CertiK Audit Report For Reserve

Request Date: 2019-05-16
Revision Date: 2019-05-21
Platform Name: Ethereum

CERTIK

Contents

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 Verification Services Agreement between CertiK and Reserve(the "Company"), or the scope of services/verification, and terms and conditions provided to the Company in connection with the verification (collectively, 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 without CertiK's prior written consent.

Exective Summary

This report has been prepared as product of the Smart Contract Audit request by Reserve. This audit was conducted to discover issues and vulnerabilities in the source code of Reserve's Smart Contracts. Utilizing CertiK's Formal Verification Platform, Static Analysis and Manual Review, a comprehensive examination has been performed. The auditing process pays special attention to the following considerations.

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessment of the codebase for best practice and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line by line manual review of the entire codebase by industry experts.

Vulnerability Classification

For every issues found, CertiK categorizes them into 3 buckets based on its risk level:

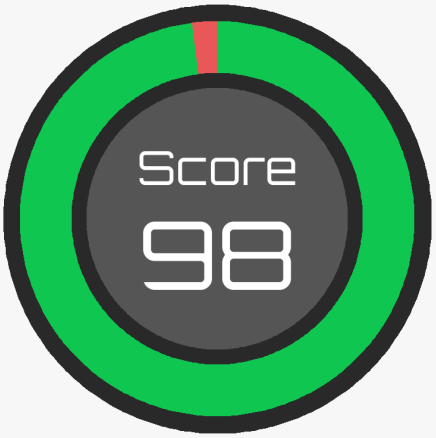
- Critical: The code implementation does not match the specification, or it could result in loss of funds for contract owner or users.
- Medium: The code implementation does not match the specification at certain condition, or it could affect the security standard by lost of access control.
- Low: The code implementation is not a best practice, or use a suboptimal design pattern, which may lead to security vulnerability, but no concern found yet.

Testing Summary

PASS

CERTIK *believes this smart contract passes security qualifications to be listed on digital asset exchanges.*

May 21, 2019



Type of Issues

CertiK smart label engine applied 100% coveraged formal verification labels on the source code, and scanned the code using our proprietary static analysis and formal verification engine to detect the follow type of issues.

Title	Description	Issues	SWC ID
Integer Overflow and Underflow	An overflow/underflow happens when an arithmetic operation reaches the maximum or minimum size of a type.	0	SWC-101
Function incorrectness	Function implementation does not meet the specification, leading to intentional or unintentional vulnerabilities.	0	
Buffer Overflow	An attacker is able to write to arbitrary storage locations of a contract if array of out bound happens	0	SWC-124
Reentrancy	A malicious contract can call back into the calling contract before the first invocation of the function is finished.	0	SWC-107
Transaction Order Dependence	A race condition vulnerability occurs when code depends on the order of the transactions submitted to it.	0	SWC-114
Timestamp Dependence	Timestamp can be influenced by minors to some degree.	0	SWC-116
Insecure Compiler Version	Using an fixed outdated compiler version or floating pragma can be problematic, if there are publicly disclosed bugs and issues that affect the current compiler version used.	0	SWC-102 SWC-103
Insecure Randomness	Block attributes are insecure to generate random numbers, as they can be influenced by minors to some degree.	0	SWC-120

"tx.origin" for authorization	tx.origin should not be used for authorization. Use msg.sender instead.	0	SWC-115
Delegatecall to Untrusted Callee	Calling into untrusted contracts is very dangerous, the target and arguments provided must be sanitized.	0	SWC-112
State Variable Default Visibility	Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.	0	SWC-108
Function Default Visibility	Functions are public by default. A malicious user is able to make unauthorized or unintended state changes if a developer forgot to set the visibility.	0	SWC-100
Uninitialized variables	Uninitialized local storage variables can point to other unexpected storage variables in the contract.	0	SWC-109
Assertion Failure	The assert() function is meant to assert invariants. Properly functioning code should never reach a failing assert statement.	0	SWC-110
Deprecated Solidity Features	Several functions and operators in Solidity are deprecated and should not be used as best practice.	0	SWC-111
Unused variables	Unused variables reduce code quality	0	

Vulnerability Details

Critical

No issue found.

Medium

No issue found.

Low

Issue 1:

- Issue 1 `code`.
- Issue 1 *emphsis*.

