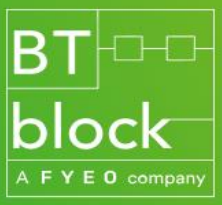

Security Assessment of the Ally for Algorand Smart Contracts

Maxos LLC

June 2022
Version 1.0

Presented by:
BTblock LLC

Corporate Headquarters
FYEO Inc.
PO Box 147044
Lakewood CO 80214
United States

Security Level
Strictly Confidential

Table of Contents

List of Figures

List of Tables

Executive Summary

Overview

Maxos LLC engaged Btblock, a FYEO Company, to perform a Security Assessment of the Ally for Algorand Smart Contracts.

The assessment was conducted remotely by the BTblock Security Team. Testing took place on May 17 - May 31, 2022, and focused on the following objectives:

- To provide the customer with an assessment of their overall security posture and any risks that were discovered within the environment during the engagement.
- To provide a professional opinion on the maturity, adequacy, and efficiency of the security measures that are in place.
- To identify potential issues and include improvement recommendations based on the results of our tests.

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

Key Findings

The following issues have been identified during the testing period. These should be prioritized for remediation to reduce to the risk they pose:

- F-MAXOS-01 – Fee funds to Ally can exceed 30%
- F-MAXOS-02 – Lack of checks for the committed Algos in pool and vaults
- F-MAXOS-03 – Pool Id can be set multiple times in Ally
- F-MAXOS-04 – Redeem price cannot be updated in Vaults
- F-MAXOS-05 – Imbalanced Algo/WAlgo ratio
- F-MAXOS-06 – Redeem amount is unnecessarily calculated twice in Vaults
- F-MAXOS-07 – The amount of distribute_algos is not validated in Pool

During the test, the following positive observations were noted regarding the scope of the engagement:

- The team was very supportive and open to discuss the design choices made

Based on formal verification we conclude that the reviewed code implements the documented functionality.

Scope and Rules of Engagement

BTblock performed a Security Assessment of the Ally for Algorand Smart Contracts. 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 through a private repository at https://github.com/MaxosLLC/ally with the commit hash 8d4bde2fc0b29b015fcd097180444f3c843a4fdf. A re-review was performed on June 9, 2022, with the commit hash 9e924ab69bb4e59cf512b33b0280a207584aa527.

Files included in the code review

```
ally/
├── admin/
│   ├── __init__.py
│   ├── claim_fee.py
│   ├── commit.py
│   ├── distribute.py
│   ├── distribute_ally.py
│   ├── set_ally_global.py
│   ├── set_governor.py
│   ├── set_pool_global.py
│   ├── set_vaults.py
│   ├── toggle_redeem.py
│   ├── vault_commit.py
│   ├── vault_redeem.py
│   ├── vault_release.py
│   ├── vault_set_governor.py
│   ├── vault_vote.py
│   └── vote.py
├── ally/
│   ├── action/
│   │   ├── admin.py
│   │   └── user.py
│   ├── contracts/
│   │   ├── __init__.py
│   │   ├── ally.py
│   │   ├── pool.py
│   │   └── vault.py
│   ├── __init__.py
│   ├── account.py
│   ├── environment.py
│   ├── process.py
│   ├── utils.py
```

```
│   └── vault.py
├── deploy/
│   ├── __init__.py
│   ├── bootstrap.py
│   ├── create.py
│   ├── destroy.py
│   └── update.py
├── helpers/
│   ├── build_msig.py
│   ├── merge_mtxn.py
│   └── setup.py
├── README.md
├── requirements.txt
└── sandbox
```

Table 1: Scope

Technical Analyses and Findings

During the Security Assessment of the Ally for Algorand Smart Contracts, we discovered:

- 4 findings with MEDIUM severity rating.
- 3 findings with LOW severity rating.

The following chart displays the findings by severity.

