KUDELSKI SECURITY

Secure Code Review of wormhole & pricecaster-v2

Findings and Recommendations Report Presented to:

Rand Labs

July 22, 2022
Version: 1.1

Presented by:

Kudelski Security, Inc.
5090 North 40th Street, Suite 450
Phoenix, Arizona 85018

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

EXECUTIVE SUMMARY

Overview

Rand Labs engaged Kudelski Security to perform a secure code assessment of both wormhole and pricecaster-v2 Algorand smart contracts.

The assessment was conducted remotely by the Kudelski Security Team. Testing took place on June 20, 2022 - July 20, 2022, and focused on the following objectives:

- Provide the customer with an assessment of the security of recently added functionality and any risks that were discovered during the engagement.
- To provide a professional opinion on the maturity, efficiency, and coding practices
- To identify potential issues and include improvement recommendations based on the result of our code review

This report summarizes the engagement, tests performed, and findings. It also contains detailed descriptions of the discovered vulnerabilities, steps the Kudelski Security Teams took to identify and validate each issue, as well as any applicable recommendations for remediation.

Key Findings

The following are the major themes and issues identified during the testing period. These, along with other items, within the findings section, should be prioritized for remediation to reduce to the risk they pose.

- A theme across the codebase was functional bugs. A prerequisite for secure code is functional correctness – working code ‘as expected’ and this is an area that would benefit from attention.
- In several areas there was little or no documentation of expected behavior beyond a higher level whitepaper. As such, assessing the correctness of the logic proved challenging and could leave the code exposed to vulnerabilities where correct behavior could not be confirmed.
- Overall, the code reflected a lower quality than we would expect for the level of risk held by the bridge. Symptoms include lower level of commenting, and coding conventions like hardcoded indices and magic values (eg. Is `18446744073709551614` a magic value?). This made it difficult to review. During our interviews we learned that some of the Algorand code was written from reverse engineering (e.g., Ethereum code), without clear explanation of the rationale or reliability of original code.

With this being said it is worth noting that some of the documentation (`MEMORY.md`, describing `TmplSig.py` for instance) was of a high quality and this should serve as a strong foundation for improving the level of documentation.

Scope and Rules of Engagement

Kudelski performed a Secure Code Review of wormhole & pricecaster-v2 for Rand Labs. The following table documents the targets in scope for the engagement. No additional systems or resources were in scope for this assessment.

The source code was supplied with the commit hashes at:

- https://github.com/certusone/wormhole/commit/3fcb35132ceadba82c283d7e89a7174fd0317634
  - Subfolder `algorand`
- https://github.com/randlabs/pricecaster-v2/commit/fcc2b411de7e5199cae421c58fd59d955f059371
  - Subfolder `teal/pyteal` (`legacy` excluded)

In-Scope Contracts	
Wormhole	Pricecaster-v2
algorand/ ├── sandbox-algorand ├── teal ├── test ├── admin.py ├── deploy.sh ├── Dockerfile ├── gentest.py ├── **globals.py** ├── **inlineasm.py** ├── **local_blob.py** ├── Makefile ├── MEMORY.md ├── NOTES.md ├── package.json ├── package-lock.json ├── Pipfile ├── Pipfile.lock ├── README.md ├── requirements.txt ├── runPythonUnitTests.sh ├── sandbox ├── test_contract.py ├── testnet-update ├── **TmplSig.py** ├── **token_bridge.py** ├── **vaa_verify.py** └── **wormhole_core.py**	pyteal/ ├── legacy ├── **globals.py** ├── **inlineasm.py** ├── **mapper.py** └── **pricecaster-v2.py**

Table 1: Scope

TECHNICAL ANALYSIS & FINDINGS

During the Secure Code Review of wormhole & pricecaster-v2, we discovered 4 findings that had a high severity rating, as well as 2 of low severity.

The following chart displays the findings by severity.

