shieldify

PANTHEON ECOSYSTEM

SECURITY REVIEW

Date: 23 September 2023

CONTENTS

1. About Shieldify

We are Shieldify Security – a company on a mission to make web3 protocols more secure, cost-efficient and user-friendly. Our team boasts extensive experience in the web3 space as both smart contract auditors and developers that have worked on top 100 blockchain projects with multi-million dollars in market capitalization.

Book an audit and learn more about us at shieldify.org or @ShieldifySec.

2. Disclaimer

This security review does not guarantee bulletproof protection against a hack or exploit. Smart contracts are a novel technological feat with many known and unknown risks. The protocol, which this report is intended for, indemnifies Shieldify Security against any responsibility for any misbehavior, bugs, or exploits affecting the audited code during any part of the project's life cycle. It is also pivotal to acknowledge that modifications made to the audited code, including fixes for the issues described in this report, may introduce new problems and necessitate additional auditing.

3. About Pantheon Ecosystem

Pantheon ecosystem is a permissionless reserve currency and store of value for ETH, fully collateralized and designed for consistent ETH appreciation. It's value benefits from the volatility of the backing asset (ETH), meaning it appreciates when ETH price rises. Unlike traditional tokens, anyone can mint `PANTHEON` ERC-20 token by depositing ETH and burn it to redeem ETH, subject to a **10%** tax (except for liquidity pool trading). This tax structure supports continuous price increase, with **6%** retained for each Mint and Burn, **3%** incentivizing Liquidity Providers of `PANTHEON` – USDC, and **1%** allocated to the team.

Learn more about Pantheon's concept and the technicalities behind it here.

4. Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4.1 Impact

- **High** – results in a significant risk for the protocol's overall well-being. Affects all or most users
- **Medium** – results in a non-critical risk for the protocol affects all or only a subset of users, but is still unacceptable
- **Low** – losses will be limited but bearable – and covers vectors similar to griefing attacks that can be easily repaired or even gas optimization techniques

4.2 Likelihood

- **High** – almost certain to happen and highly lucrative for execution by malicious actors
- **Medium** – still relatively likely, although only conditionally possible
- **Low** – requires a unique set of circumstances and poses non-lucrative cost-of-execution to rewards ratio for the actor

5. Audit Summary

The audit duration lasted 3 days and a total of 72 hours were spent by the three auditors - @ShieldifyMartin, @ShieldifyAnon and @ShieldifyGhost. This is the first audit for the Pantheon ecosystem protocol. The documentation is extensive.

The audit identified issues primarily around the proper implementation of the business logic, although they don't represent a threat to the funds of the protocol's users. Additionally, some informational findings have been found, together with a few opportunities for gas-optimization & recommendations which are not included in the audit report.

5.1 Protocol Summary

Project Name	**Pantheon ecosystem**
Repository	Pantheon ecosystem
Type of Project	ERC-20, Fully Collateralized Permisionless Reserve Currency
Audit Timeline	3 days
Review Commit Hash	55f4902535cdbb038b21e8c1c9cfb4582cbf354d
Fixes Review Commit Hash	ece738a4907f2f55cf13418dde84f5373a5400a0

5.2 Scope

The following smart contracts were in the scope of the audit:

File	**nSLOC**
PANTHEON.sol	78
Total	**78**

6. Findings Summary

- **Critical** and **High** issues: **0**
- **Medium** issues: **1**
- **Low** issues: **2**

ID	Title	Severity
[M-01]	Permanent `Fees` Loss if The Owner Forgets to Execute the `setFeeAddress()` Function	Medium
[L-01]	Ownership Role Transfer Function Implement `Single-Step` Role Transfer	Low
[L-02]	Methods `mint()` and `redeem()` are Not Following The `Checks-Effects-Interactions` Pattern	Low

7. Findings

[M-01] Permanent `Fees` Loss if The Owner Forgets to Execute the `setFeeAddress()` Function

Severity

Medium

Description

The **`FEE_ADDRESS`** address in **`PANTHEON.sol`** contract is used to store **4%** tax fees when the user calls **`mint()`** or **`redeem()`** functions. This **4%** fee is distributed as follows:

- **3%** will be used to incentivize **`liquidity`** providers manually through bribes.
- **1%** is the **`pantheon team`** profit.

