FAIRYPROOF

Panaroma Token

AUDIT REPORT

Version 1.0.0

Serial No. 2022100600012016

Presented by Fairyproof

October 6, 2022

01. Introduction

This document includes the results of the audit performed by the Fairyproof team on the Panaroma token project.

Audit Start Time:

September 5, 2022

Audit End Time:

October 6, 2022

https://github.com/Panaroma-Finance/Panaroma-Token-Smart-Contract

Audited Code's Github Commit Number When Audit Started:

bf38348f1093ae6892bd0b28f66aee37808299e9

Audited Code's Github Commit Number When Audit Ended:

eeb47b271942ddf42ce2aa3c9cf16a590877c333

Audited Source Files:

The calculated SHA-256 value for the audited file when the audit was done is as follows:

```
PanaromaToken.sol:
0xcb7056974354c027628a1b039a63f877cdaf2a7e752b577160699dbd1c54ed85
```

The source files audited include all the files with the extension "sol" as follows:

```
./
└── PanaromaToken.sol

0 directories, 1 file
```

The goal of this audit is to review Panaroma's solidity implementation for its token issuance function, study potential security vulnerabilities, its general design and architecture, and uncover bugs that could compromise the software in production.

We make observations on specific areas of the code that present concrete problems, as well as general observations that traverse the entire codebase horizontally, which could improve its quality as a whole.

This audit only applies to the specified code, software or any materials supplied by the Panaroma team for specified versions. Whenever the code, software, materials, settings, environment etc is changed, the comments of this audit will no longer apply.

— Disclaimer

Note that as of the date of publishing, the contents of this report reflect the current understanding of known security patterns and state of the art regarding system security. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk.

The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. If the audited source files are smart contract files, risks or issues introduced by using data feeds from offchain sources are not extended by this review either.

Given the size of the project, the findings detailed here are not to be considered exhaustive, and further testing and audit is recommended after the issues covered are fixed.

To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

— Methodology

The above files' code was studied in detail in order to acquire a clear impression of how the its specifications were implemented. The codebase was then subject to deep analysis and scrutiny, resulting in a series of observations. The problems and their potential solutions are discussed in this document and, whenever possible, we identify common sources for such problems and comment on them as well.

The Fairyproof auditing process follows a routine series of steps:

1. Code Review, Including:

- Project Diagnosis

Understanding the size, scope and functionality of your project's source code based on the specifications, sources, and instructions provided to Fairyproof.

- Manual Code Review

Reading your source code line-by-line to identify potential vulnerabilities.

- Specification Comparison

Determining whether your project's code successfully and efficiently accomplishes or executes its functions according to the specifications, sources, and instructions provided to Fairyproof.

2. Testing and Automated Analysis, Including:

- Test Coverage Analysis

Determining whether the test cases cover your code and how much of your code is exercised or executed when test cases are run.

- Symbolic Execution

Analyzing a program to determine the specific input that causes different parts of a program to execute its functions.

3. Best Practices Review

Reviewing the source code to improve maintainability, security, and control based on the latest established industry and academic practices, recommendations, and research.

— Structure of the document

This report contains a list of issues and comments on all the above source files. Each issue is assigned a severity level based on the potential impact of the issue and recommendations to fix it, if applicable. For ease of navigation, an index by topic and another by severity are both provided at the beginning of the report.

— Documentation

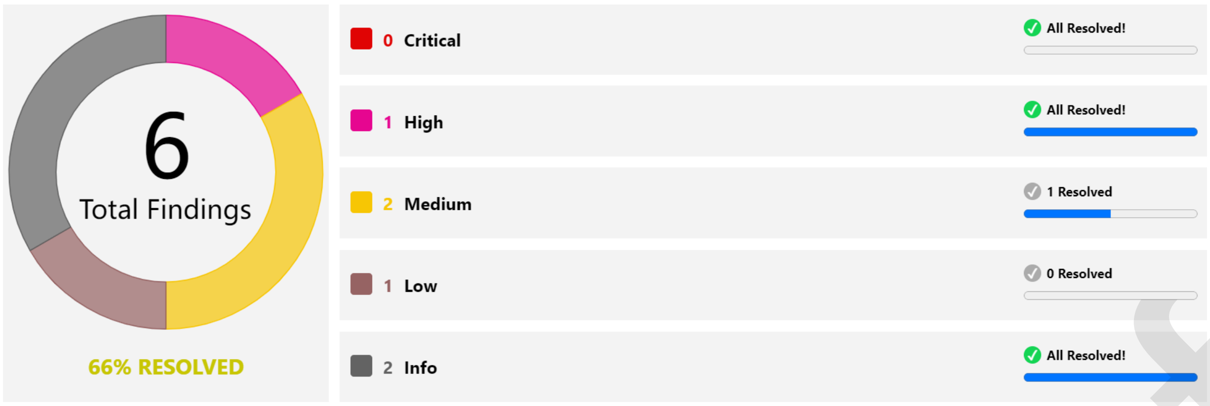
For this audit, we used the following sources of truth about how the token issuance function should work:

