HACKEN

SMART CONTRACT CODE SECONDARY REVIEW AND SECURITY ANALYSIS REPORT

Customer: Pantheon
Date: November 26, 2018
Platform: Ethereum
Language: Solidity

Document

Name	Smart Contract Code Secondary Review and Security Analysis Report for Pantheon
Platform	Ethereum / Solidity
File name	pantheon.sol
MD5 hash for first audit	8ce724e280aa42db342952ff5a067f6d
MD5 hash for secondary audit	f604b54a5bb46196d42c1b998971a414
Date of first audit	21.11.2018
Date of secondary audit	26.11.2018
Etherscan address	0xbd73e675e1fa3d60a302c797df5c82e558da7ce1

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Pantheon (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of Customer`s smart contract and its code review conducted between November 11th, 2018 - November 21th, 2018. Secondary audit was conducted between November 25th, 2018 - November 26th, 2018.

Scope

The scope of the project is **PantheonEcoSystem** smart contract.

Etherscan address: **0xbd73e675e1fa3d60a302c797df5c82e558da7ce1**

We have scanned this smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that are considered (the full list includes them but is not limited to them):

- 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

Executive Summary

According to the assessment, Customer`s smart contract is secure.

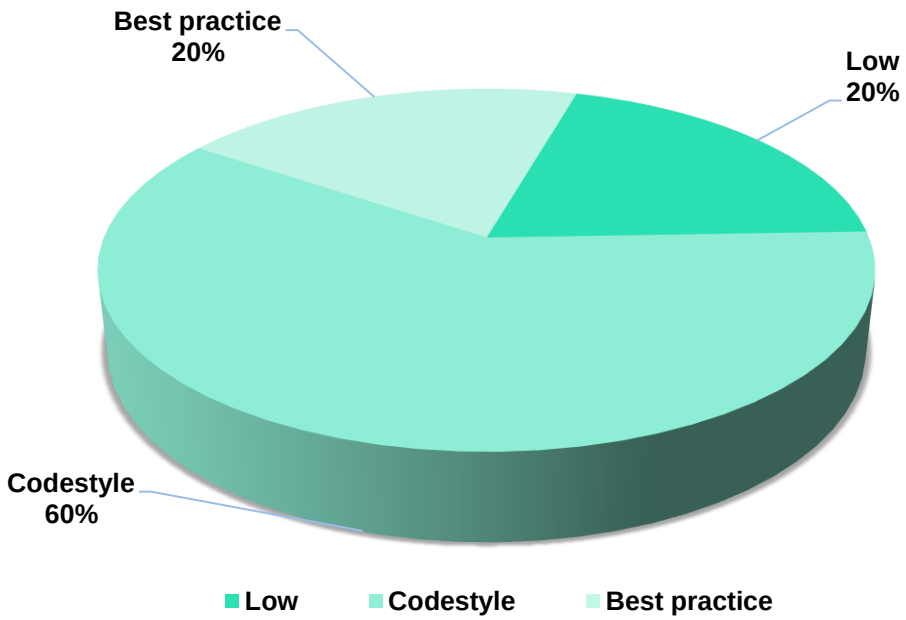
Insecure	Poor secured	Secured	Well-secured

You are here

Our team performed analysis of code functionality, manual audit and automated checks with Mythril, Slither and remix IDE (see Appendix B pic 1-5). All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in Audit overview section. General overview is presented in AS-IS section and all found issues can be found in Audit overview section.

We found 1 low vulnerability in smart contract. We also outline 3 code style issues and 1 best practice recommendation.

Graph 1. The distribution of vulnerabilities.



Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to tokens lose etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have 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 tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can't have significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations and info statements can't affect smart contract execution and can be ignored.

AS-IS overview

PantheonEcoSystem contract uses **SafeMath** and **SafeMathInt** libraries for math operations with safety checks that detects errors. It also uses **Fee** library to provide fee mechanism for smart contracts functions.

BuyToken has 1 modifier:

- **onlyValidTokenAmount** - checks whether specified amount of tokens is greater than 0 and less or equal than users tokens amount.

