JK COIN

Smart Contract Audit Report

SYSFIXED

14 Sep 2021

TABLE OF CONTENTS

AUDITED DETAILS

Audited Project

Project name	Token ticker	Blockchain
JK COIN	JK	Binance Smart Chain

Addresses

Contract address	0x1ec58fe5e681e35e490b5d4cbecdf42b29c1b063
Contract deployer address	0x46214af5dD24511ed9943CeF6691536B78cB3bdd

Project Website

https://www.jakaverse.com/

Codebase

https://bscscan.com/address/0x1ec58fe5e681e35e490b5d4cbecdf42b29c1b063#code

SUMMARY

AKAVERSE reinforces its leadership in Metaverse (virtual world) platform, ending Pre Series A funding deal from giant alliance Titan Capital Group Holdings, a leader in supporting and investing in innovative technology and startups both nationally and region with an initial funding of more than 3 million USD. to strengthen the business.

Contract Summary

Documentation Quality

JK COIN provides a very poor documentation with standard of solidity base code.

- The technical description is provided unclear and disorganized.

Code Quality

The Overall quality of the basecode is poor.

- Solidity basecode and rules are unclear and disorganized by JK COIN.

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 70, 71, 100, 111, 123, 124, 125, 148, 165, 183, 203, 223, 258, 269, 307, 316, 332, 336, 358, 368, 394, 83, 139, 350, 353 and 356.
- SWC-103 | Pragma statements can be allowed to float when a contract is intended on lines 9.
- 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 384, 243, 387, 165, 244, 336, 389, 316, 258, 285, 111, 269, 307, 196, 388, 332, 394, 183, 223 and 397.
- SWC-111 | It is recommended to use alternatives to the deprecated constructions on lines 40, 44, 48, 52, 70, 111, 123, 183, 58, 90, 171, 213, 248, 292, 300 and 149.
- SWC-119 | Review storage variable layouts on lines 244.

CONCLUSION

We have audited the JK COIN project released on September 2021 to find issues and identify potential security vulnerabilities in the JK COIN project. This process is used to find technical issues and security loopholes that may be found in smart contracts.

The security audit report yielded unsatisfactory results, discovering medium-risk and low-risk issues.

Writing a contract that does not follow the Solidity style guide can pose a significant risk. The serious and low problems we found in the smart contract are incorrect ERC20 implementation. The function could be marked as an external, built-in symbol "assert" shadowing, and an assertion violation was triggered. Function visibility is not set (prior to Solidity 0.5.0), and the function definition of "transfer" lacks a visibility specifier. Note that the compiler assumes "public" visibility by default. Function visibility should always be specified explicitly to assure the correctness of the code and improve readability.Function could be marked as external, function definition of "balanceOf" is marked "public." However, another function never directly calls it in the same contract or any of its descendants. Consider marking it as "external" instead. If an assertion violation was triggered, it is possible to trigger an assertion violation. Solidity asserts () statements should only be used to check invariants. Review the transaction trace generated for this issue and either make sure your program logic is correct or use require() instead of assert() if your goal is to constrain user inputs or enforce preconditions. Remember to validate inputs from both callers (via passed arguments) and callees (for instance, via return values).

We were recommended to keep being aware of investing in this risky smart contract.

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.	ISSUE FOUND
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes or be predictable.	PASS
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	Monday Sep 13 2021 14:11:17 GMT+0000 (Coordinated Universal Time)
Finished	Tuesday Sep 14 2021 22:09:54 GMT+0000 (Coordinated Universal Time)
Mode	Standard
Main Source File	JK.sol

Detected Issues

ID	Title	Severity	Status
SWC-000	INCORRECT ERC20 IMPLEMENTATION	medium	acknowledged
SWC-000	INCORRECT ERC20 IMPLEMENTATION	medium	acknowledged
SWC-000	INCORRECT ERC20 IMPLEMENTATION	medium	acknowledged
SWC-000	INCORRECT ERC20 IMPLEMENTATION	medium	acknowledged
SWC-000	INCORRECT ERC20 IMPLEMENTATION	medium	acknowledged
SWC-000	INCORRECT ERC20 IMPLEMENTATION	medium	acknowledged
SWC-000	INCORRECT ERC20 IMPLEMENTATION	medium	acknowledged
SWC-000	FUNCTION COULD BE MARKED AS EXTERNAL.	medium	acknowledged
SWC-000	FUNCTION COULD BE MARKED AS EXTERNAL.	medium	acknowledged
SWC-000	FUNCTION COULD BE MARKED AS EXTERNAL.	medium	acknowledged
SWC-000	FUNCTION COULD BE MARKED AS EXTERNAL.	medium	acknowledged
SWC-000	FUNCTION COULD BE MARKED AS EXTERNAL.	medium	acknowledged
SWC-000	FUNCTION COULD BE MARKED AS EXTERNAL.	medium	acknowledged
SWC-000	INCORRECT ERC20 IMPLEMENTATION	medium	acknowledged

SYSFIXED

SWC-000	FUNCTION COULD BE MARKED AS EXTERNAL.	medium	acknowledged
SWC-000	FUNCTION COULD BE MARKED AS EXTERNAL.	medium	acknowledged
SWC-000	FUNCTION COULD BE MARKED AS EXTERNAL.	medium	acknowledged
SWC-000	FUNCTION COULD BE MARKED AS EXTERNAL.	medium	acknowledged
SWC-000	FUNCTION COULD BE MARKED AS EXTERNAL.	medium	acknowledged
SWC-000	INCORRECT ERC20 IMPLEMENTATION	medium	acknowledged
SWC-000	INCORRECT ERC20 IMPLEMENTATION	medium	acknowledged
SWC-000	FUNCTION COULD BE MARKED AS EXTERNAL.	medium	acknowledged
SWC-000	FUNCTION COULD BE MARKED AS EXTERNAL.	medium	acknowledged
SWC-000	BUILTIN SYMBOL "ASSERT" SHADOWING	medium	acknowledged
SWC-110	AN ASSERTION VIOLATION WAS TRIGGERED.	medium	acknowledged
SWC-110	AN ASSERTION VIOLATION WAS TRIGGERED.	medium	acknowledged
SWC-110	AN ASSERTION VIOLATION WAS TRIGGERED.	medium	acknowledged
SWC-110	AN ASSERTION VIOLATION WAS TRIGGERED.	medium	acknowledged
SWC-110	AN ASSERTION VIOLATION WAS TRIGGERED.	medium	acknowledged
SWC-110	AN ASSERTION VIOLATION WAS TRIGGERED.	medium	acknowledged
SWC-110	AN ASSERTION VIOLATION WAS TRIGGERED.	medium	acknowledged
SWC-110	AN ASSERTION VIOLATION WAS TRIGGERED.	medium	acknowledged
SWC-110	AN ASSERTION VIOLATION WAS TRIGGERED.	medium	acknowledged