Contract Source Code

This was considered the specification, and when discrepancies arose with the actual code behavior, we consulted with the Panaroma team or reported an issue.

— Comments from Auditor

Serial Number	Auditor	Audit Time	Result
2022100600012016	Fairyproof Team	2022.09.05 - 2022.10.6	Low Risk



Summary:

The Fairyproof security team used its auto analysis tools and manual work to audit the project. During the audit, one issue of high-severity, two issues of medium-severity, one issue of low-severity and two issues of informational-severity were uncovered. The Panaroma team fixed one risk of high-severity, one issue of medium-severity and two issues of informational-severity, and acknowledged one issue of medium-severity and one issue of low-severity.

02. About Fairyproof

Fairyproof is a leading technology firm in the blockchain industry, providing consulting and security audits for organizations. Fairyproof has developed industry security standards for designing and deploying blockchain applications.

03. Major functions of audited code

The audited code mainly implements the following function:

Token Issuance:

Token Implementation: Non-standard ERC-20 Token

Address: 0xa60E0DF948708764F5397e556973a520Db378598 (Ethereum)

Token Name: Panaroma Token

Token Symbol: PANA

Token Decimals: 8

Total Supply: 500,000,000

Subsequent Minting: Yes

Contract Upgradeable: Yes

Burn: Yes

Pausable: Yes

Transfer Can be Frozen: Yes (Blacklist)

Note: this contract is implemented by referring to the USDT's implementation and is not a standard ERC-20 token. Some of the functions don't have a return value. When interacting with other smart contracts, this should be kept in mind.

04. Coverage of issues

The issues that the Fairyproof team covered when conducting the audit include but are not limited to the following ones:

- Re-entrancy Attack
- Replay Attack
- Reordering Attack
- Miner's Advantage
- Rollback Attack
- DDoS Attack
- Injection Attack
- Transaction Ordering Attack
- Race Condition
- Access Control
- Integer Overflow/Underflow
- Timestamp Attack
- Gas Consumption
- Inappropriate Callback Function
- Function Visibility
- Implementation Vulnerability
- Uninitialized Storage Pointer
- Arithmetic Precision
- Tx.origin
- Fake Deposit
- Shadow Variable
- Design Vulnerability
- Token Issuance

- Admin Rights
- Inappropriate Proxy Design
- Inappropriate Use of Slots
- Asset Security
- Contract Upgrade/Migration
- Code Improvement
- Misc

05. Severity level reference

Every issue in this report was assigned a severity level from the following:

Critical severity issues need to be fixed as soon as possible.

High severity issues will probably bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Informational is not an issue or risk but a suggestion for code improvement.

06. Major areas that need attention

Based on the provided source code the Fairyproof team focused on the possible issues and risks related to the following functions or areas.

- Function Implementation

We checked whether or not the functions were correctly implemented.

We found one issue, for more details please refer to FP-4 in "08. Issue description".

- Integer Overflow/Underflow

We checked all the code sections, which had arithmetic operations and might introduce integer overflow or underflow if no safe libraries were used. All of them used safe libraries.

We didn't find issues or risks in these functions or areas at the time of writing.

- Access Control

We checked each of the functions that could modify a state, especially those functions that could only be accessed by "owner".

We didn't find issues or risks in these functions or areas at the time of writing.

- Token Issuance & Exchange

We checked whether or not the contract files could mint tokens at will.

We found two issues, for more details please refer to FP-1 , FP-3 and FP-2 in "08. Issue description".

- State Update

We checked some key state variables which should only be set at initialization.

We didn't find issues or risks in these functions or areas at the time of writing.

- Asset Security

We checked whether or not all the functions that transfer assets were safely handled.

We didn't find issues or risks in these functions or areas at the time of writing.