BuyToken has 20 functions:

- **buy** is a public payable function - buys tokens with a specified address as a referrer.
- **sell** is a public function - sells specified amount of tokens and adds funds to users dividends. Has **onlyValidTokenAmount** modifier.
- **transfer** is a public function - transfers specified amount of tokens to a specified address. Has **onlyValidTokenAmount** modifier.
- **reinvest** is a public function - reinvests all user dividends.
- **withdraw** is a public function - withdraws all user dividends.
- **exit** is a public function - sells user tokens and calls withdraw.
- **donate** is a public payable function - donates **msg.value** to the contract.
- **totalSupply** is a public view function - returns **total_supply**.
- **balanceOf** is a public view function - returns all tokens of the specified address.
- **dividendsOf** is a public view function - returns amount of users dividends for specified address.
- **expectedTokens** is a public view function - returns amount of tokens that can be gained from given amount of funds.

- **expectedFunds** is a public view function - returns amount of funds that can be gained from given amount of tokens.
- **buyPrice** is a public view function - returns purchase price of next 1 token.
- **sellPrice** is a public view function - returns selling price of next 1 token.
- **mintTokens** is an internal function - mints specified amount of tokens to specified address.
- **burnTokens** is an internal function - burns specified amount of tokens from specified address.
- **rewardReferrer** is an internal function - rewards referrer from specified amount of funds.
- **fundsToTokens** is an internal view function - transforms funds to tokens.
- **tokensToFunds** is an internal view function - transforms tokens to funds.
- public payable fallback function that calls **buy** with **msg.data.toAddr()** as a parameter.

Audit overview

Critical

No critical severity vulnerabilities were found.

High

No high severity vulnerabilities were found.

Medium

No medium severity vulnerabilities were found.

Low

1. Compiler version is not locked. Consider locking compiler version with the latest one (see Appendix A pic 1 for evidence).

```
pragma solidity ^0.4.25; // bad: compiles w 0.4.25 and above

pragma solidity 0.4.25; // good: compiles w 0.4.25 only
```

   Fixed: Compiler version was locked.

2. Locked Ether

   Ether can be locked on the contract. Contract uses fees to collect funds. Collected fees are stored in **shared_profit** variable. **shared_profit** also contains funds, which are gained through **donate** function. When the last investor of the tokens sells all of their tokens and the **total_supply** equals to 0 and token balance equals to 0, fees will be locked on account.

   We informed Customer about the issue and he confirmed that impact is low. Customer accepts the risks and define the issue as desired behavior.

Lowest / Code style / Best Practice

Best Practice

3. **split** and **get_tax** functions should specify data location in function parameters either memory or storage.

   Fixed: Data location was specified.

4. Transaction ordering dependence

   Contract logic depends on transactions order and is vulnerable to front running attacks. Functions like **sell**, **buy**, **reinvest**, **transfer**, **exit** and **withdraw** directly impact tokens price.

   An attacker could manipulate token price. Moreover, valid users can face that the price is different from expected and heavily depends on transaction order.

   More information here - **https://consensys.github.io/smart-contract-best-practices/known_attacks/#front-running-aka-transaction-ordering-dependence**.

   We were notified by Customer that this is expected behavior of the smart contract.

Code style

5. Variable name must be in **mixedCase** on lines - 8, 9, 11, 25, 26, 27, 28, 31, 44, 49, 52, 76, 116, 134, 139, 172, 291, 306, 346, 371, 397, 400, 432, 455, 468, 471, 552.

6. Function name must be in **mixedCase** on line - 563.

7. Variable name must be in **SNAKE_CASE** on lines - 31, 34, 41.

8. Visibility modifier should be marked explicitly on line 34.

   Fixed: Visibility modifier was specified.

Conclusion

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. For the contract high level description of functionality was presented in As-is overview section of the report.

Audit report contains all found security vulnerabilities and other issues in the reviewed code.

Overall quality of reviewed contracts is good; however, it contains 1 low vulnerability. It doesn't have any serious security impact and the contract can be deployed to mainnet without changes.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

The audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only – we recommend proceeding with several independent audits and a public bug bounty program to ensure security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on blockchain platform. The platform, its programming language, and other software related to the smart contract can have own vulnerabilities that can lead to hacks. Thus, the audit can’t guarantee explicit security of the audited smart contracts.

