AlpacaToken

Smart Contract Audit Report

SYSFIXED

27 Feb 2021

TABLE OF CONTENTS

Audited Details

- Audited Project
- Blockchain
- Addresses
- Project Website
- Codebase

Summary

- Contract Summary
- Audit Findings Summary
- Vulnerabilities Summary

Conclusion

Audit Results

Smart Contract Analysis

- Detected Vulnerabilities

Disclaimer

About Us

AUDITED DETAILS

Audited Project

Project name	Token ticker	Blockchain
AlpacaToken	ALPACA	Binance Smart Chain

Addresses

Contract address	0x8f0528ce5ef7b51152a59745befdd91d97091d2f
Contract deployer address	0xC44f82b07Ab3E691F826951a6E335E1bC1bB0B51

Project Website

https://www.alpacafinance.org/

Codebase

https://bscscan.com/address/0x8f0528ce5ef7b51152a59745befdd91d97091d2f#code

SUMMARY

Alpaca Finance is the largest lending protocol allowing leveraged yield farming on BNB Chain. It helps lenders earn safe and stable yields, and offers borrowers undercollateralized loans for leveraged yield farming positions, vastly multiplying their farming principals and resulting profits. As an enabler for the entire DeFi ecosystem, Alpaca amplifies the liquidity layer of integrated exchanges, improving their capital efficiency by connecting LP borrowers and lenders. It's through this empowering function that Alpaca has become a fundamental building block within DeFi, helping bring the power of finance to each and every person's fingertips, and every alpaca's paw... Furthermore, alpacas are a virtuous breed. That's why, we are a fair-launch project with no pre-sale, no investor, and no pre-mine. So from the beginning, this has always been a product built by the people, for the people.

Contract Summary

Documentation Quality

AlpacaToken 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 AlpacaToken with the discovery of several low issues.

Test Coverage

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

Audit Findings Summary

- SWC-101 | It is recommended to use vetted safe math libraries for arithmetic operations consistently on lines 103, 115, 128, 129, 140, 150, 164, 181, 196, 197, 215, 232, 250, 270, 290, 1134, 1146, 1165, 1166, 1175, 1177, 1177, 1177, 1184, 1209, 1217, 1232, 1233, 1236, 1243, 1477, 1519, 1146, 1165, 1166, 1175, 1184, 1209, 1217, 1232 and 1233.
- 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 1331, 1459, 1461, 1477, 1505, 1522, 1554, 1578, 1595, 1601 and 1601.

- SWC-120 | It is recommended to use external sources of randomness via oracles on lines 1005, 1009, 1016, 1029, 1157, 1230, 1420, 1441, 1442, 1508, 1511, 1522, 1526, 1527, 1534, 1540 and 1551.

CONCLUSION

We have audited the AlpacaToken project released on February 2021 to discover issues and identify potential security vulnerabilities in AlpacaToken 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 a satisfactory result with some low-risk issues.

The issues found in the AlpacaToken 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 some arithmetic operation issues, public state variables with array type causing reachable exception by default, the potential use of "block.number" as a source of randomness, and out-of-bounds array access which the index access expression can cause an exception in case of the use of an invalid array index value. 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 the use of these variables introduces a certain level of trust into miners.

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.	PASS
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	ISSUE FOUND
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.	PASS
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.	PASS
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	Friday Feb 26 2021 18:11:56 GMT+0000 (Coordinated Universal Time)
Finished	Saturday Feb 27 2021 04:36:18 GMT+0000 (Coordinated Universal Time)
Mode	Standard
Main Source File	AlpacaToken.sol

Detected Issues

ID	Title	Severity	Status
SWC-101	ARITHMETIC OPERATION "+" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "-" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "*" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "/" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "/" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "%" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "+" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "-" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "*" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "/" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "/" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "%" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "-" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "/" DISCOVERED	low	acknowledged

SWC-101	ARITHMETIC OPERATION "%" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "++" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "-" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "-" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "-" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "-" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "-" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "/" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "-" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "-" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "-" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "-" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "-" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "-" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "-" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "+" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "**" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "++" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "++" DISCOVERED	low	acknowledged
SWC-101	COMPILER-REWRITABLE "<UINT> - 1" DISCOVERED	low	acknowledged