- Miscellaneous

We found one issues, for more details please refer to FP-5 and FP-6 in "08. Issue description".

07. List of issues by severity

Index	Title	Issue/Risk	Severity	Status
FP-1	Wrong Constructor Name	Design Vulnerability	High	✓ Fixed
FP-2	Uncapped Supply	Design Vulnerability	Medium	Fixed
FP-3	Excessive Access Control	Design Vulnerability	Medium	Acknowledged
FP-4	Incompatible with ERC-20	Design Vulnerability	Low	Acknowledged
FP-5	Redundant Code	Code Improvement	Info	✓ Fixed
FP-6	Code with Unknown Logic	Code Improvement	Info	✓ Fixed

08. Issue descriptions

[FP-1] Wrong Constructor Name High ✓ Fixed

Issue/Risk: Design Vulnerability

Description:

In earlier versions of Solidity , a contract and its constructor have the same name. In `PanaromaToken.sol`, the function `BoomToken` should be renamed to `PanaromaToken`. Otherwise `BoomToken` would be an external function that can be called permissionlessly.

The constructor name is incorrect and the corresponding parameters cannot be passed on deployment.

Recommendation:

Consider renaming `BoomToken` to `PanaromaToken`.

Update/Status:

The Panaroma team has fixed this.

[FP-2] Uncapped Supply Medium Fixed

Issue/Risk: Design Vulnerability

Description:

In the `PanaromaToken` contract, the owner can arbitrarily mint tokens and burn tokens of any account.

Recommendation:

Consider applying a cap on the supply.

Update:

The Panaroma team has added code to limit the minting and it won't exceed the max supply.

Status:

The Panaroma team has partially fixed this issue.

[FP-3] Excessive Access Control Medium Acknowledged

Issue/Risk: Design Vulnerability

Description:

In `PanaromaToken.sol`, token transfer can be paused, the admin can freeze or burn users' tokens. Users should be aware of this and keep this in mind.

Recommendation:

Consider managing `owner`'s access control with great care and proceeding with this access control cautiously.

Update/Status:

The Panaroma team has acknowledged this issue.

[FP-4] Incompatible with ERC-20 Low Acknowledged

Issue/Risk: Design Vulnerability

Description:

This `PanaromaToken` contract is incompatible with ERC-20. The `transfer` , `transferFrom` and `approve` functions do not return boolean values.

To call these three functions in other contracts, it is recommended to use the low-level call functions.

Recommendation:

Consider using the low-level call functions or not handling the return value.

Update/Status:

The Panaroma team has acknowledged this issue.

[FP-5] Redundant Code Info ✓ Fixed

Issue/Risk: Code Improvement

Description:

In `PanaromaToken.sol` the function `mint(uint256)` was redundant because the existing `issue(uint)` and `mintTo(address,uint256)` functions can work exactly the same.

Recommendation:

Consider removing the function.

Update/Status:

The Panaroma team has removed the `issue` function.

[FP-6] Code with Unknown Logic Info Fixed

Issue/Risk: Code Improvement

Description:

In `PanaromaToken.sol`, the `mintTo`, `mint` and `burnFrom` functions all had the following line of code:

```
require(_totalSupply >= value);
```

The logic of this conditional check is unknown.

Recommendation:

Consider implementing an understandable logic.

Update/Status:

The Panaroma team has removed the check.

09. Recommendations to enhance the overall security

We list some recommendations in this section. They are not mandatory but will enhance the overall security of the system if they are adopted.

- Consider managing the admin's access control with great care and trasferring it to a multi-sig wallet or DAO when necessary.