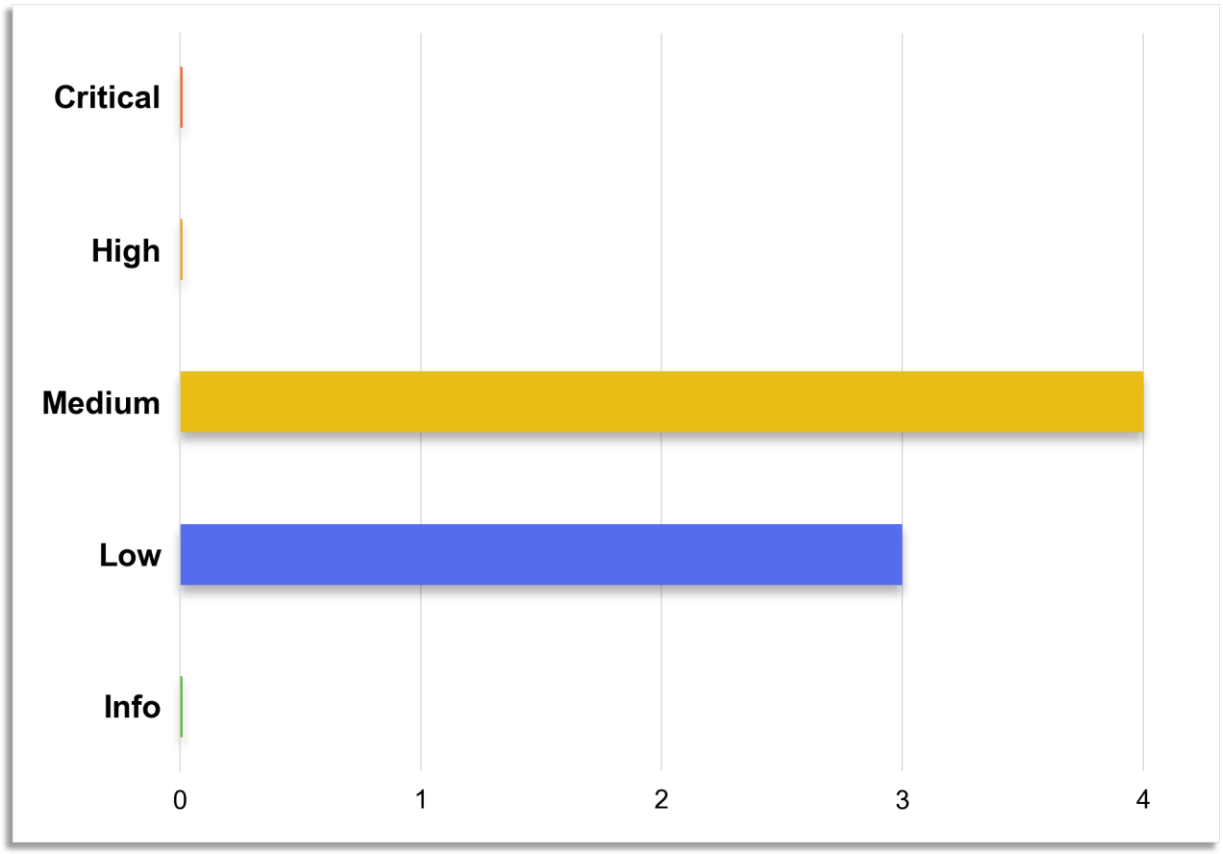

Figure 1: Findings by Severity

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.

Finding #	Severity	Description
F-MAXOS-01	Medium	Fee funds to Ally can exceed 30%
F-MAXOS-02	Medium	Lack of checks for the committed Algos in pool and vaults
F-MAXOS-03	Medium	Pool Id can be set multiple times in Ally
F-MAXOS-04	Medium	Redeem price cannot be updated in Vaults
F-MAXOS-05	Low	Imbalanced Algo/WAlgo ratio
F-MAXOS-06	Low	Redeem amount is unnecessarily calculated twice in Vaults
F-MAXOS-07	Low	The amount of distribute_algos is not validated in Pool

Table 2: Findings Overview

Technical Analysis

The source code has been manually validated to the extent that the state of the repository allowed. The validation includes confirming that the code correctly implements the intended functionality.

Conclusion

Based on formal verification, we conclude that the code implements the documented functionality to the extent of the reviewed code.

TECHNICAL FINDINGS

GENERAL OBSERVATIONS

Summary of Strengths

The Ally for Algorand code is well structured with proper variable and function names. It also contains very good documentation. In addition, the Maxos development team was very communicative, quickly providing responses to the auditing team. During the review, the following positive observations were noted regarding the scope of the engagement:

- Handles assertions correctly
- The general flow of application states is correct
- Checks for int overflows
- Handles group transactions correctly
- Checks transaction fields (AssetAmount, AssetReceiver, ReKeyTo, etc.)
- Checks AssetAmount
- Uses Type and TypeEnum to check if something is a payment or asset
- Multiplies integers before division, reducing the chance of over-frequent 0 values

Summary of Discovered Vulnerabilities

Several areas that would benefit from improvement were discovered. In addition to the documented findings, a number of important actions must be executed manually and frequently by an admin, and this assumes that the admin is always honest and available. These actions rely on the admin to operate properly, and this can become a single point of failure.

