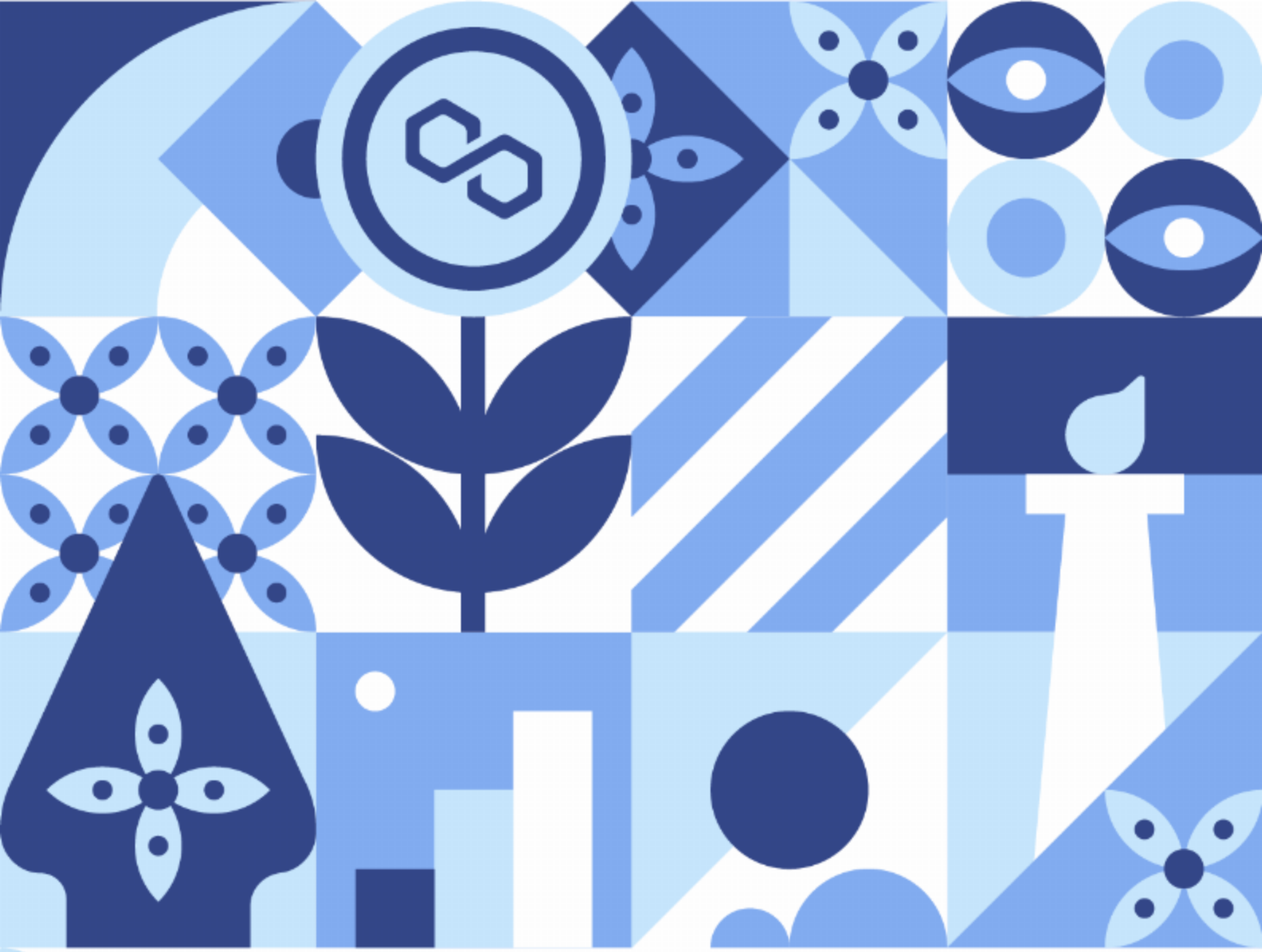

Safe Haven Token

Smart Contract Audit Report

SYSFIXED

05 May 2022

TABLE OF CONTENTS

AUDITED DETAILS

Audited Project

Project name	Token ticker	Blockchain
Safe Haven Token	SHA	Polygon Matic

Addresses

Contract address	0x534f39c5f4df9cb13e16b24ca07c7c8c0e2eadb7
Contract deployer address	0x1bde1Bae878131B919ce8316619C8409f2624E5f

