Usual V1 Protocol Audit Report - Sherlock

Bhanu Sai Enamala

November 30, 2024

Prepared by: Bhanu Sai Enamala

Lead Auditors: Bhanu Sai Enamala

Table of Contents

- Table of Contents
- Protocol Summary
- Disclaimer
- Risk Classification
- Audit Details
  - Scope
  - Issues found
- Findings
  - Low
    - [L-1] Vulnerability to Frontrunning in updateWithdrawFee Allows Exploitation of Withdraw Fee
    - [L-2] Redundant Claim Tracking Mechanism for Full Amount Merkle Proofs in Off-Chain Distribution
    - [L-3] Unauthorized Claim Execution by External Parties Due to Lack of Sender Verification in AirdropDistribution.sol contract.

Protocol Summary

Usual is a decentralized stablecoin issuer launched 3 months ago, now ranked among the top 15 with over $350M in TVL and 20k holders. The V1 release, which is the focus of this audit, includes the introduction of $USUAL, the governance token for the protocol, along with its allocation, staking and distribution logic, and the related airdrop contracts.

Link to the Usual V1 Protocol Audit Contest: https://audits.sherlock.xyz/contests/575?filter=questions

Disclaimer

The Bhanu Sai Enamala team makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the team is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

Risk Classification

		Impact		
		High	Medium	Low
	High	H	H/M	M
Likelihood	Medium	H/M	M	M/L
	Low	M	M/L	L

I use the CodeHawks severity matrix to determine severity. See the documentation for more details.

Audit Details

Scope

https://audits.sherlock.xyz/contests/575?filter=scope

nSloc = 3293

Issues found

Severity	Number of issues found
High	0
Medium	0
Low	3
Info	0
Gas	0
Total	0

Findings

Low

[L-1] Vulnerability to Frontrunning in updateWithdrawFee Allows Exploitation of Withdraw Fee

{#l-1}

Description: The updateWithdrawFee function in the UsualX contract is susceptible to frontrunning, enabling attackers to exploit fee changes before they are fully applied across the contract. Authorized roles can update the withdrawal fee, but since there is no delay or mechanism to prevent simultaneous transactions from exploiting the fee change, a malicious user could initiate a withdrawal to avoid a higher fee or benefit from a lower fee in anticipation of the update. This vulnerability could lead to financial losses for the protocol as attackers strategically time their transactions to evade fees or gain favorable terms, while regular users may face unfair disadvantages. Introducing a delay on fee updates or implementing a system to enforce consistent fees across transactions could mitigate this risk.In UsualX.sol:updateWithdrawFee, the choice to update the withdrawal fee without an anti-frontrunning mechanism is a mistake, as it allows attackers to anticipate fee changes and perform transactions before the new fee is enforced.

Impact: The protocol suffers a potential loss in anticipated withdrawal fees due to frontrunning, leading to reduced revenue from withdrawals. [The attacker avoids the increased fee, gaining the difference between the old and new fee amounts.]

Proof of Concept:

Initial Pre condition : WITHDRAW_FEE_UPDATER_ROLE calls updateWithdrawFee(uint256 newWithdrawFeeBps) on UsualX contract with newWithdrawFeeBps greater than the already existing withdrawFee on the contract.

Attack Path :

1. Attacker observes the pending fee update transaction in the mempool and identifies the new fee rate.
2. Attacker frontruns the fee change by initiating a withdrawal transaction with the current, lower fee before the update is processed.

[L-2] Redundant Claim Tracking Mechanism for Full Amount Merkle Proofs in Off-Chain Distribution

{#l-2}

Description: The claimOffChainDistribution function in the DistributionModule.sol contract (link to code) includes a mechanism to track cumulative amounts claimed by each user through the claimedByOffChainClaimer mapping. However,

this tracking is redundant due to the way claims are designed. According to the README documentation and the function implementation, the Merkle proof is intended to validate claims for the full assigned amount only, disallowing partial claims. Given that the proof is valid solely for the total allocation, there is no scenario in which users can make incremental claims. This makes the claimedByOffChainClaimer tracking mechanism unnecessary, as it does not align with the intended functionality and does not impact claim behavior.