Maxos LL Response

The biggest security issue is the process for admin actions and code upgrades. The team will work to write a signing plan for each administrative action.

FEE FUNDS TO ALLY CAN EXCEED 30%

Finding ID: F-MAXOS-01
Severity: Medium
Status: Remediated

Description

The admin is responsible for sending fee funds to ally. The admin should send funds one time only after the government period ends; however, this function `claim_fee` can be called multiple times. Thus, the total fee funds in each period can exceed 30%.

Proof of Issue

The function does not check if the funds have already been sent.

File name: ally/contracts/pool.py
Line number: 184-206

```
    def claim_fee():
        contract_address = Global.current_application_address()
        ally_address = Txn.accounts[1]
        algo_balance = Balance(contract_address)
        current_ratio = algo_walgo_ratio()
        fee_percentage = App.globalGet(fee_percentage_key)
        last_commit_price = App.globalGet(last_commit_price_key)
        amount = WideRatio(
            [fee_percentage, algo_balance, current_ratio - last_commit_price],
            [Int(PRECISION), Int(100)]
        )

        return Seq(
            Assert(Txn.sender() == governor),
            Assert(current_ratio > last_commit_price),
            Assert(algo_balance > Int(1_000)),
            Assert(amount > Int(1_000)),
            pay(ally_address, amount),
            App.globalPut(mint_price_key, algo_walgo_ratio()),
            App.globalPut(redeem_price_key, algo_walgo_ratio()),
            App.globalPut(ally_reward_rate_key, Int(0)),
            Approve()
        )
```

Severity and Impact Summary

Total fee funds for each period can be greater than 30% maximum.

Recommendation

We recommend permitting the admin to claim the fee only one time after a government period ends.

LACK OF CHECKS FOR THE COMMITTED ALGOS IN POOL AND VAULTS

Finding ID: F-MAXOS-02
Severity: Medium
Status: Remediated

Description

An Algo-holding account can sign up for a government period by sending a zero-Algo pay transaction to a designated sign-up address, with the committed number of Algos encoded in the Notes field. This field is inputted by the governor via `Txn.application_args[1]` in vaults. However,

1. The encoded `commit_amount` in `Txn.application_args[1]` is not checked against the vault's balance.
2. The second argument `Txn.application_args[2]` is also the `commit_amount` which is not checked against the commit amount in the first argument. It is recorded in the contract with key `committed_algos_key` and is never used. The pool also has this value but is never used.

Proof of Issue

First app argument `"af/gov1:j" + json.dumps({"com": commit_amount})` which is then encoded in the `TxnField.note` without validation.

File name: admin/vault_commit.py
Line number: 31-39

```
    txn = transaction.ApplicationCallTxn(
        sender=env.sender,
        sp=env.client.suggested_params(),
        index=vault_id,
        app_args=["commit", "af/gov1:j" + json.dumps({"com": commit_amount}),
commit_amount, new_redeem_price],
        accounts=[env.governance],
        foreign_assets=[env.walgo_asa_id],
        on_complete=transaction.OnComplete.NoOpOC,
    )
```

File name: ally/contracts/vault.py
Line number: 78-105

```
    def commit():
        app_call = Gtxn[0]
        committed_algos = Btoi(Txn.application_args[2])
        redeem_price = Btoi(Txn.application_args[3])

        well_formed_commit = And(
            Global.group_size() == Int(1),
            app_call.type_enum() == TxnType.ApplicationCall,
            app_call.sender() == governor,
        )
```

```
        return Seq(
            Assert(well_formed_commit),
            InnerTxnBuilder.Begin(),
            InnerTxnBuilder.SetFields(
                {
                    TxnField.type_enum: TxnType.Payment,
                    TxnField.receiver: Txn.accounts[1],
                    TxnField.amount: Int(0),
                    TxnField.note: Txn.application_args[1],
                }
            ),
            InnerTxnBuilder.Submit(),
            App.globalPut(committed_algos_key, committed_algos),
            App.globalPut(allow_redeem_key, Int(1)),
            App.globalPut(redeem_price_key, redeem_price),
            Approve(),
        )
```

Severity and Impact Summary

The inputted commit amount may be less than the current vault balance and this may lead to loss of rewards.

