FOOSTUDIO

Wasabi Security Assessment

May 12, 2023

Prepared for

Wasabi

Prepared by:

0xfoobar

Table of Contents

Summary

Overview

Wasabi is an NFT options protocol enabling peer-to-peer trading. Users can deposit NFTs or fungible tokens to receive an ERC-721 representation of their American-style covered option.

Project Scope

We reviewed the core smart contracts and test suite contained within commit hash 203286ee7ed8024b4d2158cf7a4c9d2443a81454 at https://github.com/dev-wasabi/wasabi. The Wasabi team reported that this code corresponds to contracts currently deployed on Ethereum mainnet at a May 12 snapshot of the protocol documentation: https://archive.is/VeSEt.

Summary of Findings

Severity	Findings
High	**0**
Medium	**2**
Low	**2**
Informational	**4**

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

Key Findings and Recommendations

1. Protocol breaks with nonstandard ERC20 tokens, like USDT or BNB

Severity: **Medium**
Files: AbstractWasabiPool.sol, WasabiPoolFactory.sol, WasabiConduit.sol, WasabiOptionArbitrage.sol, ERC20WasabiPool.sol, ETHWasabiPool.sol

Description

ERC20.transferFrom() and ERC20.transfer() methods are used extensively throughout the codebase. A previous Zellic audit noted that the return values of these methods should be checked, and this has been fixed in the current codebase version. However it did not account for nonstandard tokens with nonstandard return signatures, like USDT and BNB. More details can be found

Impact

Nonstandard ERC20s will not work on Wasabi pools.

Recommendation

Use the SafeERC20 library from OpenZeppelin to wrap common nonstandard behavior in all ERC20 method calls. More details can be found in this GitHub readme.

2. ecrecover return value is not checked in signature library

Severity: **Medium**
Files: Signing.sol

Description

The solidity ecrecover precompile returns the zero address rather than throwing an error when the signature is invalid. While this is expected behavior, it should be immediately wrapped with error handling in Signing.sol::recoverSigner(), rather than propagating potentially invalid signer addresses upwards. This function is consumed by a variety of files: PriceConfigValidator.verify(), Signing.getSigner(), Signing.getAskSigner(), ConduitSignatureVerifier.getSignerForBid(), ConduitSignatureVerifier.getSignerForAsk(), PoolAskVerifier.getSignerForPoolAsk(), PoolAskVerifier.getSignerForPoolBid(). While some of these have an explicit zero address check for the signer variable, it is not clear how other methods are currently used or will plan to be used. This could cause a severe authentication error down the line if this codebase continues to be iterated on.

Impact

An inheriting method may fail to check for the zero address and authenticate a wide variety of signatures.

Recommendation

Throw an error in recoverSigner() if the zero address is returned from ecrecover.

3. Use of transfer to send ETH can cause reverts

Severity: **Low**
Files: WasabiOptionArbitrage.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/.

4. Use of safeMint() exposes potentially vulnerable callback patterns

Severity: **Low**
Files: WasabiOption.sol

Description

The mint() function within WasabiOption mints and makes an external call into the receiving contract (if minted to a contract rather than an EOA) before incrementing the mint counter or assigning ownership. This is a dangerous pattern, as callbacks can trigger recursive entries.

Impact

Recursive batch minting may revert or be overwritten, causing the user to receive fewer NFTs than they should. New protocol changes may introduce further malicious vulnerabilities if the checks-effects-interactions pattern is not used.

Recommendation

Move the external safeMint call to the end of the function. Additionally, consider whether safeMint is the right function to use here, as it adds gas costs and attack surface. A simpler mint() with no callbacks may be the right choice.

5. Assembly code can be replaced with the block.chainid opcode

Severity: **Informational**
Files: PricingConfigValidator.sol, ConduitSignatureVerifier.sol, PoolAskVerifier.sol, PoolBidVerifier.sol

Description

Multiple files re-implement the getChainId() internal function, which contains a Yul block to query the current chain ID for signature verification. However this is no longer necessary from solidity 0.8.0 onwards, as the block.chainid opcode is exposed everywhere. See https://blog.soliditylang.org/2020/12/16/solidity-v0.8.0-release-announcement/ for details.

Impact

Increased deployment size and gas costs from hopping to unnecessary internal functions.

Recommendation

Delete the getChainId() function in each of these files and replace all calls to getChainID() with the block.chainid opcode.

6. Remove unused lastFactory variable

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

Description

The lastFactory variable is never read from.

Impact

Increased deployment size and gas costs from unnecessary storage writes.

Recommendation

Delete its declaration and delete its setter within toggleFactory(). If factory deployment tracking is desired, emitting events is a better way to do this.

7. Remove unused verifyPoolBid and verifyPoolAsk functions

Severity: **Informational**
Files: PoolBidVerifier.sol, PoolAskVerifier.sol

Description

The verifyPoolBid() and verifyPoolAsk() functions are marked internal, yet never called within the codebase. They cannot be used for offchain mock verification, because they are internal.

Impact

Increased deployment size.

Recommendation

Delete these declarations.

8. The loanPremiumFraction variable can be made immutable

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

Description

Since this variable is only set once in the constructor, it can be embedded into the runtime bytecode rather than using expensive storage reads.

Impact

Lower gas costs.

Recommendation

Change loanPremiumFraction to be an immutable variable.