If partial claims are strictly disallowed by design, maintaining a cumulative record of previously claimed amounts introduces redundant code and potential confusion, as it implies support for partial claims that are not possible within the current framework. To improve clarity and efficiency, the claimedByOffChainClaimer mapping could be removed or adjusted to reflect the design's one-time, full-amount claim process.

This issue, while not directly affecting user funds or functionality, adds unnecessary complexity to the claim logic and could lead to confusion among developers or auditors reviewing the code.

Root Cause: The choice to implement cumulative tracking of claimed amounts in DistributionModule.sol:claimOffChainDistribution::386 is a mistake as it does not align with the design, which only supports full, one-time claims based on a valid Merkle proof. This redundancy could mislead developers and introduces unnecessary code complexity.

Impact: In this vulnerability path, the protocol suffers unnecessary storage and gas costs due to redundant tracking of claimed amounts for full claims. The user cannot make partial claims, rendering the cumulative tracking irrelevant.

Proof of Concept:

Initial Pre-Conditions:

1. User calls claimOffChainDistribution() with a Merkle proof to set claimedByOffChainClaimer[account] for an exact amount value representing the full eligible amount according to the Merkle root.

Attack Path:

1. Contract design tracks claimedByOffChainClaimer[account] incrementally even though partial claims are not permitted, resulting in unnecessary updates.
2. Merkle proof requirement enforces that only the total eligible amount can be claimed at once, as specified in the README, ensuring no valid scenario for partial claims.
3. No mechanism exists to allow or recognize partial claims, making the tracking of claimed amounts redundant since the proof validation inherently prevents partial claims.
4. Implementation includes the cumulative tracking of claimed amounts in claimedByOffChainClaimer even though full claims only are valid, leading

to unnecessary storage updates that do not affect functionality.

[L-3] Unauthorized Claim Execution by External Parties Due to Lack of Sender Verification in AirdropDistribution.sol contract.

{#l-3}

Description: Lack of verification of the sender's address in the AirdropDistribution::claim(address account, bool isTop80, uint256 amount, bytes32[] calldata proof) function will cause an unauthorized claim execution for the intended claimant as any external actor with access to the Merkle proof and account information can call claim on behalf of the claimant without consent, potentially leading to unintended fund distribution.

Root Cause: The design choice to use the same Merkle root for the entire six-month's claim is a mistake as it exposes the claim function to unauthorized access. Specifically, it allows any actor with initial access to the Merkle proof (since it remains constant for the full claimable amount over multiple months, they can acces it from on chain data once the claimant uses the claim function first time) to repeatedly execute unauthorized claims. This static approach to the Merkle root fails to limit access to valid claimants over time, enabling potential exploitation where malicious actors could claim on behalf of other accounts, even if those accounts do not intend to claim. Consequently, tokens would be minted for those accounts without their consent, leading to an unintentional distribution of funds.

Impact: The users may suffer unexpected consequences if an unauthorized claim is made on their behalf. Potential impacts include:

- Tax Liabilities: Users may incur unwanted tax obligations if claims are processed prematurely, affecting their financial planning.
- Privacy Exposure: Users may prefer to avoid immediate on-chain visibility, and unauthorized claims compromise their privacy.
- Increased Security Risks: Premature claims could draw attention to their wallets, increasing potential security vulnerabilities.

Proof of Concept:

Initial Pre Conditions

-After the end of FIRST_AIRDROP_VESTING_CLAIMING_DATE , the claimant calls claim(address account, bool isTop80, uint256 amount, bytes32[] calldata proof) with the necessary arguments. -After the end of SECOND_AIRDROP_VESTING_CLAIMING_DATE the claimant doesnt not call the claim function as he does not want to claim at that point.

Attack Path

-Malicious actor sees the on chain data of the first claim transaction and uses the same arguments used by claimant during first claim and calls claim(address ac-

count, bool isTop80, uint256 amount, bytes32[] calldata proof) with the claimant's data as arguments after SECOND_AIRDROP_VESTING_CLAIMING_DATE

-Then the ususal token will be minted for the claimant even though he does not want to claim at that point.