Recommendation

We recommend adding checks for the commit amount.

POOL ID CAN BE SET MULTIPLE TIMES IN ALLY

Finding ID: F-MAXOS-03
Severity: Medium
Status: Remediated

Description

The parameter `pool_id` should only be allowed to be initialized once. The current implementation of the `set_pool_id` function is problematic. Firstly, when the `pool_id` is updated, this links the ally to another pool in which the `ally_rewards` balance could potentially be zero. If the new pool has zero rewards, this implies that ally rewards from previous pools are locked, and no users can claim ally tokens. Secondly, the admin can set the `pool_id` many times. The admin could link the ally to a malicious pool that sets the `ally_rewards` to any amount. They could then withdraw/claim all ally tokens, set it back to the original pool, and as a result, no users can claim any ally tokens.

Proof of Issue

File name: ally/contracts/ally.py
Line number: 111-117

```
    def set_pool_id():
        new_pool_id = Txn.application_args[1]
        return Seq(
            Assert(Txn.sender() == governor),
            App.globalPut(pool_id_key, Btoi(new_pool_id)),
            Approve()
        )
```

Severity and Impact Summary

As mentioned in the description, the ally rewards from a previous pool can be locked. Moreover, as the amount of ally rewards is read from the local state of the pool, a malicious admin can link the ally to a malicious pool and claim any amount.

Recommendation

We recommend restricting the `pool_id` to be set only once.

REDEEM PRICE CANNOT BE UPDATED IN VAULTS

Finding ID: F-MAXOS-04
Severity: Medium
Status: Remediated

Description

Redeem price can be updated anytime in the pool, but it can only be updated when the governor makes a commit. If the redeem price is updated multiple times during the government period, the redeem price in the vaults can't be updated accordingly.

Proof of Issue

`redeem_price` can only be updated when the governor makes a commit.

File name: ally/contracts/vault.py
Line number: 78-105

```
    def commit():
        app_call = Gtxn[0]
        committed_algos = Btoi(Txn.application_args[2])
        redeem_price = Btoi(Txn.application_args[3])

        well_formed_commit = And(
            Global.group_size() == Int(1),
            app_call.type_enum() == TxnType.ApplicationCall,
            app_call.sender() == governor,
        )

        return Seq(
            Assert(well_formed_commit),
            InnerTxnBuilder.Begin(),
            InnerTxnBuilder.SetFields(
                {
                    TxnField.type_enum: TxnType.Payment,
                    TxnField.receiver: Txn.accounts[1],
                    TxnField.amount: Int(0),
                    TxnField.note: Txn.application_args[1],
                }
            ),
            InnerTxnBuilder.Submit(),
            App.globalPut(committed_algos_key, committed_algos),
            App.globalPut(allow_redeem_key, Int(1)),
            App.globalPut(redeem_price_key, redeem_price),
            Approve(),
        )
```

However, `redeem_price` in the pool can be updated anytime.

File name: ally/contracts/pool.py
Line number: 312-319

```
    def set_redeem_price():
        new_redeem_price = Btoi(Txn.application_args[1])
        return Seq(
            Assert(Txn.sender() == governor),
            Assert(new_redeem_price < algo_walgo_ratio()),
            App.globalPut(redeem_price_key, new_redeem_price),
            Approve()
        )
```

Severity and Impact Summary

This can lead to price mismatch, and users can redeem more or less than they should have.

Recommendation

We recommend adding another `set_redeem_price` function in the vaults.

IMBALANCED ALGO/WALGO RATIO

Finding ID: F-MAXOS-05
Severity: Low
Status: Open

Description

The `WideRatio` expression is used to calculate expressions of the form

`(N_1 * N_2 * N_3 * ...) / (D_1 * D_2 * D_3 * ...)`

and it will round down to an integer. If users mint/redeem some very small amount, they will receive zero WAlgo/Algo. Thus, even if users mint and redeem at 1/1, there would still be more Algo than WAlgo.

Proof of Issue

For example, suppose `redeem_price_key < 1000000` and `amount = 1`. Because `redeem_price_key * amount / PRECISION = 0.99`, the resulting `algos_to_redeem` will return `0`.

