Algorand Wormhole
Smart Contract Audit

coinspect

WORMHOLE

Algorand Wormhole

Smart Contract Audit

V221007 Prepared for Wormhole • October 2022

1. Executive Summary

In **September 2022, Wormhole** engaged Coinspect to perform a source code review of the implementation of the **Wormhole** smart contracts on Algorand. The objective of the project was to evaluate the security of the smart contracts.

The following issues were identified during the initial assessment:

High Risk	Medium Risk	Low Risk
0	**1**	**2**
Fixed 0	Fixed 0	Fixed 0

Coinspect did not find any high risk issues on the Algorand implementation. The medium risk issue, WMH-1, reflects the fact that correctly using the Core contract is harder than in the Ethereum implementation. Low risk issues related to admin scripts and missing transaction validations that could be abused as phishing aids are reflected in WMH-2 and WMH-3. WMH-4 and WMH-5 are informational risks that currently present no risk but Coinspect believed Wormhole should be aware of.

2. Assessment and Scope

The audit started on **September 5** and was conducted on the **dev.v2** branch of the git repository at https://github.com/wormhole-foundation/wormhole as of commit **`811e17afb01bd94cb6d8beede9bbd170c9f43b17`** of **September 3.** The audit was conducted for three non-consecutive weeks.

The Algorand implementation of the contract is fairly similar to the one found in its other chains: there's a Core contract which is only responsible for emitting and verifying messages and a TokenBridge contract which uses the core functionality to provide bridging of assets between different chains. The main difference is found in its usage of Algorand's logic signatures as storage.

Coinspect found the design to be elegant: responsibilities are clearly set for every piece of the system and the reliance on the `verifyVAA` method in the contract makes it easy to reason about. Coinspect spent most of the time making sure interactions with `verifyVAA` where safe, as we believe this is the most critical part of the system. This method plus its more important user, `completeTransfer` on the Token Bridge, were considered the most critical in the whole system.

In general, the PyTEAL is clear and easy to follow. The memory layout is complex but this is due to Alogrand's restraints and it is well documented. Nevertheless, auditors found that auxiliary scripts meant to perform administrative functions were not working and did not follow Python's best practices. The testing of the contract is also lacking, a risk Coinspect helped reduce by improving the testing suite (see **Testing suite**).

Threat model

Coinspect's conducted the audit based on the the following threat model:

1. Validators are **trusted:** if the validator network colludes or turns malicious the whole Wormhole ecosystem would collapse. The application or operational security of the validators was not in scope for this audit.
2. The ultimate goal for an attacker is to bypass VAA verification so arbitrary or invalid messages are accepted by unsuspecting contracts.
3. The Wormhole Token Bridge is an ideal victim for VAA verification bypass.

