Smart Contract Audit

GARD

Smart Contract Audit

V220325

Prepared for GARD • March 2022

1. Executive Summary

In **March 2022, GARD** engaged Coinspect to perform a source code review of **GARD**. The objective of the project was to evaluate the security of the smart contracts.

Coinspect finds the protocol design concerning due to relying on off-chain mechanisms to ensure consistency on the voting mechanism. Besides this, no major issues were identified regarding design.

The following issues were identified during the assessment:

High Risk	Medium Risk	Low Risk
4	0	0
Fixed	Fixed	Fixed
4	0	0

2. Assessment and Scope

The audit started on **March 11, 2022** and was conducted on the `main` branch of the git repository at https://github.com/Tapera-Finance/CodeAudit as of commit **c67130a2e8fa8802c9b013d010bfddc2f7d2792e**.

```
f1f84e2920d410473b0bf36f018da31896ca5f20eea432748439810469c9b212  cdp_escrow.py
0e5d67c0e6d3420e9c442ae8394518c3cd3ebb1dd18ccfec92dbaba72860dc35  price_validator.py
8064f02c4684ef6835db12c90302bec4042c55c5fe532047647816fa3eaff41f  reserve_logic.py
c9881b55a97a1b451c496e65fe3a999694cd5ff306fbcb4f4d415be135f6173d  Stake.py
d4eda173d165f2fac42fc268a20f11851a898f90919dc49f7cd3775294359256  treasury.py
3bf90bd839873f546dbdbb6fe7c1f0d5f4e84cb7827ae25b9ca9bd1e9c439269  utils.py
d886b68fe15e341827f1b1f4d05613e68ef2d62cf58ae5aacc706467ae58d6d0  Vote_fee.py
931121df64c071b76eb3c3fb212bf2e2ff091daf3519ec1853e367a18df9899e  Vote_lib.py
594671f45cd601ebf32a1deddd259ea39697d2bebf3e3c4b64d493acd1c884a0  Vote_manager.py
```

The GARD protocol implements an algorithmic stablecoin on the Algorand blockchain. Similarly to other algorithmic stablecoin systems, it works by distributing new assets against users' ALGOs as collateral. In addition, users can still use their ALGOs to participate in Algorand Governance. It also implements a decentralized voting system to appoint a manager and modify the opening and closing fees.

The documentation and tests provided by the GARD team were minimal.

Four high impact issues were found in the protocol implementation. GARD-1 allows attackers to **delete or update three of the main smart contracts**. GARD-2 arises due to an error in the auction price function, and GARD-4 allows malicious liquidators to avoid paying liquidated users the auction price by sending it instead to the fee address. GARD-3 allows attackers to bypass the reserve validations by providing a malicious token, resulting in the loss of the security assumptions made by the price validator.

3. Summary of Findings

Id	Title	Total Risk	Fixed
GARD-1	Contracts can be deleted or updated	High	✔
GARD-2	Faulty auction_price function	High	✔
GARD-3	Attackers can bypass reserve validations	High	✔
GARD-4	Liquidated user loses all collateral	High	✔
GARD-5	has_voted is always true for Vote_id 0	Info	Fixed

4. Detailed Findings

GARD-1 Contracts can be deleted or updated

Total Risk	Impact	Location
High	High	`Vote_fee.py` `Vote_manager.py` `Stake.py`
Fixed	Likelihood	
✓	High	

Description

Several contracts fail to correctly validate `DeleteApplication` or `UpdateApplication` calls, allowing attackers to delete them, or update the code.

```
#Stake.py
program = Cond(
    [Txn.application_id() == Int(0), Approve()],
    [Txn.on_completion() == OnComplete.CloseOut, close_out(sender, asset_id)],
    [Txn.application_args[0] == Bytes("Add_vote"), add_vote_app],
    [Txn.application_args[0] == Bytes("Remove_vote"), remove_vote_app],
    [Txn.application_args[0] == Bytes("Lock_vote"), lock_vote_app],
    [Txn.application_args[0] == Bytes("Stake"), stake],
    [Txn.application_args[0] == Bytes("Unstake"), unstake],
    [Txn.application_args[0] == Bytes("Activate"), activate],
    [Txn.on_completion() == OnComplete.OptIn, Approve()],
    [Txn.on_completion() == OnComplete.DeleteApplication, Reject()],
    [Txn.on_completion() == OnComplete.UpdateApplication, Reject()],
)

#Vote_fee.py
program = Cond(
    [Txn.application_id() == Int(0), on_creation],
    [Txn.on_completion() == OnComplete.CloseOut, on_closeout],
    [Txn.application_args[0] == Bytes("Vote"), send_vote],
    [Txn.application_args[0] == Bytes("Cancel"), cancel_vote],
    [Txn.application_args[0] == Bytes("Init"), init_vote],
    [Txn.application_args[0] == Bytes("Close"), close_vote],
    [Txn.on_completion() == OnComplete.OptIn, Approve()],
    [Txn.on_completion() == OnComplete.DeleteApplication, Reject()],
    [Txn.on_completion() == OnComplete.UpdateApplication, Reject()],
)

#Vote_manager.py
program = Cond(
    [Txn.application_id() == Int(0), on_creation],
    [Txn.on_completion() == OnComplete.CloseOut, on_closeout],
    [Txn.application_args[0] == Bytes("Vote"), send_vote],
    [Txn.application_args[0] == Bytes("Cancel"), cancel_vote],
    [Txn.application_args[0] == Bytes("Init"), init_vote],
    [Txn.application_args[0] == Bytes("Close"), close_vote],
    [Txn.on_completion() == OnComplete.OptIn, Approve()],
    [Txn.on_completion() == OnComplete.DeleteApplication, Reject()],
    [Txn.on_completion() == OnComplete.UpdateApplication, Reject()],
)
```

The rejection of the `DeleteApplication` and `UpdateApplication` calls happen after all the other conditions are evaluated. This allows the attackers to perform those actions in any of the previous functions.