Project Website

https://safehaven.io/

Codebase

https://polygonscan.com/address/0x534f39c5f4df9cb13e16b24ca07c7c8c0e2eadb7#code

SUMMARY

Established in 2017, Safe Haven aims to provide advanced FinTech solutions powered by blockchain. However, unlike many others, our solutions are patented globally, and we take the time to do things right. Focusing on security, we provide decentralized financial, backup, inheritance, and data transfer products for individual consumers and established organizations.

Contract Summary

Documentation Quality

Safe Haven Token provides a very good documentation with standard of solidity base code.

- The technical description is provided clearly and structured and also dont have any high risk issue.

Code Quality

The Overall quality of the basecode is standard.

- Standard solidity basecode and rules are already followed by Safe Haven Token with the discovery of several low issues.

Test Coverage

Test coverage of the project is 100% (Through Codebase)

Audit Findings Summary

- SWC-100 SWC-108 | Explicitly define visibility for all state variables on lines 81, 84 and 87.
- SWC-103 | Pragma statements can be allowed to float when a contract is intended on lines 5.
- SWC-110 SWC-123 | It is recommended to use of revert(), assert(), and require() in Solidity, and the new REVERT opcode in the EVM on lines 534, 515 and 274.
- SWC-111 | It is recommended to use alternatives to the deprecated constructions on lines 220, 256, 286, 299, 324, 444 and 494.
- SWC-120 | It is recommended to use external sources of randomness via oracles on lines 129, 185, 192, 208, 221, 287, 367, 481 and 483.

CONCLUSION

We have audited the Safe Haven Token project released in May 2022 to discover issues and identify potential security vulnerabilities in Safe Haven Token Project. This process is used to find technical issues and security loopholes which might be found in the smart contract.

The security audit report provides satisfactory results with low-risk issues.

The issues found in the Safe Haven Token smart contract code do not pose a considerable risk. The writing of the contract is close to the standard of writing contracts in general. The low-risk issues found are that a floating pragma is set, a state variable visibility is not set, the potential use of "block.number" as a source of randomness, the "constant" state mutability modifier is deprecated, and the requirement violation. 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. A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments). Using "constant" as a state mutability modifier in function "getValueAt" is disallowed as of Solidity version 0.5.0. Use "view" instead.

AUDIT RESULT

Article	Category	Description	Result
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	ISSUE FOUND
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	PASS
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	PASS
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	ISSUE FOUND
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	PASS
Unprotected Ether Withdrawal	SWC-105	Due to missing or insufficient access controls, malicious parties can withdraw from the contract.	PASS
SELFDESTRUCT Instruction	SWC-106	The contract should not be self-destructible while it has funds belonging to users.	PASS
Reentrancy	SWC-107	Check effect interaction pattern should be followed if the code performs recursive call.	PASS
Uninitialized Storage Pointer	SWC-109	Uninitialized local storage variables can point to unexpected storage locations in the contract.	PASS
Assert Violation	SWC-110 SWC-123	Properly functioning code should never reach a failing assert statement.	ISSUE FOUND
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	ISSUE FOUND
Delegate call to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	PASS

DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless required.	PASS
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	PASS
Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	PASS
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	PASS
Signature Unique ID	SWC-117 SWC-121 SWC-122	Signed messages should always have a unique id. A transaction hash should not be used as a unique id.	PASS
Incorrect Constructor Name	SWC-118	Constructors are special functions that are called only once during the contract creation.	PASS
Shadowing State Variable	SWC-119	State variables should not be shadowed.	PASS
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes or be predictable.	ISSUE FOUND
Write to Arbitrary Storage Location	SWC-124	The contract is responsible for ensuring that only authorized user or contract accounts may write to sensitive storage locations.	PASS
Incorrect Inheritance Order	SWC-125	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order. The rule of thumb is to inherit contracts from more /general/ to more /specific/.	PASS
Insufficient Gas Griefing	SWC-126	Insufficient gas griefing attacks can be performed on contracts which accept data and use it in a sub-call on another contract.	PASS
Arbitrary Jump Function	SWC-127	As Solidity doesnt support pointer arithmetics, it is impossible to change such variable to an arbitrary value.	PASS

Typographical Error	SWC-129	A typographical error can occur for example when the intent of a defined operation is to sum a number to a variable.	PASS
Override control character	SWC-130	Malicious actors can use the Right-To-Left-Override unicode character to force RTL text rendering and confuse users as to the real intent of a contract.	PASS
Unused variables	SWC-131 SWC-135	Unused variables are allowed in Solidity and they do not pose a direct security issue.	PASS
Unexpected Ether balance	SWC-132	Contracts can behave erroneously when they strictly assume a specific Ether balance.	PASS
Hash Collisions Variable	SWC-133	Using abi.encodePacked() with multiple variable length arguments can, in certain situations, lead to a hash collision.	PASS
Hardcoded gas amount	SWC-134	The transfer() and send() functions forward a fixed amount of 2300 gas.	PASS
Unencrypted Private Data	SWC-136	It is a common misconception that private type variables cannot be read.	PASS

SMART CONTRACT ANALYSIS

Started	Wednesday May 04 2022 10:55:46 GMT+0000 (Coordinated Universal Time)
Finished	Thursday May 05 2022 17:12:28 GMT+0000 (Coordinated Universal Time)
Mode	Standard
Main Source File	SafeHavenToken.sol

Detected Issues

ID	Title	Severity	Status
SWC-103	A FLOATING PRAGMA IS SET.	low	acknowledged
SWC-108	STATE VARIABLE VISIBILITY IS NOT SET.	low	acknowledged
SWC-108	STATE VARIABLE VISIBILITY IS NOT SET.	low	acknowledged
SWC-108	STATE VARIABLE VISIBILITY IS NOT SET.	low	acknowledged
SWC-111	USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED.	low	acknowledged
SWC-111	USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED.	low	acknowledged
SWC-111	USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED.	low	acknowledged
SWC-111	USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED.	low	acknowledged
SWC-111	USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED.	low	acknowledged

SWC-111	USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED.	low	acknowledged
SWC-111	USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED.	low	acknowledged
SWC-120	POTENTIAL USE OF "BLOCK.NUMBER" AS SOURCE OF RANDOMNESS.	low	acknowledged
SWC-120	POTENTIAL USE OF "BLOCK.NUMBER" AS SOURCE OF RANDOMNESS.	low	acknowledged
SWC-120	POTENTIAL USE OF "BLOCK.NUMBER" AS SOURCE OF RANDOMNESS.	low	acknowledged
SWC-120	POTENTIAL USE OF "BLOCK.NUMBER" AS SOURCE OF RANDOMNESS.	low	acknowledged
SWC-120	POTENTIAL USE OF "BLOCK.NUMBER" AS SOURCE OF RANDOMNESS.	low	acknowledged
SWC-120	POTENTIAL USE OF "BLOCK.NUMBER" AS SOURCE OF RANDOMNESS.	low	acknowledged
SWC-120	POTENTIAL USE OF "BLOCK.NUMBER" AS SOURCE OF RANDOMNESS.	low	acknowledged
SWC-120	POTENTIAL USE OF "BLOCK.NUMBER" AS SOURCE OF RANDOMNESS.	low	acknowledged
SWC-120	POTENTIAL USE OF "BLOCK.NUMBER" AS SOURCE OF RANDOMNESS.	low	acknowledged
SWC-123	REQUIREMENT VIOLATION.	low	acknowledged
SWC-123	REQUIREMENT VIOLATION.	low	acknowledged
SWC-123	REQUIREMENT VIOLATION.	low	acknowledged

SWC-103 | A FLOATING PRAGMA IS SET.