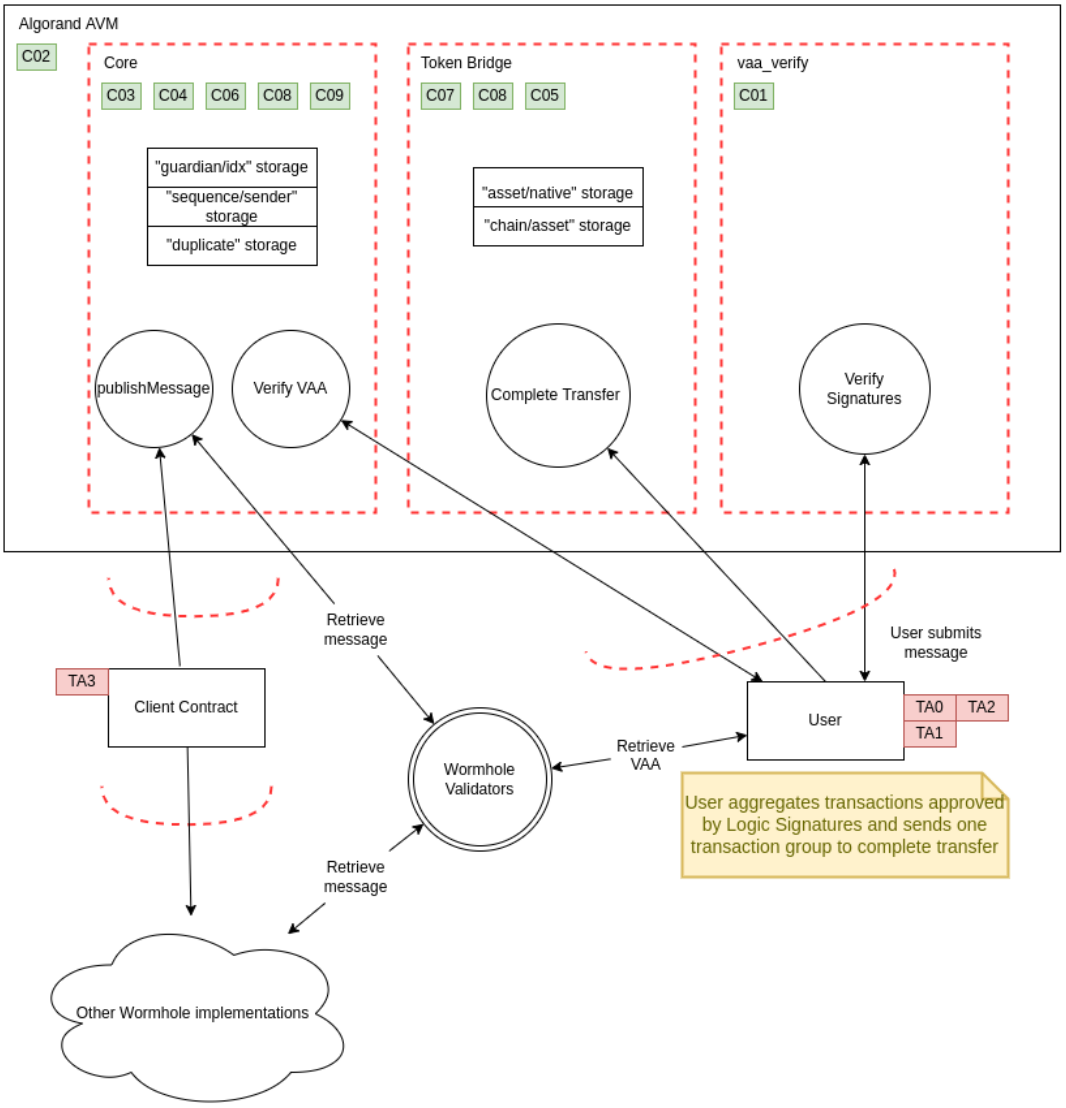
These main tenants guided the inspection and testing of the codebase. Although other possible attacks such as denial of service were not ignored, the priority was to bypass said protections.

Auditors explored several ideas which an attacker could try to leverage to attack the system:

1. By writing to certain memory slots of the *logic signatures* used by the contracts as storage or, conversely, by making the contract read from a wrong location.
2. By abusing logic errors in the parsing of VAAs which would lead to the same VAA to be considered different, leading to *replay attacks*.
3. By breaking the logical signatures invariant that hold that they only must be able to do a *rekey* transaction to their master contract and nothing else.
4. By abusing the fact that transactions in Algorand have many fields which are difficult and not obvious to check which can allow an attacker to spoof calls to other contracts or methods.

Security Controls	
ID	**Description**
C01	Verify Signatures: will check that provided signatures are valid for an **arbitrary** array of addresses.
C02	Algorand LogicSig Sender: the sender must be the hash of the expected program.
C03	Guardian Amount Guards: not zero but more than two thirds
C04	Expiration time check for Guardian set
C05	Duplicates check for VAA using emmiter_chain/emitter_addr/sequence
C06	Guardian reuse check for signatures
C07	Correct emitter: checks token bridge of other chain emitted the message
C08	Tx group checks: correct sender, arguments, previous calls are OK
C09	Sequence number is strictly increasing

Threat Actors	
ID	**Intention**
TA0	Squatter: control storage addresses so contracts can not use them
TA1	Minter: mint assets without a valid VAA
TA2	Denier: prevent issuance of new message by a contract
TA3	Impersonator: publish messages as another contract



A graphical representation of Coinspect threat models for the `completeTransfer`, attackers considered and the security controls found that make sure attackers are not successful.