Recommendation

Reject invalid transactions first and then execute methods, or check for the correct `OnComplete` operation code.

Status

Followed recommendation.

GARD-2 Faulty auction_price function

Total Risk	Impact	Location
High	High	`price_validator.py`
Fixed ✓	Likelihood High	

Description

The `auction_price` function does not correctly compute the auction price for the liquidated ALGOs.

```
# Gets current price of collateral in the auction
# Decreases price linearly from 115% to 105% over 6 minutes
@Subroutine(TealType.uint64)
def auction_price():
  temp = ScratchVar(TealType.uint64)
  main = Seq(
    temp.store(App.localGet(Txn.sender(), Bytes("GARD_DEBT"))*Int(23)/Int(20)),
    If(Global.latest_timestamp() > App.localGet(Txn.sender(),
      Bytes("UNIX_START"))).Then(
        Seq(
          temp.store(temp.load()-(App.localGet(Txn.sender(),
            Bytes("GARD_DEBT"))*(Global.latest_timestamp() -
              App.localGet(Txn.sender(), Bytes("UNIX_START")))/Int(24))
        )
      )
    )
  )
  return Seq(main, Return(temp.load()))
```

The function computes `DEBT*1.15 - DEBT*Δt/24`. Replacing delta with 360 (6 minutes) and an arbitrary DEBT, we have: `50*1.15 - 50*360/24 = -692.5`.

Recommendation

Replace the subtraction second term dividend (24) with 3600.

Status

The function was changed to decrease the price linearly from 115% to 100%.

GARD-3 Attackers can bypass reserve validations

Total Risk	Impact	Location
High	High	`price_validators.py`
Fixed	Likelihood	
✔	High	

Description

Attackers can use crafted assets to bypass core validations and assumptions. These checks are critical for the safety of the protocol and can lead to funds stolen.

The price validator assumes the asset id passed in the Foreign Assets array corresponds to the stablecoin id. This is not a safe assumption since it can be manipulated by the users.

For instance, in `new_position`:

```
# application args["NewPosition", Int(unix_start)]
# (asset array args[stable_id, account_id])
new_position = And(
    Txn.applications[1] == price_app_id,
    Txn.applications[2] == open_app_id,
    Txn.rekey_to() == Global.zero_address(),
    Global.latest_timestamp() <= Btoi(Gtxn[0].application_args[1])
        + Int(30),
    Global.latest_timestamp() >= Btoi(Gtxn[0].application_args[1])
        - Int(30),
    Gtxn[3].asset_amount() <= Int(600000000000000000),
    Gtxn[3].asset_amount() >= Int(1000000),
    Gtxn[2].amount()
        >= Btoi(BytesDiv(BytesMul(Itob(Gtxn[3].asset_amount()
                * open_fee), Itob(Int(10) ** decimals)), Itob(Int(1000)
                * price))),
    Gtxn[3].asset_amount() * Int(7) / Int(5)
        <= Btoi(BytesDiv(BytesMul(Itob(Gtxn[1].amount()), Itob(price)),
                Itob(Int(10) ** decimals))),
    Seq(
        Assert(App.localGet(Int(1), Bytes('GARD_DEBT')) == Int(0)),
        Assert(get_reserve() == Gtxn[3].sender()),
        App.localPut(Int(1), Bytes('GARD_DEBT'),
                     Gtxn[3].asset_amount()),
        App.localPut(Int(1), Bytes('UNIX_START'),
                     Btoi(Gtxn[0].application_args[1]) / Int(2)
                     * Int(2)),
        App.localPut(Int(1), Bytes('EXTERNAL_APPCOUNT'), Int(0)),
        Int(1),
        ),
    )
```

Debt can be created for the sender, without reserve validation.

Recommendation

Validate Foreign assets array for expected values.

Status

Followed recommendation.

GARD-4 Liquidated user loses all collateral

Total Risk	Impact	Location
High	High	`price_validator.py`
Fixed ✓	Likelihood High	

Description

Malicious liquidators can send the remaining `GARD` from `auction_price` intended for the user to the `dev_fee` address.

```
Gtxn[2].asset_amount() + Gtxn[3].asset_amount() + Gtxn[4].asset_amount() >=
  Max(App.localGet(Txn.sender(), Bytes("GARD_DEBT")), auction_price()),
Gtxn[2].asset_amount() == App.localGet(Txn.sender(), Bytes("GARD_DEBT")),
Gtxn[3].asset_amount() >= Gtxn[4].asset_amount()/Int(5),
```

By having Gtxn[3] be a transfer of zero GARD, the liquidated user won't receive any of their corresponding assets.

Recommendation

Check that the liquidated user receives the remaining `GARD` from auction.

Status

Followed recommendation.

GARD-5 has_voted is always true for Vote_id 0

Total Risk	Impact	Location
Info	-	Vote_lib.py
Fixed	Likelihood -	

Description

Since after the first call to `init_vote_core` the `Vote_id` is 1, this has no impact, but if this changes in the future, it could lead to undesired behavior.

```
def has_voted(address: TealType.bytes, app_id: TealType.uint64) -> Expr:
       # Checks if `address` has voted in the last vote in `app_id`

       current_vote_id = global_must_get(Bytes("Vote_id"), app_id)
       last_voted_id = App.localGetEx(address, app_id, Bytes("Vote_id"))

        # We don't check hasValue because a user may *never* have voted in this
        #    vote, so a 0 value check is sufficient
       return Seq(last_voted_id, current_vote_id == last_voted_id.value())
```

Recommendation

Check for `hasValue`.

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.