SWC-110	AN ASSERTION VIOLATION WAS TRIGGERED.	medium	acknowledged
SWC-110	AN ASSERTION VIOLATION WAS TRIGGERED.	medium	acknowledged
SWC-110	AN ASSERTION VIOLATION WAS TRIGGERED.	medium	acknowledged
SWC-110	AN ASSERTION VIOLATION WAS TRIGGERED.	medium	acknowledged
SWC-110	AN ASSERTION VIOLATION WAS TRIGGERED.	medium	acknowledged
SWC-110	AN ASSERTION VIOLATION WAS TRIGGERED.	medium	acknowledged
SWC-110	AN ASSERTION VIOLATION WAS TRIGGERED.	medium	acknowledged
SWC-110	AN ASSERTION VIOLATION WAS TRIGGERED.	medium	acknowledged
SWC-110	AN ASSERTION VIOLATION WAS TRIGGERED.	medium	acknowledged
SWC-110	AN ASSERTION VIOLATION WAS TRIGGERED.	medium	acknowledged
SWC-110	AN ASSERTION VIOLATION WAS TRIGGERED.	medium	acknowledged
SWC-100	FUNCTION VISIBILITY IS NOT SET (PRIOR TO SOLIDITY 0.5.0)	low	acknowledged
SWC-100	FUNCTION VISIBILITY IS NOT SET (PRIOR TO SOLIDITY 0.5.0)	low	acknowledged
SWC-100	FUNCTION VISIBILITY IS NOT SET (PRIOR TO SOLIDITY 0.5.0)	low	acknowledged
SWC-100	FUNCTION VISIBILITY IS NOT SET (PRIOR TO SOLIDITY 0.5.0)	low	acknowledged
SWC-100	FUNCTION VISIBILITY IS NOT SET (PRIOR TO SOLIDITY 0.5.0)	low	acknowledged
SWC-100	FUNCTION VISIBILITY IS NOT SET (PRIOR TO SOLIDITY 0.5.0)	low	acknowledged
SWC-100	FUNCTION VISIBILITY IS NOT SET (PRIOR TO SOLIDITY 0.5.0)	low	acknowledged
SWC-100	FUNCTION VISIBILITY IS NOT SET (PRIOR TO SOLIDITY 0.5.0)	low	acknowledged

SYSFIXED

SWC-100	FUNCTION VISIBILITY IS NOT SET (PRIOR TO SOLIDITY 0.5.0)	low	acknowledged
SWC-100	FUNCTION VISIBILITY IS NOT SET (PRIOR TO SOLIDITY 0.5.0)	low	acknowledged
SWC-100	FUNCTION VISIBILITY IS NOT SET (PRIOR TO SOLIDITY 0.5.0)	low	acknowledged
SWC-100	FUNCTION VISIBILITY IS NOT SET (PRIOR TO SOLIDITY 0.5.0)	low	acknowledged
SWC-100	FUNCTION VISIBILITY IS NOT SET (PRIOR TO SOLIDITY 0.5.0)	low	acknowledged
SWC-100	FUNCTION VISIBILITY IS NOT SET (PRIOR TO SOLIDITY 0.5.0)	low	acknowledged
SWC-100	FUNCTION VISIBILITY IS NOT SET (PRIOR TO SOLIDITY 0.5.0)	low	acknowledged
SWC-100	FUNCTION VISIBILITY IS NOT SET (PRIOR TO SOLIDITY 0.5.0)	low	acknowledged
SWC-100	FUNCTION VISIBILITY IS NOT SET (PRIOR TO SOLIDITY 0.5.0)	low	acknowledged
SWC-100	FUNCTION VISIBILITY IS NOT SET (PRIOR TO SOLIDITY 0.5.0)	low	acknowledged
SWC-100	FUNCTION VISIBILITY IS NOT SET (PRIOR TO SOLIDITY 0.5.0)	low	acknowledged
SWC-100	FUNCTION VISIBILITY IS NOT SET (PRIOR TO SOLIDITY 0.5.0)	low	acknowledged
SWC-100	FUNCTION VISIBILITY IS NOT SET (PRIOR TO SOLIDITY 0.5.0)	low	acknowledged
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-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-111	USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED.	low	acknowledged
SWC-111	USE OF THE "THROW" KEYWORD IS DEPRECATED.	low	acknowledged
SWC-111	USE OF THE "THROW" KEYWORD IS DEPRECATED.	low	acknowledged
SWC-111	USE OF THE "THROW" KEYWORD IS DEPRECATED.	low	acknowledged
SWC-111	USE OF THE "THROW" KEYWORD IS DEPRECATED.	low	acknowledged
SWC-111	USE OF THE "THROW" KEYWORD IS DEPRECATED.	low	acknowledged
SWC-111	USE OF THE "THROW" KEYWORD IS DEPRECATED.	low	acknowledged
SWC-111	USE OF THE "THROW" KEYWORD IS DEPRECATED.	low	acknowledged
SWC-111	USE OF THE "VAR" KEYWORD IS DEPRECATED.	low	acknowledged
SWC-119	STATE VARIABLE SHADOWS ANOTHER STATE VARIABLE.	low	acknowledged

SWC-000 | INCORRECT ERC20 IMPLEMENTATION

LINE 100

medium SEVERITY

Contract "BasicToken" looks like its trying to implement the ERC20 standard, but its missing a required externally accessible function with signature "function transfer(address, uint256) returns (bool)". A similar "transfer" function is defined in contract "BasicToken", but its signature deviates from the standard.

Source File

- JK.sol

Locations

```
99   */
100   function transfer(address _to, uint _value) onlyPayloadSize(2 * 32) {
101   balances[msg.sender] = balances[msg.sender].sub(_value);
102   balances[_to] = balances[_to].add(_value);
103   Transfer(msg.sender, _to, _value);
104
```

SWC-000 | INCORRECT ERC20 IMPLEMENTATION

LINE 80

medium SEVERITY

Contract "BasicToken" looks like its trying to implement the ERC20 standard, but its missing a required event with signature "event Approval(address indexed, address indexed, uint256)"

Source File

- JK.sol

Locations

```
79   */
80   contract BasicToken is ERC20Basic {
81   using SafeMath for uint;
82
83   mapping(address => uint) balances;
84
```

SWC-000 | INCORRECT ERC20 IMPLEMENTATION

LINE 137

medium SEVERITY

Contract "StandardToken" looks like its trying to implement the ERC20 standard, but its missing a required externally accessible function with signature "function transfer(address, uint256) returns (bool)". A similar "transfer" function is defined in contract "BasicToken", but its signature deviates from the standard.

Source File

- JK.sol

Locations

```
136   */
137   contract StandardToken is BasicToken, ERC20 {
138
139   mapping (address => mapping (address => uint)) allowed;
140
141
```

SWC-000 | INCORRECT ERC20 IMPLEMENTATION

LINE 148

medium SEVERITY

Contract "StandardToken" looks like its trying to implement the ERC20 standard, but its missing a required externally accessible function with signature "function transferFrom(address, address, uint256) returns (bool)". A similar "transferFrom" function is defined in contract "StandardToken", but its signature deviates from the standard.

