KUDELSKI SECURITY

Secure Code Review of Wormhole & Pricecaster-v2

Findings and Recommendations Report Presented to:

Rand Labs

August 16, 2022
Version: 3.0

Presented by:

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

FOR PUBLIC RELEASE

TABLE OF CONTENTS

TABLE OF CONTENTS 2
LIST OF FIGURES 3
LIST OF TABLES 3
EXECUTIVE SUMMARY 4
Overview 4
Key Findings 4
Scope and Rules of Engagement 5
TECHNICAL ANALYSIS & FINDINGS 6
Findings 7
WORMHOLE 8
KS-WH-01 – Invalid asset check on ASA 8
KS-WH-02 – Missing check for chain on contract upgrade 10
KS-WH-03 – Code Duplication 12
PRICECASTER-V2 15
KS-PC-01 – Missing transaction property checks 15
METHODOLOGY 16
Tools 17
Vulnerability Scoring Systems 18

LIST OF FIGURES

Figure 1: Findings by Severity 6

LIST OF TABLES

Table 1: Scope 5
Table 2: Findings Overview 7

EXECUTIVE SUMMARY

Overview

Rand Labs engaged Kudelski Security to perform a secure code assessment of the Wormhole Algorand smart contracts and related pricecaster-v2 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 the code review.

This report summarizes the engagement, tests performed, and findings. It also contains detailed descriptions of the discovered vulnerabilities, steps the Kudelski Security Team 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 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. Assessing the correctness of the logic proved to be challenging without the required details. This could leave the code exposed to vulnerabilities where intended behavior could not be confirmed.

It was observed that documentation (eg. `MEMORY.md`, describing `TmplSig.py`) did exist that was of a high quality and this can serve as a strong foundation for continuing to improve the level of documentation across the project.

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)

A further round of review was performed by Kudelski Security 8th - 11th August, 2022 on remediations with the commit hashes at:

- https://github.com/certusone/wormhole/commit/1c2ab0e5767d8761de742d13bc462be8f4350a29
- https://github.com/randlabs/pricecaster-v2/commit/f1833d7ec20014bde0876777c53b8213adb73542

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, two (2) medium severity findings, one (1) low severity finding, and one (1) informational findings were identified.

The following chart displays the findings by severity.

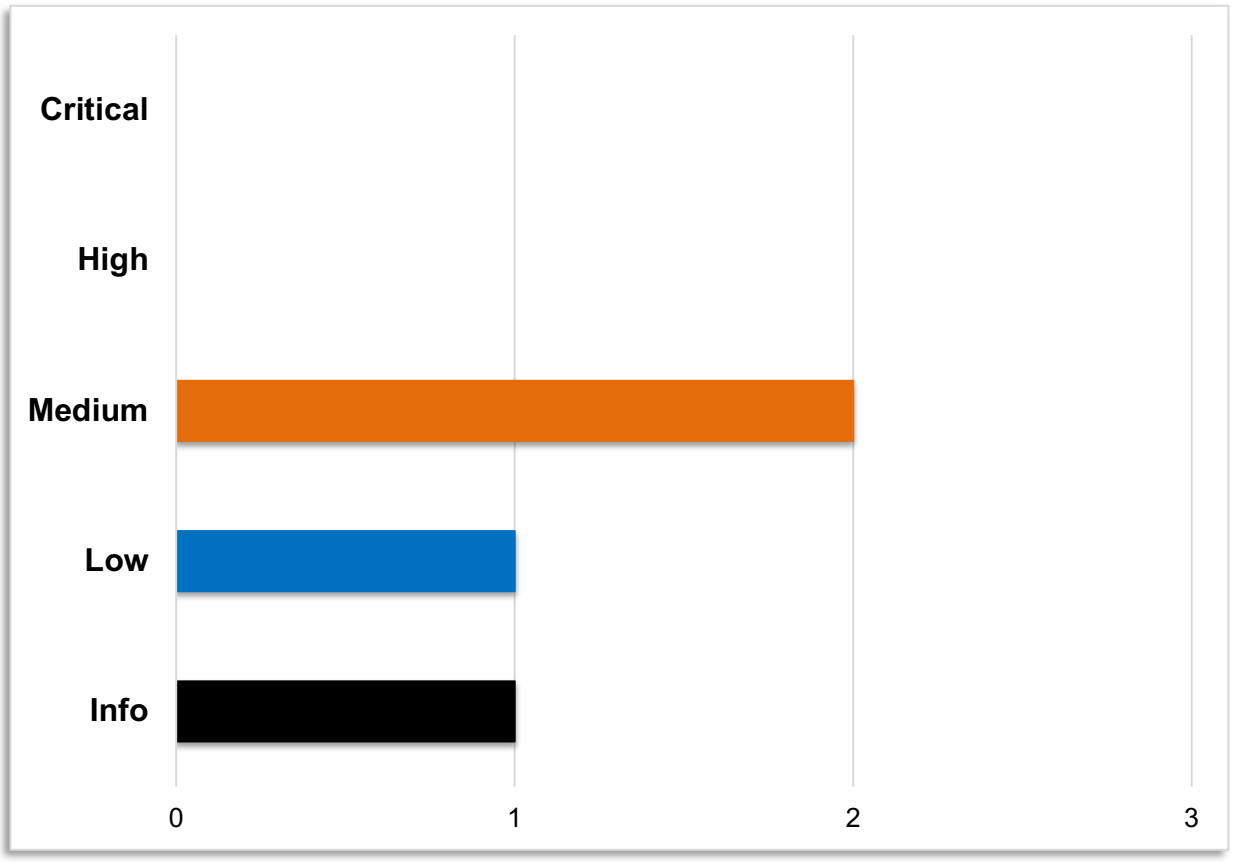

Figure 1: Findings by Severity

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	Status
Wormhole			
KS-WH-01	**Medium**	Invalid asset check on ASA	**Resolved**
KS-WH-02	**Medium**	Missing check for chain on contract upgrade	**Resolved**
KS-WH-03	**Informational**	Code Duplication	**Acknowledged**
Pricecaster-v2			
KS-PC-01	**Low**	Missing transaction property checks	**Resolved**

Table 2: Findings Overview

WORMHOLE

KS-WH-01 – Invalid asset check on ASA

Severity	MEDIUM
Status	RESOLVED

Impact	Likelihood	Difficulty
Low	High	Low

Description

Kudelski Security noticed that the `token_bridge.py` contract contained instances where validated values did not match the allowed values of the Algorand chain. In these cases, the contract rejected any Algorand Standard Assets (ASAs) defined with more than 16 decimal and at the time of this report Algorand Standard Assets (ASAs) can be defined with up to 19 decimals.

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

Impact

Transactions with an ASA id greater than 9999999999999999 (0x2386F26FC0FFFF) is rejected. A transaction with an ASA with a decimals parameter greater than 9999999999999999 (0x2386F26FC0FFFF) may be unintentionally rejected causing failures, denial of service, and an inability to interact with assets with decimals that are noted beyond this range.

With regard to the `UnitName`, 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

```
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         )
```

```
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 184)
`algorand/token_bridge.py` (line 382)

Recommendation

Adjust the limits inside the `getFactor` function to allow for the correct ASA limits. This should prevent unintentional rejection of valid values. As an example, see below:

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)))]`

Throughout the application, ensure that validations align with the Algorand standards and ensure there is a periodic alignment throughout the lifecycle of the application through an automated or manual process

Reference

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

KS-WH-02 – Missing check for chain on contract upgrade

Severity	MEDIUM
Status	RESOLVED

Impact	Likelihood	Difficulty
High	Low	High

Description

The Kudelski Security team noticed in `wormhole_core.py`, in the `hdlGovernance()` method that the smart contract checked the target of the governance message. The function only accepted the value 8 for Algorand or 0 for all chains. (Line 216)

Then it checks if the governance message is one of:

i) a contract upgrade (lines 220-228)
ii) an update of the guardian set (lines 229-267)
iii) a message fee, or (lines 268-274)
iv) a Payment. (lines 275-293)

If the governance is a contract upgrade, the smart contract accepted an upgrade even if the target value was 0.

Impact

A transaction involving a contract upgrade could be approved, specified to target all chains. Without any mitigating controls the result of this could be that messages submitted to the wormhole network could all be routed to this new smart contract, thereby opening up the risk of a denial of service or other network failure.
External to the contracts in scope for this review, the risk is mitigated downstream by the Guardians who would not sign a contract upgrade VAA with target chain set to 0.

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
```