Formal Verification Results

How to read

Detail for Request 1

transferFrom to same address

Verification date	20, Oct 2018
Verification timespan	395.38 ms

CERTIK *label location*	Line 30-34 in File howtoread.sol

CERTIK *label*

```
30  /*@CTK FAIL "transferFrom to same address"
31      @tag assume_completion
32      @pre from == to
33      @post __post.allowed[from][msg.sender] ==
34  */
```

Raw code location	Line 35-41 in File howtoread.sol

Raw code

```
35  function transferFrom(address from, address to
        ) {
36      balances[from] = balances[from].sub(tokens
37      allowed[from][msg.sender] = allowed[from][
38      balances[to] = balances[to].add(tokens);
39      emit Transfer(from, to, tokens);
40      return true;
41  }
```

Counterexample This code violates the specification

Initial environment

```
1  Counter Example:
2  Before Execution:
3      Input = {
4          from = 0x0
5          to = 0x0
6          tokens = 0x6c
7      }
8      This = 0
53         balance: 0x0
54     }
55 }
56
```

Post environment

```
57 After Execution:
58     Input = {
59         from = 0x0
60         to = 0x0
61         tokens = 0x6c
```

Formal Verification Request 1

If method completes, integer overflow would not happen.

21, May 2019
7.74 ms

Line 67 in File SlowWallet.sol

```
67    //@CTK NO_OVERFLOW
```

Line 73-76 in File SlowWallet.sol

```
73    constructor(address tokenAddress) public {
74        token = IERC20(tokenAddress);
75        owner = msg.sender;
76    }
```

The code meets the specification

Formal Verification Request 2

Buffer overflow / array index out of bound would never happen.

21, May 2019
0.36 ms

Line 68 in File SlowWallet.sol

```
68    //@CTK NO_BUF_OVERFLOW
```

Line 73-76 in File SlowWallet.sol

```
73    constructor(address tokenAddress) public {
74        token = IERC20(tokenAddress);
75        owner = msg.sender;
76    }
```

The code meets the specification

Formal Verification Request 3

Method will not encounter an assertion failure.

21, May 2019
0.35 ms

Line 69 in File SlowWallet.sol

```
69    //@CTK NO_ASF
```

Line 73-76 in File SlowWallet.sol

```
73    constructor(address tokenAddress) public {
74        token = IERC20(tokenAddress);
75        owner = msg.sender;
76    }
```

The code meets the specification

Formal Verification Request 4

SlowWallet constructor correctness

21, May 2019
0.77 ms

Line 70-72 in File SlowWallet.sol

```
70  /*@CTK "SlowWallet constructor correctness"
71    @post __post.owner == msg.sender
72   */
```

Line 73-76 in File SlowWallet.sol

```
73  constructor(address tokenAddress) public {
74      token = IERC20(tokenAddress);
75      owner = msg.sender;
76  }
```

The code meets the specification

Formal Verification Request 5

Buffer overflow / array index out of bound would never happen.

21, May 2019
28.41 ms

Line 84 in File SlowWallet.sol

```
84  //@CTK NO_BUF_OVERFLOW
```

Line 97-117 in File SlowWallet.sol

```
97  function propose(address destination, uint256 value, string calldata notes)
        external onlyOwner {
98      // Delay by at least two weeks.
99      // We are relying on block.timestamp for this, and aware of the possibility of
            its
100     // manipulation by miners. But we are working at a timescale that is already
            much
101     // longer than the variance in timestamps we have observed and expect in the
            future,
102     // so we are satisfied with this choice.
103     // solium-disable-next-line security/no-block-members
104     uint256 delayUntil = now + delay;
105     require(delayUntil >= now, "delay overflowed");
106
107     proposals[proposalsLength] = TransferProposal(
108         destination,
109         value,
110         delayUntil,
111         notes,
112         false
113     );
114     proposalsLength++;
115
```

```
        emit TransferProposed(proposalsLength-1, destination, value, delayUntil, notes);
    }
```

The code meets the specification

Formal Verification Request 6

Method will not encounter an assertion failure.

21, May 2019
0.81 ms

Line 85 in File SlowWallet.sol

```
    //@CTK NO_ASF
```

Line 97-117 in File SlowWallet.sol

```
    function propose(address destination, uint256 value, string calldata notes) external onlyOwner {
        // Delay by at least two weeks.
        // We are relying on block.timestamp for this, and aware of the possibility of its
        // manipulation by miners. But we are working at a timescale that is already much
        // longer than the variance in timestamps we have observed and expect in the future,
        // so we are satisfied with this choice.
        // solium-disable-next-line security/no-block-members
        uint256 delayUntil = now + delay;
        require(delayUntil >= now, "delay overflowed");

        proposals[proposalsLength] = TransferProposal(
            destination,
            value,
            delayUntil,
            notes,
            false
        );
        proposalsLength++;

        emit TransferProposed(proposalsLength-1, destination, value, delayUntil, notes);
    }
```

