingly.
- To clarify the purpose of the `nonreentrant` pragma, the documentation should specify that it applies only to non-pure external functions.
- Documentation for the `reentrant` modifier applied to global variables should be added.

Remediation

Update the documentation to cover all new features and properly describe their usage.