Coinspect also considered attacks on other components of the system, such as:

1. Denial of service attacks, for example by being able to squat addresses that the contract needs to function.
2. Bypass of the `vaa_verify.py` logic signature
3. Griefing attacks against validators of the Wormhole network.
4. Initialization problems on the contract.
5. Governance problems.

For the purposes of the audit, the Wormhole Core contract was considered as a library contract: it provides no functionality by itself, and instead just provides verifications capabilities to other contracts (for example, the Token Bridge). Because of this, ease of use by its clients was considered as part of its threat model. Risks reflecting difficult interactions with the Core contract are reflected in WMH-1.

One type of attack that **was not** considered was a subversion of the validator sets: it is assumed for all of Wormhole operations that more than ⅓ of the validators are honest to ensure integrity and that more than ⅔ are honest to ensure liveness.

Testing suite

When running and examining the tests to interact with the contracts, Coinspect noted that the codebase was not well tested and that there were zero adversarial tests in the codebase.

Adversarial tests mimic inputs and actions that an attacker would try out. For example, submitting the same VAA twice to have a double-mint on the Token Bridge.

During the last week of the engagement, Coinspect improved this situation. The objective was to provide Wormhole with a base to continue developing adversarial tests so future upgrades to the contracts can be done swiftly and securely.

Overall, Coinspect added 2 randomized tests for the Token Bridge which indirectly test the Core `verifyVAA` function as well. Randomized tests are particularly important because they allow the testing suite to ensure that many different `tokenTransfer` VAA types uphold the basic invariants.

Coinspect also added several tests to the other critical parts of the system: `TmplSig.py`, `vaa_verify.py`, and `wormhole_core.py`.

These additional tests were provided to Wormhole and are easy to integrate into the current codebase.

3. Summary of Findings

Id	Title	Total Risk	Fixed
WMH-1	Hard for clients of Core to correctly call verifyVAA	Medium	✘
WMH-2	Admin helper script fails for some transactions	Low	✘
WMH-3	Missing transaction validations aid phishing attempts	Low	✘
WMH-4	VAAs are malleable due to version not being signed	Info	✘
WMH-5	Addresses can opt-in to the Core contract without rekeying	Info	✘

4. Detailed Findings

WMH-1 Hard for clients of Core to correctly call verifyVAA

Total Risk	Impact	Location
Medium	Medium	`/wormhole/algorand/wormhole_core.py`
Fixed ✘	Likelihood Medium	

Description

The Core contract is harder to use safely than it needs to be. By not adhering to `ARC04`, the contract forces clients to implement custom logic to use inner transactions or to rely exclusively on group transactions to very VAAs. Both are error prone.

What is more: even if `ARC04` was implemented, the `verifyVAA` does not return any useful data; forcing clients to parse VAAs manually to implement replay protection.

Contrast it with the Ethereum implementation as shown in the Messaging example client, where the client can just call the core contract and get a parsed VAA in response. This simple system is impossible in the Algorand Wormhole implementation.

Recommendation

Implement ARC04. Make it so clients can use an inner transition to retrieve key data from the verifyVAA call, for example the `(emitter, sequence)` tuple. This would make it more similar to the Ethereum implementation.

Otherwise, make sure that documentation clearly and unambiguously states how to parse VAAs and detect duplicate messages on Algorand.

Status

Open

WMH-2 Admin helper script fails for some transactions

Total Risk	Impact	Location
Low	Low	/wormhole/algorand/admin.py
Fixed	Likelihood	
✘	Low	

Description

There are bugs in the admin helper script that can make it fail. In an emergency, they could delay a prompt response.