The code meets the specification

Formal Verification Request 7

SlowWallet propose correctness

21, May 2019
6.04 ms

Line 86-96 in File SlowWallet.sol

```
86    /*@CTK "SlowWallet propose correctness"
87      @tag assume_completion
88      @pre now + delay > now
89      @post owner == msg.sender
90      @post __post.proposals[proposalsLength].destination == destination
91      @post __post.proposals[proposalsLength].value == value
92      @post __post.proposals[proposalsLength].notes == notes
93      @post __post.proposals[proposalsLength].time == now + delay
94      @post __post.proposals[proposalsLength].closed == false
95      @post __post.proposalsLength == proposalsLength + 1
96     */
```

Line 97-117 in File SlowWallet.sol

```
97    function propose(address destination, uint256 value, string calldata notes)
          external onlyOwner {
98        // Delay by at least two weeks.
99        // We are relying on block.timestamp for this, and aware of the possibility of
              its
100       // manipulation by miners. But we are working at a timescale that is already
              much
101       // longer than the variance in timestamps we have observed and expect in the
              future,
102       // so we are satisfied with this choice.
103       // solium-disable-next-line security/no-block-members
104       uint256 delayUntil = now + delay;
105       require(delayUntil >= now, "delay overflowed");
106
107       proposals[proposalsLength] = TransferProposal(
108           destination,
109           value,
110           delayUntil,
111           notes,
112           false
113       );
114       proposalsLength++;
115
116       emit TransferProposed(proposalsLength-1, destination, value, delayUntil, notes)
              ;
117   }
```

✅ The code meets the specification

Formal Verification Request 8

If method completes, integer overflow would not happen.

📅 21, May 2019
⏱ 87.01 ms

Line 120 in File SlowWallet.sol

```
120   //@CTK NO_OVERFLOW
```

Line 132-139 in File SlowWallet.sol

```
132   function cancel(uint256 index, address addr, uint256 value) external onlyOwner {
133       // Check authorization.
```

```
        requireMatchingOpenProposal(index, addr, value);

        // Cancel transfer.
        proposals[index].closed = true;
        emit TransferCancelled(index, addr, value, proposals[index].notes);
    }
```

The code meets the specification

Formal Verification Request 9

Buffer overflow / array index out of bound would never happen.

21, May 2019
3.53 ms

Line 121 in File SlowWallet.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 132-139 in File SlowWallet.sol

```
    function cancel(uint256 index, address addr, uint256 value) external onlyOwner {
        // Check authorization.
        requireMatchingOpenProposal(index, addr, value);

        // Cancel transfer.
        proposals[index].closed = true;
        emit TransferCancelled(index, addr, value, proposals[index].notes);
    }
```

The code meets the specification

Formal Verification Request 10

Method will not encounter an assertion failure.

21, May 2019
3.67 ms

Line 122 in File SlowWallet.sol

```
    //@CTK NO_ASF
```

Line 132-139 in File SlowWallet.sol

```
    function cancel(uint256 index, address addr, uint256 value) external onlyOwner {
        // Check authorization.
        requireMatchingOpenProposal(index, addr, value);

        // Cancel transfer.
        proposals[index].closed = true;
        emit TransferCancelled(index, addr, value, proposals[index].notes);
    }
```

The code meets the specification

Formal Verification Request 11

SlowWallet cancel correctness

21, May 2019
6.44 ms

Line 123-131 in File SlowWallet.sol

```
123 /*@CTK "SlowWallet cancel correctness"
124   @tag assume_completion
125   @post msg.sender == owner
126   @post index < proposalsLength
127   @post proposals[index].destination == addr
128   @post proposals[index].value == value
129   @post proposals[index].closed == false
130   @post __post.proposals[index].closed == true
131 */
```

Line 132-139 in File SlowWallet.sol

```
132 function cancel(uint256 index, address addr, uint256 value) external onlyOwner {
133     // Check authorization.
134     requireMatchingOpenProposal(index, addr, value);
135
136     // Cancel transfer.
137     proposals[index].closed = true;
138     emit TransferCancelled(index, addr, value, proposals[index].notes);
139 }
```

The code meets the specification

Formal Verification Request 12

If method completes, integer overflow would not happen.

21, May 2019
12.81 ms

Line 142 in File SlowWallet.sol

```
142 //@CTK NO_OVERFLOW
```

Line 150-153 in File SlowWallet.sol

```
150 function voidAll() external onlyOwner {
151     proposalsLength = 0;
152     emit AllTransfersCancelled();
153 }
```

The code meets the specification

Formal Verification Request 13

Buffer overflow / array index out of bound would never happen.