File name: ally/contracts/pool.py
Line number: 80-87

```
@Subroutine(TealType.uint64)
def algos_to_redeem(amount):
    algos = WideRatio(
        [App.globalGet(redeem_price_key), amount],
        [Int(PRECISION)]
    )
```

Similarly, users get 0 `wAlgo` if `mint_price_key increases > 1000000` and `amount = 1`. Because `PRECISION * amount / mint_price_key = 0.99`, the resulting `walgos_to_mint` will return `0`.

File name: ally/contracts/pool.py
Line number: 69-76

```
@Subroutine(TealType.uint64)
def walgos_to_mint(algos):
    amount = WideRatio(
        [Int(PRECISION), algos],
        [App.globalGet(mint_price_key)]
    )
```

Severity and Impact Summary

If users mint/redeem some very small amount, they will receive zero WAlgo/Algo and lose the Algo/WAlgo. Additionally, even if users mint and redeem at a 1:1 ratio, there would still be more Algo than WAlgo.

Recommendation

We recommend adding another assert to `walgos_to_mint` and `algos_to_redeem`, i.e., `Assert(amount > 0)` to `walgos_to_mint` and `Assert(algos > 0)` to `algos_to_redeem`.

This may not resolve this problem completely because numbers are always rounded down by `WideRatio`.

REDEEM AMOUNT IS UNNECESSARILY CALCULATED TWICE IN VAULTS

Finding ID: F-MAXOS-06
Severity: Low
Status: Remediated

Description

The variable `redeem_amount` is calculated twice. This will cost more compute units than needed.

Proof of Issue

File name: ally/contracts/vault.py
Line number: 130-157

```
    def redeem():
        contract_address = Global.current_application_address()
        pool_address = App.globalGet(pool_address_key)
        pool_token = App.globalGet(pool_token_key)
        algo_balance = Balance(contract_address)
        app_call = Gtxn[0]
        asset_xfer = Gtxn[1]
        redeem_amount = algos_to_redeem(asset_xfer.asset_amount())
        return Seq(
            Assert(App.globalGet(allow_redeem_key)),
            Assert(
                And(
                    Global.group_size() == Int(2),
                    app_call.type_enum() == TxnType.ApplicationCall,
                    app_call.assets[0] == pool_token,
                    asset_xfer.type_enum() == TxnType.AssetTransfer,
                    asset_xfer.sender() == app_call.sender(),
                    asset_xfer.asset_receiver() == pool_address,
                    asset_xfer.xfer_asset() == pool_token,
                    algo_balance > redeem_amount
                )
            ),
            pay(
                asset_xfer.sender(),
                algos_to_redeem(asset_xfer.asset_amount())
            ),
            Approve(),
        )
```

Severity and Impact Summary

This will add several extra units of cost.

Recommendation

We recommend using `redeem_amount` for `pay`.

THE AMOUNT OF DISTRIBUTE_ALGOS IS NOT VALIDATED IN POOL

Finding ID: F-MAXOS-07
Severity: Low
Status: Remediated

Description

In the `distribute_algo()` function, the inputted `algo_amount` is not checked against the current balance of the pool. The `algo_amount` should be less than the current balance (or equal to one third of the pool - `one_algo` in the event of even distribution).

Proof of Issue

File name: ally/contracts/pool.py
Line number: 253-259

```
        def distribute_algo():
        algo_amount = Btoi(Txn.application_args[1])
        group = Txn.application_args[2]
        sub_commit_algos = Div(
            algo_amount,
            GROUP_COUNT
        )
```

Severity and Impact Summary

This may cause the transaction to fail.

Recommendation

We recommend adding checks for `algo_amount`.

Our Process

Methodology

BTblock, a FYEO Company, uses the following high-level methodology when approaching engagements. They are broken up into the following phases.

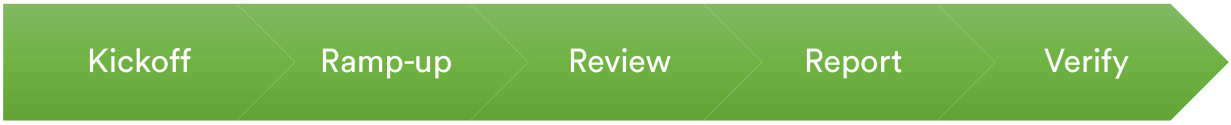

Figure 2: Methodology Flow

Kickoff

The project is kicked off as the sales process has concluded. We typically set up a kickoff meeting where project stakeholders are gathered to discuss the project as well as the responsibilities of participants. During this meeting we verify the scope of the engagement and discuss the project activities. It's an opportunity for both sides to ask questions and get to know each other. By the end of the kickoff there is an understanding of the following:

- Designated points of contact
- Communication methods and frequency
- Shared documentation
- Code and/or any other artifacts necessary for project success
- Follow-up meeting schedule, such as a technical walkthrough
- Understanding of timeline and duration

Ramp-up

Ramp-up consists of the activities necessary to gain proficiency on the particular project. This can include the steps needed for familiarity with the codebase or technological innovation utilized. This may include, but is not limited to:

- Reviewing previous work in the area including academic papers
- Reviewing programming language constructs for specific languages
- Researching common flaws and recent technological advancements

Review

The review phase is where most of the work on the engagement is completed. This is the phase where we analyze the project for flaws and issues that impact the security posture. Depending on the project

this may include an analysis of the architecture, a review of the code, and a specification matching to match the architecture to the implemented code.

In this code audit, we performed the following tasks:

1. Security analysis and architecture review of the original protocol
2. Review of the code written for the project
3. Compliance of the code with the provided technical documentation

The review for this project was performed using manual methods and utilizing the experience of the reviewer. No dynamic testing was performed, only the use of custom-built scripts and tools were used to assist the reviewer during the testing. We discuss our methodology in more detail in the following sections.

Code Safety

We analyzed the provided code, checking for issues related to the following categories:

- General code safety and susceptibility to known issues
- Poor coding practices and unsafe behavior
- Leakage of secrets or other sensitive data through memory mismanagement
- Susceptibility to misuse and system errors
- Error management and logging

This list is general and not comprehensive, meant only to give an understanding of the issues we are looking for.

Technical Specification Matching

We analyzed the provided documentation and checked that the code matches the specification. We checked for things such as:

- Proper implementation of the documented protocol phases
- Proper error handling
- Adherence to the protocol logical description

Reporting

BTblock, a FYEO Company, delivers a draft report that contains an executive summary, technical details, and observations about the project.

The executive summary contains an overview of the engagement including the number of findings as well as a statement about our general risk assessment of the project. We may conclude that the overall risk is low but depending on what was assessed we may conclude that more scrutiny of the project is needed.

We not only report security issues identified but also informational findings for improvement categorized into several buckets:

- Critical
- High
- Medium
- Low
- Informational

The technical details are aimed more at developers, describing the issues, the severity ranking and recommendations for mitigation.

As we perform the audit, we may identify issues that aren't security related, but are general best practices and steps that can be taken to lower the attack surface of the project. We will call those out as we encounter them and as time permits.

As an optional step, we can agree on the creation of a public report that can be shared and distributed with a larger audience.

VERIFY

After the preliminary findings have been delivered, this could be in the form of the approved communication channel or delivery of the draft report, we will verify any fixes within a window of time specified in the project. After the fixes have been verified, we will change the status of the finding in the report from open to remediated.

The output of this phase will be a final report with any mitigated findings noted.

ADDITIONAL NOTE

It is important to note that, although we did our best in our analysis, no code audit or assessment is a guarantee of the absence of flaws. Our effort was constrained by resource and time limits along with the scope of the agreement.

While assessing the severity of the findings, we considered the impact, ease of exploitability, and the probability of attack. This is a solid baseline for severity determination.

THE CLASSIFICATION OF VULNERABILITIES

Security vulnerabilities and areas for improvement are weighted into one of several categories using, but is not limited to, the criteria listed below:

Critical – vulnerability will lead to a loss of protected assets

- This is a vulnerability that would lead to immediate loss of protected assets

- The complexity to exploit is low
- The probability of exploit is high

High - vulnerability has potential to lead to a loss of protected assets

- All discrepancies found where there is a security claim made in the documentation that cannot be found in the code
- All mismatches from the stated and actual functionality
- Unprotected key material
- Weak encryption of keys
- Badly generated key materials
- Txn signatures not verified
- Spending of funds through logic errors
- Calculation errors overflows and underflows

Medium - vulnerability hampers the uptime of the system or can lead to other problems

- Insecure calls to third party libraries
- Use of untested or nonstandard or non-peer-reviewed crypto functions
- Program crashes, leaves core dumps or writes sensitive data to log files

Low – vulnerability has a security impact but does not directly affect the protected assets

- Overly complex functions
- Unchecked return values from 3rd party libraries that could alter the execution flow

Informational

- General recommendations