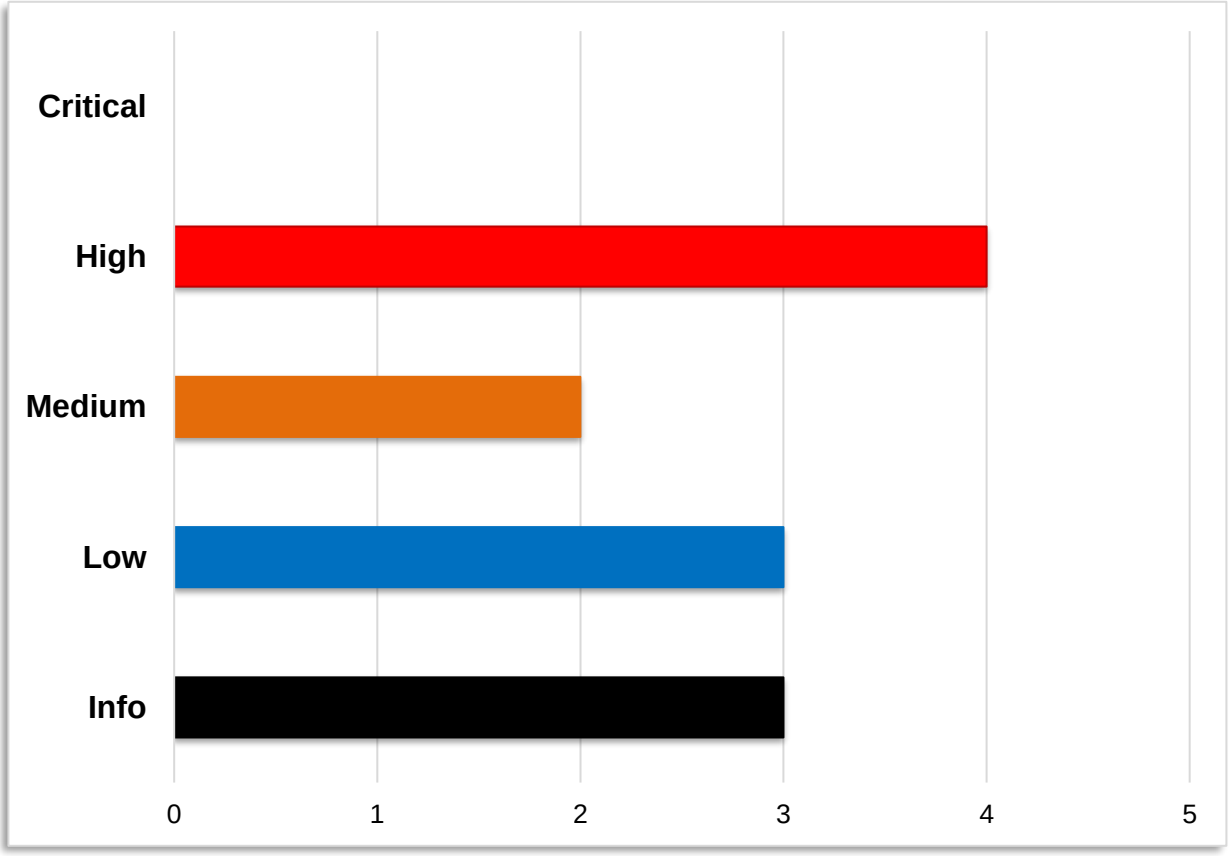

Figure 1: Findings by Severity

Observations

As introduced previously, we observed that part of the code reflected a lack of quality, and this on different aspects of the project, with a direct impact on code auditing. Let us detail some important steps of this process.

File tree

The file tree - discovered on the first connection to the repo - is the first source of information as to the content to be audited. The questions asked during this initial phase are:

- How is the project structured?
- What are the assets (smart contracts) to be audited? Where are they located?

Looking at Table 1 : Scope give us first insights:

- Wormhole ↔ not structured
  - All files are at the same level
    - `*.json`, `*.lock`, `*.md`, `*.py`, `*.sh`, `*.txt`, and `*.py` files
    - `*.py` files: how many smart contracts are there?
      - Some of them seem to be for testing…

  → First difficulty: identify assets

- Pricecaster-v2 ↔ structured
  - Smart contracts are directly identified in a dedicated "`pyteal`" folder
    - "`pyteal`" programming language
    - 4 files

Documentation

In code audit, one of the main tasks is to check compliance with the code with the provided technical documentation:

- Any document format is welcome (presentation, specification, diagrams, …)
- If they allow to understand – in details – your business flow

The `algorand/MEMORY.md` file is a very good example of such a documentation which includes:

- Background
- The "allocator" program: a. Instantiation, off-chain b. Instantiation, on-chain
- Many links to source code: `algorand/TmplSig.py`, `algorand/wormhole_core.py`, `algorand/admin.py`
- The `algorand/TmplSig.py`, 144 lines, is detailed almost line by line

Unfortunately, we could not find anything like that for all the other files, especially for the `algorand/token_bridge.py` file and it is 1021 lines

Source Code

The general observation we made about the source code is that there are no rules, whether for the container or for the content.