21, May 2019
0.42 ms

Line 143 in File SlowWallet.sol

```
143    //@CTK NO_BUF_OVERFLOW
```

Line 150-153 in File SlowWallet.sol

```
150    function voidAll() external onlyOwner {
151        proposalsLength = 0;
152        emit AllTransfersCancelled();
153    }
```

The code meets the specification

Formal Verification Request 14

Method will not encounter an assertion failure.

21, May 2019
0.41 ms

Line 144 in File SlowWallet.sol

```
144    //@CTK NO_ASF
```

Line 150-153 in File SlowWallet.sol

```
150    function voidAll() external onlyOwner {
151        proposalsLength = 0;
152        emit AllTransfersCancelled();
153    }
```

The code meets the specification

Formal Verification Request 15

SlowWallet voidAll correctness

21, May 2019
1.38 ms

Line 145-149 in File SlowWallet.sol

```
145    /*@CTK "SlowWallet voidAll correctness"
146      @tag assume_completion
147      @post msg.sender == owner
148      @post __post.proposalsLength == 0
149    */
```

Line 150-153 in File SlowWallet.sol

```
150    function voidAll() external onlyOwner {
151        proposalsLength = 0;
152        emit AllTransfersCancelled();
153    }
```

The code meets the specification

Formal Verification Request 16

If method completes, integer overflow would not happen.

21, May 2019
72.1 ms

Line 156 in File SlowWallet.sol

```
156    //@CTK NO_OVERFLOW
```

Line 168-182 in File SlowWallet.sol

```
168    function confirm(uint256 index, address destination, uint256 value) external
           onlyOwner {
169        // Check authorization.
170        requireMatchingOpenProposal(index, destination, value);
171
172        // See commentary above about using ‘now‘.
173        // solium-disable-next-line security/no-block-members
174        require(proposals[index].time < now, "too early");
175
176        // Record execution of transfer.
177        proposals[index].closed = true;
178        emit TransferConfirmed(index, destination, value, proposals[index].notes);
179
180        // Proceed with execution of transfer.
181        require(token.transfer(destination, value));
182    }
```

The code meets the specification

Formal Verification Request 17

Buffer overflow / array index out of bound would never happen.

21, May 2019
6.47 ms

Line 157 in File SlowWallet.sol

```
157    //@CTK NO_BUF_OVERFLOW
```

Line 168-182 in File SlowWallet.sol

```
168    function confirm(uint256 index, address destination, uint256 value) external
           onlyOwner {
169        // Check authorization.
170        requireMatchingOpenProposal(index, destination, value);
171
172        // See commentary above about using ‘now‘.
173        // solium-disable-next-line security/no-block-members
174        require(proposals[index].time < now, "too early");
175
176        // Record execution of transfer.
177        proposals[index].closed = true;
178        emit TransferConfirmed(index, destination, value, proposals[index].notes);
179
180        // Proceed with execution of transfer.
```

```
        require(token.transfer(destination, value));
    }
```

The code meets the specification

Formal Verification Request 18

Method will not encounter an assertion failure.

21, May 2019
6.73 ms

Line 158 in File SlowWallet.sol

```
    //@CTK NO_ASF
```

Line 168-182 in File SlowWallet.sol

```
    function confirm(uint256 index, address destination, uint256 value) external
        onlyOwner {
        // Check authorization.
        requireMatchingOpenProposal(index, destination, value);

        // See commentary above about using ‘now‘.
        // solium-disable-next-line security/no-block-members
        require(proposals[index].time < now, "too early");

        // Record execution of transfer.
        proposals[index].closed = true;
        emit TransferConfirmed(index, destination, value, proposals[index].notes);

        // Proceed with execution of transfer.
        require(token.transfer(destination, value));
    }
```

The code meets the specification

Formal Verification Request 19

SlowWallet confirm correctness

21, May 2019
9.1 ms

Line 159-167 in File SlowWallet.sol

```
    /*@CTK "SlowWallet confirm correctness"
      @tag assume_completion
      @post msg.sender == owner
      @post index < proposalsLength
      @post proposals[index].destination == destination
      @post proposals[index].value == value
      @post proposals[index].closed == false
      @post __post.proposals[index].closed == true
    */
```

Line 168-182 in File SlowWallet.sol

```
168     function confirm(uint256 index, address destination, uint256 value) external
            onlyOwner {
169         // Check authorization.
170         requireMatchingOpenProposal(index, destination, value);
171
172         // See commentary above about using ‘now‘.
173         // solium-disable-next-line security/no-block-members
174         require(proposals[index].time < now, "too early");
175
176         // Record execution of transfer.
177         proposals[index].closed = true;
178         emit TransferConfirmed(index, destination, value, proposals[index].notes);
179
180         // Proceed with execution of transfer.
181         require(token.transfer(destination, value));
182     }
```

✅ The code meets the specification

Formal Verification Request 20

If method completes, integer overflow would not happen.

📅 21, May 2019
⏱ 388.35 ms

Line 67 in File ReserveRights.sol

```
67  //@CTK NO_OVERFLOW
```

Line 80-84 in File ReserveRights.sol

```
80  function transfer(address to, uint256 value) public returns (bool) {
81    // Tokens belonging to Reserve team members and early investors are locked until
          network launch.
82    require(!reserveTeamMemberOrEarlyInvestor[msg.sender]);
83    return super.transfer(to, value);
84  }
```

✅ The code meets the specification

Formal Verification Request 21

Buffer overflow / array index out of bound would never happen.

📅 21, May 2019
⏱ 10.42 ms

Line 68 in File ReserveRights.sol

```
68  //@CTK NO_BUF_OVERFLOW
```

Line 80-84 in File ReserveRights.sol

```
80  function transfer(address to, uint256 value) public returns (bool) {
81    // Tokens belonging to Reserve team members and early investors are locked until
          network launch.
```

```
  require(!reserveTeamMemberOrEarlyInvestor[msg.sender]);
  return super.transfer(to, value);
}
```

The code meets the specification

Formal Verification Request 22

Method will not encounter an assertion failure.

21, May 2019
9.71 ms

Line 69 in File ReserveRights.sol

```
//@CTK NO_ASF
```

Line 80-84 in File ReserveRights.sol

```
function transfer(address to, uint256 value) public returns (bool) {
  // Tokens belonging to Reserve team members and early investors are locked until network launch.
  require(!reserveTeamMemberOrEarlyInvestor[msg.sender]);
  return super.transfer(to, value);
}
```

The code meets the specification

Formal Verification Request 23

ReserveRightsToken transfer correctness

21, May 2019
112.8 ms

Line 70-79 in File ReserveRights.sol

```
/*@CTK "ReserveRightsToken transfer correctness"
  @tag assume_completion
  @post to != 0x0
  @post value <= _balances[msg.sender]
  @post _paused == false
  @post reserveTeamMemberOrEarlyInvestor[msg.sender] == false
  @post to != msg.sender -> __post._balances[msg.sender] == _balances[msg.sender] - value
  @post to != msg.sender -> __post._balances[to] == _balances[to] + value
  @post to == msg.sender -> __post._balances[msg.sender] == _balances[msg.sender]
 */
```

Line 80-84 in File ReserveRights.sol

```
function transfer(address to, uint256 value) public returns (bool) {
  // Tokens belonging to Reserve team members and early investors are locked until network launch.
  require(!reserveTeamMemberOrEarlyInvestor[msg.sender]);
  return super.transfer(to, value);
}
```

The code meets the specification

Formal Verification Request 24

If method completes, integer overflow would not happen.

21, May 2019
340.96 ms

Line 86 in File ReserveRights.sol

```
86 //@CTK NO_OVERFLOW
```

Line 100-104 in File ReserveRights.sol

```
100 function transferFrom(address from, address to, uint256 value) public returns (bool)
        {
101   // Tokens belonging to Reserve team members and early investors are locked until
        network launch.
102   require(!reserveTeamMemberOrEarlyInvestor[from]);
103   return super.transferFrom(from, to, value);
104 }
```

The code meets the specification

Formal Verification Request 25

Buffer overflow / array index out of bound would never happen.

21, May 2019
10.53 ms

Line 87 in File ReserveRights.sol

```
87 //@CTK NO_BUF_OVERFLOW
```

Line 100-104 in File ReserveRights.sol

```
100 function transferFrom(address from, address to, uint256 value) public returns (bool)
        {
101   // Tokens belonging to Reserve team members and early investors are locked until
        network launch.
102   require(!reserveTeamMemberOrEarlyInvestor[from]);
103   return super.transferFrom(from, to, value);
104 }
```

The code meets the specification

Formal Verification Request 26

Method will not encounter an assertion failure.

21, May 2019
9.68 ms

Line 88 in File ReserveRights.sol

```
88  //@CTK NO_ASF
```

Line 100-104 in File ReserveRights.sol

```
100 function transferFrom(address from, address to, uint256 value) public returns (bool)
        {
101   // Tokens belonging to Reserve team members and early investors are locked until
        network launch.
102   require(!reserveTeamMemberOrEarlyInvestor[from]);
103   return super.transferFrom(from, to, value);
104 }
```

The code meets the specification

Formal Verification Request 27

ReserveRightsToken transferFrom correctness

21, May 2019
353.85 ms

Line 89-99 in File ReserveRights.sol

```
89  /*@CTK "ReserveRightsToken transferFrom correctness"
90    @tag assume_completion
91    @post to != 0x0
92    @post value <= _balances[from] && value <= _allowed[from][msg.sender]
93    @post _paused == false
94    @post reserveTeamMemberOrEarlyInvestor[from] == false
95    @post to != from -> __post._balances[from] == _balances[from] - value
96    @post to != from -> __post._balances[to] == _balances[to] + value
97    @post to == from -> __post._balances[from] == _balances[from]
98    @post __post._allowed[from][msg.sender] == _allowed[from][msg.sender] - value
99   */
```

Line 100-104 in File ReserveRights.sol

```
100 function transferFrom(address from, address to, uint256 value) public returns (bool)
        {
101   // Tokens belonging to Reserve team members and early investors are locked until
        network launch.
102   require(!reserveTeamMemberOrEarlyInvestor[from]);
103   return super.transferFrom(from, to, value);
104 }
```

The code meets the specification

Static Analysis Results

INSECURE_COMPILER_VERSION

Line 1 in File SlowWallet.sol

```
1 pragma solidity ^0.5.8;
```

Version to compile has the following bug: 0.5.8: DynamicConstructorArgumentsClippedABIV2

TIMESTAMP_DEPENDENCY

Line 104 in File SlowWallet.sol

```
104         uint256 delayUntil = now + delay;
```

"now" can be influenced by minors to some degree

TIMESTAMP_DEPENDENCY

Line 105 in File SlowWallet.sol

```
105         require(delayUntil >= now, "delay overflowed");
```

"now" can be influenced by minors to some degree

TIMESTAMP_DEPENDENCY

Line 174 in File SlowWallet.sol

```
174         require(proposals[index].time < now, "too early");
```

"now" can be influenced by minors to some degree

INSECURE_COMPILER_VERSION

Line 1 in File ReserveRights.sol

```
1 pragma solidity ^0.4.24;
```

Version to compile has the following bug: 0.4.24: DynamicConstructorArgumentsClippedABIV2, UninitializedFunctionPointerInConstructor_0.4.x, IncorrectEventSignatureInLibraries_0.4.x, ABIEncoderV2PackedStorage_0.4.x, ExpExponentCleanup, EventStructWrongData 0.4.25: DynamicConstructorArgumentsClippedABIV2, UninitializedFunctionPointerInConstructor_0.4.x, IncorrectEventSignatureInLibraries_0.4.x, ABIEncoderV2PackedStorage_0.4.x 0.4.26: DynamicConstructorArgumentsClippedABIV2

Manual Review Notes

Review Details

Source Code SHA-256 Checksum

- **ReserveRights.sol** `3543daebc4fddcddb195892320c6234a69d849aa86af3a775b216f411b0f3e4d`
- **TokenVesting.sol** `f00f607605e5fa87d615e37ebbfce543f794f60828ecb1928f2b912da337d364`
- **SlowWallet.sol** `8b90267c9761c4771f8587e4c54b47d79dae7421b2024849c8de0ed4db104991`

Summary

CertiK was chosen by The Reserve team to audit the design and implementations of its to-be released ERC20 based smart contract. The source code has been analyzed under different perspectives and with different tools such as CertiK formal verification engine, as well as manual reviews by smart contract experts. We have been actively collaborating with client-side engineers around potential loopholes and recommended design changes during the audit process. The Reserve team has been actively providing updates to the source code and feedback about the business logic.

Reserve, is a team who has demonstrated their professional, and knowledgeable understanding of the project Rsr https://github.com/reserve-protocol/rsrgithub repository, which is the subject of this audit. As a production ready repository, available source code in solidity, and go, are written in high quality. Unit tests cover the majority of its business scenarios. Readme documents provide brief information for the intentions, functionalities, and responsibilities of each smart contract. The Reserve Team should be commended for providing easy to read, and clean documentation that is approachable for individuals with or without strong technical backgrounds.

Overall we found the smart contracts to follow good practices. To mention a few, 1) the RSR token is the standard implementation with a reasonable amount of features on top of the ERC20 including administrative controls by the token issuer. The team has strictly followed the rules by hardcoding all its team and investor level allocations in the contracts; 2) the TokenVesting contract is prototyped from the open source library and passed the formal verification checkings; 3) SlowWallet is a commonly seen wallet contract with delay functionality, which could better protect the execution of proposals. With the final update of source code and delivery of the audit report, we conclude that the contract is not vulnerable to any classically known anti-patterns or security issues. The audit report itself is not necessarily a guarantee of correctness or trustworthiness, and we always recommend seeking multiple opinions, more test coverage and sandbox deployments before the mainnet release.

Recommendations

Items in this section are low impact to the overall aspects of the smart contracts, thus will let client to decide whether to have those reflected in the final deployed version of source codes.

SlowWallet.sol

- **pragma solidity version** – Recommend using same solidity version across `*.sol`, current `SlowWallet.sol` is using `^0.5.8` and the others are `^0.4.24` .
- **import IERC20** – Recommend importing the interface from the library instead of redefining in source code `openzeppelin-solidity/contracts/token/ERC20/IERC20.sol`.
- **constructor()** – Recommend setting the `delay` variable thru the constructor rather than hard code.
- **mapping (uint256 ...) public proposals** – Recommend setting the proposals mapping as `internal` or `private`, and adding a `lookupProposal(uint256 index)` with `require(index < proposalsLength)` for retrieving valid data.

  One of the concerns is that after `voidAll()` is called, the `proposalLength` is reset to 0. However, all the voided proposals are still accessible by `SlowWallet.proposals(uint256)`.

  To illustrate:

  1. Create 2 new proposes:

     ```
     SlowWallet.propose("0x9aD910FC9414C110B863B34FfdC678Bc640348F6'', 90, ''
         test 1")
     SlowWallet.propose("0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c'', 10, ''
         test 2")
     ```

  2. Check for `proposalLength`: `SlowWallet.proposalsLength() // 2`
  3. Check `record[0]` and `record[1]`:

     ```
     SlowWallet.proposals(0)
     /* 0x9aD910FC9414C110B863B34FfdC678Bc640348F6: {
       1: uint256: value 90
       2: uint256: time 1559610948
       3: string: notes test 1
       4: bool: closed false
     } */
     SlowWallet.proposals(1)
     /* 0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c: {
       1: uint256: value 10
       2: uint256: time 1559612001
       3: string: notes test 2
       4: bool: closed false
     } */
     ```

  4. Void all records: `SlowWallet.voidAll()`
  5. Check for `proposalsLength`: `SlowWallet.proposalsLength() // 0`
  6. Check `record[1]`:

     ```
     SlowWallet.proposals(1)
     /* 0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c: {
       1: uint256: value 10
       2: uint256: time 1559612001
       3: string: notes test 2
       4: bool: closed false
     } */
     ```

Propsals won't be reset as `closed = true` after the `voidAll()` call, which may confuse the users when looking up.

ReserveRightsToken.sol

- **lockMyTokensForever()** – Recommend declaring a constant variable at contract level for the confirmation message "I understand that I am locking my account forever, or at least until the next token upgrade.", since it never changes. One time declaration saves gas in the long run.

  ```
  bytes32 constant private acknowledgement = keccak256(abi.encodePacked("I understand
   that I am locking my account forever, or at least until the next token upgrade."
  ))
  ```

  Also consider adding error message for `require()`.

TokenVesting.sol

- **constructor()** – Consider providing error messages for `require()`.
- **release()** – Consider providing an error message for `require(unreleased > 0)`.

Source Code with CertiK Labels

File SlowWallet.sol

```
pragma solidity ^0.5.8;

/**
 * @title The standard ERC20 interface
 * @dev see https://eips.ethereum.org/EIPS/eip-20
 */
interface IERC20 {
    /*@CTK "transfer mock"
      @tag spec
      @post !__has_assertion_failure
      @post !__has_buf_overflow
      @post __addr_map == __addr_map__post
      @post msg == msg__post
      @post !__reverted -> !__has_overflow
      @post !__reverted -> __return == true
     */
    function transfer(address, uint256) external returns (bool);
    function approve(address, uint256) external returns (bool);
    function transferFrom(address, address, uint256) external returns (bool);
    function totalSupply() external view returns (uint256);
    function balanceOf(address) external view returns (uint256);
    function allowance(address, address) external view returns (uint256);
    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed holder, address indexed spender, uint256 value);
}

/// @title Time-delayed ERC-20 wallet contract.
/// Can only transfer tokens after publicly recording the intention to do so
/// at least two weeks in advance.
contract SlowWallet {

    // TYPES

    struct TransferProposal {
        address destination;
        uint256 value;
        uint256 time;
        string notes;
        bool closed;
    }

    // DATA

    IERC20 public token;
    uint256 public constant delay = 2 weeks;
    address public owner;

    // PROPOSALS

    mapping (uint256 => TransferProposal) public proposals;
    uint256 public proposalsLength;

    // EVENTS