Source File

- JK.sol

Locations

```
147   */
148   function transferFrom(address _from, address _to, uint _value) onlyPayloadSize(3 *
32) {
149   var _allowance = allowed[_from][msg.sender];
150
151   // Check is not needed because sub(_allowance, _value) will already throw if this
condition is not met
152
```

SWC-000 | INCORRECT ERC20 IMPLEMENTATION

LINE 239

medium SEVERITY

Contract "MintableToken" looks like its trying to implement the ERC20 standard, but its missing a required externally accessible function with signature "function transfer(address, uint256) returns (bool)". A similar "transfer" function is defined in contract "BasicToken", but its signature deviates from the standard.

Source File

- JK.sol

Locations

```
238
239   contract MintableToken is StandardToken, Ownable {
240   event Mint(address indexed to, uint value);
241   event MintFinished();
242
243
```

SWC-000 | INCORRECT ERC20 IMPLEMENTATION

LINE 330

medium SEVERITY

Contract "PausableToken" looks like its trying to implement the ERC20 standard, but its missing a required externally accessible function with signature "function approve(address, uint256) returns (bool)". A similar "approve" function is defined in contract "StandardToken", but its signature deviates from the standard.

Source File

- JK.sol

Locations

```
329
330   contract PausableToken is StandardToken, Pausable {
331
332   function transfer(address _to, uint _value) whenNotPaused {
333   super.transfer(_to, _value);
334
```

SWC-000 | INCORRECT ERC20 IMPLEMENTATION

LINE 384

medium SEVERITY

Contract "JK" looks like its trying to implement the ERC20 standard, but its missing a required externally accessible function with signature "function transfer(address, uint256) returns (bool)". A similar "transfer" function is defined in contract "PausableToken", but its signature deviates from the standard.

Source File

- JK.sol

Locations

```
383   */
384   contract JK is PausableToken, MintableToken {
385   using SafeMath for uint256;
386
387   string public name = "JK COIN";
388
```

SWC-000 | FUNCTION COULD BE MARKED AS EXTERNAL.

LINE 70

medium SEVERITY

The function definition of "balanceOf" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source File

- JK.sol

Locations

```
69   uint public totalSupply;
70   function balanceOf(address who) constant returns (uint);
71   function transfer(address to, uint value);
72   event Transfer(address indexed from, address indexed to, uint value);
73   }
74
```

SWC-000 | FUNCTION COULD BE MARKED AS EXTERNAL.

LINE 71

medium SEVERITY

The function definition of "transfer" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source File

- JK.sol

Locations

```
70   function balanceOf(address who) constant returns (uint);
71   function transfer(address to, uint value);
72   event Transfer(address indexed from, address indexed to, uint value);
73   }
74
75
```

SWC-000 | FUNCTION COULD BE MARKED AS EXTERNAL.

LINE 111

medium SEVERITY

The function definition of "balanceOf" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source File

- JK.sol

Locations

```
110   */
111   function balanceOf(address _owner) constant returns (uint balance) {
112   return balances[_owner];
113   }
114
115
```

SWC-000 | FUNCTION COULD BE MARKED AS EXTERNAL.

LINE 123

medium SEVERITY

The function definition of "allowance" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source File

- JK.sol

Locations

```
122   contract ERC20 is ERC20Basic {
123   function allowance(address owner, address spender) constant returns (uint);
124   function transferFrom(address from, address to, uint value);
125   function approve(address spender, uint value);
126   event Approval(address indexed owner, address indexed spender, uint value);
127
```

SWC-000 | FUNCTION COULD BE MARKED AS EXTERNAL.

LINE 124

medium SEVERITY

The function definition of "transferFrom" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source File

- JK.sol

Locations

```
123   function allowance(address owner, address spender) constant returns (uint);
124   function transferFrom(address from, address to, uint value);
125   function approve(address spender, uint value);
126   event Approval(address indexed owner, address indexed spender, uint value);
127   }
128
```

SWC-000 | FUNCTION COULD BE MARKED AS EXTERNAL.

LINE 125

medium SEVERITY

The function definition of "approve" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source File

- JK.sol

Locations

```
124    function transferFrom(address from, address to, uint value);
125    function approve(address spender, uint value);
126    event Approval(address indexed owner, address indexed spender, uint value);
127    }
128
129
```

SWC-000 | INCORRECT ERC20 IMPLEMENTATION

LINE 165

medium SEVERITY

Contract "StandardToken" looks like its trying to implement the ERC20 standard, but its missing a required externally accessible function with signature "function approve(address, uint256) returns (bool)". A similar "approve" function is defined in contract "StandardToken", but its signature deviates from the standard.

Source File

- JK.sol

Locations

```
164   */
165   function approve(address _spender, uint _value) {
166
167   // To change the approve amount you first have to reduce the addresses`
168   //  allowance to zero by calling `approve(_spender, 0)` if it is not
169
```

SWC-000 | FUNCTION COULD BE MARKED AS EXTERNAL.

LINE 183

medium SEVERITY

The function definition of "allowance" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source File

- JK.sol

Locations

```
182   */
183   function allowance(address _owner, address _spender) constant returns (uint
remaining) {
184   return allowed[_owner][_spender];
185   }
186
187
```

SWC-000 | FUNCTION COULD BE MARKED AS EXTERNAL.

LINE 223

medium SEVERITY

The function definition of "transferOwnership" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source File

- JK.sol

Locations

```
222   */
223   function transferOwnership(address newOwner) onlyOwner {
224   if (newOwner != address(0)) {
225   owner = newOwner;
226   }
227
```

SWC-000 | FUNCTION COULD BE MARKED AS EXTERNAL.

LINE 269

medium SEVERITY

The function definition of "finishMinting" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source File

- JK.sol

Locations

```
268   */
269   function finishMinting() onlyOwner returns (bool) {
270   mintingFinished = true;
271   MintFinished();
272   return true;
273
```

SWC-000 | FUNCTION COULD BE MARKED AS EXTERNAL.

LINE 307

medium SEVERITY

The function definition of "pause" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source File

- JK.sol

Locations

```
306   */
307   function pause() onlyOwner whenNotPaused returns (bool) {
308   paused = true;
309   Pause();
310   return true;
311
```

SWC-000 | FUNCTION COULD BE MARKED AS EXTERNAL.

LINE 316

medium SEVERITY

The function definition of "unpause" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source File

- JK.sol

Locations

```
315   */
316   function unpause() onlyOwner whenPaused returns (bool) {
317   paused = false;
318   Unpause();
319   return true;
320
```

SWC-000 | INCORRECT ERC20 IMPLEMENTATION

LINE 332

medium SEVERITY

Contract "PausableToken" looks like its trying to implement the ERC20 standard, but its missing a required externally accessible function with signature "function transfer(address, uint256) returns (bool)". A similar "transfer" function is defined in contract "PausableToken", but its signature deviates from the standard.

Source File

- JK.sol

Locations

```
331
332   function transfer(address _to, uint _value) whenNotPaused {
333   super.transfer(_to, _value);
334   }
335
336
```

SWC-000 | INCORRECT ERC20 IMPLEMENTATION

LINE 336

medium SEVERITY

Contract "PausableToken" looks like its trying to implement the ERC20 standard, but its missing a required externally accessible function with signature "function transferFrom(address, address, uint256) returns (bool)". A similar "transferFrom" function is defined in contract "PausableToken", but its signature deviates from the standard.

Source File

- JK.sol

Locations

```
335
336   function transferFrom(address _from, address _to, uint _value) whenNotPaused {
337   super.transferFrom(_from, _to, _value);
338   }
339   }
340
```

SWC-000 | FUNCTION COULD BE MARKED AS EXTERNAL.

LINE 368

medium SEVERITY

The function definition of "claim" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source File

- JK.sol

Locations

```
367   */
368   function claim() {
369   require(msg.sender == beneficiary);
370   require(now >= releaseTime);
371
372
```

SWC-000 | FUNCTION COULD BE MARKED AS EXTERNAL.

LINE 394

medium SEVERITY

The function definition of "mintTimelocked" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source File

- JK.sol

Locations

```
393   */
394   function mintTimelocked(address _to, uint256 _amount, uint256 _releaseTime)
395   onlyOwner canMint returns (TokenTimelock) {
396
397   TokenTimelock timelock = new TokenTimelock(this, _to, _releaseTime);
398
```

SWC-000 | BUILTIN SYMBOL "ASSERT" SHADOWING

LINE 56

medium SEVERITY

Definition "assert" uses the same name as a built-in symbol. Reserved names should not be used to avoid confusion.

Source File

- JK.sol

Locations

```
55
56   function assert(bool assertion) internal {
57   if (!assertion) {
58   throw;
59   }
60
```

SWC-110 | AN ASSERTION VIOLATION WAS TRIGGERED.

LINE 384

medium SEVERITY

It is possible to trigger an assertion violation. Note that Solidity assert() statements should only be used to check invariants. Review the transaction trace generated for this issue and either make sure your program logic is correct, or use require() instead of assert() if your goal is to constrain user inputs or enforce preconditions. Remember to validate inputs from both callers (for instance, via passed arguments) and callees (for instance, via return values).

Source File

- JK.sol

Locations

```
383   */
384   contract JK is PausableToken, MintableToken {
385   using SafeMath for uint256;
386
387   string public name = "JK COIN";
388
```

SWC-110 | AN ASSERTION VIOLATION WAS TRIGGERED.

LINE 243

medium SEVERITY

It is possible to trigger an assertion violation. Note that Solidity assert() statements should only be used to check invariants. Review the transaction trace generated for this issue and either make sure your program logic is correct, or use require() instead of assert() if your goal is to constrain user inputs or enforce preconditions. Remember to validate inputs from both callers (for instance, via passed arguments) and callees (for instance, via return values).

Source File

- JK.sol

Locations

```
242
243   bool public mintingFinished = false;
244   uint public totalSupply = 0;
245
246
247
```

SWC-110 | AN ASSERTION VIOLATION WAS TRIGGERED.

LINE 387

medium SEVERITY

It is possible to trigger an assertion violation. Note that Solidity assert() statements should only be used to check invariants. Review the transaction trace generated for this issue and either make sure your program logic is correct, or use require() instead of assert() if your goal is to constrain user inputs or enforce preconditions. Remember to validate inputs from both callers (for instance, via passed arguments) and callees (for instance, via return values).

Source File

- JK.sol

Locations

```
386
387   string public name = "JK COIN";
388   string public symbol = "JK";
389   uint public decimals = 18;
390
391
```

SWC-110 | AN ASSERTION VIOLATION WAS TRIGGERED.

LINE 165

medium SEVERITY

It is possible to trigger an assertion violation. Note that Solidity assert() statements should only be used to check invariants. Review the transaction trace generated for this issue and either make sure your program logic is correct, or use require() instead of assert() if your goal is to constrain user inputs or enforce preconditions. Remember to validate inputs from both callers (for instance, via passed arguments) and callees (for instance, via return values).

Source File

- JK.sol

Locations

```
164   */
165   function approve(address _spender, uint _value) {
166
167   // To change the approve amount you first have to reduce the addresses`
168   //  allowance to zero by calling `approve(_spender, 0)` if it is not
169
```

SWC-110 | AN ASSERTION VIOLATION WAS TRIGGERED.

LINE 244

medium SEVERITY

It is possible to trigger an assertion violation. Note that Solidity assert() statements should only be used to check invariants. Review the transaction trace generated for this issue and either make sure your program logic is correct, or use require() instead of assert() if your goal is to constrain user inputs or enforce preconditions. Remember to validate inputs from both callers (for instance, via passed arguments) and callees (for instance, via return values).

Source File

- JK.sol

Locations

```
243   bool public mintingFinished = false;
244   uint public totalSupply = 0;
245
246
247   modifier canMint() {
248
```

SWC-110 | AN ASSERTION VIOLATION WAS TRIGGERED.

LINE 336

medium SEVERITY

It is possible to trigger an assertion violation. Note that Solidity assert() statements should only be used to check invariants. Review the transaction trace generated for this issue and either make sure your program logic is correct, or use require() instead of assert() if your goal is to constrain user inputs or enforce preconditions. Remember to validate inputs from both callers (for instance, via passed arguments) and callees (for instance, via return values).

Source File

- JK.sol

Locations

```
335
336   function transferFrom(address _from, address _to, uint _value) whenNotPaused {
337   super.transferFrom(_from, _to, _value);
338   }
339   }
340
```

SWC-110 | AN ASSERTION VIOLATION WAS TRIGGERED.

LINE 389

medium SEVERITY

It is possible to trigger an assertion violation. Note that Solidity assert() statements should only be used to check invariants. Review the transaction trace generated for this issue and either make sure your program logic is correct, or use require() instead of assert() if your goal is to constrain user inputs or enforce preconditions. Remember to validate inputs from both callers (for instance, via passed arguments) and callees (for instance, via return values).

Source File

- JK.sol

Locations

```
388   string public symbol = "JK";
389   uint public decimals = 18;
390
391   /**
392   * @dev mint timelocked tokens
393
```

SWC-110 | AN ASSERTION VIOLATION WAS TRIGGERED.

LINE 316

medium SEVERITY

It is possible to trigger an assertion violation. Note that Solidity assert() statements should only be used to check invariants. Review the transaction trace generated for this issue and either make sure your program logic is correct, or use require() instead of assert() if your goal is to constrain user inputs or enforce preconditions. Remember to validate inputs from both callers (for instance, via passed arguments) and callees (for instance, via return values).

Source File

- JK.sol

Locations

```
315   */
316   function unpause() onlyOwner whenPaused returns (bool) {
317   paused = false;
318   Unpause();
319   return true;
320
```

SWC-110 | AN ASSERTION VIOLATION WAS TRIGGERED.

LINE 258

medium SEVERITY

It is possible to trigger an assertion violation. Note that Solidity assert() statements should only be used to check invariants. Review the transaction trace generated for this issue and either make sure your program logic is correct, or use require() instead of assert() if your goal is to constrain user inputs or enforce preconditions. Remember to validate inputs from both callers (for instance, via passed arguments) and callees (for instance, via return values).

Source File

- JK.sol

Locations

```
257   */
258   function mint(address _to, uint _amount) onlyOwner canMint returns (bool) {
259   totalSupply = totalSupply.add(_amount);
260   balances[_to] = balances[_to].add(_amount);
261   Mint(_to, _amount);
262
```

SWC-110 | AN ASSERTION VIOLATION WAS TRIGGERED.

LINE 285

medium SEVERITY

It is possible to trigger an assertion violation. Note that Solidity assert() statements should only be used to check invariants. Review the transaction trace generated for this issue and either make sure your program logic is correct, or use require() instead of assert() if your goal is to constrain user inputs or enforce preconditions. Remember to validate inputs from both callers (for instance, via passed arguments) and callees (for instance, via return values).

Source File

- JK.sol

Locations

```
284
285   bool public paused = false;
286
287
288   /**
289
```

SWC-110 | AN ASSERTION VIOLATION WAS TRIGGERED.

LINE 111

medium SEVERITY

It is possible to trigger an assertion violation. Note that Solidity assert() statements should only be used to check invariants. Review the transaction trace generated for this issue and either make sure your program logic is correct, or use require() instead of assert() if your goal is to constrain user inputs or enforce preconditions. Remember to validate inputs from both callers (for instance, via passed arguments) and callees (for instance, via return values).

Source File

- JK.sol

Locations

```
110   */
111   function balanceOf(address _owner) constant returns (uint balance) {
112   return balances[_owner];
113   }
114
115
```

SWC-110 | AN ASSERTION VIOLATION WAS TRIGGERED.

LINE 269

medium SEVERITY

It is possible to trigger an assertion violation. Note that Solidity assert() statements should only be used to check invariants. Review the transaction trace generated for this issue and either make sure your program logic is correct, or use require() instead of assert() if your goal is to constrain user inputs or enforce preconditions. Remember to validate inputs from both callers (for instance, via passed arguments) and callees (for instance, via return values).

Source File

- JK.sol

Locations

```
268   */
269   function finishMinting() onlyOwner returns (bool) {
270   mintingFinished = true;
271   MintFinished();
272   return true;
273
```

SWC-110 | AN ASSERTION VIOLATION WAS TRIGGERED.

LINE 307

medium SEVERITY

It is possible to trigger an assertion violation. Note that Solidity assert() statements should only be used to check invariants. Review the transaction trace generated for this issue and either make sure your program logic is correct, or use require() instead of assert() if your goal is to constrain user inputs or enforce preconditions. Remember to validate inputs from both callers (for instance, via passed arguments) and callees (for instance, via return values).

Source File

- JK.sol

Locations

```
306   */
307   function pause() onlyOwner whenNotPaused returns (bool) {
308   paused = true;
309   Pause();
310   return true;
311
```

SWC-110 | AN ASSERTION VIOLATION WAS TRIGGERED.

LINE 196

medium SEVERITY

It is possible to trigger an assertion violation. Note that Solidity assert() statements should only be used to check invariants. Review the transaction trace generated for this issue and either make sure your program logic is correct, or use require() instead of assert() if your goal is to constrain user inputs or enforce preconditions. Remember to validate inputs from both callers (for instance, via passed arguments) and callees (for instance, via return values).

Source File

- JK.sol

Locations

```
195   contract Ownable {
196   address public owner;
197
198
199   /**
200
```

SWC-110 | AN ASSERTION VIOLATION WAS TRIGGERED.

LINE 388

medium SEVERITY

It is possible to trigger an assertion violation. Note that Solidity assert() statements should only be used to check invariants. Review the transaction trace generated for this issue and either make sure your program logic is correct, or use require() instead of assert() if your goal is to constrain user inputs or enforce preconditions. Remember to validate inputs from both callers (for instance, via passed arguments) and callees (for instance, via return values).

Source File

- JK.sol

Locations

```
387   string public name = "JK COIN";
388   string public symbol = "JK";
389   uint public decimals = 18;
390
391   /**
392
```

SWC-110 | AN ASSERTION VIOLATION WAS TRIGGERED.

LINE 332

medium SEVERITY

It is possible to trigger an assertion violation. Note that Solidity assert() statements should only be used to check invariants. Review the transaction trace generated for this issue and either make sure your program logic is correct, or use require() instead of assert() if your goal is to constrain user inputs or enforce preconditions. Remember to validate inputs from both callers (for instance, via passed arguments) and callees (for instance, via return values).

Source File

- JK.sol

Locations

```
331
332   function transfer(address _to, uint _value) whenNotPaused {
333   super.transfer(_to, _value);
334   }
335
336
```

SWC-110 | AN ASSERTION VIOLATION WAS TRIGGERED.

LINE 394

medium SEVERITY

It is possible to trigger an assertion violation. Note that Solidity assert() statements should only be used to check invariants. Review the transaction trace generated for this issue and either make sure your program logic is correct, or use require() instead of assert() if your goal is to constrain user inputs or enforce preconditions. Remember to validate inputs from both callers (for instance, via passed arguments) and callees (for instance, via return values).

Source File

- JK.sol

Locations

```
393   */
394   function mintTimelocked(address _to, uint256 _amount, uint256 _releaseTime)
395   onlyOwner canMint returns (TokenTimelock) {
396
397   TokenTimelock timelock = new TokenTimelock(this, _to, _releaseTime);
398
```

SWC-110 | AN ASSERTION VIOLATION WAS TRIGGERED.

LINE 183

medium SEVERITY

It is possible to trigger an assertion violation. Note that Solidity assert() statements should only be used to check invariants. Review the transaction trace generated for this issue and either make sure your program logic is correct, or use require() instead of assert() if your goal is to constrain user inputs or enforce preconditions. Remember to validate inputs from both callers (for instance, via passed arguments) and callees (for instance, via return values).

Source File

- JK.sol

Locations

```
182   */
183   function allowance(address _owner, address _spender) constant returns (uint
remaining) {
184   return allowed[_owner][_spender];
185   }
186
187
```

SWC-110 | AN ASSERTION VIOLATION WAS TRIGGERED.

LINE 223

medium SEVERITY

It is possible to trigger an assertion violation. Note that Solidity assert() statements should only be used to check invariants. Review the transaction trace generated for this issue and either make sure your program logic is correct, or use require() instead of assert() if your goal is to constrain user inputs or enforce preconditions. Remember to validate inputs from both callers (for instance, via passed arguments) and callees (for instance, via return values).

Source File

- JK.sol

Locations

```
222   */
223   function transferOwnership(address newOwner) onlyOwner {
224   if (newOwner != address(0)) {
225   owner = newOwner;
226   }
227
```

SWC-110 | AN ASSERTION VIOLATION WAS TRIGGERED.

LINE 397

medium SEVERITY

It is possible to trigger an assertion violation. Note that Solidity assert() statements should only be used to check invariants. Review the transaction trace generated for this issue and either make sure your program logic is correct, or use require() instead of assert() if your goal is to constrain user inputs or enforce preconditions. Remember to validate inputs from both callers (for instance, via passed arguments) and callees (for instance, via return values).

Source File

- JK.sol

Locations

```
396
397   TokenTimelock timelock = new TokenTimelock(this, _to, _releaseTime);
398   mint(timelock, _amount);
399
400   return timelock;
401
```

SWC-100 | FUNCTION VISIBILITY IS NOT SET (PRIOR TO SOLIDITY 0.5.0)

LINE 70

low SEVERITY

The function definition of "balanceOf" lacks a visibility specifier. Note that the compiler assumes "public" visibility by default. Function visibility should always be specified explicitly to assure correctness of the code and improve readability.

Source File

- JK.sol

Locations

```
69   uint public totalSupply;
70   function balanceOf(address who) constant returns (uint);
71   function transfer(address to, uint value);
72   event Transfer(address indexed from, address indexed to, uint value);
73   }
74
```

SWC-100 | FUNCTION VISIBILITY IS NOT SET (PRIOR TO SOLIDITY 0.5.0)

LINE 71

low SEVERITY

The function definition of "transfer" lacks a visibility specifier. Note that the compiler assumes "public" visibility by default. Function visibility should always be specified explicitly to assure correctness of the code and improve readability.

Source File

- JK.sol

Locations

```
70   function balanceOf(address who) constant returns (uint);
71   function transfer(address to, uint value);
72   event Transfer(address indexed from, address indexed to, uint value);
73   }
74
75
```

SWC-100 | FUNCTION VISIBILITY IS NOT SET (PRIOR TO SOLIDITY 0.5.0)

LINE 100

low SEVERITY

The function definition of "transfer" lacks a visibility specifier. Note that the compiler assumes "public" visibility by default. Function visibility should always be specified explicitly to assure correctness of the code and improve readability.

Source File

- JK.sol

Locations

```
99    */
100    function transfer(address _to, uint _value) onlyPayloadSize(2 * 32) {
101    balances[msg.sender] = balances[msg.sender].sub(_value);
102    balances[_to] = balances[_to].add(_value);
103    Transfer(msg.sender, _to, _value);
104
```

SWC-100 | FUNCTION VISIBILITY IS NOT SET (PRIOR TO SOLIDITY 0.5.0)

LINE 111

low SEVERITY

The function definition of "balanceOf" lacks a visibility specifier. Note that the compiler assumes "public" visibility by default. Function visibility should always be specified explicitly to assure correctness of the code and improve readability.

Source File

- JK.sol

Locations

```
110   */
111   function balanceOf(address _owner) constant returns (uint balance) {
112   return balances[_owner];
113   }
114
115
```

SWC-100 | FUNCTION VISIBILITY IS NOT SET (PRIOR TO SOLIDITY 0.5.0)

LINE 123

low SEVERITY

The function definition of "allowance" lacks a visibility specifier. Note that the compiler assumes "public" visibility by default. Function visibility should always be specified explicitly to assure correctness of the code and improve readability.

Source File

- JK.sol

Locations

```
122   contract ERC20 is ERC20Basic {
123   function allowance(address owner, address spender) constant returns (uint);
124   function transferFrom(address from, address to, uint value);
125   function approve(address spender, uint value);
126   event Approval(address indexed owner, address indexed spender, uint value);
127
```

SWC-100 | FUNCTION VISIBILITY IS NOT SET (PRIOR TO SOLIDITY 0.5.0)

LINE 124

low SEVERITY

The function definition of "transferFrom" lacks a visibility specifier. Note that the compiler assumes "public" visibility by default. Function visibility should always be specified explicitly to assure correctness of the code and improve readability.

Source File

- JK.sol

Locations

```
123   function allowance(address owner, address spender) constant returns (uint);
124   function transferFrom(address from, address to, uint value);
125   function approve(address spender, uint value);
126   event Approval(address indexed owner, address indexed spender, uint value);
127   }
128
```

SWC-100 | FUNCTION VISIBILITY IS NOT SET (PRIOR TO SOLIDITY 0.5.0)

LINE 125

low SEVERITY

The function definition of "approve" lacks a visibility specifier. Note that the compiler assumes "public" visibility by default. Function visibility should always be specified explicitly to assure correctness of the code and improve readability.

Source File

- JK.sol

Locations

```
124   function transferFrom(address from, address to, uint value);
125   function approve(address spender, uint value);
126   event Approval(address indexed owner, address indexed spender, uint value);
127   }
128
129
```

SWC-100 | FUNCTION VISIBILITY IS NOT SET (PRIOR TO SOLIDITY 0.5.0)

LINE 148

low SEVERITY

The function definition of "transferFrom" lacks a visibility specifier. Note that the compiler assumes "public" visibility by default. Function visibility should always be specified explicitly to assure correctness of the code and improve readability.

Source File

- JK.sol

Locations

```
147   */
148   function transferFrom(address _from, address _to, uint _value) onlyPayloadSize(3 *
32) {
149   var _allowance = allowed[_from][msg.sender];
150
151   // Check is not needed because sub(_allowance, _value) will already throw if this
condition is not met
152
```

SWC-100 | FUNCTION VISIBILITY IS NOT SET (PRIOR TO SOLIDITY 0.5.0)

LINE 165

low SEVERITY

The function definition of "approve" lacks a visibility specifier. Note that the compiler assumes "public" visibility by default. Function visibility should always be specified explicitly to assure correctness of the code and improve readability.

Source File

- JK.sol

Locations

```
164   */
165   function approve(address _spender, uint _value) {
166
167   // To change the approve amount you first have to reduce the addresses`
168   //  allowance to zero by calling `approve(_spender, 0)` if it is not
169
```

SWC-100 | FUNCTION VISIBILITY IS NOT SET (PRIOR TO SOLIDITY 0.5.0)

LINE 183

low SEVERITY

The function definition of "allowance" lacks a visibility specifier. Note that the compiler assumes "public" visibility by default. Function visibility should always be specified explicitly to assure correctness of the code and improve readability.

Source File

- JK.sol

Locations

```
182   */
183   function allowance(address _owner, address _spender) constant returns (uint
remaining) {
184   return allowed[_owner][_spender];
185   }
186
187
```

SWC-100 | FUNCTION VISIBILITY IS NOT SET (PRIOR TO SOLIDITY 0.5.0)

LINE 203

low SEVERITY

The function definition of "Ownable" lacks a visibility specifier. Note that the compiler assumes "public" visibility by default. Function visibility should always be specified explicitly to assure correctness of the code and improve readability.

Source File

- JK.sol

Locations

```
202   */
203   function Ownable() {
204   owner = msg.sender;
205   }
206
207
```

SWC-100 | FUNCTION VISIBILITY IS NOT SET (PRIOR TO SOLIDITY 0.5.0)

LINE 223

low SEVERITY

The function definition of "transferOwnership" lacks a visibility specifier. Note that the compiler assumes "public" visibility by default. Function visibility should always be specified explicitly to assure correctness of the code and improve readability.

Source File

- JK.sol

Locations

```
222   */
223   function transferOwnership(address newOwner) onlyOwner {
224   if (newOwner != address(0)) {
225   owner = newOwner;
226   }
227
```

SWC-100 | FUNCTION VISIBILITY IS NOT SET (PRIOR TO SOLIDITY 0.5.0)

LINE 258

low SEVERITY

The function definition of "mint" lacks a visibility specifier. Note that the compiler assumes "public" visibility by default. Function visibility should always be specified explicitly to assure correctness of the code and improve readability.

Source File

- JK.sol

Locations

```
257   */
258   function mint(address _to, uint _amount) onlyOwner canMint returns (bool) {
259   totalSupply = totalSupply.add(_amount);
260   balances[_to] = balances[_to].add(_amount);
261   Mint(_to, _amount);
262
```

SWC-100 | FUNCTION VISIBILITY IS NOT SET (PRIOR TO SOLIDITY 0.5.0)

LINE 269

low SEVERITY

The function definition of "finishMinting" lacks a visibility specifier. Note that the compiler assumes "public" visibility by default. Function visibility should always be specified explicitly to assure correctness of the code and improve readability.

Source File

- JK.sol

Locations

```
268   */
269   function finishMinting() onlyOwner returns (bool) {
270   mintingFinished = true;
271   MintFinished();
272   return true;
273
```

SWC-100 | FUNCTION VISIBILITY IS NOT SET (PRIOR TO SOLIDITY 0.5.0)

LINE 307

low SEVERITY

The function definition of "pause" lacks a visibility specifier. Note that the compiler assumes "public" visibility by default. Function visibility should always be specified explicitly to assure correctness of the code and improve readability.

Source File

- JK.sol

Locations

```
306   */
307   function pause() onlyOwner whenNotPaused returns (bool) {
308   paused = true;
309   Pause();
310   return true;
311
```

SWC-100 | FUNCTION VISIBILITY IS NOT SET (PRIOR TO SOLIDITY 0.5.0)

LINE 316

low SEVERITY

The function definition of "unpause" lacks a visibility specifier. Note that the compiler assumes "public" visibility by default. Function visibility should always be specified explicitly to assure correctness of the code and improve readability.

Source File

- JK.sol

Locations

```
315   */
316   function unpause() onlyOwner whenPaused returns (bool) {
317   paused = false;
318   Unpause();
319   return true;
320
```

SWC-100 | FUNCTION VISIBILITY IS NOT SET (PRIOR TO SOLIDITY 0.5.0)

LINE 332

low SEVERITY

The function definition of "transfer" lacks a visibility specifier. Note that the compiler assumes "public" visibility by default. Function visibility should always be specified explicitly to assure correctness of the code and improve readability.

Source File

- JK.sol

Locations

```
331
332   function transfer(address _to, uint _value) whenNotPaused {
333   super.transfer(_to, _value);
334   }
335
336
```

SWC-100 | FUNCTION VISIBILITY IS NOT SET (PRIOR TO SOLIDITY 0.5.0)

LINE 336

low SEVERITY

The function definition of "transferFrom" lacks a visibility specifier. Note that the compiler assumes "public" visibility by default. Function visibility should always be specified explicitly to assure correctness of the code and improve readability.

Source File

- JK.sol

Locations

```
335
336   function transferFrom(address _from, address _to, uint _value) whenNotPaused {
337   super.transferFrom(_from, _to, _value);
338   }
339   }
340
```

SWC-100 | FUNCTION VISIBILITY IS NOT SET (PRIOR TO SOLIDITY 0.5.0)

LINE 358

low SEVERITY

The function definition of "TokenTimelock" lacks a visibility specifier. Note that the compiler assumes "public" visibility by default. Function visibility should always be specified explicitly to assure correctness of the code and improve readability.

Source File

- JK.sol

Locations

```
357
358   function TokenTimelock(ERC20Basic _token, address _beneficiary, uint _releaseTime)
{
359   require(_releaseTime > now);
360   token = _token;
361   beneficiary = _beneficiary;
362
```

SWC-100 | FUNCTION VISIBILITY IS NOT SET (PRIOR TO SOLIDITY 0.5.0)

LINE 368

low SEVERITY

The function definition of "claim" lacks a visibility specifier. Note that the compiler assumes "public" visibility by default. Function visibility should always be specified explicitly to assure correctness of the code and improve readability.

Source File

- JK.sol

Locations

```
367   */
368   function claim() {
369   require(msg.sender == beneficiary);
370   require(now >= releaseTime);
371
372
```

SWC-100 | FUNCTION VISIBILITY IS NOT SET (PRIOR TO SOLIDITY 0.5.0)

LINE 394

low SEVERITY

The function definition of "mintTimelocked" lacks a visibility specifier. Note that the compiler assumes "public" visibility by default. Function visibility should always be specified explicitly to assure correctness of the code and improve readability.

Source File

- JK.sol

Locations

```
393   */
394   function mintTimelocked(address _to, uint256 _amount, uint256 _releaseTime)
395   onlyOwner canMint returns (TokenTimelock) {
396
397   TokenTimelock timelock = new TokenTimelock(this, _to, _releaseTime);
398
```

SWC-103 | A FLOATING PRAGMA IS SET.

LINE 9

low SEVERITY

The current pragma Solidity directive is ""^0.4.11"". 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

- JK.sol

Locations

```
8
9    pragma solidity ^0.4.11;
10
11
12    /**
13
```

SWC-108 | STATE VARIABLE VISIBILITY IS NOT SET.

LINE 83

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

- JK.sol

Locations

```
82
83   mapping(address => uint) balances;
84
85   /**
86   * @dev Fix for the ERC20 short address attack.
87
```

SWC-108 | STATE VARIABLE VISIBILITY IS NOT SET.

LINE 139

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

- JK.sol

Locations

```
138
139    mapping (address => mapping (address => uint)) allowed;
140
141
142    /**
143
```

SWC-108 | STATE VARIABLE VISIBILITY IS NOT SET.