10. Appendices

10.1 Unit Test File

```
const { expect } = require("chai");
const { ethers } = require("hardhat");

describe("PanaromaToken", function () {
    let instance;
    let owner,user1,user2,users;
    const init_supply = ethers.utils.parseUnits("2000000",6);
    const max_supply = ethers.BigNumber.from("50000000000000000");
    before(async () => {
        [owner,user1,user2,...users] = await ethers.getSigners();
    });

    beforeEach(async() => {
        const PanaromaToken = await ethers.getContractFactory("PanaromaToken");
        instance = await PanaromaToken.deploy(init_supply,"Panaroma","PAN",6);
    });

    describe("Meta test", function() {
        it("name | symbol | decimals test", async () => {
            expect(await instance.name()).to.be.equal("Panaroma");
            expect(await instance.symbol()).to.be.equal("PAN");
            expect(await instance.decimals()).to.be.equal(6);
        });
    });

    describe("Init status test", function() {
        it("init_supply test", async () => {
            expect(await instance.totalSupply()).to.be.equal(init_supply);
            expect(await
instance.balanceOf(owner.address)).to.be.equal(init_supply);
        });
    });

    describe("approve and allowance test", function() {
        it("approve should change allowance", async () => {
            expect(await
instance.allowance(user1.address,user2.address)).to.be.equal(0);
            await
expect(instance.connect(user1).approve(user2.address,10000)).to.be.emit(
                instance,"Approval"
            ).withArgs(user1.address,user2.address,10000);
            expect(await
instance.allowance(user1.address,user2.address)).to.be.equal(10000);
```

```
        });
    });

    describe("transferFrom and transfer test", function() {
        it("transfer should change balance", async () => {
            await expect(instance.transfer(user1.address,1000)).to.be.emit(
                instance,"Transfer"
            ).withArgs(owner.address,user1.address,1000);
            expect(await
instance.balanceOf(owner.address)).to.be.equal(init_supply.sub(1000));
            expect(await instance.balanceOf(user1.address)).to.be.equal(1000);
            expect(await instance.totalSupply()).to.be.equal(init_supply);
        });

        it("transfer to self shouldn't change balance", async () => {
            await instance.transfer(owner.address,1000);
            expect(await
instance.balanceOf(owner.address)).to.be.equal(init_supply);
            expect(await instance.totalSupply()).to.be.equal(init_supply);
        });

        it("transfer should failed while sender has insufficient tokens", async ()
=> {
            await
expect(instance.connect(user1).transfer(user2.address,10)).to.be.reverted;
        });

        it("transferFrom without approval should be failed", async () => {
            await
expect(instance.connect(user1).transferFrom(owner.address,user2.address,10)).to.be
.reverted;
        });

        it("transferFrom should change balance and allowance", async () => {
            await instance.approve(user1.address, 10000);
            await
expect(instance.connect(user1).transferFrom(owner.address,user2.address,3000)).to.
be.emit(
                instance,"Transfer"
            ).withArgs(owner.address,user2.address,3000);
            expect(await
instance.balanceOf(owner.address)).to.be.equal(init_supply.sub(3000));
            expect(await instance.balanceOf(user1.address)).to.be.equal(0);
            expect(await instance.balanceOf(user2.address)).to.be.equal(3000);
            expect(await instance.totalSupply()).to.be.equal(init_supply);
            expect(await
instance.allowance(owner.address,user1.address)).to.be.equal(7000);
        });
    });

    describe("minter and burn", async () => {

        it("mint or burn without owner should be failed", async () => {
            await
expect(instance.connect(user1).mintTo(user1.address,100)).to.be.reverted;
```

```
            await expect(instance.connect(user1).mint(100)).to.be.reverted;
            await
expect(instance.connect(user1).burnFrom(owner.address,100)).to.be.reverted;
        })
        it("mintTo should change state and emit event ", async () => {
            await expect(instance.mintTo(user1.address,10000)).to.be.emit(
                instance,"Transfer"
            ).withArgs(ethers.constants.AddressZero,user1.address,10000);

            expect(await
instance.balanceOf(owner.address)).to.be.equal(init_supply);
            expect(await instance.balanceOf(user1.address)).to.be.equal(10000);
            expect(await
instance.totalSupply()).to.be.equal(init_supply.add(10000));
        });

        it("mint should change state and emit event ", async () => {
            await expect(instance.mint(10000)).to.be.emit(
                instance,"Transfer"
            ).withArgs(ethers.constants.AddressZero,owner.address,10000);

            expect(await
instance.balanceOf(owner.address)).to.be.equal(init_supply.add(10000));
            expect(await
instance.totalSupply()).to.be.equal(init_supply.add(10000));
        });

        it("mint beyond max_supply should be failed", async () => {
            let value = max_supply.sub(init_supply).add(1)
            await expect(instance.mint(value)).to.be.reverted;
        });

        it("BurnFrom should change state and emit event", async () => {
            await instance.mintTo(user1.address,10000);
            await instance.connect(user1).approve(owner.address,10000);
            expect(await instance.balanceOf(user1.address)).to.be.equal(10000);
            expect(await
instance.allowance(user1.address,owner.address)).to.be.equal(10000);
            await expect(instance.burnFrom(user1.address,100)).to.be.emit(
                instance,"Transfer"
            ).withArgs(user1.address,ethers.constants.AddressZero,100);
            expect(await instance.balanceOf(user1.address)).to.be.equal(9900);
            expect(await
instance.allowance(user1.address,owner.address)).to.be.equal(9900);
            await expect(instance.burnFrom(user1.address,10000)).to.be.reverted;

        });
    });

    describe("isBlackListed and Pausable test", function() {
        it("transfer should be failed while paused", async () => {
            await instance.mint(10000);
            await expect(instance.pause()).to.be.emit(
                instance,"Pause"
            );
            await expect(instance.transfer(user1.address,100)).to.be.reverted;
            await instance.unpause();
            await instance.transfer(user1.address,100);
        });

        it("transfer should be failed while in blacklist", async () => {
            await instance.mintTo(user1.address,100);
            await instance.addBlackList(user1.address);
            await
expect(instance.connect(user1).transfer(user2.address,20)).to.be.reverted;

            await instance.removeBlackList(user1.address);
            await instance.connect(user1).transfer(user2.address,20);
            expect(await instance.balanceOf(user1.address)).to.be.equal(80);
            expect(await instance.balanceOf(user2.address)).to.be.equal(20);
        });

        it("destroyBlackFunds should change state", async () => {
            await instance.transfer(user1.address,100);
            expect(await instance.totalSupply()).to.be.equal(init_supply);
            expect(await instance.balanceOf(user1.address)).to.be.equal(100);
            await instance.addBlackList(user1.address);
            await instance.destroyBlackFunds(user1.address);
            expect(await
instance.totalSupply()).to.be.equal(init_supply.sub(100));
            expect(await instance.balanceOf(user1.address)).to.be.equal(0);

        });
    });

});
```