LINE 5

low SEVERITY

The current pragma Solidity directive is ""^0.4.24"". 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.

Source File

- SafeHavenToken.sol

Locations

```
4
5   pragma solidity ^0.4.24;
6
7   // Safe Haven Token Sale
8   //
9
```

SWC-108 | STATE VARIABLE VISIBILITY IS NOT SET.

LINE 81

low SEVERITY

It is best practice to set the visibility of state variables explicitly. The default visibility for "balances" is internal. Other possible visibility settings are public and private.

Source File

- SafeHavenToken.sol

Locations

```
80   //  occurred is also included in the map
81   mapping (address => Checkpoint[]) balances;
82
83   // `allowed` tracks any extra transfer rights as in all ERC20 tokens
84   mapping (address => mapping (address => uint256)) allowed;
85
```

SWC-108 | STATE VARIABLE VISIBILITY IS NOT SET.

LINE 84

low SEVERITY

It is best practice to set the visibility of state variables explicitly. The default visibility for "allowed" is internal. Other possible visibility settings are public and private.

Source File

- SafeHavenToken.sol

Locations

```
83   // `allowed` tracks any extra transfer rights as in all ERC20 tokens
84   mapping (address => mapping (address => uint256)) allowed;
85
86   // Tracks the history of the `totalSupply` of the token
87   Checkpoint[] totalSupplyHistory;
88
```

SWC-108 | STATE VARIABLE VISIBILITY IS NOT SET.

LINE 87

low SEVERITY

It is best practice to set the visibility of state variables explicitly. The default visibility for "totalSupplyHistory" is internal. Other possible visibility settings are public and private.

Source File

- SafeHavenToken.sol

Locations

```
86   // Tracks the history of the `totalSupply` of the token
87   Checkpoint[] totalSupplyHistory;
88
89   // Flag that determines if the token is transferable or not.
90   bool public transfersEnabled;
91
```

SWC-111 | USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED.

LINE 220

low SEVERITY

Using "constant" as a state mutability modifier in function "balanceOf" is disallowed as of Solidity version 0.5.0. Use "view" instead.

Source File

- SafeHavenToken.sol

Locations

```
219   /// @return The balance of `_owner` at the current block
220   function balanceOf(address _owner) public constant returns (uint256 balance) {
221   return balanceOfAt(_owner, block.number);
222   }
223
224
```

SWC-111 | USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED.

LINE 256

low SEVERITY

Using "constant" as a state mutability modifier in function "allowance" is disallowed as of Solidity version 0.5.0. Use "view" instead.

Source File

- SafeHavenToken.sol

Locations

```
255   ///  to spend
256   function allowance(address _owner, address _spender
257   ) public constant returns (uint256 remaining)
258   {
259   return allowed[_owner][_spender];
260
```

SWC-111 | USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED.

LINE 286

low SEVERITY

Using "constant" as a state mutability modifier in function "totalSupply" is disallowed as of Solidity version 0.5.0. Use "view" instead.

Source File

- SafeHavenToken.sol

Locations

```
285   /// @return The total number of tokens
286   function totalSupply() public constant returns (uint) {
287   return totalSupplyAt(block.number);
288   }
289
290
```

SWC-111 | USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED.

LINE 299

low SEVERITY

Using "constant" as a state mutability modifier in function "balanceOfAt" is disallowed as of Solidity version 0.5.0. Use "view" instead.

Source File

- SafeHavenToken.sol

Locations

```
298   /// @return The balance at `_blockNumber`
299   function balanceOfAt(address _owner, uint _blockNumber) public constant
300   returns (uint)
301   {
302   // These next few lines are used when the balance of the token is
303
```

SWC-111 | USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED.