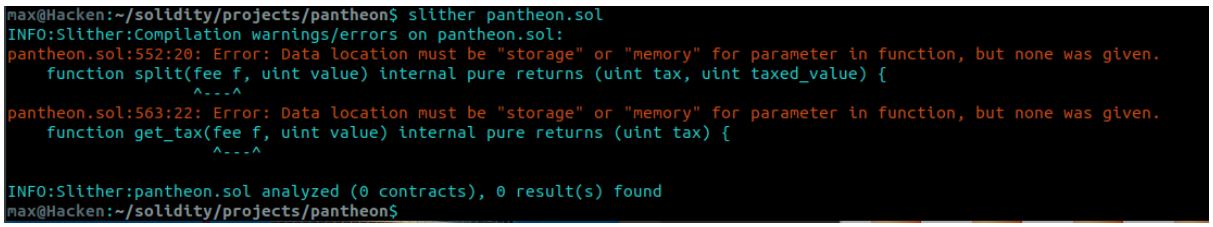
Appendix A. Evidences

Pic 1. Compiler version not locked:

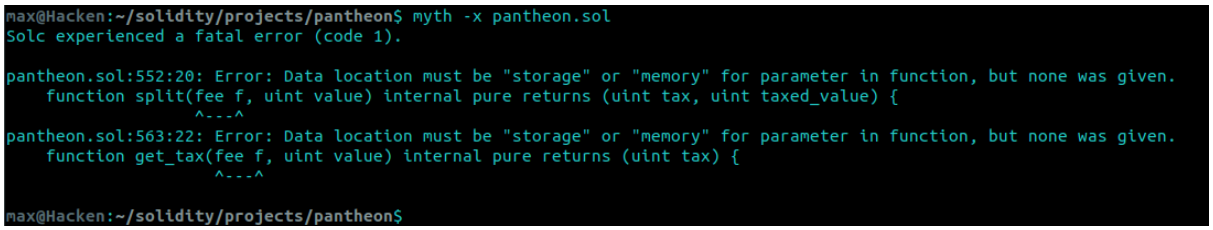
```
1  pragma solidity ^0.4.25;
```

Appendix B. Automated tools reports

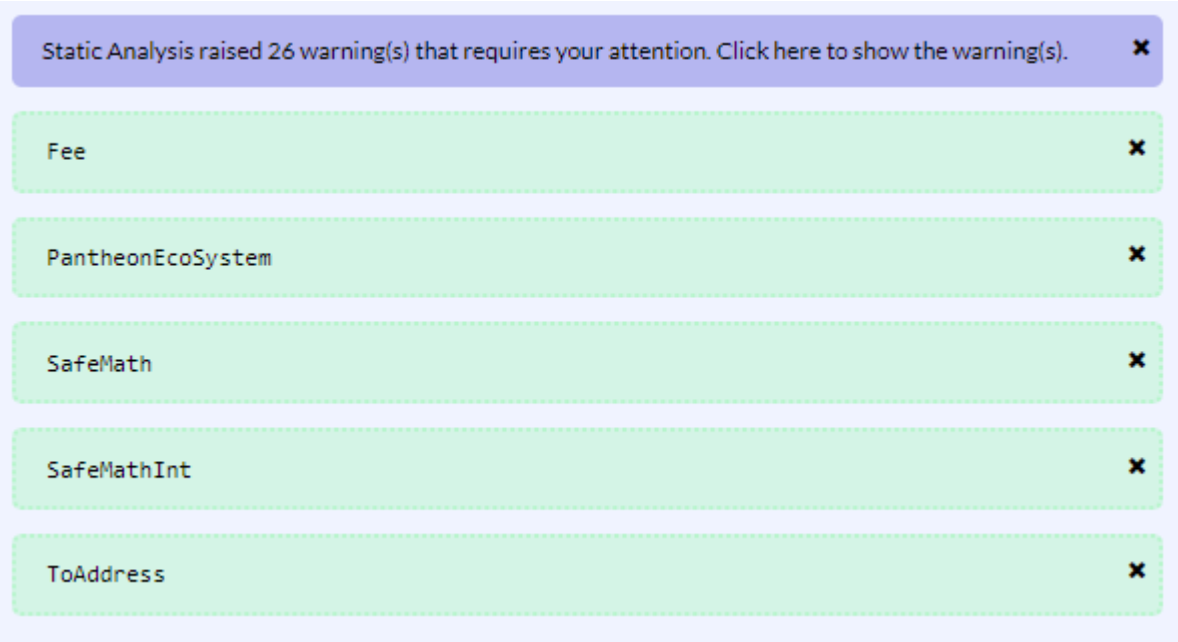
Pic 1. Slither automated report:



Pic 2. Mythril automated report:



Pic 3. Remix IDE automated report part 1:



Pic 4. Remix IDE automated report part 2:

Pic 5. Remix IDE automated report part 3: