EtherAuthority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	Pulse Litecoin
Platform:	Pulse Chain & Ethereum Network
Language:	Solidity
Date:	October 14th, 2024

Table of contents

`

Introduction

EtherAuthority was contracted by Pulse Litecoin to perform the Security audit of the Pulse Litecoin smart contract code. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on October 14th, 2024.

The purpose of this audit was to address the following:

- Ensure that all claimed functions exist and function correctly.
- Identify any security vulnerabilities that may be present in the smart contract.

Project Background

- Pulse Litecoin Contracts handle multiple contracts, and all contracts have different functions.
- Here's a brief overview of the key components and functionalities of the provided code:
  - **PulseBitcoin:** The PulseLitecoin contract expands on the PulseBitcoin mining concept by introducing an ERC20 token for the mineable asset (pLTC). It allows users to leverage their ASIC tokens to mine pLTC and implements a penalty-based system to encourage timely actions when ending mining operations. The integration of ReentrancyGuard ensures that operations remain secure and prevents abuse of the mining logic.
  - **PulseBitcoinMineable:** The PulseBitcoinMineable contract is designed to be a flexible, abstract base for any contracts that wish to interact with the PulseBitcoin mining system. It offers mechanisms to start and end mining, manage penalties, and handle miner data efficiently. The use of abstract contracts for Asic and PulseBitcoin ensures modularity, allowing the contract to be extended and integrated into more complex systems like the PulseLitecoin contract.
- There are 2 smart contracts files that were included in the audit scope.
- The contracts inherit the ERC20 and ReentrancyGuard standard smart contracts from the OpenZeppelin library.
- These OpenZeppelin contracts are considered community-audited and time-tested and hence are not part of the audit scope.

Audit scope

Name	Code Review and Security Analysis Report for Pulse Litecoin Smart Contract
Platform	**Pulse Chain & Ethereum Network / Solidity**
File 1	PulseLitecoin.sol
File 1 MD5 hash	35B6373F1B2B8C2B958D02AB397ED042
File 2	PulseBitcoinMineable.sol
File 2 MD5 hash	BB4FEE22A9570986C82C462775CA6EAF
Audit Date	October 14th, 2024

Claimed Smart Contract Features

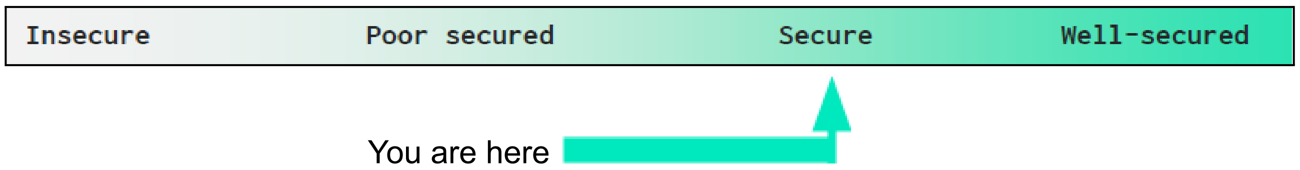
Claimed Feature Detail	Our Observation
File 1: PulseLitecoin.sol **Tokenomics:** ● Name: PulseLitecoin ● Symbol: pLTC ● Decimals: 12 ● OpenZeppelin library used. **Key Points:** ● This contract introduces a custom token (pLTC) and interacts with external tokens like ASIC and pBTC as part of its mining and reward mechanism. ● There are checks to prevent reentrancy attacks using the nonReentrant modifier. ● The contract likely relies on the inherited PulseBitcoinMineable for mining-related calculations. **Other Specification:** ● Reentrancy Protection: Both minerStart and minerEnd are marked as nonReentrant, preventing reentrancy attacks. ● Penalties: The minerEnd function implements a penalty for miners who go beyond a certain number of days without ending their mining contract. This penalty halves both the mined pLTC and the returned ASIC coins. **Ownership Control:** ● There are no owner functions, which makes it 100% decentralized.	**YES, This is valid.**
File 2: PulseBitcoinMineable.sol ● The PulseBitcoinMineable contract effectively integrates mining functionality, handles miner data, and interacts	**YES, This is valid.**

seamlessly with the PulseBitcoin and Asic contracts. ● The setup allows miners to be easily started and ended while keeping track of relevant mining data. ● However, careful attention should be paid to gas costs, error handling, and security implications, especially when working with external contracts and user funds.	

Note: pBTC is the same as PLSB

Audit Summary

According to the standard audit assessment, Customer`s solidity-based smart contracts are **“secured”**. These contracts do not contain any ownership control, hence they are **100% decentralized**.



We used various tools like Slither, Solhint, and Remix IDE. At the same time, this finding is based on a critical analysis of the manual audit.
All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit Overview section. The general overview is presented in the AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 1 high, 0 medium, 1 low, and 2 very low-level issues.

We confirm that 1 high and 1 low severity issues are fixed/acknowledged in the revised smart contract code.

Investor Advice: A technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner-controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	The solidity version is not specified	Passed
	The solidity version is too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Moderated
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Passed
Gas Optimization	“Out of Gas” Issue	Moderated
	High consumption ‘for/while’ loop	Passed
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage is not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

Business Risk Analysis

Category	Result
● Buy Tax	0%
● Sell Tax	0%
● Cannot Buy	No
● Cannot Sell	No
● Max Tax	0%
● Modify Tax	No
● Fee Check	Not Detected
● Is Honeypot	Not Detected
● Trading Cooldown	Not Detected
● Can Pause Trade?	Not Detected
● Pause Transfer?	No
● Max Tax?	No
● Is it Anti-whale?	Not Detected
● Is Anti-bot?	Not Detected
● Is it a Blacklist?	No
● Blacklist Check	No
● Can Mint?	No
● Is it a Proxy Contract?	No
● Can Take Ownership?	No
● Creator Percentage?	0.00%
● Hidden Owner?	Not Detected
● Self Destruction?	Not Detected
● Auditor Confidence	High

Overall Audit Result: PASSED

Code Quality

This audit scope has 2 smart contracts. Smart contracts contain Libraries, Smart contracts, inherits, and Interfaces. This is a compact and well-written smart contract.