The impact is a loss of liquidity and Pantheon team fees.

In case the owner of the contract forgets to call **`setFeeAddress()`** function, **`FEE_ADDRESS`** storage variable will be initialized by default to `address(0x0)`. Therefore when someone calls **`mint()`** or **`redeem()`** functions the fees will be sent to the zero address, which is a loss of tax fees for the protocol.

Location of Affected Code

File: PANTHEON.sol#L35-L38

```
address payable private FEE_ADDRESS;
...
function setFeeAddress(address _address) external onlyOwner {
  require(_address != address(0x0));
  FEE_ADDRESS = payable(_address);
}
```

Recommendation

It is recommended to pre-set the **`FEE_ADDRESS`** in the constructor as follows:

```
- address payable private FEE_ADDRESS;
+ address payable public feeAddress;

+ error ZeroAddressNotAllowed();

constructor(address _feeAddress) payable ERC20("Pantheon", "PANTHEON") {
+ if(_feeAddress == address(0)) revert ZeroAddressNotAllowed();

  _mint(msg.sender, msg.value * MIN);
  totalEth = msg.value;

+ feeAddress = payable(_feeAddress);

  transfer(0x000000000000000000000000000000000000dEaD, 10000);
}
```

Team Response

Acknowledged, will be mitigated.

[L-01] Ownership Role Transfer Function Implement `Single-Step` Role Transfer

Severity

Low Risk

Description

The current ownership transfer process for all the contracts inheriting from **`Ownable`** involves the current owner calling the **`transferOwnership()`** function. If the nominated **`EOA`** account is not a valid account, it is entirely possible that the owner may accidentally transfer ownership to an uncontrolled account, losing access to all functions with the **`onlyOwner`** modifier.

Example: OpenZeppelin Ownable

```
/**
* @dev Transfers ownership of the contract to a new account (`newOwner`).
* Can only be called by the current owner.
*/
function transferOwnership(address newOwner) public virtual onlyOwner {
  require(newOwner != address(0), "Ownable: new owner is the zero address
     ");
  _transferOwnership(newOwner);
}
```

Location of Affected Code

File: PANTHEON.sol#L8

```
contract PANTHEON is ERC20Burnable, Ownable, ReentrancyGuard {
```

Recommendation

It is recommended to implement a two-step process where the owner nominates an account and the nominated account needs to call an **`acceptOwnership()`** function for the transfer of the ownership to fully succeed. This ensures the nominated EOA account is a valid and active account. This can be easily achieved by using OpenZeppelin's **`Ownable2Step`** contract instead of **`Ownable`**.

```
- import "@openzeppelin/contracts/access/Ownable.sol";
+ import {Ownable2Step} from "@openzeppelin/contracts/access/Ownable2Step
    .sol";

- contract PANTHEON is ERC20Burnable, Ownable, ReentrancyGuard {
+ contract PANTHEON is ERC20Burnable, Ownable2Step, ReentrancyGuard {
```

Team Response

Acknowledged and fixed as proposed.

[L-02] Methods `mint()` and `redeem()` are Not Following The `CEI` Pattern

Severity

Low Risk

Description

In **`mint()`** & **`redeem()`** functions, even though there is a **`nonReentrant`** modifier, the **`Checks-Effects-Interactions`** pattern is not followed. The **`totalEth`** state in the both function is executed after the ether transfers. It is recommended to always first change the state before doing external calls - while the code is not vulnerable right now due to the **`nonReentrant`** modifier, it is still a best practice to be followed.

Location of Affected Code

File: PANTHEON.sol#L62

```
function mint(address reciever) external payable nonReentrant {
```

File: PANTHEON.sol#L40

```
function redeem(uint256 pantheon) external nonReentrant {
```

Recommendation

Apply the **`Checks-Effects-Interactions`** Pattern for both the **`mint()`** and **`redeem()`** functions.

Team Response

Acknowledged, CEI Pattern will implemented.

shieldify

Thank you!