shieldify

Flare

SECURITY REVIEW

Date: 20 January 2025

CONTENTS

1. About Shieldify

Positioned as the first hybrid Web3 Security company, Shieldify shakes things up with a unique subscription-based auditing model that entitles the customer to unlimited audits within its duration, as well as top-notch service quality thanks to a disruptive 6-layered security approach. The company works with very well-established researchers in the space and has secured multiple millions in TVL across protocols, also can audit codebases written in Solidity, Rust, Go, Vyper, Move and Cairo.

Learn more about us at shieldify.org.

2. Disclaimer

This security review does not guarantee bulletproof protection against a hack or exploit. Smart contracts are a novel technological feat with many known and unknown risks. The protocol, which this report is intended for, indemnifies Shieldify Security against any responsibility for any misbehavior, bugs, or exploits affecting the audited code during any part of the project's life cycle. It is also pivotal to acknowledge that modifications made to the audited code, including fixes for the issues described in this report, may introduce new problems and necessitate additional auditing.

3. About Flare

Flare is a hyper-deflationary token built on the TitanX protocol, incorporating a unique daily auction system and a capped supply distributed over 11 weeks. Following the initial distribution, the protocol enters a phase of full deflation. Flare dedicates 92% of system value to buy-and-burns and liquidity within TitanX and its future protocols.

Flare's Key Features:

- Deflationary Token: Controlled token distribution with rapid buy-burn mechanics.
- Daily Auctions: Structured to increase scarcity and value.
- 11-Week Full Distribution: The initial distribution phase ends in 11 weeks.

Auto X28 Minting: 76% of the ETH and TitanX used for Minting or Daily Auction will mint X28.

4. Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4.1 Impact

- **High** – results in a significant risk for the protocol's overall well-being. Affects all or most users
- **Medium** – results in a non-critical risk for the protocol affects all or only a subset of users, but is still unacceptable
- **Low** – losses will be limited but bearable – and covers vectors similar to griefing attacks that can be easily repaired

4.2 Likelihood

- **High** – almost certain to happen and highly lucrative for execution by malicious actors
- **Medium** – still relatively likely, although only conditionally possible
- **Low** – requires a unique set of circumstances and poses non-lucrative cost-of-execution to rewards ratio for the actor

5. Security Review Summary

The security review lasted 7 days with a total of 224 hours dedicated by 4 researchers from the Shieldify team.

Overall, the code is super well-written. The audit report identified three Low-severity issues related to misconfigured addresses, missing timestamp validation, and unrestricted function calls in Flare contracts, along with other informational recommendations about constants setting.

The Flare dev team has been very responsive to the Shieldify research team's inquiries, demonstrating a strong commitment and dedication to the protocol's development. Their proactive security approach is evident, as they have conducted multiple security reviews with various companies, showcasing their thoroughness and dedication to securing their platform.

5.1 Protocol Summary

Project Name	**Flare**
Repository	flare-contracts
Type of Project	DeFi, Buy and Burn, Auction, Mint
Audit Timeline	7 days
Review Commit Hash	a5a481abba4100b72b5224c411495b3974e5c469
Fixes Review Commit Hash	948338f4d8bcac3e1a8a72daf2ee8f0e889cf23d

5.2 Scope

The following smart contracts were in the scope of the security review:

File	nSLOC
src/BuyNBurn/BaseBuyNBurn.sol	143
src/const/Constants.sol	42
src/core/FlareAuctionBuy.sol	185
src/core/Flare.sol	84
src/core/FlareAuction.sol	181
src/core/FlareBuyNBurn.sol	71
src/core/FlareMinting.sol	236
src/libs/OracleLibrary.sol	49

src/libs/PoolAddress.sol	30
src/interfaces/IWETH.sol	3
src/interfaces/IX28.sol	4
Total	**1028**

6. Findings Summary

The following number of issues have been identified, sorted by their severity:

- **Low** issues: **3**
- **Informational** issues: **1**

ID	Title	Severity	Status
[L-01]	TitanXInfBnB In `FlareMinting.sol` Is Set To Liquidity Pool Address Instead Of BuyNBurn Address In The Scripts	Low	Fixed
[L-02]	`Block.timestamp` Is Not Checked In `FlareAuction.sol` and `FlareAuctionBuy.sol`	Low	Fixed
[L-03]	The `SetFlareAuctionTreasury()` Function In `FlareMinting.sol` Can Be Called More Than Once	Low	Fixed
[I-01]	Reminder To Change Constants To Desired Values	Informational	Acknowledged

7. Findings

[L-01] TitanXInfBnB In `FlareMinting.sol` Is Set To Liquidity Pool Address Instead Of BuyNBurn Address In The Scripts

Severity

Low Risk

Description

In `FlareMinting.sol`, whenever a user calls `mint()` or `mintETH()`, `_distributeTitanX()` is called which will send some titanX to `mintingState.titanXInfBnB` after four weeks.

The docs mention:

> After the 4 weeks are over, that 8% is sent to the Inferno Buy And Burn V2 forever.

```
function _distributeTitanX(uint256 _amount) internal returns (uint256) {
    if (addedLiquidity) {
        if (block.timestamp <= startTimestamp + FOUR_WEEKS) {
            mintingState.titanX.transfer(FlARE_LP, wmul(_amount,
               TO_FLARE_LP));
        } else {
            mintingState.titanX.transfer(mintingState.titanXInfBnB, wmul(
               _amount, TO_INFERNO_BNB));
        }

        mintingState.titanX.transfer(FLARE_LP_WEBBING, wmul(_amount,
           TO_FLARE_LP));
    }
    return mintingState.titanX.balanceOf(address(this));
 }
```

In DeployConfig.s.sol, `titanXInfBnB` is set as the `_titanXInfernoPool` instead, which is the TitanX and Bnb UniswapV3 liquidity pool:

```
function config() public pure returns (Config memory _config) {
_titanXInfernoPool: 0x1E90B67149e688DfB95fD73Acacd8ADefd16d88D,
```

Impact

Funds will be sent to the wrong contract.

Recommendation

Set `_titanXInfernoPool` as the Buy And Burn contract instead.

Team Response

Fixed.

[L-02] `Block.timestamp` Is Not Checked In `FlareAuction.sol` and `FlareAuctionBuy.sol`

Severity

Low Risk

Description

In `FlareAuction.sol` and `FlareAuctionBuy.sol`, `_startTimestamp` is not checked when the constructor is called, which means that the Auction can start at any time, including a time in the past.

Location of Affected Code

File: FlareAuction.sol
File: FlareAuctionBuy.sol

```
constructor(AuctionState memory _state, SwapActionParams memory _s,
    uint32 _startTimestamp)
     notAddress0(_state.titanX)
     notAddress0(address(_state.flare))
     notAddress0(address(_state.flareMinting))
     notAddress0(address(_state.flareBnB))
     notAddress0(_state.titanXInfBnB)
     notAddress0(_state.X28)
     notAddress0(_state.WETH)
     notAddress0(_state.v3Router)
     SwapActions(_s)
{
     state = _state;

@>   startTimestamp = _startTimestamp;
}
```

Recommendation

If not intended, recommend setting a `block.timestamp` check like how it is done in the other contracts as well, such as `FlareMinting.sol`:

```
constructor(MintingState memory _mintingState, uint32 _startTimestamp,
    SwapActionParams memory _s)
require((_startTimestamp % Time.SECONDS_PER_DAY) == Time.TURN_OVER_TIME,
    "_startTimestamp must be 2PM UTC");
```

Team Response

Fixed.

[L-03] The `SetFlareAuctionTreasury()` Function In `FlareMinting.sol` Can Be Called More Than Once

Severity

Low Risk

Description

In the `FlareMinting.sol` contract, the `setFlareAuctionTreasury()` function is called by the owner to set the flare auction treasury address and mint 1 trillion flares to that address. The issue is that the owner can call this function again to mint another trillion flare.

Location of Affected Code

File: src/core/FlareMinting.sol#L204

```
function setFlareAuctionTreasury(address _FlareAuctionTreasury)
    external
    onlyOwner
    notAddress0(_FlareAuctionTreasury)
{
    FlareAuctionTreasury = _FlareAuctionTreasury;
    mintingState.flare.mint(_FlareAuctionTreasury,
       INITIAL_FLARE_FOR_AUCTION);
}
```

Impact

If `setFlareAuctionTreasury()` is called again, the extra minted tokens will cause a price discrepancy.

Recommendation

It is recommended to allow the owner to call it once, like how it is done in `createAndFundLP()` by checking a variable and setting that variable to a true value.

```
 function createAndFundLP(uint32 _deadline, uint256 _amountFlareMin,
    uint256 _amountX28Min)
    external
    onlyOwner
    notExpired(_deadline)
    notAmount0(_amountFlareMin)
    notAmount0(_amountX28Min)
{
    uint256 deadline = _deadline;
    if (mintingState.X28.balanceOf(address(this)) < INITIAL_X28_FLARE_LP
       + 1) {
        revert NotEnoughBalanceForLp();
    }

@> if (addedLiquidity) revert LiquidityAlreadyAdded();
@> addedLiquidity = true;

   // code
}
```

Team Response

Fixed.

[I-01] Reminder To Change Constants To Desired Values

Severity

Informational

Description

In the current code in `Constants.sol`, some addresses are set to the zero address, but these addresses are used in other parts of the code. The code is just a placeholder right now, ensure that the addresses are changed and tested prior to deployment.

For Genesis addresses, ensure they are capable of receiving ether as well.

Location of Affected Code

File: src/const/Constants.sol

```
address constant DEAD_ADDR = 0x000000000000000000000000000000000000dEaD;
address constant GENESIS = 0x000000000000000000000000000000000000dEaD; //
    @todo -> TBD
address constant GENESIS_TWO = 0x000000000000000000000000000000000000dEaD
    ; //@todo -> TBD
address constant OWNER = 0x000000000000000000000000000000000000dEaD; //
    @todo -> TBD
address constant FlARE_LP = 0x000000000000000000000000000000000000dEaD;
    //@todo -> TBD
address constant FLARE_LP_WEBBING = 0
    x000000000000000000000000000000000000dEaD; //@todo -> TBD
```

Recommendation

Change the constants to desired values.

Team Response

Acknowledged.

shieldify

Thank you!