The libraries in Pulse Litecoin are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties/methods can be reused many times by other contracts in the Pulse Litecoin.

The Pulse Litecoin team has provided scenario and unit test scripts, which have helped to determine the integrity of the code in an automated way.

Code parts are well commented on in smart contracts. Ethereum's NatSpec commenting style is used, which is a good thing.

Documentation

We were given a Pulse Litecoin smart contract code in the form of a file. The hash of that code is mentioned in the table above.

As mentioned above, the code parts are well-commented and the logic is straightforward. So, it is easy to understand the programming flow and complex code logic quickly. Comments are very helpful in understanding the overall architecture of the protocol.

Use of Dependencies

As per our observation, the libraries used in this smart contract infrastructure are based on well-known industry standard open-source projects.

Apart from libraries, its functions are used in external smart contract calls.

AS-IS overview

PulseBitcoin.sol : Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	decimals	read	Passed	No Issue
3	minerStart	external	nonReentrant	No Issue
4	minerEnd	external	nonReentrant	No Issue
5	_minerStart	internal	Passed	No Issue
6	_minerEnd	internal	Passed	No Issue
7	_minerAt	internal	Passed	No Issue
8	_minerLoad	internal	Passed	No Issue
9	_minerAdd	internal	Passed	No Issue
10	_minerRemove	internal	Passed	No Issue
11	_minerIndexSearch	internal	Passed	No Issue
12	minerCount	external	Passed	No Issue
13	_miningDuration	internal	Passed	No Issue
14	_miningDuration	internal	Passed	No Issue
15	_daysForPenalty	internal	Passed	No Issue
16	_lastMinerIndex	internal	Passed	No Issue
17	_currentDay	internal	Passed	No Issue
18	minerCount	read	Passed	No Issue
19	minerStart	write	Passed	No Issue
20	minerEnd	write	Passed	No Issue
21	currentDay	read	Passed	No Issue
22	approve	write	Passed	No Issue
23	balanceOf	read	Passed	No Issue
24	transfer	write	Passed	No Issue
25	transferFrom	write	Passed	No Issue
26	nonReentrant	modifier	Passed	No Issue
27	_nonReentrantBefore	write	Passed	No Issue
28	_nonReentrantAfter	write	Passed	No Issue
29	_reentrancyGuardEntered	internal	Passed	No Issue
30	name	read	Passed	No Issue
31	symbol	read	Passed	No Issue
32	decimals	read	Passed	No Issue
33	totalSupply	write	Passed	No Issue
34	balanceOf	read	Passed	No Issue
35	transfer	write	Passed	No Issue
36	allowance	read	Passed	No Issue
37	approve	write	Passed	No Issue
38	transferFrom	write	Passed	No Issue
39	_transfer	internal	Passed	No Issue
40	_update	internal	Passed	No Issue
41	_mint	internal	Passed	No Issue
42	_burn	internal	Passed	No Issue

43	_approve	internal	Passed	No Issue
44	_approve	internal	Passed	No Issue
45	_spendAllowance	internal	Passed	No Issue

PulseBitcoinMineable.sol : Functions

Sl.	**Functions**	**Type**	**Observation**	**Conclusion**
1	constructor	write	Passed	No Issue
2	_minerStart	internal	Passed	No Issue
3	_minerEnd	internal	Passed	No Issue
4	_minerAt	internal	Passed	No Issue
5	_minerLoad	internal	Passed	No Issue
6	_minerAdd	internal	Passed	No Issue
7	_minerRemove	internal	Gas Optimizations	Refer Audit Findings
8	_minerIndexSearch	internal	Gas Optimizations	Refer Audit Findings
9	minerCount	external	Passed	No Issue
10	_miningDuration	internal	Passed	No Issue
11	_withdrawGracePeriod	internal	Passed	No Issue
12	_daysForPenalty	internal	Passed	No Issue
13	_lastMinerIndex	internal	Passed	No Issue
14	_currentDay	internal	Passed	No Issue
15	minerCount	read	Passed	No Issue
16	minerStart	write	Passed	Fixed
17	minerEnd	write	Anybody can execute the minerEnd	Refer Audit Findings
18	currentDay	read	Passed	Fixed
19	approve	write	Passed	Fixed
20	balanceOf	read	Passed	Fixed
21	transfer	write	Passed	Fixed
22	transferFrom	write	Passed	Fixed
23	approve	write	Passed	No Issue
24	balanceOf	read	Passed	No Issue
25	transfer	write	Passed	No Issue
26	transferFrom	write	Passed	No Issue

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
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
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 a 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.

Audit Findings

Critical Severity

No critical severity vulnerabilities were found.

High Severity

(1) Anybody can execute the minerEnd: **PulseBitcoinMineable.sol**

There is no check for the caller and the miner for which the minerEnd function executes. This allows the caller to get pLTC if the miner ends in a penalty period.

Resolution: We suggest correcting the logic. If this is part of the plan then please disregard this issue.

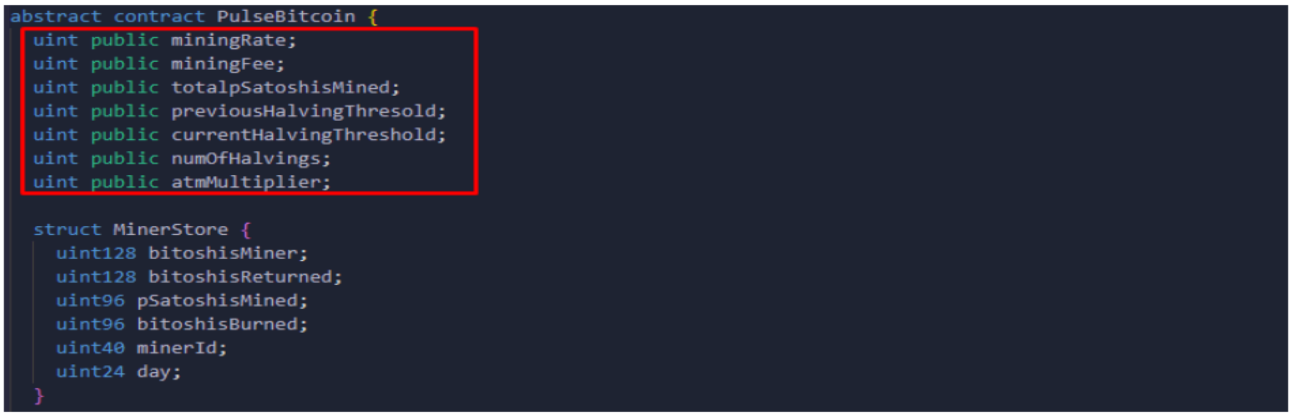
Status: Acknowledged

Medium

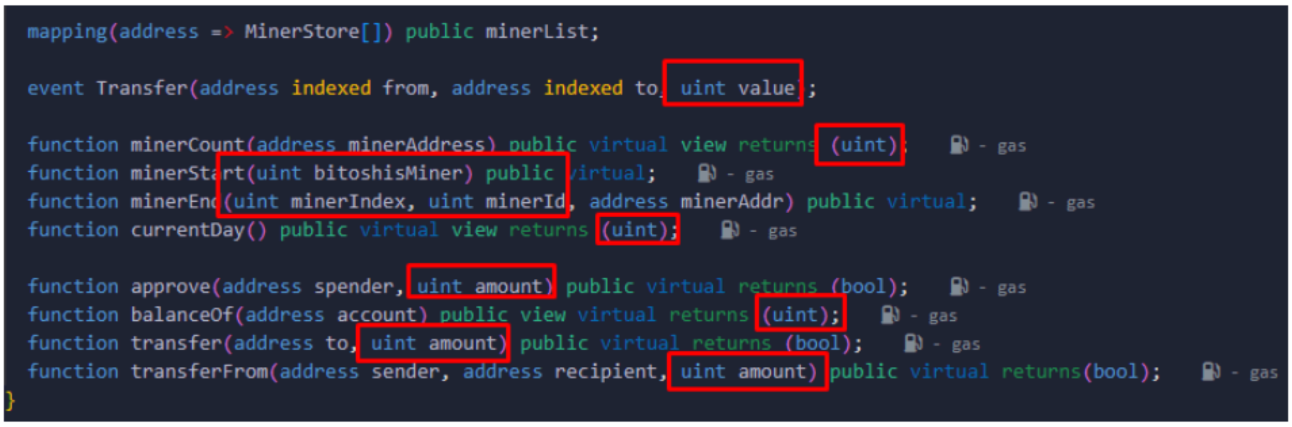
No medium-severity vulnerabilities were found.

Low

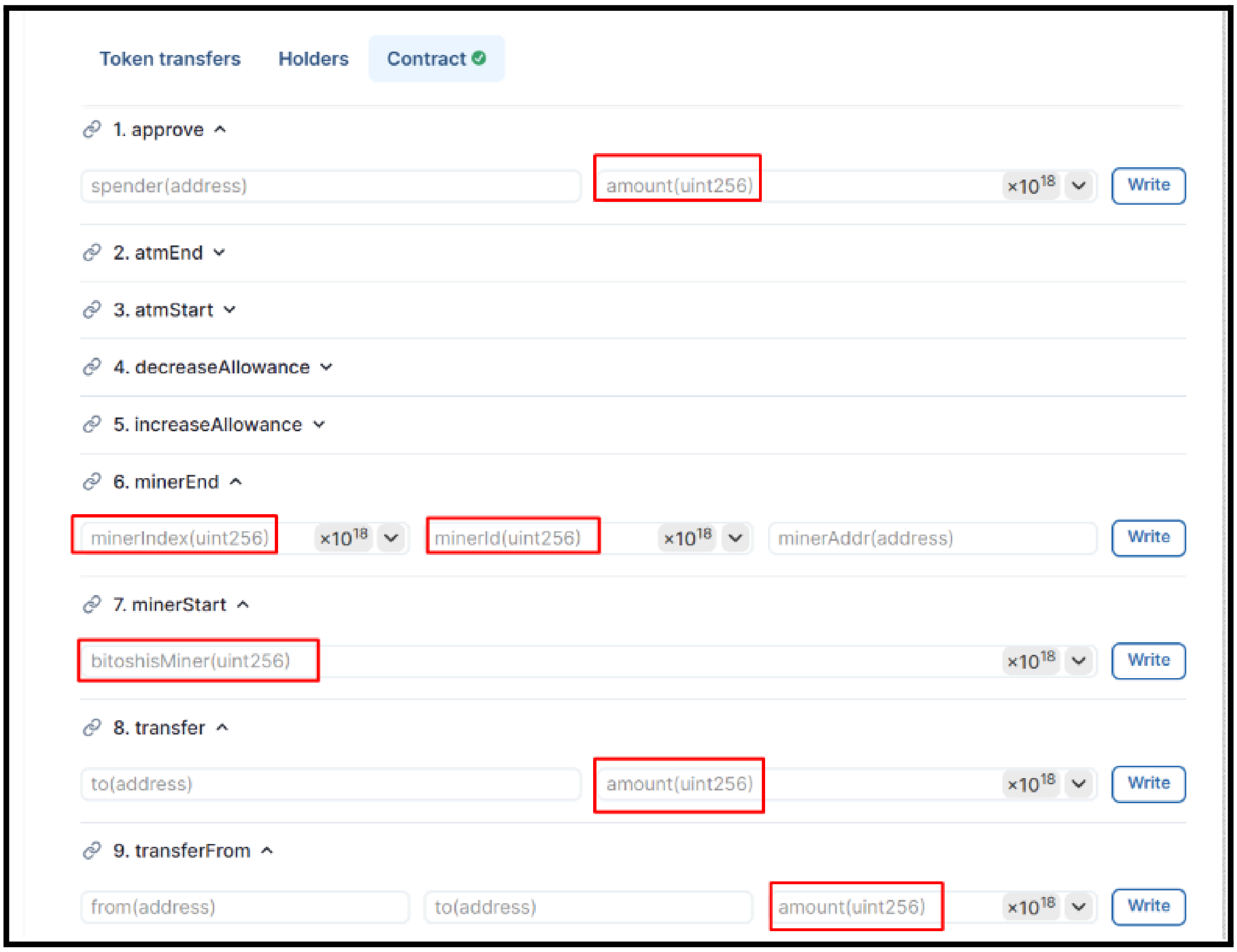
(1) Parameter type is different: **PulseBitcoinMineable.sol**



```
abstract contract PulseBitcoin {
  uint public miningRate;
  uint public miningFee;
  uint public totalpSatoshisMined;
  uint public previousHalvingThresold;
  uint public currentHalvingThreshold;
  uint public numOfHalvings;
  uint public atmMultiplier;

  struct MinerStore {
    uint128 bitoshisMiner;
    uint128 bitoshisReturned;
    uint96 pSatoshisMined;
    uint96 bitoshisBurned;
    uint40 minerId;
    uint24 day;
  }
```



```
mapping(address => MinerStore[]) public minerList;

event Transfer(address indexed from, address indexed to, uint value);

function minerCount(address minerAddress) public virtual view returns (uint);   - gas
function minerStart(uint bitoshisMiner) public virtual;   - gas
function minerEnd(uint minerIndex, uint minerId, address minerAddr) public virtual;   - gas
function currentDay() public virtual view returns (uint);   - gas

function approve(address spender, uint amount) public virtual returns (bool);   - gas
function balanceOf(address account) public view virtual returns (uint);   - gas
function transfer(address to, uint amount) public virtual returns (bool);   - gas
function transferFrom(address sender, address recipient, uint amount) public virtual returns(bool);   - gas
}
```



Token transfers | Holders | Contract

1. approve
spender(address) | amount(uint256) ×10^18 | Write

2. atmEnd

3. atmStart

4. decreaseAllowance

5. increaseAllowance

6. minerEnd
minerIndex(uint256) ×10^18 | minerId(uint256) ×10^18 | minerAddr(address) | Write

7. minerStart
bitoshisMiner(uint256) ×10^18 | Write

8. transfer
to(address) | amount(uint256) ×10^18 | Write

9. transferFrom
from(address) | to(address) | amount(uint256) ×10^18 | Write

The parameter types defined in the original contract and used in the PulseBitcoinMineable contract are different.

Resolution: We suggest correcting the parameter type to avoid any failure of the function in case of a large number.

Status: **Fixed**

Very Low / Informational / Best practices:

(1) Access Control: **PulseBitcoinMineable.sol**

No specific access control mechanisms (such as onlyOwner) are present in this contract. However, since the primary functions are user-specific (i.e., each user manages their own mining), this may not be required. If further administrative control is needed, consider implementing owner-only functions for managing the contract.

(2) Gas Optimizations: **PulseBitcoinMineable.sol**

The use of loops in functions like _minerIndexSearch() and _minerRemove() could potentially lead to high gas costs if there are a large number of miners. While this seems mitigated by the practical usage limits (i.e., the number of miners a user might control), it is still a concern to consider in gas-heavy environments.

Centralization Risk

The Pulse Litecoin smart contract does not have any ownership control, **hence it is 100% decentralized.**

Therefore, there is **no** centralization risk.

Conclusion

We were given a contract code in the form of a file, and we have used all possible tests based on given objects. We have observed 1 high, 1 low, and 2 Informational severity issues. We confirm that 1 high and 1 low severity issues are fixed/acknowledged in the revised smart contract code. **So, the smart contract is ready for mainnet deployment.**

Since possible test cases can be unlimited for such smart contracts protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover the maximum possible test cases to scan everything.

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

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

The security state of the reviewed smart contract, based on standard audit procedure scope, is **"Secured".**

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of the systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:
In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:
Our audit techniques included manual code analysis, user interface interaction, and white box penetration testing. We look at the project's website to get a high-level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this, we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally, we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract 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).

Due to the fact that the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free 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 also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

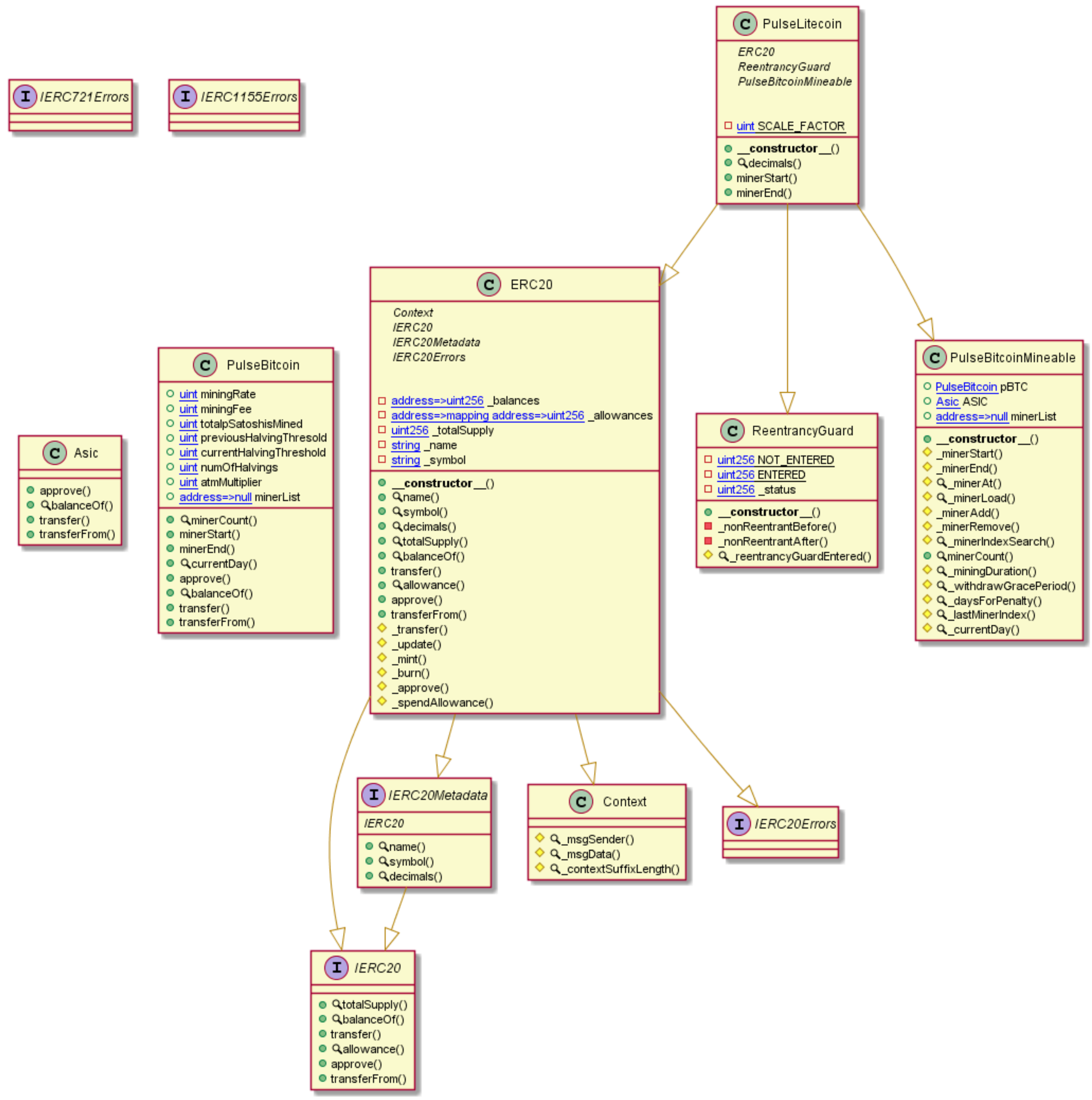
Technical Disclaimer

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

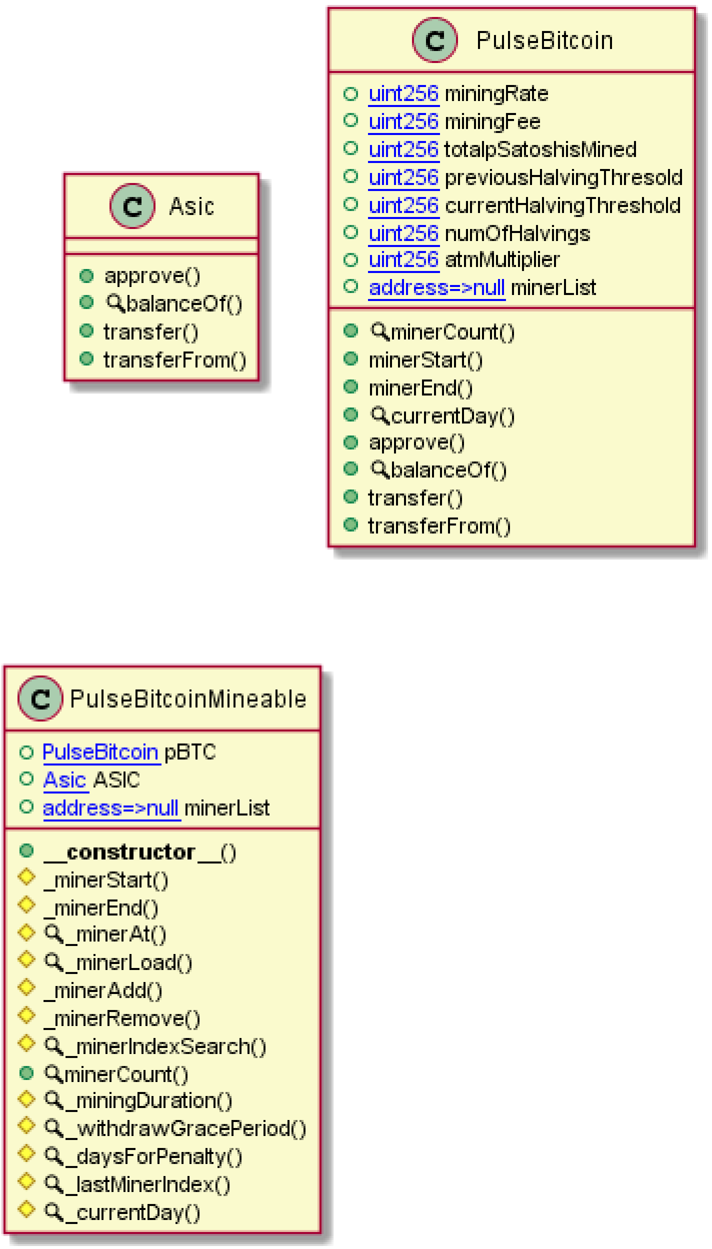
Appendix

Code Flow Diagram - Pulse Litecoin

PulseBitcoin Diagram



PulseBitcoinMineable Diagram



Slither Results Log

Slither Log >> PulseLitecoin.sol

```
INFO:Detectors:
PulseLitecoin.minerEnd(int256,uint256,uint256,address) (PulseLitecoin.sol#791-817) ignores
return value by pBTC.transfer(minerOwner,miner.pSatoshisMined) (PulseLitecoin.sol#814)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer
INFO:Detectors:
PulseBitcoinMineable.constructor() (PulseLitecoin.sol#550-556) ignores return value by
ASIC.approve(address(pBTC),type()(uint256).max) (PulseLitecoin.sol#555)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
Reentrancy in PulseLitecoin.minerEnd(int256,uint256,uint256,address)
(PulseLitecoin.sol#791-817):
	External calls:
	- miner = _minerEnd(minerIndex,minerOwnerIndex,minerId,minerOwner)
(PulseLitecoin.sol#793)
		- pBTC.minerEnd(uint256(minerIndex),minerId,address(this)) (PulseLitecoin.sol#627)
	State variables written after the call(s):
	- _mint(minerOwner,pltcMined / 2) (PulseLitecoin.sol#803)
		- _balances[from] = fromBalance - value (PulseLitecoin.sol#295)
		- _balances[to] += value (PulseLitecoin.sol#307)
	- _mint(msg.sender,pltcMined / 2) (PulseLitecoin.sol#804)
		- _balances[from] = fromBalance - value (PulseLitecoin.sol#295)
		- _balances[to] += value (PulseLitecoin.sol#307)
	- _mint(minerOwner,pltcMined) (PulseLitecoin.sol#811)
		- _balances[from] = fromBalance - value (PulseLitecoin.sol#295)
		- _balances[to] += value (PulseLitecoin.sol#307)
	- _mint(minerOwner,pltcMined / 2) (PulseLitecoin.sol#803)
		- _totalSupply += value (PulseLitecoin.sol#287)
		- _totalSupply -= value (PulseLitecoin.sol#302)
	- _mint(msg.sender,pltcMined / 2) (PulseLitecoin.sol#804)
		- _totalSupply += value (PulseLitecoin.sol#287)
		- _totalSupply -= value (PulseLitecoin.sol#302)
	- _mint(minerOwner,pltcMined) (PulseLitecoin.sol#811)
		- _totalSupply += value (PulseLitecoin.sol#287)
		- _totalSupply -= value (PulseLitecoin.sol#302)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Context._contextSuffixLength() (PulseLitecoin.sol#109-111) is never used and should be
removed
Context._msgData() (PulseLitecoin.sol#105-107) is never used and should be removed
ERC20._burn(address,uint256) (PulseLitecoin.sol#337-342) is never used and should be
removed
```

```
ReentrancyGuard._reentrancyGuardEntered() (PulseLitecoin.sol#469-471) is never used and
