Agents.Fun
Security Review

Cantina Managed review by:
MiloTruck, Lead Security Researcher
0x4non, Associate Security Researcher

December 22, 2024

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Valory is a research and deployment company at the intersection of crypto and AI. Specifically, the premier creator of open-source frameworks for co-owned AI. Valory's mission is to enable communities, organizations, and countries to co-own AI systems, beginning with decentralized autonomous agents. Valory is the VC-backed team of engineers, researchers, and commercial executors that co-founded the Olas DAO, contribute to the Olas Protocol and Olas Stack, and built the first agents using it.

From Dec 17th to Dec 19th the Cantina team conducted a review of meme-ooorr on commit hash 3fa080c8 along with the following on-chain contracts:

- Base: `0x82A9c823332518c32a0c0eDC050Ef00934Cf04D4`
- Celo: `0xEea5F1e202dc43607273d54101fF8b58FB008A99`

The team identified a total of **5** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 0
- Medium Risk: 0
- Low Risk: 3
- Gas Optimizations: 0
- Informational: 2

3 Findings

3.1 Low Risk

3.1.1 Uniswap V3 pool cardinality should not be hardcoded to 60 in MemeFactory._createUniswapPair()

Severity: Medium Risk

Context: MemeFactory.sol#L229

Description: In MemeFactory._createUniswapPair(), whenever a new Uniswap V3 pool is created, increaseObservationCardinalityNext() is called with a hardcoded value of 60:

```
// Increase observation cardinality
IUniswapV3(pool).increaseObservationCardinalityNext(60);
```

However, this leads to two issues:

1. A cardinality of 60 is too small.
2. Cardinality should not be hardcoded to the same value across different chains.

Calling increaseObservationCardinalityNext() increases the cardinality of the pool, which is the maximum number of observations that can be stored. Considering that an observation is stored at most once per block, the cardinality of the pool must be large enough to cover the desired TWAP window when observe() is called (ie. should be at least twapWindowSeconds / secondsPerBlock).

Regarding (1), Base produces a block every 2 seconds. Assuming an observation is stored every block, setting cardinality = 60 limits the oldest observation to 2 * 60 = 120 seconds ago, which is extremely short and susceptible to manipulation.

Regarding (2), the cardinality should be set depending on the block time of each chain. Since BuyBackBurner uses an observation window of 1800 seconds (BuyBackBurner.sol#L57-L58), the minimum cardinality for Base and Celo should be:

- Base - 1800 / 2 = 900.
- Celo - 1800 / 5 = 360.

Recommendation: Consider calling increaseObservationCardinalityNext() with a virtual _observationCardinalityNext() function that is overridden in MemeBase and MemeCelo to return the appropriate values:

```
  // Increase observation cardinality
- IUniswapV3(pool).increaseObservationCardinalityNext(60);
+ IUniswapV3(pool).increaseObservationCardinalityNext(_observationCardinalityNext());
```

Valory: Fixed in PR 110. Since calling increaseObservationCardinalityNext() with a large cardinality costs a huge amount of gas (eg. cardinality = 900 for Base as recommended above would exceed the block gas limit), we have decided to go with the following values for _observationCardinalityNext():

- Arbitrum: 240.
- Base: 120.
- All other chains: 60.

Cantina Managed: Verified. The recommendation above has been implemented with smaller cardinality values to maintain reasonable gas costs. This is an acceptable risk as it still leaves a minimum observation window of 240 seconds and the TWAP oracle would be non-trivial to manipulate.

Additionally, the currently deployed contracts on Base and Celo, which have their cardinality set to 60, have a minimum observation window of 120 seconds which is considered acceptable.

As a last resort, should the team decide that the current cardinality values are too small in the future, they can directly call increaseObservationCardinalityNext() of the respective Uniswap V3 pool.

3.1.2 Duplicate address accounts can lead to overwritten contributions

Severity: Low Risk

Context: MemeBase.sol#L60

Description: On deployment the constructor of `MemeBase` will receive an array of accounts to set the contribution of each account. The `_launchCampaignSetup` function currently sets the contribution amount of each address in `memeHearters` by assigning the value directly. If the accounts array contains the same address more than once, the value assigned last will overwrite the previous one, effectively ignoring earlier entries. This can lead to inaccurate accounting of contributions, causing the final totalAmount calculation to potentially mismatch expected totals.

Proof of Concept: Consider the following input arrays:

```
accounts = [0x1234..., 0xABC..., 0x1234...]
amounts  = [100,        200,     50]
```

During iteration:

1. For `0x1234...` (first occurrence), `memeHearters[launchCampaignNonce][0x1234...]` is set to 100.
2. For `0xABC...`, `memeHearters[launchCampaignNonce][0xABC...]` is set to 200.
3. When `0x1234...` appears again, `memeHearters[launchCampaignNonce][0x1234...]` is overwritten and set to 50, losing the original contribution of 100.

This results in incorrect data storage, as the final accounting will only consider the last occurrence of a given address rather than the sum of its contributions.