LINE 324

low SEVERITY

Using "constant" as a state mutability modifier in function "totalSupplyAt" is disallowed as of Solidity version 0.5.0. Use "view" instead.

Source File

- SafeHavenToken.sol

Locations

```
323   /// @return The total amount of tokens at `_blockNumber`
324   function totalSupplyAt(uint _blockNumber) public constant returns(uint) {
325
326   // These next few lines are used when the totalSupply of the token is
327   //  requested before a check point was ever created for this token, it
328
```

SWC-111 | USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED.

LINE 444

low SEVERITY

Using "constant" as a state mutability modifier in function "getValueAt" is disallowed as of Solidity version 0.5.0. Use "view" instead.

Source File

- SafeHavenToken.sol

Locations

```
443   /// @return The number of tokens being queried
444   function getValueAt(Checkpoint[] storage checkpoints, uint _block)
445   constant internal returns (uint)
446   {
447   if (checkpoints.length == 0) {
448
```

SWC-111 | USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED.

LINE 494

low SEVERITY

Using "constant" as a state mutability modifier in function "isContract" is disallowed as of Solidity version 0.5.0. Use "view" instead.

Source File

- SafeHavenToken.sol

Locations

```
493   /// @return True if `_addr` is a contract
494   function isContract(address _addr) constant internal returns(bool) {
495   uint size;
496   if (_addr == 0) {
497   return false;
498
```

SWC-120 | POTENTIAL USE OF "BLOCK.NUMBER" AS SOURCE OF RANDOMNESS.

LINE 129

low SEVERITY

The environment variable "block.number" looks like it might be used as a source of randomness. 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. 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.

Source File

- SafeHavenToken.sol

Locations

```
128   transfersEnabled = _transfersEnabled;
129   creationBlock = block.number;
130   }
131
132
133
```

SWC-120 | POTENTIAL USE OF "BLOCK.NUMBER" AS SOURCE OF RANDOMNESS.

LINE 185

low SEVERITY

The environment variable "block.number" looks like it might be used as a source of randomness. 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. 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.

Source File

- SafeHavenToken.sol

Locations

```
184
185   require(parentSnapShotBlock < block.number);
186
187   // Do not allow transfer to 0x0 or the token contract itself
188   require((_to != 0) && (_to != address(this)));
189
```

SWC-120 | POTENTIAL USE OF "BLOCK.NUMBER" AS SOURCE OF RANDOMNESS.

LINE 192

low SEVERITY

The environment variable "block.number" looks like it might be used as a source of randomness. 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. 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.

Source File

- SafeHavenToken.sol

Locations

```
191   //  account the transfer returns false
192   uint256 previousBalanceFrom = balanceOfAt(_from, block.number);
193   if (previousBalanceFrom < _amount) {
194   return false;
195   }
196
```

SWC-120 | POTENTIAL USE OF "BLOCK.NUMBER" AS SOURCE OF RANDOMNESS.

LINE 208

low SEVERITY

The environment variable "block.number" looks like it might be used as a source of randomness. 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. 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.

Source File

- SafeHavenToken.sol

Locations

```
207   //  receiving the tokens
208   uint256 previousBalanceTo = balanceOfAt(_to, block.number);
209   require(previousBalanceTo + _amount >= previousBalanceTo); // Check for overflow
210   updateValueAtNow(balances[_to], previousBalanceTo + _amount);
211
212
```

SWC-120 | POTENTIAL USE OF "BLOCK.NUMBER" AS SOURCE OF RANDOMNESS.

LINE 221

low SEVERITY

The environment variable "block.number" looks like it might be used as a source of randomness. 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. 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.

Source File

- SafeHavenToken.sol

Locations

```
220   function balanceOf(address _owner) public constant returns (uint256 balance) {
221   return balanceOfAt(_owner, block.number);
222   }
223
224   /// @notice `msg.sender` approves `_spender` to spend `_amount` tokens on
225
```

SWC-120 | POTENTIAL USE OF "BLOCK.NUMBER" AS SOURCE OF RANDOMNESS.