should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.20 (PulseLitecoin.sol#4) necessitates a version too recent to be trusted.
Consider deploying with 0.8.18.
solc-0.8.20 is not recommended for deployment
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Variable PulseBitcoinMineable.ASIC (PulseLitecoin.sol#528) is not in mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-c
onventions
INFO:Slither:PulseLitecoin.sol analyzed (12 contracts with 93 detectors), 14 result(s) found
```

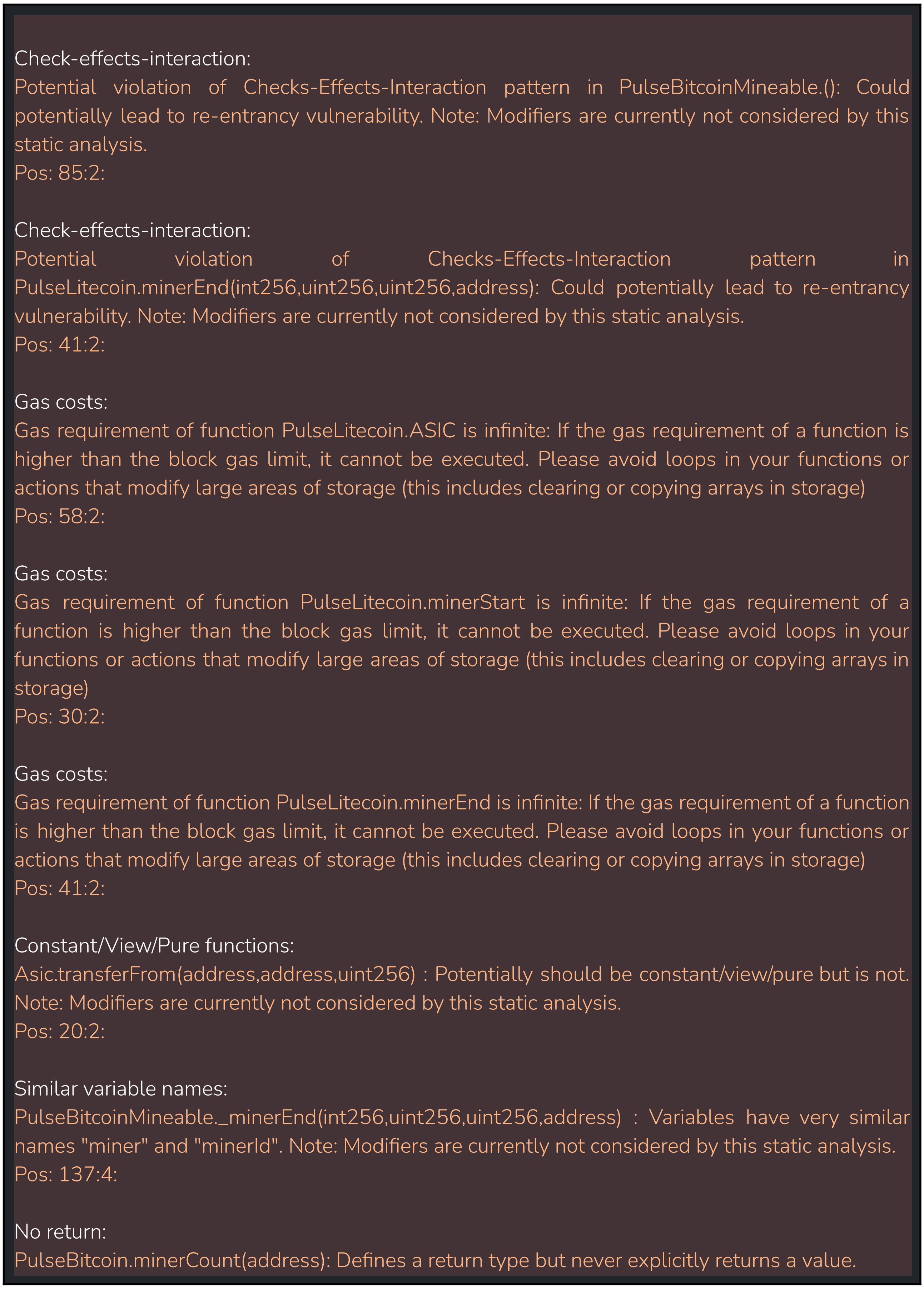
Slither Log >> PulseBitcoinMineable.sol

```
INFO:Detectors:
PulseBitcoinMineable._currentDay() (PulseBitcoinMineable.sol#295-297) is never used and
should be removed
PulseBitcoinMineable._daysForPenalty() (PulseBitcoinMineable.sol#287-289) is never used and
should be removed
PulseBitcoinMineable._lastMinerIndex() (PulseBitcoinMineable.sol#291-293) is never used and
should be removed
PulseBitcoinMineable._minerAdd(PulseBitcoinMineable.MinerStore[],PulseBitcoinMineable.Miner
Cache) (PulseBitcoinMineable.sol#206-218) is never used and should be removed
PulseBitcoinMineable._minerAt(uint256) (PulseBitcoinMineable.sol#168-186) is never used and
should be removed
PulseBitcoinMineable._minerEnd(int256,uint256,uint256,address)
(PulseBitcoinMineable.sol#124-166) is never used and should be removed
PulseBitcoinMineable._minerIndexSearch(PulseBitcoinMineable.MinerCache)
(PulseBitcoinMineable.sol#254-273) is never used and should be removed
PulseBitcoinMineable._minerLoad(uint256,address) (PulseBitcoinMineable.sol#188-204) is
never used and should be removed
PulseBitcoinMineable._minerRemove(PulseBitcoinMineable.MinerStore[],PulseBitcoinMineable.M
inerCache) (PulseBitcoinMineable.sol#220-250) is never used and should be removed
PulseBitcoinMineable._minerStart(uint256) (PulseBitcoinMineable.sol#101-114) is never used
and should be removed
PulseBitcoinMineable._miningDuration() (PulseBitcoinMineable.sol#279-281) is never used and
should be removed
PulseBitcoinMineable._withdrawGracePeriod() (PulseBitcoinMineable.sol#283-285) is never
used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.19 (PulseBitcoinMineable.sol#2) necessitates a version too recent to be
```

```
trusted. Consider deploying with 0.8.18.
solc-0.8.26 is not recommended for deployment
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Variable PulseBitcoinMineable.ASIC (PulseBitcoinMineable.sol#58) is not in mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-c
onventions
INFO:Slither:PulseBitcoinMineable.sol analyzed (3 contracts with 93 detectors), 16 result(s)
found
```