In particular, Coinspect found and fixed two bugs.

1. When submitting a VAA of token transfer type, the `index` of the application is taken to be the full `ToAddress`. In reality, only the last eight bytes are needed. Trying to convert the whole address to int makes the Python process crash with an `OverflowError`.

```
if p["Meta"] == "TokenBridge Transfer With Payload":
    m = abi.Method("portal_transfer", [abi.Argument("byte[]")], abi.Returns("byte[]"))
    txns.append(transaction.ApplicationCallTxn(
        sender=sender.getAddress(),
        index=int.from_bytes(bytes.fromhex(p["ToAddress"]), "big"),
        on_complete=transaction.OnComplete.NoOpOC,
        app_args=[m.get_selector(), m.args[0].type.encode(vaa)],
        foreign_assets = foreign_assets,
        sp=sp
    ))
```

2. When the amount of signatures to be validated is a multiple of 9, an empty signature-block is made, which makes the transaction group fail.

```
# How many signatures can we process in a single txn... we can do 9!
bsize = (9*66)
blocks = int(len(p["signatures"]) / bsize) + 1
```

The second bug can affect governance messages.

Besides these concrete issues, running pylint returns 517 problems in the `admin.py` file, of which 19 are errors which could cause a program crash:

```
************* Module admin
admin.py:667:8: E0602: Undefined variable 'ret' (undefined-variable)
admin.py:667:35: E0602: Undefined variable 'ret' (undefined-variable)
```

```
admin.py:668:8: E0602: Undefined variable 'ret' (undefined-variable)
admin.py:669:23: E0602: Undefined variable 'ret' (undefined-variable)
admin.py:670:12: E0602: Undefined variable 'ret' (undefined-variable)
admin.py:671:15: E0602: Undefined variable 'ret' (undefined-variable)
admin.py:672:8: E0602: Undefined variable 'ret' (undefined-variable)
admin.py:673:8: E0602: Undefined variable 'ret' (undefined-variable)
admin.py:675:8: E0602: Undefined variable 'ret' (undefined-variable)
admin.py:677:8: E0602: Undefined variable 'ret' (undefined-variable)
admin.py:678:8: E0602: Undefined variable 'ret' (undefined-variable)
admin.py:680:8: E0602: Undefined variable 'ret' (undefined-variable)
admin.py:682:8: E0602: Undefined variable 'ret' (undefined-variable)
admin.py:684:8: E0602: Undefined variable 'ret' (undefined-variable)
admin.py:928:16: E1101: Instance of 'PortalCore' has no 'asset_optin' member (no-member)
admin.py:931:20: E1101: Instance of 'PortalCore' has no 'asset_optin' member (no-member)
admin.py:963:70: E0602: Undefined variable 'vaa' (undefined-variable)
admin.py:1168:16: E1101: Instance of 'PortalCore' has no 'asset_optin' member (no-member)
admin.py:1171:20: E1101: Instance of 'PortalCore' has no 'asset_optin' member (no-member)
```

Recommendation

Merge the adversarial testing suite into the main branch, as it contains fixes for the two main reported bugs. These fixes were required to make the adversarial testing suite work correctly.

Fix the 19 critical errors reported by `pylint` and inspect the file and the full output of the linter to evaluate if other recommendations of the tool are relevant.

Status

Open

WMH-3 Missing transaction validations aid phishing attempts

Total Risk	Impact	Location
Low	Low	`/wormhole/algorand/wormhole_core.py` `/wormhole/algorand/token_bridge.py`
Fixed	Likelihood	
✘	Low	

Description

Some methods do not check that the `rekey_to` or `close_remain_to` fields are set to zero. This can be leveraged by scammers to provide a transaction that *looks* reasonable, including setting the correct destination address, but then rekey the victim accounts or close assets to the scammer.

The following table summarizes where Coinspect found missing checks. Keep in mind that all transactions in the group should be checked.

