Tipcoin Security Assessment

Oct 22, 2023

Prepared for

Tipcoin

Prepared by:

0xfoobar

All published reports visible at:

https://github.com/0xfoobar/audits

Table of Contents

Summary

Overview

Tipcoin is a project for onchain tipping linked to social media accounts.

Project Scope

We reviewed the core smart contracts and test suite contained within commit hash `bf5228e` on branch `develop` on a private repo. Fixes were reviewed at commit hash `3d71ae4` at https://github.com/viralitysystems/tip-pool.

Severity Classification

High - Assets can be stolen/lost/compromised directly (or indirectly if there is a valid attack path that does not have hand-wavy hypotheticals).
Medium - Assets not at direct risk, but the function of the protocol or its availability could be impacted, or leak value with a hypothetical attack path with stated assumptions, but external requirements.
Low - Assets are not at risk: state handling, function incorrect as to spec, issues with comments.
Informational - Code style, clarity, syntax, versioning, off-chain monitoring (events, etc)

Summary of Findings

Severity	Findings	Resolved
High	**1**	**1**
Medium	**0**	**0**
Low	**3**	**3**
Informational	**7**	**6**

Disclaimer

This security assessment should not be used as investment advice.

We do not provide any guarantees on eliminating all possible security issues. foostudio recommends proceeding with several other independent audits and a public bug bounty program to ensure smart contract security.

We did not assess the following areas that were outside the scope of this engagement:

- Website frontend components
- Offchain order management infrastructure
- Multisig or EOA private key custody
- Metadata generation

Key Findings and Recommendations

1. ECDSA.recover() returns unchecked address(0) on failure

Severity: **High**
Files: pool.sol

Description

ETH withdrawals from the contract are gated by an offchain signature from an admin `withdrawalSigner` address. However the signature validity check lacks a critical check on error which could lead to an attacker draining the contract. As stated in OpenZeppelin docs for ECDSA.recover(), **“This call *does not revert* if the signature is invalid, or if the signer is otherwise unable to be retrieved. In those scenarios, the zero address is returned.”** This means that if `withdrawalSigner` is left uninitialized to address(0), any signature from any private key will return address(0) in recover and be treated as valid in `_verifyWithdrawal()`. Even though there are further checks in `withdrawETH()`, such as checking the `withdrawalsEnabled` boolean toggle, it’s feasible that an admin could accidentally toggle withdrawals enabled before initializing the proper withdrawalSigner address, and safeguards in the core signature verification function would prevent this in any case.

Impact

Loss of all funds.

Recommendation

`_verifyWithdrawal()` should explicitly revert if ECDSA.recover() returns address(0). This can be done by adding a line to the end of the function:

require(recoveredAddress != address(0), “invalid signature”);

Response

Fixed.

2. ECDSA variable names do not match code logic

Severity: **Low**
Files: pool.sol

Description

ECDSA signatures follow three steps, as described in https://solidity-by-example.org/signature/:

1. Recreate hash from the original message
2. Recover signer from signature and hash
3. Compare recovered signer to claimed signer

The signature verification logic follows these properly, however shifts the names of the outputs off by one which makes the intention harder to follow and could break external integrations.

Impact

Potential broken integrations.

Recommendation

Rename the result of `keccak256(abi.encodePacked(index, expiresAt, msg.sender, amount))` from `dataHash` to `messageHash`.
Rename the result of `ECDSA.toEthSignedMessageHash(dataHash)` from `messageHash` to `ethSignedMessageHash`.

Response

Fixed.

3. Smart contract wallets cannot use the protocol

Severity: **Low**
Files: pool.sol

Description

`depositETH()` and `receive()` functions have a `require(msg.sender == tx.origin)` limitation. This is not necessary and breaks useful integrations such as Gnosis Safe wallets or account abstraction wallets, while simultaneously not preventing botting or MEV.

Impact

Smart contract wallets cannot use the protocol.

Recommendation

Remove these limitations or clarify which behavior you intend to prevent.

Response

Fixed.

4. Use of transfer to send ETH can cause reverts

Severity: **Low**
Files: pool.sol

Description

The protocol employs Solidity's .transfer method to send ETH. However, .transfer is limited to a hardcoded gas amount of 2300, which can break if contracts have fallback logic. This is also potentially incompatible with future changes to Ethereum's gas costs.

Impact

Sending ETH to contracts with fallback logic will revert.

Recommendation

Use the .call() method as described in https://solidity-by-example.org/sending-ether/.

Response

Fixed.

5. Use latest compiler version and associated features

Severity: **Informational**
Files: pool.sol

Description

All files currently have the solidity pragma at 0.8.6. Best practice is to target the latest 0.8.21 version. This will have three benefits: cheaper deployment costs, gas optimization and new language features. This protocol is targeting Ethereum, so there are no PUSH0 incompatibility issues. However if multichain expansion is a plan in the future, it'll be important to explicitly set the compiler's target EVM version as "paris" rather than "shanghai" to avoid PUSH0 opcode incompatibility.

Impact

Increased deployment size and gas costs.

Recommendation

Use a floating ^0.8.21 pragma everywhere, and target the paris EVM version with `evm_version = "paris"` in foundry.toml

Response

Fixed.

6. Unnecessary balance check in recoverToken

Severity: **Informational**
Files: pool.sol

Description

Each ERC20 checks that the transferrer has the necessary balance to make the transfer internally, and will revert if not. So it's not necessary to do external balance checks in recoverToken() before making the call. It may also be desirable to generalize this function to enable retrieval of ERC721 and ERC1155 as well.

Impact

Extra code size and added gas costs for the owner.

Recommendation

Remove the check on line 97 since it's duplicated internally in the ERC20.

Response

Fixed.

7. Unnecessary ReentrancyGuard nonReentrant modifiers

Severity: **Informational**
Files: pool.sol

Description

Re-entrancy guards are needed to protect against making external calls before internal state has been updated, that can be maliciously looped. Since both depositETH() and withdrawETH() follow the checks-effects-interaction pattern cleanly, reentrancy guards are unnecessary here. In the case of depositETH(), there are no external calls whatsoever. In the case of withdrawETH(), all state is updated prior to making the external ETH transfer and so there is no danger here.

Impact

Code bloat and increased gas costs for users.

Recommendation

Delete the `nonReentrant` modifiers and remove the ReentrancyGuard import.

Response

Fixed.

8. Use custom errors instead of require strings

Severity: **Informational**
Files: *.sol

Description

Custom errors are slimmer from a deployment codesize perspective, and also use less gas on reverts. They’re also better for grouping similar errors into canonical categories without the repetition of similar-but-distinct revert strings. Compiler versions will need to be ^0.8.4 or higher instead of ^0.8.0 for compatibility here.

Impact

Lower deployment and usage gas costs.

Recommendation

Update require statements to revert with custom errors.

Response

Not implemented.

9. Unused imports

Severity: **Informational**
Files: pool.sol

Description

Both `import "@openzeppelin/contracts/utils/cryptography/MerkleProof.sol";` and `import "hardhat/console.sol";` are unused, can be deleted.

Impact

Unnecessary code size.

Recommendation

Remove.

Response

Fixed.

10. Fix typos

Severity: **Informational**
Files: *.sol

Description

Replace “withdrawl” with “withdrawal” in all variable names
Use mixedCase for variable names as described here:
https://docs.soliditylang.org/en/latest/style-guide.html#naming-styles
Use CapitalizedWords for event names.

Impact

Cleaner code.

Recommendation

Update spelling.

Response

Fixed.

11. Replace private mappings + public getters with public mappings

Severity: **Informational**
Files: pool.sol

Description

Solidity automatically populates getter functions for public mappings. So it’s cleaner to replace private mappings that expose one-to-one getter functions with a public mapping. This can be done for `withdrawalIndex`, `userDeposits`, and `userWithdrawals`.

Impact

Code bloat.

Recommendation

Simplify these into public mappings.

Response

Fixed.