Output:

```
    PanaromaToken
    Meta test
      ✓ name | symbol | decimals test
    Init status test
      ✓ init_supply test
    approve and allowance test
      ✓ approve should change allowance
    transferFrom and transfer test
      ✓ transfer should change balance
      ✓ transfer to self shouldn't change balance
      ✓ transfer should failed while sender has insufficient tokens
      ✓ transferFrom without approval should be failed
      ✓ transferFrom should change balance and allowance (51ms)
    minter and burn
      ✓ mint or burn without owner should be failed
      ✓ mintTo should change state and emit event
      ✓ mint should change state and emit event  (42ms)
      ✓ mint beyond max_supply should be failed
      ✓ BurnFrom should change state and emit event (82ms)
    isBlackListed and Pausable test
      ✓ transfer should be failed while paused (50ms)
      ✓ transfer should be failed while in blacklist (66ms)
      ✓ destroyBlackFunds should change state (54ms)


  16 passing (2s)
```

10.2 External Functional Checkpoints

(Empty elements in the table represent things that are not required or relevant)

File:PanaromaToken.sol

contract: PanaromaToken is Pausable, StandardToken, BlackList

Index	Function	Visibility	Re-entrancy Check	Permission Check	Unit Test	Notes
1	unpause()	public		onlyOwner	Passed	whenPaused
2	transfer(address,uint)	public			Passed	whenNotPaused
3	transferFrom(address,address,uint)	public			Passed	whenNotPaused
4	balanceOf(address)	public			Passed	View
5	approve(address,uint)	public			Passed	
6	allowance(address,address)	public			Passed	View
7	deprecate(address)	public		onlyOwner	Passed	
8	totalSupply()	public			Passed	View
9	setParams(uint,uint)	public		onlyOwner		
10	addFounders(address)	public		onlyOwner		redundant
11	addInvestors(address)	public		onlyOwner		redundant
12	addTeamMember(address)	public		onlyOwner		redundant
13	addAdvisors(address)	public		onlyOwner		redundant
14	getFounders()	public				View
15	getInvestors()	public				View
16	getTeam()	public				View
17	getAdvisors()	public				View
18	mintTo(address,uint256)	public		onlyOwner	Passed	
19	mint(uint256)	public		onlyOwner	Passed	
20	burnFrom(address,uint256)	public		onlyOwner	Passed	
21	transferOwnership(address)	public		onlyOwner		
22	pause()	public		onlyOwner	Passed	whenNotPaused

FAIRYPROOF

https://medium.com/@FairyproofT

https://twitter.com/FairyproofT

https://www.linkedin.com/company/fairyproof-tech

https://t.me/Fairyproof_tech

Reddit: https://www.reddit.com/user/FairyproofTech