LINE 350

low SEVERITY

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

Source File

- JK.sol

Locations

```
349   // ERC20 basic token contract being held
350   ERC20Basic token;
351
352   // beneficiary of tokens after they are released
353   address beneficiary;
354
```

SWC-108 | STATE VARIABLE VISIBILITY IS NOT SET.

LINE 353

low SEVERITY

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

Source File

- JK.sol

Locations

```
352   // beneficiary of tokens after they are released
353   address beneficiary;
354
355   // timestamp where token release is enabled
356   uint releaseTime;
357
```

SWC-108 | STATE VARIABLE VISIBILITY IS NOT SET.

LINE 356

low SEVERITY

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

Source File

- JK.sol

Locations

```
355   // timestamp where token release is enabled
356   uint releaseTime;
357
358   function TokenTimelock(ERC20Basic _token, address _beneficiary, uint _releaseTime)
{
359   require(_releaseTime > now);
360
```

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

LINE 40

low SEVERITY

Using "constant" as a state mutability modifier in function "max64" is disallowed as of Solidity version 0.5.0. Use "view" instead.

Source File

- JK.sol

Locations

```
39
40   function max64(uint64 a, uint64 b) internal constant returns (uint64) {
41   return a >= b ? a : b;
42   }
43
44
```

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

LINE 44

low SEVERITY

Using "constant" as a state mutability modifier in function "min64" is disallowed as of Solidity version 0.5.0. Use "view" instead.

Source File

- JK.sol

Locations

```
43
44   function min64(uint64 a, uint64 b) internal constant returns (uint64) {
45   return a < b ? a : b;
46   }
47
48
```

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

LINE 48

low SEVERITY

Using "constant" as a state mutability modifier in function "max256" is disallowed as of Solidity version 0.5.0. Use "view" instead.

Source File

- JK.sol

Locations

```
47
48   function max256(uint256 a, uint256 b) internal constant returns (uint256) {
49   return a >= b ? a : b;
50   }
51
52
```

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

LINE 52

low SEVERITY

Using "constant" as a state mutability modifier in function "min256" is disallowed as of Solidity version 0.5.0. Use "view" instead.

Source File

- JK.sol

Locations

```
51
52   function min256(uint256 a, uint256 b) internal constant returns (uint256) {
53   return a < b ? a : b;
54   }
55
56
```

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

LINE 70

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

- JK.sol

Locations

```
69   uint public totalSupply;
70   function balanceOf(address who) constant returns (uint);
71   function transfer(address to, uint value);
72   event Transfer(address indexed from, address indexed to, uint value);
73   }
74
```

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

LINE 111

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

- JK.sol

Locations

```
110   */
111   function balanceOf(address _owner) constant returns (uint balance) {
112   return balances[_owner];
113   }
114
115
```

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

LINE 123

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

- JK.sol

Locations

```
122   contract ERC20 is ERC20Basic {
123   function allowance(address owner, address spender) constant returns (uint);
124   function transferFrom(address from, address to, uint value);
125   function approve(address spender, uint value);
126   event Approval(address indexed owner, address indexed spender, uint value);
127
```

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

LINE 183

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

- JK.sol

Locations

```
182    */
183    function allowance(address _owner, address _spender) constant returns (uint
remaining) {
184    return allowed[_owner][_spender];
185    }
186
187
```

SWC-111 | USE OF THE "THROW" KEYWORD IS DEPRECATED.

LINE 58

low SEVERITY

"throw" is disallowed as of Solidity version 0.5.0. Use one of "revert()", "require()" or "assert()" instead

Source File

- JK.sol

Locations

```
57   if (!assertion) {
58   throw;
59   }
60   }
61   }
62
```

SWC-111 | USE OF THE "THROW" KEYWORD IS DEPRECATED.

LINE 90

low SEVERITY

"throw" is disallowed as of Solidity version 0.5.0. Use one of "revert()", "require()" or "assert()" instead

Source File

- JK.sol

Locations

```
89   if(msg.data.length < size + 4) {
90   throw;
91   }
92   _;
93   }
94
```

SWC-111 | USE OF THE "THROW" KEYWORD IS DEPRECATED.

LINE 171

low SEVERITY

"throw" is disallowed as of Solidity version 0.5.0. Use one of "revert()", "require()" or "assert()" instead

Source File

- JK.sol

Locations

```
170   //  https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
171   if ((_value != 0) && (allowed[msg.sender][_spender] != 0)) throw;
172
173   allowed[msg.sender][_spender] = _value;
174   Approval(msg.sender, _spender, _value);
175
```

SWC-111 | USE OF THE "THROW" KEYWORD IS DEPRECATED.

LINE 213

low SEVERITY

"throw" is disallowed as of Solidity version 0.5.0. Use one of "revert()", "require()" or "assert()" instead

Source File

- JK.sol

Locations

```
212   if (msg.sender != owner) {
213   throw;
214   }
215   _;
216   }
217
```

SWC-111 | USE OF THE "THROW" KEYWORD IS DEPRECATED.

LINE 248

low SEVERITY

"throw" is disallowed as of Solidity version 0.5.0. Use one of "revert()", "require()" or "assert()" instead

Source File

- JK.sol

Locations

```
modifier canMint() {
if(mintingFinished) throw;
_;
}
```

SWC-111 | USE OF THE "THROW" KEYWORD IS DEPRECATED.

LINE 292

low SEVERITY

"throw" is disallowed as of Solidity version 0.5.0. Use one of "revert()", "require()" or "assert()" instead

Source File

- JK.sol

Locations

```
291   modifier whenNotPaused() {
292   if (paused) throw;
293   _;
294   }
295
296
```

SWC-111 | USE OF THE "THROW" KEYWORD IS DEPRECATED.

LINE 300

low SEVERITY

"throw" is disallowed as of Solidity version 0.5.0. Use one of "revert()", "require()" or "assert()" instead

Source File

- JK.sol

Locations

```
299   modifier whenPaused {
300   if (!paused) throw;
301   _;
302   }
303
304
```

SWC-111 | USE OF THE "VAR" KEYWORD IS DEPRECATED.

LINE 149

low SEVERITY

The keyword "var" is disallowed as of Solidity version 0.5.0. It is not possible anymore to create variable declarations without static types. Note that it is always preferable to be as explicit as possible when writing Solidity code.

Source File

- JK.sol

Locations

```
148   function transferFrom(address _from, address _to, uint _value) onlyPayloadSize(3 * 32) {
149   var _allowance = allowed[_from][msg.sender];
150
151   // Check is not needed because sub(_allowance, _value) will already throw if this condition is not met
152   // if (_value > _allowance) throw;
153
```

SWC-119 | STATE VARIABLE SHADOWS ANOTHER STATE VARIABLE.

LINE 244

low SEVERITY

The state variable "totalSupply" in contract "MintableToken" shadows another state variable with the same name "totalSupply" in contract "ERC20Basic".

Source File

- JK.sol

Locations

```
243   bool public mintingFinished = false;
244   uint public totalSupply = 0;
245
246
247   modifier canMint() {
248
```

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.