Contract	Method	Missing Rekey Check	Missing `CloseRemainderTo` check
Core	`on_create`	MISSING	
	`on_update`	MISSING	
	`init`	MISSING	
	`nop`	MISSING	
	`publishMessage`	MISSING	MISSING
	`verifySigs`	MISSING	
	`verifyVAA`	MISSING	
	`optin`		MISSING

Token Bridge	`on_create`	MISSING	
	`on_update`	MISSING	
	`nop`	MISSING	
	`receiveAttest`	MISSING	
	`sendTransfer`	MISSING	
	`attestToken`	MISSING	
	`optin`		MISSING

Recommendation

Add the missing checks.

Status

Open

WMH-4	VAAs are malleable due to version not being signed

Total Risk	Impact	Location
Info	-	-
Fixed	**Likelihood**	
✘	-	

Description

The VAA format contains a *version* in the header which is **not** signed. Depending on implementation factors, this can allow an attacker to **change the version of a VAA** and cause it to be interpreted differently by the system.

Coinspect first noticed this issue in the Algorand implementation, but later also confirmed it in the Ethereum one. To be exploitable, the issue requires that clients support different VAA versions and use the verifyVAA method in TEAL or verifyVM in Solidity.

The issue is less likely in the Solidity implementation as a parseAndVerifyVM method is provided, which indeed asserts that the only version supported is 1. The Algorand contract provides no such parsing capabilities, leaving that responsibility to the client.

Recommendation

For the version field malleability, the only root-cause solution is to include the *version* field in the signed message. This would involve a considerable effort of engineering, as it would imply redeploying most of the contracts to support this new format.

If no new versions are foreseen in the future, put assertions in the codebase that only version 1 is supported in `verifyVAA`.

For the lack of parsing on Algorand's contract, it is recommended that the core contract returns the actual hash value of the VAA the user should use instead of making it a responsibility of the user.

Status

Wormhole team has acknowledged the issue and stated that it is a tradeoff and that replay protection can be implemented safely. We agree replay protection is not at risk as long as the hash is derived correctly (or any other part of the *signed message* is used to implement it, as in the Algorand implementation).

Nevertheless, there are risks with the message version being unsigned. Consider a toy example of V1 and V2 where the payload is simple: an int256 for V1 and uint256 for V2. A validator could sign a very big *negative* number, and then an attacker could change the version for V2 so when verifying, the message is interpreted as a very big *positive* number.

This risk is currently not present, so the finding is left as informative. Care should be taken when implementing new versions of VAAs so one cannot be confused for the other.

WMH-5 Addresses can opt-in to the Core contract without rekeying

Total Risk	Impact	Location
Info	-	/wormhole/algorand/wormhole_core.py
Fixed ✘	Likelihood -	

Description

The Core contract only validates fields of the second transaction in the group performing the opt-in operation. This allows subsequent transactions to optin without the proper requirements being enforced:

```
        @Subroutine(TealType.uint64)
        def optin():
            # Alias for readability
            algo_seed = Gtxn[0]
            optin = Gtxn[1]

            well_formed_optin = And(
                # Check that we're paying it
                algo_seed.type_enum() == TxnType.Payment,
                algo_seed.amount() == Int(seed_amt),
                algo_seed.receiver() == optin.sender(),
                # Check that its an opt in to us
                optin.type_enum() == TxnType.ApplicationCall,
                optin.on_completion() == OnComplete.OptIn,
                # Not strictly necessary since we wouldn't be seeing this unless it
was us, but...
                optin.application_id() == Global.current_application_id(),
                optin.rekey_to() == Global.current_application_address(),
                optin.application_args.length() == Int(0)
            )
```

Coinspect verified that the Token Bridge contract correctly valitates `Txn`.

Recommendation

Validate the current transaction (`Txn`) instead of `Gtxn[1]`.

Status

Open.

5. Disclaimer

The information presented in this document is provided "as is" and without warranty. The present security audit does not cover any off-chain systems or frontends that communicate with the contracts, nor the general operational security of the organization that developed the code.

---------- rawText-block