About filenames, many conventions are used:

- PascalCase: `algorand/TmplSig.py`
- kebab-case: `pyteal/pricecaster-v2.py`
- snake_case: `algorand/token_bridge.py`
- all-lowercase: `algorand/inlineasm.py` & `pyteal/inlineasm.py`

→ As reference see https://peps.python.org/pep-0008/#package-and-module-names

While the lack of a file naming convention is not really a big deal, it becomes much more important when it comes to their content:

- With or without shebang?
- With or without header?
  - With or without copyright notice?
  - With or without license notice?
  - With or without description?
    `vaa_verify.py` and `pricecaster-v2.py` are the only ones documented here
- With or without comments?
- With or without docstrings?
  - As they are used by modern editor/IDE to show details when the cursor hovers some code, it is a very valuable tool for all readers (developer, reviewer, …)
- Which naming convention for constants, classes, functions, variables?
  - One example among many

```
35  max_keys = 15
36  max_bytes_per_key = 127
37  bits_per_byte = 8
38
39  bits_per_key = max_bytes_per_key * bits_per_byte
40  max_bytes = max_bytes_per_key * max_keys
41  max_bits = bits_per_byte * max_bytes
42
43  portal_transfer_selector = MethodSignature("portal_transfer(byte[])byte[]")
44
45  def fullyCompileContract(genTeal, client: AlgodClient, contract: Expr, name, devmode) -> bytes:…
62
63  def clear_token_bridge():…
65
66  def approve_token_bridge(seed_amt: int, tmpl_sig: TmplSig, devMode: bool):
67      blob = LocalBlob()
68      tidx = ScratchVar()
69      mfee = ScratchVar()
70
71      def MagicAssert(a) -> Expr:…
77
78      @Subroutine(TealType.uint64)
79      def governanceSet() -> Expr:…
82
```

    - First time we observed that the very widely used rule of SCREAMING_CASE for constants is not applied, convention used later in the same file
    - 5 functions ↔ 3 conventions
    - Not a single comment, nor docstring

- Hard-coded values vs constants?
  - One example among many

```
off.store(off.load()+Int(43)),

MagicAssert(Int(2) ==      Btoi(Extract(Txn.application_args[1], off.load(),      Int(1)))),
Address.store(             Extract(Txn.application_args[1], off.load() + Int(1),  Int(32))),

FromChain.store(      Btoi(Extract(Txn.application_args[1], off.load() + Int(33), Int(2)))),
Decimals.store(       Btoi(Extract(Txn.application_args[1], off.load() + Int(35), Int(1)))),
Symbol.store(              Extract(Txn.application_args[1], off.load() + Int(36), Int(32))),
Name.store(                Extract(Txn.application_args[1], off.load() + Int(68), Int(32))),
```

    - Do you think it is easy to check that correct offsets and lengths are used?
    - If the structure evolves, it will be necessary to update the whole code, line by line → very error prone
    - Whereas with constants like ITEM OFFSET and ITEM SIZE
      → it is immediately readable and understandable
      → a single update and the whole code remains correct
- Common vs. case-by-case processing?
  - Example of function parameters:
    - Number of parameters is sometimes checked, but often is not checked
    - Sometimes at the beginning, sometimes later in the processing
    - Parameter sometimes used without any check

Therefore, since all files are different from each other, from naming to processing, the main difficulty is the readability and comprehensibility of the code.

Recommendations

Adopting best practices from the software engineering industry would help to code the right way:

- Define coding conventions
  - Standardized conventions help keep the code relevant and useful for clients, future developers, and the coders themselves
- Enforce coding conventions
  - This task can be automated with a static analyzer (e.g., PyLint) to:
    - Provide consistency in engineering (and audit) teams
    - Provide insight into code without executing it
    - Search for bugs at the early stages (although not all)
    - Find of security problems at an early stage
- Enhance code readability
  - Try to always write code that can be easily understood by others
    - Allocate appropriate names to all classes, functions, variables
    - Avoid abbreviations as much as possible to reduce ambiguity
- Define a standard/systematic control flow
  - First, check all parameters
    - Number of, value/range, consistency
  - According to the specification, process step by step
    - Abort/reject transaction at the slightest deviation

Findings

The *Findings* section provides detailed information on each of the findings, including methods of discovery, explanation of severity determination, recommendations, and applicable references.

The following table provides an overview of the findings.

#	Severity	Description
KS-RL-01	**High**	Wrong Parameter Check → Send an Invalid Transaction
KS-RL-02	**High**	Wrong Targeted Chain Check
KS-RL-03	**High**	Algorand smart contract not aligned with Ethereum smart contract
KS-RL-04	**High**	Invalid Asset Check → Transaction Falsely Rejected
KS-RL-05	**Medium**	Undocumented limitation of the UnitName of asset
KS-RL-06	**Medium**	No Boundaries Check
KS-RL-07	**Low**	Unused application_args
KS-RL-08	**Low**	Too Many Hardcoded Values
KS-RL-09	**Low**	Code Duplication
KS-RL-10	**Info**	Algorand guidelines not followed
KS-RL-11	**Info**	Python Code Quality
KS-RL-12	**Info**	Inappropriate Content

Table 2: Findings Overview

KS-RL-01 – Wrong Parameter Check → Send an Invalid Transaction

Severity	HIGH

Impact	Likelihood	Difficulty
High	High	Low

Description

In `token_bridge.py`, in `sendTransfer()` method, the expected number of arguments is 6 (if no payload) or 7 (if payload). But the `sendTransfer()` never checks exactly this condition.

