MalwareCare

Solidity Contract Security Audit

Overview

The task was to conduct a Smart Contract Manual Code Review & Security Analysis for a Solidity file. The objective of the assessment was to measure the security posture and identify and present any vulnerabilities discovered.

Target(s)

The scope of the test included the following in-scope information assets:

- Hodlv3.sol

Timetable

The following testing timetable is shown below:

- Test Start: 11/9/2021
- Test End: 11/9/2021

Project Scope

The scope of the project is a smart contract. I have scanned this smart contract for commonly known and more specific vulnerabilities, below are those considered (the full list includes but is not limited to):

- Reentrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Byte array vulnerabilities
- Style guide violation
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Unchecked external call - Unchecked math
- Unsafe type inference
- Implicit visibility level

Summary

I performed a manual audit, which was completed with MythX, Mythril, Slither and remix IDE. All issues found during analysis were reviewed, and important vulnerabilities are presented in the “Findings” section.

As a result of this test, four (4) vulnerabilities were identified. Three (3) of the findings were identified to be a low-level vulnerability, and one (1) was a best-practice recommendation.

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to asset loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions.
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to asset loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that can't have a significant impact on execution.
Lowest/Best Practice	Lowest-level vulnerabilities, code style violations, and info statements can't affect smart contract execution and can be ignored.

Audit Overview

The table below is designed to provide a view of all the identified findings and their respective risk rating. Please see the following section for a detailed listing of the identified findings.

#	Finding Title	Instances	Rating
1.	A floating pragma is set	4	Low
2.	Potential use of “block.number as a source of randomness	1	Low
3.	Use of “t.x origin” as a part of authorization control	1	Low
4.	State variable is not set	3	Best Practice

Finding(s)

1. Floating pragma is set Low

Description:

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Pragma statements can be allowed to float when a contract is intended for consumption by other developers.

Impact:

Locking the pragma helps to ensure that contracts do not accidentally get deployed using an outdated compiler version that might introduce bugs that affect the contract system negatively.

Finding Comments:

The current pragma Solidity directive is “>=0.6.8”. Consider known bugs for the compiler version(s) that are chosen.

- L: 3
- L: 960
- L: 1194
- L: 1262

Recommendations:

It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

2. Potential use of "block.number" as source of randomness Low

Description:

The ability to generate random numbers is very helpful in all kinds of applications. The environment variable "block.number" looks like it might be used as a source of randomness.

Impact:

Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks.

Finding Comments:

N/A

Recommendations:

Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

3. Use of "tx.origin" as a part of authorization control **Low**

Description:

“.tx.origin” is a global variable in Solidity which returns the address of the account that sent the transaction. Using the variable for authorization could make a contract vulnerable if an authorized account calls into a malicious contract.

Impact:

Note that using "tx.origin" as a security control might cause a situation where a user inadvertently authorizes a smart contract to perform an action on their behalf.

Finding Comments:

The tx.origin environment variable has been found to influence a control flow decision.

Recommendations:

It is recommended to use "msg.sender" instead.

4. State variable visibility is not set **Best Practice**

Description:

It is best practice to set the visibility of state variables explicitly. Other possible visibility settings are public and private.

Impact:

Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.

Finding Comments:

- Default visibility for "inSwapAndLiquify" is internal
- Default visibility for "isBlacklisted" is internal
- Default visibility for "userWalletAllowance" is internal

Recommendations:

Variables can be specified as being public, internal, or private. Explicitly define visibility for all state variables.

Conclusion

I was given a smart contract file and have used all the latest static and dynamic tools and manual observations to review everything in the given timeframe. Upon final review, I found 3 (three) low-level vulnerabilities and 1 (one) best-practice recommendation. Overall, the security state of the reviewed contract is ***"well-secured"***.

Disclaimer Notice

THIS DOCUMENT MAY CONTAIN CONFIDENTIAL INFORMATION ABOUT ITS SYSTEMS AND INTELLECTUAL PROPERTY OF THE CUSTOMER AS WELL AS INFORMATION ABOUT POTENTIAL VULNERABILITIES AND METHODS OF THEIR EXPLOITATION. THE REPORT CONTAINING CONFIDENTIAL INFORMATION CAN BE USED INTERNALLY BY THE CUSTOMER OR IT CAN BE DISCLOSED PUBLICLY AFTER ALL VULNERABILITIES ARE FIXED - UPON DECISION OF THE CUSTOMER.