wormhole_core.py

In the above code extract, the target chain is checked to be either 8 (Algorand) or 0 (all chains).

```
220                     [a.load() == Int(1), Seq([
221                         # ContractUpgrade is a VAA that instructs an implementation on a specific chain to
                            upgrade itself
222                         #
223                         # In the case of Algorand, it contains the hash of the program that we are allowed to
                            upgrade ourselves to.  We would then run the upgrade program itself
224                         # to perform the actual upgrade
225                         off.store(off.load() + Int(3)),
226
227                         App.globalPut(Bytes("validUpdateApproveHash"), Extract(Txn.application_args[1], off.
                            load(), Int(32)))
228                     ])],
```

wormhole_core.py

The code snippet above triggers a contract upgrade, that will be sent to whichever target chain has been specified in the target chain parameter.

Affected Resource

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

Recommendation

While the Guardians protect against this risk outside of reviewed code, the smart contract should only accept a contract upgrade with Algorand as the target chain (value of 8).

Reference

N/A

KS-WH-03 – Code Duplication

Severity	INFORMATIONAL

Impact	Likelihood	Difficulty
N/A	N/A	N/A

Description

The Kudelski Security team noticed that the project used multiple instances of the same code and variables. This may have been overlooked due to expansive code or duplicate work. Duplicate code is often a code smell for interesting areas and is used as a metric for code quality for repeatability, standardisation and ease of operation.

Impact

Duplicated code has a reasonable potential impact due to human error. If functions or values such as constants are intended to be the same throughout the project and they are defined in multiple places, there may be unforeseen issues where the expected value and actual value do not match. Acquiring and using the correct value may differ between implementations and may result in unexpected behavior. This potential impact becomes especially relevant if typos occur between implementations and where code hardening processes are not in place.

Evidence

- Some constants

`local_blob.py`

```
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)
```

`wormhole_core.py`

```
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`

`local_blob.py`

```
131  @Subroutine(TealType.bytes)
132  def encode_uvarint(val: Expr, b: Expr):
133      buff = ScratchVar()
134      return Seq(
135          buff.store(b),
136          Concat(
137              buff.load(),
138              If(
139                      val >= Int(128),
140                      encode_uvarint(
141                          val >> Int(7),
142                          Extract(Itob((val & Int(255)) | Int(128)), Int(7), Int(1)),
143                      ),
144                      Extract(Itob(val & Int(255)), Int(7), Int(1)),
145              ),
146          ),
147      )
```

`wormhole_core.py`

```
71       @Subroutine(TealType.bytes)
72       def encode_uvarint(val: Expr, b: Expr):
73           buff = ScratchVar()
74           return Seq(
75               buff.store(b),
76               Concat(
77                   buff.load(),
78                   If(
79                       val >= Int(128),
80                       encode_uvarint(
81                           val >> Int(7),
82                           Extract(Itob((val & Int(255)) | Int(128)), Int(7), Int(1)),
83                       ),
84                       Extract(Itob(val & Int(255)), Int(7), Int(1)),
85                   ),
86               ),
87           )
```

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`, and centralise reused code or constants. This will allow the team to make changes in a central location rather than manually identifying all iterations of the same code.

Reference

N/A

PRICECASTER-V2

KS-PC-01 – Missing transaction property checks

Severity	**LOW**
Status	**RESOLVED**

Impact	Likelihood	Difficulty
Low	Low	Low

Description

The Kudelski Security team noticed that the `pricecaster-v2` project did not implement some best practice checks for transactions. It was noted that the `RekeyTo` property was not used in transaction objects and not validated. The `RekeyTo` property specifies the authorized address to transfer ownership of an account to and could be used to authorize future transactions.

Similarly the `CloseRemainderTo` or `AssetCloseTo` should be set to the intended recipient or equal to global `ZeroAddress`, to prevent an adversary from performing an unexpected close operation.

Impact

An attacker could gain control of the account and publish incorrect price data.

Evidence

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

Affected Resource

`pyteal/pricecaster-v2.py`

Recommendation

Implement checks for the `rekey_to, close_remainder_to`, and `asset_close_to` properties to ensure they are set to `ZeroAddress`, unless the contracts specifically require rekeying, closing the sender's account, or opting out of the asset.

Reference

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

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