It checks (line 766) if the `Txn.application_args.length()` is equal to 6, and it checks (line 777 and 781) if the `Txn.application_args.length()` is equal to 7.

Impact

If an attacker sends a "`sendTransfer`" transaction with 8 parameters, the smart contract will accept the transaction and the behavior of the smart contract will be impacted.

The smart contract will publish a message of type "`TransferWithPayload`" (line 768), with the fee value in the "`FromAddress`" field of the message (line 777) and with no payload.

Then the "**`magic line`**" (as it is mentioned in the code) will not be executed (line 781).

As a result, the transaction will be accepted by the smart contract and an invalid message "`TransferWithPayload`" will be sent.

It is difficult to estimate the impact as we do not know how the target chain will behave, but there is a risk of token lock in the Algorand smart contract.

Evidence

```
765             p.store(Concat(
766                 If(Txn.application_args.length() == Int(6),
767                    Bytes("base16", "01"),
768                    Bytes("base16", "03")),
769                 Extract(zb.load(), Int(0), Int(24)),
770                 Itob(amount.load()),  # 8 bytes
771                 Extract(zb.load(), Int(0), Int(32) - Len(Address.load())),
772                 Address.load(),
773                 FromChain.load(),
774                 Extract(zb.load(), Int(0), Int(32) - Len(Txn.application_args[3])),
775                 Txn.application_args[3],
776                 Extract(Txn.application_args[4], Int(6), Int(2)),
777                 If(Txn.application_args.length() == Int(7), Concat(Txn.sender(), Txn.application_args[6]), Concat(Extract(zb.load(), Int(0), Int(24)), Itob(fee.load()))))
778             )),
779
780             # This one magic line should protect us from overruns/underruns and trickery..
781             If(Txn.application_args.length() == Int(7),
782                MagicAssert(Len(p.load()) == Int(133) + Len(Txn.application_args[6])),
783                MagicAssert(Len(p.load()) == Int(133))),
```

Affected Resource

- `algorand/token_bridge.py` (lines 766-783)

Recommendation

In `sendTransfer()`, check that `application_args` length is equal to 6 or 7.
Do not accept any other value.

Reference

N/A

KS-RL-02 – Wrong Targeted Chain Check

Severity	HIGH

Impact	Likelihood	Difficulty
High	High	Low

Description

In `wormhole_core.py`, in `hdlGovernance()` method, the smart contract checks the target of the governance message. It only accepts the target value 8 (Algorand) or 0 (all chains).
Then it checks if the governance message is a contract upgrade, an update of the guardian set, a message fee, or a Payment.
If the governance is a contract upgrade, the smart contract will accept an upgrade even if the target value is 0.

Impact

The smart contract could accept a contract upgrade sends to all chains.
Upgrading a contract is always a sensitive part in a smart contract. It should be protected against unexpected upgrade. The impact could be high depending on the content of the new contract.

Evidence

```
213                 # What is the target of this governance message?
214                 tchain.store(Extract(Txn.application_args[1], off.load() + Int(1), Int(2))),
215                 # Needs to point at us or to all chains
216                 MagicAssert(Or(tchain.load() == Bytes("base16", "0008"), tchain.load() == Bytes("base16", "0000"))),
217
```

Affected Resource

- `algorand/wormhole_core.py` (line 216, 220/228)

Recommendation

The smart contract should only accept contract upgrade with Algorand target value (8).

Reference

N/A

KS-RL-03 – Algorand smart contract not aligned with Ethereum smart contract

Severity	HIGH

Impact	Likelihood	Difficulty
High	High	Medium

Description

A wormhole core contract is implemented on each supported chain.
Concerning the governance messages handled by the `wormhole_core.py`, there are at least two differences with what is done on Ethereum smart contract:

1. When receiving a governance message with a Guardian Set update action, the Ethereum smart contract checks if the index is incrementing by 1.
   This is not done in the `wormhole_core.py` for Algorand.

2. When receiving a governance message with a Transfer Fee, the Ethereum smart contract accepts its own chainId and chainId 0.
   The `wormhole_core.py` only accepts its own chainId.

Impact

Its difficult to estimate the impact as we do not know which smart contract is correctly implemented (no documentation that explains the implementation or the design choices).

Evidence

Ethereum smart contract
https://github.com/certusone/wormhole/blob/dev.v2/ethereum/contracts/Governance.sol

Ethereum

```
96          // Verify that the index is incrementing via a predictable +1 pattern
97          require(upgrade.newGuardianSetIndex == getCurrentGuardianSetIndex() + 1, "index must increase in steps of 1");
98
```

Algorand

```
242                         # Make sure it is different and we can only walk forward
243                         If(isBoot == Int(0), Seq(
244                                 MagicAssert(Txn.accounts[3] != Txn.accounts[2]),
245                                 MagicAssert(idx.load() > (set.load()))
246                         )),
```

Ethereum

```
115       function submitTransferFees(bytes memory _vm) public {
116           Structs.VM memory vm = parseVM(_vm);
117
118           // Verify the VAA is valid before processing it
119           (bool isValid, string memory reason) = verifyGovernanceVM(vm);
120           require(isValid, reason);
121
122           // Obtains the transfer from the VAA payload
123           GovernanceStructs.TransferFees memory transfer = parseTransferFees(vm.payload);
124
125           // Verify the VAA is for this module
126           require(transfer.module == module, "invalid Module");
127
128           // Verify the VAA is for this chain
129           require(transfer.chain == chainId() || transfer.chain == 0, "invalid Chain");
130
```

Algorand

```
275                   [a.load() == Int(4), Seq([
276                       off.store(off.load() + Int(1)),
277                       MagicAssert(tchain.load() == Bytes("base16", "0008")),
278                       off.store(off.load() + Int(26)),
```

Affected Resource

- `algorand/wormhole_core.py` (245 and 277)

Recommendation

Add comments in the `wormhole_core.py` to explain the implementation choice and/or do the modification.

Reference

N/A

KS-RL-04 – Invalid Asset Check → Transaction Falsely Rejected

Severity	HIGH

Impact	Likelihood	Difficulty
High	High	Low

Description

The number of decimals is limited to 16, while an ASA can be identified with up to 19.

Impact

Transaction with an ASA id greater than 9999999999999999 (0x2386F26FC0FFFF) is rejected.

Evidence

```
173     @Subroutine(TealType.uint64)
174     def getFactor(dec: Expr):
175         return Cond(
176             [dec == Int(9),  Int(10)],
177             [dec == Int(10), Int(100)],
178             [dec == Int(11), Int(1000)],
179             [dec == Int(12), Int(10000)],
180             [dec == Int(13), Int(100000)],
181             [dec == Int(14), Int(1000000)],
182             [dec == Int(15), Int(10000000)],
183             [dec == Int(16), Int(100000000)],
184             [dec >  Int(16), Seq(Reject(), Int(1))],
185             [dec < Int(9), Int(1)]
186         )
```

Affected Resource

`algorand/token_bridge.py` (line 184)

Recommendation

1. Document the function, especially constraints, limitations
2. Follow a logical process:
   a. Lower limit `[dec  < Int(9), Int(1)]`
   b. In between limits `[dec == Int(9), Int(10)]`
      `[dec == Int(…), Int(…)]`
   c. Upper limit `[dec  > Int(19), Seq(Reject(),Int(1)))]`
3. Maybe refactor the code with an optimized function to reduce its size.

Reference

https://developer.algorand.org/docs/get-details/transactions/transactions/#asset-parameters

KS-RL-05 – Undocumented limitation of the UnitName of asset

Severity	MEDIUM

Impact	Likelihood	Difficulty
High	Medium	Medium

Description

In `token_bridge.py`, in `receiveAttest()` method, the `UnitName` is limited to 7, while the max size of a unit name of an asset is 8 bytes on Algorand.

Impact

If the `UnitName` of an asset is 8 bytes length, the smart contract will remove the last byte, and creates an asset with a modified name.

Evidence

```
380             # Lets trim this... seems these are limited to 7 characters
381             Symbol.store(trim_bytes(Symbol.load())),
382             If (Len(Symbol.load()) > Int(7), Symbol.store(Extract(Symbol.load(), Int(0), Int(7)))),
383             Name.store(Concat(trim_bytes(Name.load()), Bytes(" (Wormhole)"))),
```

Affected Resource

`algorand/token_bridge.py` (line 382)

Recommendation

Explain why it is limited to 7 or change this limitation to 8 for Algorand.

Reference

https://developer.algorand.org/docs/get-details/transactions/transactions/#asset-configuration-transaction

KS-RL-06 – No Boundaries Check

Severity	MEDIUM

Impact	Likelihood	Difficulty
Medium	Medium	Low

Description

There is no check on the index for the blob read and write operations.

Impact

Possible overflow.

Evidence

```
36
37  def _key_and_offset(idx: Int) -> Tuple[Int, Int]:
38      return idx / page_size, idx % page_size
39
```

Affected Resource

`algorand/local_blob.py`

Recommendation

Limit the index to the maximum available size.

Reference

N/A

KS-RL-07 – Unused application_args

Severity	LOW

Impact	Likelihood	Difficulty
Medium	Low	Low

Description

In `token_bridge.py`, the `sendTransfer()` method accepts transaction with 6, 7 (or more) arguments. The `Txn.application_args[2]` is never used.

Impact

The impact needs to be evaluated by the developer. As there is no information on the implementation choice, we do not know if this parameter is unneeded or if it has been forgotten in the `sendTransfer` implementation.

Evidence

Check the code from line 661 to 679.
The `Txn.application_args[2]` is never used.

Affected Resource

`algorand/token_bridge.py` (lines 661-679)

Recommendation

Use it in the implementation, or remove this argument if not needed.

Reference

N/A

KS-RL-08 – Too Many Hardcoded Values

Severity	LOW

Impact	Likelihood	Difficulty
Medium	Low	Low

Description

Much of the code consists of reading or writing data from larger structures; arrays, byte arrays (VAA / blob). To do this, the basic operation is to read/write with 2 parameters: offset and length.
We observed that in most cases these 2 parameters were hard-coded values.

Impact

The very first impact is that such a code is unreadable/incomprehensible.
Looking at the few samples below:

- Who can know what the developer intended?
- What parameter is it?

→ You constantly need to consult the description of the structure in use

Second impact is for the development team:

- Same remarks as above for a new developer
- At the slightest change in the structure, you have to review all the code to apply the changes in the right places

→ Very error prone, especially if the code is not fully covered by automated tests

Evidence

```
918         return Seq(
919             # VM only is version 1
920             MagicAssert(Btoi(Extract(Txn.application_args[1], Int(0), Int(1))) == Int(1)),
921
922             off.store(Btoi(Extract(Txn.application_args[1], Int(5), Int(1))) * Int(66) + Int(14)),
923
924             # emitter is chain/contract-address
925             emitter.store(Extract(Txn.application_args[1], off.load(), Int(34))),
926             sequence.store(Btoi(Extract(Txn.application_args[1], off.load() + Int(34), Int(8)))),
927
```

```
85                          InlineAssembly(
86                              "ecdsa_pk_recover Secp256k1",
87                              dhash,
88                              Btoi(Extract(signatures, si.load() + Int(65), Int(1))),
89                              Extract(signatures, si.load() + Int(1), Int(32)),
90                              Extract(signatures, si.load() + Int(33), Int(32)),
91                              type=TealType.none),
```

```
166                 packed_price_data.store(Concat(
167                     Extract(attestation_data.load(), Int(64), Int(41)),
168                     Extract(attestation_data.load(), Int(109), Int(8)),
169                     Extract(attestation_data.load(), Int(133), Int(8)),
170                 )),
```

Affected Resource

```
algorand/token_brigde.py
algorand/vaa_verify.py
algorand/wormhole_core.py
pyteal/pricecaster-v2.py
```

Recommendation

1. Think twice before hardcoding a value
2. Define constants with appropriate names and constants reflecting the structure:

```
ITEM_OFFSET  = X
ITEM_LENGTH  = Y
OTHER_OFFSET = ITEM_OFFSET + ITEM_LENGTH
OTHER_LEGNTH = Z
```

Reference

N/A

KS-RL-09 – Code Duplication

Severity	LOW

Impact	Likelihood	Difficulty
Medium	Low	Low

Description

Duplication of code in different files.

Impact

Duplicated code is used as a code maturity metric in the industry to point out how maintainable the code base is. It is also a possible entry point for new bugs as code duplication leads to mistakes when updating/rewriting the codebase.

Evidence

- Some constants

```
27  _max_keys = 15
28  _page_size = 128 - 1  # need 1 byte for key
29  _max_bytes = _max_keys * _page_size
30  _max_bits = _max_bytes * 8
31
32  max_keys = Int(_max_keys)
33  page_size = Int(_page_size)
34  max_bytes = Int(_max_bytes)
```

```
35  max_keys = 15
36  max_bytes_per_key = 127
37  bits_per_byte = 8
38
39  bits_per_key = max_bytes_per_key * bits_per_byte
40  max_bytes = max_bytes_per_key * max_keys
41  max_bits = bits_per_byte * max_bytes
```

- Some methods
  - `get_sig_address`
    - calling the method `encode_uvarint` also duplicated
  - `encode_uvarint`
  - `checkForDuplicate`
    - 30 duplicated lines
    - Excepted 1 extra line in wormhole_core.py

```python
@Subroutine(TealType.bytes)
def encode_uvarint(val: Expr, b: Expr):
    buff = ScratchVar()
    return Seq(
        buff.store(b),
        Concat(
            buff.load(),
            If(
                    val >= Int(128),
                    encode_uvarint(
                        val >> Int(7),
                        Extract(Itob((val & Int(255)) | Int(128)), Int(7), Int(1)),
                    ),
                    Extract(Itob(val & Int(255)), Int(7), Int(1)),
            ),
        ),
    )
```

```python
    @Subroutine(TealType.bytes)
    def encode_uvarint(val: Expr, b: Expr):
        buff = ScratchVar()
        return Seq(
            buff.store(b),
            Concat(
                buff.load(),
                If(
                    val >= Int(128),
                    encode_uvarint(
                        val >> Int(7),
                        Extract(Itob((val & Int(255)) | Int(128)), Int(7), Int(1)),
                    ),
                    Extract(Itob(val & Int(255)), Int(7), Int(1)),
                ),
            ),
        )
```

Affected Resource

- Constants
  - algorand/local_blob.py (lines 27-34)
  - algorand/token_bridge.py (lines 35-41)
- Methods
  - algorand/token_bridge.py (encode_uvarint lines 131-147 / get_sig_address lines 189-216)
  - algorand/wormhole_core.py (encode_uvarint lines 71-87 / get_sig_address lines 90-118)

Recommendation

Refactor duplicated code, e.g., in `global.py`.

Reference

N/A

KS-RL-10 – Algorand guidelines not followed

Severity	INFORMATIONAL

Description

One of the first Algorand recommendation is to "*Always verify that the* `RekeyTo` *property of any transaction is set to the* `ZeroAddress` *unless the contract is specifically involved in a rekeying operation*".

Another recommendation is "`CloseRemainderTo` *or* `AssetCloseTo` *should be the intended recipient or equal to global* `ZeroAddress`".

Impact

Code should validate as many `txn` fields as possible.
If not done, things not checked could become set to anything.

Evidence

No `Txn.rekey_to(), Txn.close_remainder_to(), Txn.asset_close_to()` check.

Affected Resource

`pyteal/pricecaster-v2.py`

Recommendation

Follow Algorand guidelines.

Even if other checks are implemented to validate the transactions, basic checks should be systematically applied to embody best practices.

Reference

https://developer.algorand.org/docs/get-details/dapps/avm/teal/guidelines/

KS-RL-11 – Python Code Quality

Severity	INFO

Description

One way of trying to define code quality is to look at one end of the spectrum: high-quality code. Hopefully, you can agree on the following high-quality code identifiers:

- It does what it is supposed to do, without any defects or problems
- It is easy to read & review, maintain, and extend

Impact

What happens when code does not meet above requirements:

- It does **not** do what it is supposed to do
  - Meeting requirements is the basis of any product, software, including security ones.
  - Software is developed to do something. If in the end, it does not do it… what about security impacts?
- It is **difficult** to read, maintain, or extend
  - Imagine the person who wrote the original code is gone. Now it is up to you to replace that person, and to make sense of the code that is already there
  - If the code is easy to comprehend, you will be able to analyze the problem and come up with a solution much quicker. If the code is complex and convoluted, you will probably take longer and possibly make some wrong assumptions → even applied for security
  - If the code is not easy to extend, your new feature could break other things.
  - No one wants to be in the position where they have to read, maintain, or extend low-quality code.

Evidence

One way to evaluate code is to use standard tools.
E.g., Pylint (default rcfile + disable `unused-wildcard-import`)

- pyteal/
  - mapper.py 3.90/10
  - pricecaster-v2.py 3.86/10
- algorand/
  - local_blob.py 7.55/10
  - TmplSig.py 3.19/10
  - token_bridge.py **0.00**/10
  - vaa_ferify.py 1.54/10
  - wormhole_core.py **0.00**/10

Here after samples of "`unused-variable`" warnings:

- `algorand/admin.py:357:18: W0612: Unused variable 'clear'`
- `algorand/admin.py:784:8: W0612: Unused variable 'bits_set'`
- `algorand/admin.py:1324:18: W0612: Unused variable 'clear'`
- `algorand/test_contract.py:47:4: W0612: Unused variable 'me'`
- `algorand/TmplSig.py:113:12: W0612: Unused variable 'admin_app_id'`