Solidity Static Analysis

PulseLitecoin.sol



```
Check-effects-interaction:
Potential violation of Checks-Effects-Interaction pattern in PulseBitcoinMineable.(): Could
potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this
static analysis.
Pos: 85:2:

Check-effects-interaction:
Potential violation of Checks-Effects-Interaction pattern in
PulseLitecoin.minerEnd(int256,uint256,uint256,address): Could potentially lead to re-entrancy
vulnerability. Note: Modifiers are currently not considered by this static analysis.
Pos: 41:2:

Gas costs:
Gas requirement of function PulseLitecoin.ASIC is infinite: If the gas requirement of a function is
higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or
actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 58:2:

Gas costs:
Gas requirement of function PulseLitecoin.minerStart is infinite: If the gas requirement of a
function is higher than the block gas limit, it cannot be executed. Please avoid loops in your
functions or actions that modify large areas of storage (this includes clearing or copying arrays in
storage)
Pos: 30:2:

Gas costs:
Gas requirement of function PulseLitecoin.minerEnd is infinite: If the gas requirement of a function
is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or
actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 41:2:

Constant/View/Pure functions:
Asic.transferFrom(address,address,uint256) : Potentially should be constant/view/pure but is not.
Note: Modifiers are currently not considered by this static analysis.
Pos: 20:2:

Similar variable names:
PulseBitcoinMineable._minerEnd(int256,uint256,uint256,address) : Variables have very similar
names "miner" and "minerId". Note: Modifiers are currently not considered by this static analysis.
Pos: 137:4:

No return:
PulseBitcoin.minerCount(address): Defines a return type but never explicitly returns a value.
```