SWC-101	COMPILER-REWRITABLE "<UINT> - 1" DISCOVERED	low	acknowledged
SWC-101	COMPILER-REWRITABLE "<UINT> - 1" DISCOVERED	low	acknowledged
SWC-101	COMPILER-REWRITABLE "<UINT> - 1" DISCOVERED	low	acknowledged
SWC-101	COMPILER-REWRITABLE "<UINT> - 1" DISCOVERED	low	acknowledged
SWC-101	COMPILER-REWRITABLE "<UINT> - 1" DISCOVERED	low	acknowledged
SWC-101	COMPILER-REWRITABLE "<UINT> - 1" DISCOVERED	low	acknowledged
SWC-101	COMPILER-REWRITABLE "<UINT> - 1" DISCOVERED	low	acknowledged
SWC-101	COMPILER-REWRITABLE "<UINT> - 1" DISCOVERED	low	acknowledged
SWC-110	PUBLIC STATE VARIABLE WITH ARRAY TYPE CAUSING REACHABLE EXCEPTION BY DEFAULT.	low	acknowledged
SWC-110	OUT OF BOUNDS ARRAY ACCESS	low	acknowledged
SWC-110	OUT OF BOUNDS ARRAY ACCESS	low	acknowledged
SWC-110	OUT OF BOUNDS ARRAY ACCESS	low	acknowledged
SWC-110	OUT OF BOUNDS ARRAY ACCESS	low	acknowledged
SWC-110	OUT OF BOUNDS ARRAY ACCESS	low	acknowledged
SWC-110	OUT OF BOUNDS ARRAY ACCESS	low	acknowledged
SWC-110	OUT OF BOUNDS ARRAY ACCESS	low	acknowledged
SWC-110	OUT OF BOUNDS ARRAY ACCESS	low	acknowledged
SWC-110	OUT OF BOUNDS ARRAY ACCESS	low	acknowledged

SWC-110	OUT OF BOUNDS ARRAY ACCESS	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-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-101 | ARITHMETIC OPERATION "+" DISCOVERED

LINE 103

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- AlpacaToken.sol

Locations

```
102   function tryAdd(uint256 a, uint256 b) internal pure returns (bool, uint256) {
103   uint256 c = a + b;
104   if (c < a) return (false, 0);
105   return (true, c);
106   }
107
```

SWC-101 | ARITHMETIC OPERATION "-" DISCOVERED

LINE 115

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- AlpacaToken.sol

Locations

```
114   if (b > a) return (false, 0);
115   return (true, a - b);
116   }
117
118   /**
119
```

SWC-101 | ARITHMETIC OPERATION "*" DISCOVERED

LINE 128

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- AlpacaToken.sol

Locations

```
127   if (a == 0) return (true, 0);
128   uint256 c = a * b;
129   if (c / a != b) return (false, 0);
130   return (true, c);
131   }
132
```

SWC-101 | ARITHMETIC OPERATION "/" DISCOVERED

LINE 129

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- AlpacaToken.sol

Locations

```
128   uint256 c = a * b;
129   if (c / a != b) return (false, 0);
130   return (true, c);
131   }
132
133
```

SWC-101 | ARITHMETIC OPERATION "/" DISCOVERED

LINE 140

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- AlpacaToken.sol

Locations

```
139   if (b == 0) return (false, 0);
140   return (true, a / b);
141   }
142
143   /**
144
```

SWC-101 | ARITHMETIC OPERATION "%" DISCOVERED

LINE 150

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- AlpacaToken.sol

Locations

```
149   if (b == 0) return (false, 0);
150   return (true, a % b);
151   }
152
153   /**
154
```

SWC-101 | ARITHMETIC OPERATION "+" DISCOVERED

LINE 164

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- AlpacaToken.sol

Locations

```
163   function add(uint256 a, uint256 b) internal pure returns (uint256) {
164   uint256 c = a + b;
165   require(c >= a, "SafeMath: addition overflow");
166   return c;
167   }
168
```

SWC-101 | ARITHMETIC OPERATION "-" DISCOVERED

LINE 181

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- AlpacaToken.sol

Locations

```
180   require(b <= a, "SafeMath: subtraction overflow");
181   return a - b;
182   }
183
184   /**
185
```

SWC-101 | ARITHMETIC OPERATION "*" DISCOVERED

LINE 196

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- AlpacaToken.sol

Locations

```
195   if (a == 0) return 0;
196   uint256 c = a * b;
197   require(c / a == b, "SafeMath: multiplication overflow");
198   return c;
199   }
200
```

SWC-101 | ARITHMETIC OPERATION "/" DISCOVERED

LINE 197

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- AlpacaToken.sol

Locations

```
196   uint256 c = a * b;
197   require(c / a == b, "SafeMath: multiplication overflow");
198   return c;
199   }
200
201
```

SWC-101 | ARITHMETIC OPERATION "/" DISCOVERED

LINE 215

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- AlpacaToken.sol

Locations

```
214   require(b > 0, "SafeMath: division by zero");
215   return a / b;
216   }
217
218   /**
219
```

SWC-101 | ARITHMETIC OPERATION "%" DISCOVERED

LINE 232

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- AlpacaToken.sol

Locations

```
231   require(b > 0, "SafeMath: modulo by zero");
232   return a % b;
233   }
234
235   /**
236
```

SWC-101 | ARITHMETIC OPERATION "-" DISCOVERED

LINE 250

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- AlpacaToken.sol

Locations

```
249   require(b <= a, errorMessage);
250   return a - b;
251   }
252
253   /**
254
```

SWC-101 | ARITHMETIC OPERATION "/" DISCOVERED

LINE 270

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- AlpacaToken.sol

Locations

```
269   require(b > 0, errorMessage);
270   return a / b;
271   }
272
273   /**
274
```

SWC-101 | ARITHMETIC OPERATION "%" DISCOVERED

LINE 290

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- AlpacaToken.sol

Locations

```
289   require(b > 0, errorMessage);
290   return a % b;
291   }
292   }
293
294
```

SWC-101 | ARITHMETIC OPERATION "++" DISCOVERED

LINE 1134

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- AlpacaToken.sol

Locations

```
1133   require(signatory != address(0), "ALPACA::delegateBySig: invalid signature");
1134   require(nonce == nonces[signatory]++, "ALPACA::delegateBySig: invalid nonce");
1135   require(now <= expiry, "ALPACA::delegateBySig: signature expired");
1136   return _delegate(signatory, delegatee);
1137   }
1138
```

SWC-101 | ARITHMETIC OPERATION "-" DISCOVERED

LINE 1146

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- AlpacaToken.sol

Locations

```
1145   uint32 nCheckpoints = numCheckpoints[account];
1146   return nCheckpoints > 0 ? checkpoints[account][nCheckpoints - 1].votes : 0;
1147   }
1148
1149   /**
1150
```

SWC-101 | ARITHMETIC OPERATION "-" DISCOVERED

LINE 1165

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- AlpacaToken.sol

Locations

```
1164   // First check most recent balance
1165   if (checkpoints[account][nCheckpoints - 1].fromBlock <= blockNumber) {
1166   return checkpoints[account][nCheckpoints - 1].votes;
1167   }
1168
1169
```

SWC-101 | ARITHMETIC OPERATION "-" DISCOVERED

LINE 1166

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- AlpacaToken.sol

Locations

```
1165   if (checkpoints[account][nCheckpoints - 1].fromBlock <= blockNumber) {
1166   return checkpoints[account][nCheckpoints - 1].votes;
1167   }
1168
1169   // Next check implicit zero balance
1170
```

SWC-101 | ARITHMETIC OPERATION "-" DISCOVERED

LINE 1175

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- AlpacaToken.sol

Locations

```
1174   uint32 lower = 0;
1175   uint32 upper = nCheckpoints - 1;
1176   while (upper > lower) {
1177   uint32 center = upper - (upper - lower) / 2; // ceil, avoiding overflow
1178   Checkpoint memory cp = checkpoints[account][center];
1179
```

SWC-101 | ARITHMETIC OPERATION "-" DISCOVERED

LINE 1177

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- AlpacaToken.sol

Locations

```
1176   while (upper > lower) {
1177   uint32 center = upper - (upper - lower) / 2; // ceil, avoiding overflow
1178   Checkpoint memory cp = checkpoints[account][center];
1179   if (cp.fromBlock == blockNumber) {
1180   return cp.votes;
1181
```

SWC-101 | ARITHMETIC OPERATION "/" DISCOVERED

LINE 1177

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- AlpacaToken.sol

Locations

```
1176   while (upper > lower) {
1177   uint32 center = upper - (upper - lower) / 2; // ceil, avoiding overflow
1178   Checkpoint memory cp = checkpoints[account][center];
1179   if (cp.fromBlock == blockNumber) {
1180   return cp.votes;
1181
```

SWC-101 | ARITHMETIC OPERATION "-" DISCOVERED

LINE 1177

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- AlpacaToken.sol

Locations

```
1176   while (upper > lower) {
1177   uint32 center = upper - (upper - lower) / 2; // ceil, avoiding overflow
1178   Checkpoint memory cp = checkpoints[account][center];
1179   if (cp.fromBlock == blockNumber) {
1180   return cp.votes;
1181
```

SWC-101 | ARITHMETIC OPERATION "-" DISCOVERED

LINE 1184

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- AlpacaToken.sol

Locations

```
1183   } else {
1184   upper = center - 1;
1185   }
1186   }
1187   return checkpoints[account][lower].votes;
1188
```

SWC-101 | ARITHMETIC OPERATION "-" DISCOVERED

LINE 1209

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- AlpacaToken.sol

Locations

```
1208   uint32 srcRepNum = numCheckpoints[srcRep];
1209   uint256 srcRepOld = srcRepNum > 0 ? checkpoints[srcRep][srcRepNum - 1].votes : 0;
1210   uint256 srcRepNew = srcRepOld.sub(amount);
1211   _writeCheckpoint(srcRep, srcRepNum, srcRepOld, srcRepNew);
1212   }
1213
```

SWC-101 | ARITHMETIC OPERATION "-" DISCOVERED

LINE 1217

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- AlpacaToken.sol

Locations

```
1216   uint32 dstRepNum = numCheckpoints[dstRep];
1217   uint256 dstRepOld = dstRepNum > 0 ? checkpoints[dstRep][dstRepNum - 1].votes : 0;
1218   uint256 dstRepNew = dstRepOld.add(amount);
1219   _writeCheckpoint(dstRep, dstRepNum, dstRepOld, dstRepNew);
1220   }
1221
```

SWC-101 | ARITHMETIC OPERATION "-" DISCOVERED

LINE 1232

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- AlpacaToken.sol

Locations

```
1231
1232   if (nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock ==
blockNumber) {
1233   checkpoints[delegatee][nCheckpoints - 1].votes = newVotes;
1234   } else {
1235   checkpoints[delegatee][nCheckpoints] = Checkpoint(blockNumber, newVotes);
1236
```

SWC-101 | ARITHMETIC OPERATION "-" DISCOVERED

LINE 1233

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- AlpacaToken.sol

Locations

```
1232   if (nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock ==
blockNumber) {
1233   checkpoints[delegatee][nCheckpoints - 1].votes = newVotes;
1234   } else {
1235   checkpoints[delegatee][nCheckpoints] = Checkpoint(blockNumber, newVotes);
1236   numCheckpoints[delegatee] = nCheckpoints + 1;
1237
```

SWC-101 | ARITHMETIC OPERATION "+" DISCOVERED

LINE 1236

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- AlpacaToken.sol

Locations

```
1235   checkpoints[delegatee][nCheckpoints] = Checkpoint(blockNumber, newVotes);
1236   numCheckpoints[delegatee] = nCheckpoints + 1;
1237   }
1238
1239   emit DelegateVotesChanged(delegatee, oldVotes, newVotes);
1240
```

SWC-101 | ARITHMETIC OPERATION "**" DISCOVERED

LINE 1243

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- AlpacaToken.sol

Locations

```
1242   function safe32(uint256 n, string memory errorMessage) internal pure returns
(uint32) {
1243   require(n < 2**32, errorMessage);
1244   return uint32(n);
1245   }
1246
1247
```

SWC-101 | ARITHMETIC OPERATION "++" DISCOVERED

LINE 1477

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- AlpacaToken.sol

Locations

```
1476   uint256 alpacaReward =
multiplier.mul(alpacaPerBlock).mul(pool.allocPoint).div(totalAllocPoint);
1477   accAlpacaPerShare = accAlpacaPerShare.add(alpacaReward.mul(1e12).div(lpSupply));
1478   }
1479   return user.amount.mul(accAlpacaPerShare).div(1e12).sub(user.rewardDebt);
1480   }
1481
```

SWC-101 | ARITHMETIC OPERATION "++" DISCOVERED

LINE 1519

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- AlpacaToken.sol

Locations

```
1518
1519   // Deposit Staking tokens to FairLaunchToken for ALPACA allocation.
1520   function deposit(address _for, uint256 _pid, uint256 _amount) public override {
1521   PoolInfo storage pool = poolInfo[_pid];
1522   UserInfo storage user = userInfo[_pid][_for];
1523
```

SWC-101 | COMPILER-REWRITABLE "<UINT> - 1" DISCOVERED

LINE 1146

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- AlpacaToken.sol

Locations

```
1145   uint32 nCheckpoints = numCheckpoints[account];
1146   return nCheckpoints > 0 ? checkpoints[account][nCheckpoints - 1].votes : 0;
1147   }
1148
1149   /**
1150
```

SWC-101 | COMPILER-REWRITABLE "<UINT> - 1" DISCOVERED

LINE 1165

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- AlpacaToken.sol

Locations

```
1164   // First check most recent balance
1165   if (checkpoints[account][nCheckpoints - 1].fromBlock <= blockNumber) {
1166   return checkpoints[account][nCheckpoints - 1].votes;
1167   }
1168
1169
```

SWC-101 | COMPILER-REWRITABLE "<UINT> - 1" DISCOVERED

LINE 1166

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- AlpacaToken.sol

Locations

```
1165   if (checkpoints[account][nCheckpoints - 1].fromBlock <= blockNumber) {
1166   return checkpoints[account][nCheckpoints - 1].votes;
1167   }
1168
1169   // Next check implicit zero balance
1170
```

SWC-101 | COMPILER-REWRITABLE "<UINT> - 1" DISCOVERED

LINE 1175

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- AlpacaToken.sol

Locations

```
1174   uint32 lower = 0;
1175   uint32 upper = nCheckpoints - 1;
1176   while (upper > lower) {
1177   uint32 center = upper - (upper - lower) / 2; // ceil, avoiding overflow
1178   Checkpoint memory cp = checkpoints[account][center];
1179
```

SWC-101 | COMPILER-REWRITABLE "<UINT> - 1" DISCOVERED

LINE 1184

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- AlpacaToken.sol

Locations

```
1183   } else {
1184   upper = center - 1;
1185   }
1186   }
1187   return checkpoints[account][lower].votes;
1188
```

SWC-101 | COMPILER-REWRITABLE "<UINT> - 1" DISCOVERED

LINE 1209

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- AlpacaToken.sol

Locations

```
1208   uint32 srcRepNum = numCheckpoints[srcRep];
1209   uint256 srcRepOld = srcRepNum > 0 ? checkpoints[srcRep][srcRepNum - 1].votes : 0;
1210   uint256 srcRepNew = srcRepOld.sub(amount);
1211   _writeCheckpoint(srcRep, srcRepNum, srcRepOld, srcRepNew);
1212   }
1213
```

SWC-101 | COMPILER-REWRITABLE "<UINT> - 1" DISCOVERED

LINE 1217

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- AlpacaToken.sol

Locations

```
1216   uint32 dstRepNum = numCheckpoints[dstRep];
1217   uint256 dstRepOld = dstRepNum > 0 ? checkpoints[dstRep][dstRepNum - 1].votes : 0;
1218   uint256 dstRepNew = dstRepOld.add(amount);
1219   _writeCheckpoint(dstRep, dstRepNum, dstRepOld, dstRepNew);
1220   }
1221
```

SWC-101 | COMPILER-REWRITABLE "<UINT> - 1" DISCOVERED

LINE 1232

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- AlpacaToken.sol

Locations

```
1231
1232   if (nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock ==
blockNumber) {
1233   checkpoints[delegatee][nCheckpoints - 1].votes = newVotes;
1234   } else {
1235   checkpoints[delegatee][nCheckpoints] = Checkpoint(blockNumber, newVotes);
1236
```

SWC-101 | COMPILER-REWRITABLE "<UINT> - 1" DISCOVERED

LINE 1233

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- AlpacaToken.sol

Locations

```
1232   if (nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock ==
blockNumber) {
1233   checkpoints[delegatee][nCheckpoints - 1].votes = newVotes;
1234   } else {
1235   checkpoints[delegatee][nCheckpoints] = Checkpoint(blockNumber, newVotes);
1236   numCheckpoints[delegatee] = nCheckpoints + 1;
1237
```

SWC-110 | PUBLIC STATE VARIABLE WITH ARRAY TYPE CAUSING REACHABLE EXCEPTION BY DEFAULT.

LINE 1331

low SEVERITY

The public state variable "poolInfo" in "FairLaunch" contract has type "struct FairLaunch.PoolInfo[]" and can cause an exception in case of use of invalid array index value.

Source File

- AlpacaToken.sol

Locations

```
1330   // Info of each pool.
1331   PoolInfo[] public poolInfo;
1332   // Info of each user that stakes Staking tokens.
1333   mapping(uint256 => mapping(address => UserInfo)) public userInfo;
1334   // Total allocation poitns. Must be the sum of all allocation points in all pools.
1335
```

SWC-110 | OUT OF BOUNDS ARRAY ACCESS

LINE 1459

low SEVERITY

The index access expression can cause an exception in case of use of invalid array index value.

Source File

- AlpacaToken.sol

Locations

```
1458   if (_currentBlock <= bonusEndBlock) {
1459   return _currentBlock.sub(_lastRewardBlock).mul(bonusMultiplier);
1460   }
1461   if (_lastRewardBlock >= bonusEndBlock) {
1462   return _currentBlock.sub(_lastRewardBlock);
1463
```

SWC-110 | OUT OF BOUNDS ARRAY ACCESS

LINE 1461

low SEVERITY

The index access expression can cause an exception in case of use of invalid array index value.

Source File

- AlpacaToken.sol

Locations

```
1460   }
1461   if (_lastRewardBlock >= bonusEndBlock) {
1462   return _currentBlock.sub(_lastRewardBlock);
1463   }
1464   // This is the case where bonusEndBlock is in the middle of _lastRewardBlock and
_currentBlock block.
1465
```

SWC-110 | OUT OF BOUNDS ARRAY ACCESS

LINE 1477

low SEVERITY

The index access expression can cause an exception in case of use of invalid array index value.

Source File

- AlpacaToken.sol

Locations

```
1476   uint256 alpacaReward =
multiplier.mul(alpacaPerBlock).mul(pool.allocPoint).div(totalAllocPoint);
1477   accAlpacaPerShare = accAlpacaPerShare.add(alpacaReward.mul(1e12).div(lpSupply));
1478   }
1479   return user.amount.mul(accAlpacaPerShare).div(1e12).sub(user.rewardDebt);
1480   }
1481
```

SWC-110 | OUT OF BOUNDS ARRAY ACCESS

LINE 1505

low SEVERITY

The index access expression can cause an exception in case of use of invalid array index value.

Source File

- AlpacaToken.sol

Locations

```
1504   alpaca.mint(address(this), alpacaReward);
1505   pool.accAlpacaPerShare =
pool.accAlpacaPerShare.add(alpacaReward.mul(1e12).div(lpSupply));
1506   // update accAlpacaPerShareTilBonusEnd
1507   if (block.number <= bonusEndBlock) {
1508   alpaca.lock(devaddr, alpacaReward.div(10).mul(bonusLockUpBps).div(10000));
1509
```

SWC-110 | OUT OF BOUNDS ARRAY ACCESS

LINE 1522

low SEVERITY

The index access expression can cause an exception in case of use of invalid array index value.

Source File

- AlpacaToken.sol

Locations

```
1521   PoolInfo storage pool = poolInfo[_pid];
1522   UserInfo storage user = userInfo[_pid][_for];
1523   if (user.fundedBy != address(0)) require(user.fundedBy == msg.sender, "bad sof");
1524   require(pool.stakeToken != address(0), "deposit: not accept deposit");
1525   updatePool(_pid);
1526
```

SWC-110 | OUT OF BOUNDS ARRAY ACCESS

LINE 1554

low SEVERITY

The index access expression can cause an exception in case of use of invalid array index value.

Source File

- AlpacaToken.sol

Locations

```
1553   user.bonusDebt = user.amount.mul(pool.accAlpacaPerShareTilBonusEnd).div(1e12);
1554   if (pool.stakeToken != address(0)) {
1555   IERC20(pool.stakeToken).safeTransfer(address(msg.sender), _amount);
1556   }
1557   emit Withdraw(msg.sender, _pid, user.amount);
1558
```

SWC-110 | OUT OF BOUNDS ARRAY ACCESS

LINE 1578

low SEVERITY

The index access expression can cause an exception in case of use of invalid array index value.

Source File

- AlpacaToken.sol

Locations

```
1577   safeAlpacaTransfer(_to, pending);
1578   alpaca.lock(_to, bonus.mul(bonusLockUpBps).div(10000));
1579   }
1580
1581   // Withdraw without caring about rewards. EMERGENCY ONLY.
1582
```

SWC-110 | OUT OF BOUNDS ARRAY ACCESS

LINE 1595

low SEVERITY

The index access expression can cause an exception in case of use of invalid array index value.

Source File

- AlpacaToken.sol

Locations

```
1594   if (_amount > alpacaBal) {
1595   alpaca.transfer(_to, alpacaBal);
1596   } else {
1597   alpaca.transfer(_to, _amount);
1598   }
1599
```

SWC-110 | OUT OF BOUNDS ARRAY ACCESS

LINE 1601

low SEVERITY

The index access expression can cause an exception in case of use of invalid array index value.

Source File

- AlpacaToken.sol

Locations

```
1600
1601   }
1602
```

SWC-110 | OUT OF BOUNDS ARRAY ACCESS

LINE 1601

low SEVERITY

The index access expression can cause an exception in case of use of invalid array index value.

Source File

- AlpacaToken.sol

Locations

```
1600
1601   }
1602
```

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

LINE 1005

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

- AlpacaToken.sol

Locations

```
1004   // When block number less than startReleaseBlock, no ALPACAs can be unlocked
1005   if (block.number < startReleaseBlock) {
1006   return 0;
1007   }
1008   // When block number more than endReleaseBlock, all locked ALPACAs can be unlocked
1009
```

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

LINE 1009

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

- AlpacaToken.sol

Locations

```
1008   // When block number more than endReleaseBlock, all locked ALPACAs can be unlocked
1009   else if (block.number >= endReleaseBlock) {
1010   return _locks[_account];
1011   }
1012   // When block number is more than startReleaseBlock but less than endReleaseBlock,
1013
```

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

LINE 1016

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

- AlpacaToken.sol

Locations

```
1015   {
1016   uint256 releasedBlock = block.number.sub(_lastUnlockBlock[_account]);
1017   uint256 blockLeft = endReleaseBlock.sub(_lastUnlockBlock[_account]);
1018   return _locks[_account].mul(releasedBlock).div(blockLeft);
1019   }
1020
```

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

LINE 1029

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

- AlpacaToken.sol

Locations

```
1028   _locks[msg.sender] = _locks[msg.sender].sub(amount);
1029   _lastUnlockBlock[msg.sender] = block.number;
1030   _totalLock = _totalLock.sub(amount);
1031   }
1032
1033
```

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

LINE 1157

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

- AlpacaToken.sol

Locations

```
1156   function getPriorVotes(address account, uint256 blockNumber) external view returns
(uint256) {
1157   require(blockNumber < block.number, "ALPACA::getPriorVotes: not yet determined");
1158
1159   uint32 nCheckpoints = numCheckpoints[account];
1160   if (nCheckpoints == 0) {
1161
```

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

LINE 1230

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

- AlpacaToken.sol

Locations

```
1229   ) internal {
1230   uint32 blockNumber = safe32(block.number, "ALPACA::_writeCheckpoint: block number
exceeds 32 bits");
1231
1232   if (nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock ==
blockNumber) {
1233   checkpoints[delegatee][nCheckpoints - 1].votes = newVotes;
1234
```

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

LINE 1420

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

- AlpacaToken.sol

Locations

```
1419   function setPool(
1420   uint256 _pid,
1421   uint256 _allocPoint,
1422   bool _withUpdate
1423   ) public override onlyOwner {
1424
```