Recommendation: Accumulate Contributions: Instead of directly assigning the amount, sum the contributions for repeated addresses:

```
@@ -57,7 +57,7 @@ contract MemeBase is MemeFactory {
  uint256 totalAmount;
  for (uint256 i = 0; i < accounts.length; ++i) {
      totalAmount += amounts[i];
-     memeHearters[launchCampaignNonce][accounts[i]] = amounts[i];
+     memeHearters[launchCampaignNonce][accounts[i]] += amounts[i];
      // to match original hearter events
      emit Hearted(accounts[i], launchCampaignNonce, amounts[i]);
  }
```

Alternative you can enforce uniqueness, ensure that the accounts array contains no duplicates by passing it as a sorted array and checking during iteration:

```
for (uint256 i = 0; i < accounts.length; ++i) {
  require(i == 0 || accounts[i] > accounts[i-1], "Duplicate or unsorted accounts array detected");
  totalAmount += amounts[i];
  memeHearters[launchCampaignNonce][accounts[i]] = amounts[i];
  // to match original hearter events
  emit Hearted(accounts[i], launchCampaignNonce, amounts[i]);
}
```

Valory: Fixed in PR 110.

Cantina Managed: Verified, contributions are now added for repeated addresses as recommended above.

3.1.3 Potential front-running grief attack on `unleashThisMeme()`

Severity: Low Risk

Context: MemeFactory.sol#L194-L195

Description: When a new meme token is unleashed, the contract ensures that no pre-existing Uniswap V3 pool with the given token pair exists, expecting the returned pool address to be zero before creation. If an attacker anticipates the token address calculating the random salt used for createing the token and frontruns the `unleashThisMeme()` transaction by creating the Uniswap pool beforehand, it can cause the `unleashThisMeme()` call to fail. This could result in a short-term denial-of-service (DoS) scenario.

However, this issue is more theoretical in nature, particularly in L2 environments where frontrunning is significantly more difficult or non-existent due to the use of sequencers rather than public mempools. Additionally, the contract's logic for token creation and salt determination ensures a new token address configuration each attempt, preventing a permanent blockage.

Recommendation: Since this scenario primarily affects blockchains with a public mempool and the attacker gains no direct economic benefit, the impact is minimal, and can be avoided using a private mempool. However, for improved resilience:

- Document the theoretical DoS vector to inform users.
- Consider using a more sophisticated address derivation process or introducing additional entropy for token address generation, further reducing the predictability of the new meme token address.

Valory: Acknowledged.

Cantina Managed: Acknowledged. This is an acceptable risk for the currently deployed contracts on Base and Celo as:

1. The DOS is not permanent as the meme token address changes every block.
2. Front-running is not possible on most L2s, since they do not have a public mempool.

3.2 Informational

3.2.1 Minor improvements to code and comments

Severity: Informational

Context: MemeFactory.sol#L8, MemeFactory.sol#L119, MemeFactory.sol#L186, MemeFactory.sol#L223, MemeFactory.sol#L262, MemeFactory.sol#L315-L319, MemeFactory.sol#L441, MemeFactory.sol#L584

Description:

1. MemeFactory.sol#L8-L28 -- The `IBuyBackBurner` and `IERC20` interfaces should be in their own separate files.
2. MemeFactory.sol#L118-L119 -- The `olas` variable is not used in the in-scope code and can be removed.
3. MemeFactory.sol#L185-L186 -- The variable name `priceX96` is misleading as price is scaled by `1e18` and `not 2 ** 96`. Consider renaming it to `price`. Additionally, consider using `FixedPointMathLib.divWadDown()`:

```
  // Calculate the price ratio (amount1 / amount0) scaled by 1e18 to avoid floating point issues
- uint256 priceX96 = (amount1 * 1e18) / amount0;
+ uint256 price = FixedPointMathLib.divWadDown(amount1, amount0);
```

4. MemeFactory.sol#L223 -- This line can be simplified:

```
- uint256 nativeLeftovers = isNativeFirst ? (nativeTokenAmount - amount0) : (nativeTokenAmount - amount1);
+ uint256 nativeLeftovers = nativeTokenAmount - (isNativeFirst ? amount0 : amount1);
```

5. MemeFactory.sol#L253-L262 -- This part of the code can be simplified to:

```
(uint256 nativeAmountForOLASBurn, uint256 memeAmountToBurn) = isNativeFirst ? (amount0, amount1) : (amount1,
↪  amount0);
```

6. MemeFactory.sol#L315-L322 -- The code here can be simplified to:

```
memeToken = address(new Meme{salt: randomNonce}(name, symbol, DECIMALS, totalSupply));
```

7. MemeFactory.sol#L438-L441 -- This part of the code can be simplified:

```
  // Update meme token map values
- uint256 totalNativeTokenCommitted = memeSummon.nativeTokenContributed;
- totalNativeTokenCommitted += msg.value;
- memeSummon.nativeTokenContributed = totalNativeTokenCommitted;
+ memeSummon.nativeTokenContributed += msg.value;
```

8. MemeFactory.sol#L584 -- `collectFees()` should be declared as `external` instead of `public`.

Valory: Fixed in PR 110.

Cantina Managed: Verified. Issue 1 has been acknowledged, the remaining issues have been fixed.

3.2.2 `numTokens` state variable can be replaced with a getter function

Severity: Informational

Context: MemeFactory.sol#L127-L128, MemeFactory.sol#L365-L367

Description: In the `MemeFactory` contract, the `numTokens` state variable always set to the length of the `memeTokens` array:

```
// Number of meme tokens
uint256 public numTokens;
```

```
// Push token into the global list of tokens
memeTokens.push(memeToken);
numTokens = memeTokens.length;
```

As such, the `numTokens` variable is redundant and wastes gas.

Recommendation: Consider removing the `numTokens` state variable and implementing a getter instead, for example:

```
function numTokens() external view returns (uint256) {
    return memeTokens.length;
}
```

Valory: Fixed in PR 110.

Cantina Managed: Verified, the recommendation was implemented.