LINE 287

low SEVERITY

The environment variable "block.number" looks like it might be used as a source of randomness. 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. 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.

Source File

- SafeHavenToken.sol

Locations

```
286   function totalSupply() public constant returns (uint) {
287   return totalSupplyAt(block.number);
288   }
289
290
291
```

SWC-120 | POTENTIAL USE OF "BLOCK.NUMBER" AS SOURCE OF RANDOMNESS.

LINE 367

low SEVERITY

The environment variable "block.number" looks like it might be used as a source of randomness. 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. 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.

Source File

- SafeHavenToken.sol

Locations

```
366   if (_snapshotBlock == 0) {
367   _snapshotBlock = block.number;
368   }
369
370   MiniMeToken cloneToken = tokenFactory.createCloneToken(
371
```

SWC-120 | POTENTIAL USE OF "BLOCK.NUMBER" AS SOURCE OF RANDOMNESS.

LINE 481

low SEVERITY

The environment variable "block.number" looks like it might be used as a source of randomness. 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. 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.

Source File

- SafeHavenToken.sol

Locations

```
480   {
481   if ((checkpoints.length == 0) || (checkpoints[checkpoints.length-1].fromBlock <
block.number)) {
482   Checkpoint storage newCheckPoint = checkpoints[checkpoints.length++];
483   newCheckPoint.fromBlock = uint128(block.number);
484   newCheckPoint.value = uint128(_value);
485
```

SWC-120 | POTENTIAL USE OF "BLOCK.NUMBER" AS SOURCE OF RANDOMNESS.

LINE 483

low SEVERITY

The environment variable "block.number" looks like it might be used as a source of randomness. 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. 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.

Source File

- SafeHavenToken.sol

Locations

```
482   Checkpoint storage newCheckPoint = checkpoints[checkpoints.length++];
483   newCheckPoint.fromBlock = uint128(block.number);
484   newCheckPoint.value = uint128(_value);
485   } else {
486   Checkpoint storage oldCheckPoint = checkpoints[checkpoints.length-1];
487
```

SWC-123 | REQUIREMENT VIOLATION.

LINE 534

low SEVERITY

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

Source File

- SafeHavenToken.sol

Locations

```
533   uint balance = token.balanceOf(this);
534   token.transfer(controller, balance);
535   emit ClaimedTokens(_token, controller, balance);
536   }
537
538
```

SWC-123 | REQUIREMENT VIOLATION.

LINE 515

low SEVERITY

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

Source File

- SafeHavenToken.sol

Locations

```
514   require(isContract(controller));
515   require(TokenController(controller).proxyPayment.value(msg.value)(msg.sender));
516   }
517
518   //////////
519
```

SWC-123 | REQUIREMENT VIOLATION.

LINE 274

low SEVERITY

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

Source File

- SafeHavenToken.sol

Locations

```
273
274   ApproveAndCallFallBack(_spender).receiveApproval(
275   msg.sender,
276   _amount,
277   this,
278
```

DISCLAIMER

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to, or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without Sysfixed's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Sysfixed to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model, or legal compliance.

This is a limited report on our findings based on our analysis, in accordance with good industry practice as of the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the below disclaimer below – please make sure to read it in full.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and Sysfixed and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (Sysfixed) owe no duty of care.

ABOUT US

Sysfixed is a blockchain security certification organization established in 2021 with the objective to provide smart contract security services and verify their correctness in blockchain-based protocols. Sysfixed automatically scans for security vulnerabilities in Ethereum and other EVM-based blockchain smart contracts. Sysfixed a comprehensive range of analysis techniques—including static analysis, dynamic analysis, and symbolic execution—can accurately detect security vulnerabilities to provide an in-depth analysis report. With a vibrant ecosystem of world-class integration partners that amplify developer productivity, Sysfixed can be utilized in all phases of your project's lifecycle. Our team of security experts is dedicated to the research and improvement of our tools and techniques used to fortify your code.