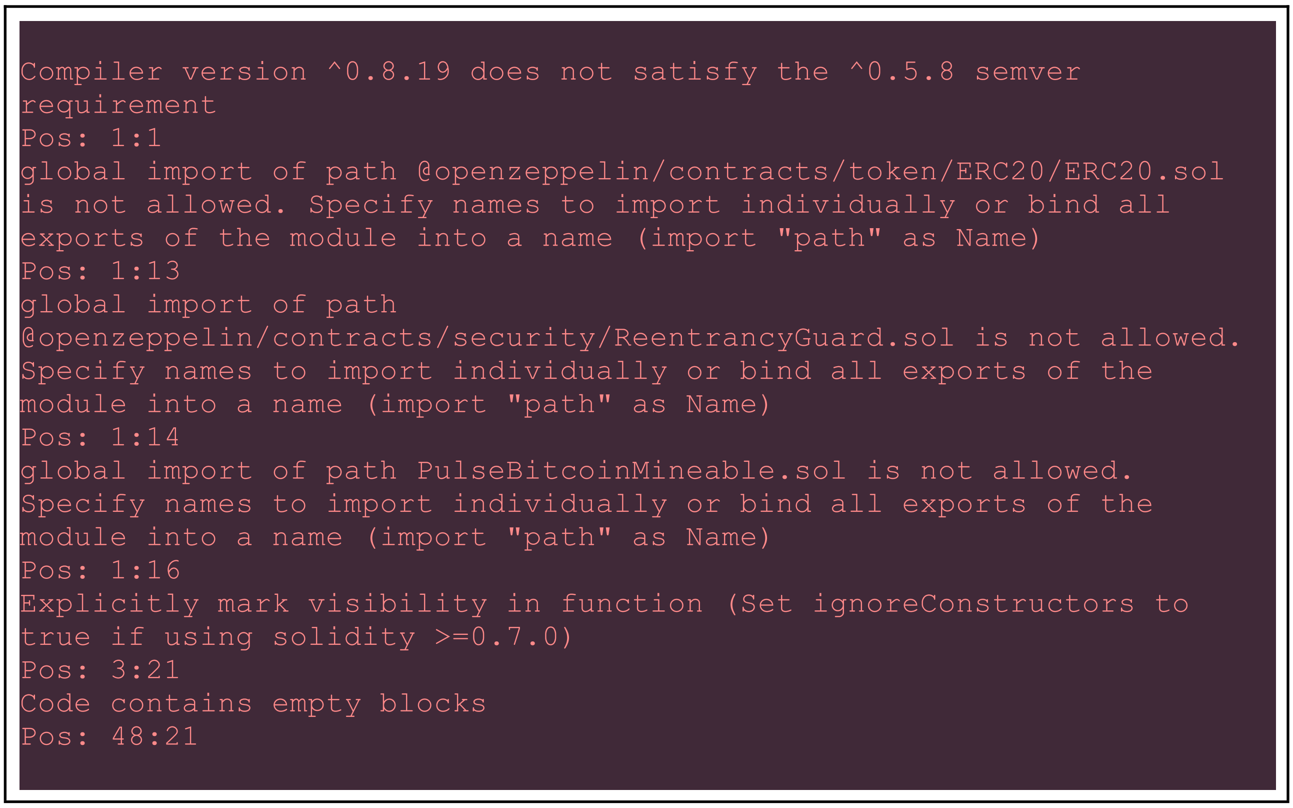
```
Pos: 45:2:

Data truncated:
Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of
0.1 since the result is an integer again. This does not hold for division of (only) literal values since
those yield rational constants.
Pos: 57:32:
```

PulseBitcoinMineable.sol

```
Check-effects-interaction:
Potential violation of Checks-Effects-Interaction pattern in
PulseBitcoinMineable._minerIndexSearch(struct PulseBitcoinMineable.MinerCache): Could
potentially lead to re-entrancy vulnerability.
Pos: 258:2:

Constant/View/Pure functions:
PulseBitcoinMineable._minerAt(uint256) : Is constant but potentially should not be.
Pos: 172:2:

Constant/View/Pure functions:
PulseBitcoinMineable._minerAdd(struct PulseBitcoinMineable.MinerStore[],struct
PulseBitcoinMineable.MinerCache) : Potentially should be constant/view/pure but is not.
Pos: 210:2:

Similar variable names:
PulseBitcoinMineable._minerEnd(int256,uint256,uint256,address) : Variables have very similar
names "miner" and "minerId".
Pos: 140:24:

No return:
PulseBitcoin.minerCount(address): Defines a return type but never explicitly returns a value.
Pos: 45:2:

No return:
PulseBitcoin.transferFrom(address,address,uint256): Defines a return type but never explicitly
returns a value.
Pos: 53:2:
```

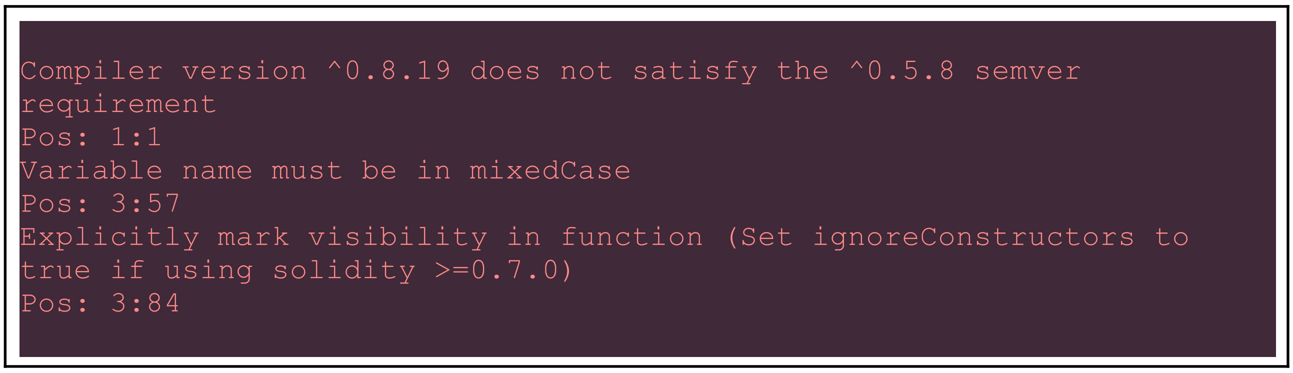
Solhint Linter

PulseLitecoin.sol



```
Compiler version ^0.8.19 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:1
global import of path @openzeppelin/contracts/token/ERC20/ERC20.sol
is not allowed. Specify names to import individually or bind all
exports of the module into a name (import "path" as Name)
Pos: 1:13
global import of path
@openzeppelin/contracts/security/ReentrancyGuard.sol is not allowed.
Specify names to import individually or bind all exports of the
module into a name (import "path" as Name)
Pos: 1:14
global import of path PulseBitcoinMineable.sol is not allowed.
Specify names to import individually or bind all exports of the
module into a name (import "path" as Name)
Pos: 1:16
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 3:21
Code contains empty blocks
Pos: 48:21
```

PulseBitcoinMineable.sol



```
Compiler version ^0.8.19 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:1
Variable name must be in mixedCase
Pos: 3:57
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 3:84
```

Software analysis result:

This software reported many false positive results and some are informational issues. So, those issues can be safely ignored.