- `algorand/TmplSig.py:114:12: W0612: Unused variable 'admin_address'`
- `algorand/token_bridge.py:308:8: W0612: Unused variable 'c'`
- `algorand/token_bridge.py:309:8: W0612: Unused variable 'a'`
- `algorand/token_bridge.py:443:8: W0612: Unused variable 'me'`
- `algorand/token_bridge.py:458:8: W0612: Unused variable 'd'`
- `algorand/token_bridge.py:664:8: W0612: Unused variable 'd'`
- `algorand/token_bridge.py:825:8: W0612: Unused variable 'asset'`
- `algorand/token_bridge.py:833:8: W0612: Unused variable 'Address'`
- `algorand/token_bridge.py:834:8: W0612: Unused variable 'FromChain'`
- `algorand/wormhole_core.py:184:12: W0612: Unused variable 'emitter'`
- `algorand/wormhole_core.py:555:8: W0612: Unused variable 'progHash'`
- `algorand/wormhole_core.py:556:8: W0612: Unused variable 'progSet'`
- `algorand/wormhole_core.py:557:8: W0612: Unused variable 'clearHash'`
- `algorand/wormhole_core.py:558:8: W0612: Unused variable 'clearSet'`

→ Every time someone reads the code, they will need to understand what everything does

- Is that variable ever used for anything?

Affected Resource

All python files

Recommendation

Define a quality policy and enforce it using standard tools.

Reference

Code audit tool for Python: https://pypi.org/project/pylama/

KS-RL-12 – Inappropriate Content

Severity	INFO

Description

Raising an exception with the message "`you suck`" is not very helpful, nor meaningful.

Impact

Whether for a developer, reviewer, or anyone else, what can they understand/learn from such a message?

Evidence

```
311         if type == 'uint256' or type == 'bytes32':
312             return encode_single(type, val).hex()[64-(64):64]
313         raise Exception("you suck")
```

```
168         if chain == 5:
169             return "0000000000000000000000005a58505a96d1dbf8df91cb21b54419fc36e93fde"
170         raise Exception("you suck")
```

Affected Resource

algorand/admin.py (line 313)
algorand/gentest.py (line 170)

Recommendation

Out of respect for the people who will read/update/use your code, please do not use inappropriate messages, comments, or any type of irrelevant information. Always remember to give useful information that could help them better understand your code.

Reference

N/A

METHODOLOGY

During this source code review, the Kudelski Security Services team reviewed code within the project within an appropriate IDE. During every review, the team spends considerable time working with the client to determine correct and expected functionality, business logic, and content to ensure that findings incorporate this business logic into each description and impact. Following this discovery phase the team works through the following categories:

- Authentication
- Authorization and Access Control
- Auditing and Logging
- Injection and Tampering
- Configuration Issues
- Logic Flaws
- Cryptography

These categories incorporate common vulnerabilities such as the OWASP Top 10

Tools

The following tools were used during this portion of the test. A link for more information about the tool is provided as well.

- Visual Studio Code
- Pylint

Vulnerability Scoring Systems

Kudelski Security utilizes a vulnerability scoring system based on impact of the vulnerability, likelihood of an attack against the vulnerability, and the difficulty of executing an attack against the vulnerability based on a high, medium, and low rating system

Impact
The overall effect of the vulnerability against the system or organization based on the areas of concern or affected components discussed with the client during the scoping of the engagement.

> **High:**
> The vulnerability has a severe affect on the company and systems or has an affect within one of the primary areas of concern noted by the client
>
> **Medium:**
> It is reasonable to assume that the vulnerability would have a measurable affect on the company and systems that may cause minor financial or reputational damage.
>
> **Low:**
> There is little to no affect from the vulnerability being compromised. These vulnerabilities could lead to complex attacks or create footholds used in more severe attacks.

Likelihood
The likelihood of an attacker discovering a vulnerability, exploiting it, and obtaining a foothold varies based on a variety of factors including compensating controls, location of the application, availability of commonly used exploits, and institutional knowledge

> **High:**
> It is extremely likely that this vulnerability will be discovered and abused
>
> **Medium:**
> It is likely that this vulnerability will be discovered and abused by a skilled attacker
>
> **Low:**
> It is unlikely that this vulnerability will be discovered or abused when discovered.

Difficulty
Difficulty is measured according to the ease of exploit by an attacker based on availability of readily available exploits, knowledge of the system, and complexity of attack. It should be noted that a LOW difficulty results in a HIGHER severity.

> **Low:**
> The vulnerability is easy to exploit or has readily available techniques for exploit
>
> **Medium:**
> The vulnerability is partially defended against, difficult to exploit, or requires a skilled attacker to exploit.
>
> **High:**
> The vulnerability is difficult to exploit and requires advanced knowledge from a skilled attacker to write an exploit

Severity
Severity is the overall score of the weakness or vulnerability as it is measured from Impact, Likelihood, and Difficulty