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

LINE 1441

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

- AlpacaToken.sol

Locations

```
1440   function isDuplicatedPool(address _stakeToken) public view returns (bool) {
1441   uint256 length = poolInfo.length;
1442   for (uint256 _pid = 0; _pid < length; _pid++) {
1443   if(poolInfo[_pid].stakeToken == _stakeToken) return true;
1444   }
1445
```

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

LINE 1442

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

- AlpacaToken.sol

Locations

```
1441   uint256 length = poolInfo.length;
1442   for (uint256 _pid = 0; _pid < length; _pid++) {
1443   if(poolInfo[_pid].stakeToken == _stakeToken) return true;
1444   }
1445   return false;
1446
```

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

LINE 1508

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

- AlpacaToken.sol

Locations

```
1507   if (block.number <= bonusEndBlock) {
1508   alpaca.lock(devaddr, alpacaReward.div(10).mul(bonusLockUpBps).div(10000));
1509   pool.accAlpacaPerShareTilBonusEnd = pool.accAlpacaPerShare;
1510   }
1511   if(block.number > bonusEndBlock && pool.lastRewardBlock < bonusEndBlock) {
1512
```

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

LINE 1511

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

- AlpacaToken.sol

Locations

```
1510   }
1511   if(block.number > bonusEndBlock && pool.lastRewardBlock < bonusEndBlock) {
1512   uint256 alpacaBonusPortion =
bonusEndBlock.sub(pool.lastRewardBlock).mul(bonusMultiplier).mul(alpacaPerBlock).mul(pool
.allocPoint).div(totalAllocPoint);
1513   alpaca.lock(devaddr, alpacaBonusPortion.div(10).mul(bonusLockUpBps).div(10000));
1514   pool.accAlpacaPerShareTilBonusEnd =
pool.accAlpacaPerShareTilBonusEnd.add(alpacaBonusPortion.mul(1e12).div(lpSupply));
1515
```

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

LINE 1522

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

- AlpacaToken.sol

Locations

```
1521   PoolInfo storage pool = poolInfo[_pid];
1522   UserInfo storage user = userInfo[_pid][_for];
1523   if (user.fundedBy != address(0)) require(user.fundedBy == msg.sender, "bad sof");
1524   require(pool.stakeToken != address(0), "deposit: not accept deposit");
1525   updatePool(_pid);
1526
```

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

LINE 1526

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

- AlpacaToken.sol

Locations

```
1525   updatePool(_pid);
1526   if (user.amount > 0) _harvest(_for, _pid);
1527   if (user.fundedBy == address(0)) user.fundedBy = msg.sender;
1528   IERC20(pool.stakeToken).safeTransferFrom(address(msg.sender), address(this),
_amount);
1529   user.amount = user.amount.add(_amount);
1530
```

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

LINE 1527

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

- AlpacaToken.sol

Locations

```
1526   if (user.amount > 0) _harvest(_for, _pid);
1527   if (user.fundedBy == address(0)) user.fundedBy = msg.sender;
1528   IERC20(pool.stakeToken).safeTransferFrom(address(msg.sender), address(this),
_amount);
1529   user.amount = user.amount.add(_amount);
1530   user.rewardDebt = user.amount.mul(pool.accAlpacaPerShare).div(1e12);
1531
```

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

LINE 1534

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

- AlpacaToken.sol

Locations

```
1533   }
1534
1535   // Withdraw Staking tokens from FairLaunchToken.
1536   function withdraw(address _for, uint256 _pid, uint256 _amount) public override {
1537   _withdraw(_for, _pid, _amount);
1538
```

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

LINE 1540

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

- AlpacaToken.sol

Locations

```
1539
1540   function withdrawAll(address _for, uint256 _pid) public override {
1541   _withdraw(_for, _pid, userInfo[_pid][_for].amount);
1542   }
1543
1544
```

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

LINE 1551

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

- AlpacaToken.sol

Locations

```
1550   _harvest(_for, _pid);
1551   user.amount = user.amount.sub(_amount);
1552   user.rewardDebt = user.amount.mul(pool.accAlpacaPerShare).div(1e12);
1553   user.bonusDebt = user.amount.mul(pool.accAlpacaPerShareTilBonusEnd).div(1e12);
1554   if (pool.stakeToken != address(0)) {
1555
```

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.