```

```
    event TransferProposed(
        uint256 index,
        address indexed destination,
        uint256 value,
        uint256 delayUntil,
        string notes
    );
    event TransferConfirmed(uint256 index, address indexed destination, uint256 value,
         string notes);
    event TransferCancelled(uint256 index, address indexed destination, uint256 value,
         string notes);
    event AllTransfersCancelled();

    // FUNCTIONALITY
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK "SlowWallet constructor correctness"
      @post __post.owner == msg.sender
     */
    constructor(address tokenAddress) public {
        token = IERC20(tokenAddress);
        owner = msg.sender;
    }

    modifier onlyOwner() {
        require(msg.sender == owner, "must be owner");
        _;
    }

    /// Propose a new transfer, which can be confirmed after two weeks.
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK "SlowWallet propose correctness"
      @tag assume_completion
      @pre now + delay > now
      @post owner == msg.sender
      @post __post.proposals[proposalsLength].destination == destination
      @post __post.proposals[proposalsLength].value == value
      @post __post.proposals[proposalsLength].notes == notes
      @post __post.proposals[proposalsLength].time == now + delay
      @post __post.proposals[proposalsLength].closed == false
      @post __post.proposalsLength == proposalsLength + 1
     */
    function propose(address destination, uint256 value, string calldata notes)
        external onlyOwner {
        // Delay by at least two weeks.
        // We are relying on block.timestamp for this, and aware of the possibility of
            its
        // manipulation by miners. But we are working at a timescale that is already
            much
        // longer than the variance in timestamps we have observed and expect in the
            future,
        // so we are satisfied with this choice.
        // solium-disable-next-line security/no-block-members
        uint256 delayUntil = now + delay;
        require(delayUntil >= now, "delay overflowed");

```

```
        proposals[proposalsLength] = TransferProposal(
            destination,
            value,
            delayUntil,
            notes,
            false
        );
        proposalsLength++;

        emit TransferProposed(proposalsLength-1, destination, value, delayUntil, notes);
    }

    /// Cancel a proposed transfer.
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK "SlowWallet cancel correctness"
      @tag assume_completion
      @post msg.sender == owner
      @post index < proposalsLength
      @post proposals[index].destination == addr
      @post proposals[index].value == value
      @post proposals[index].closed == false
      @post __post.proposals[index].closed == true
    */
    function cancel(uint256 index, address addr, uint256 value) external onlyOwner {
        // Check authorization.
        requireMatchingOpenProposal(index, addr, value);

        // Cancel transfer.
        proposals[index].closed = true;
        emit TransferCancelled(index, addr, value, proposals[index].notes);
    }

    /// Mark all proposals "void", in O(1).
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK "SlowWallet voidAll correctness"
      @tag assume_completion
      @post msg.sender == owner
      @post __post.proposalsLength == 0
    */
    function voidAll() external onlyOwner {
        proposalsLength = 0;
        emit AllTransfersCancelled();
    }

    /// Confirm and execute a proposed transfer, if enough time has passed since it was proposed.
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK "SlowWallet confirm correctness"
      @tag assume_completion
      @post msg.sender == owner
      @post index < proposalsLength
```

```
     @post proposals[index].destination == destination
     @post proposals[index].value == value
     @post proposals[index].closed == false
     @post __post.proposals[index].closed == true
   */
   function confirm(uint256 index, address destination, uint256 value) external onlyOwner {
       // Check authorization.
       requireMatchingOpenProposal(index, destination, value);

       // See commentary above about using `now`.
       // solium-disable-next-line security/no-block-members
       require(proposals[index].time < now, "too early");

       // Record execution of transfer.
       proposals[index].closed = true;
       emit TransferConfirmed(index, destination, value, proposals[index].notes);

       // Proceed with execution of transfer.
       require(token.transfer(destination, value));
   }

   /// Throw unless the given transfer proposal exists and matches `destination` and `value`.
   function requireMatchingOpenProposal(uint256 index, address destination, uint256 value) private view {
       require(index < proposalsLength, "index too high, or transfer voided");
       require(!proposals[index].closed, "transfer already closed");

       // Slither reports "dangerous strict equality" for each of these, but it's OK.
       // These equalities are to confirm that the transfer entered is equal to the
       // matching previous transfer. We're vetting data entry; strict equality is appropriate.
       require(proposals[index].destination == destination, "destination mismatched");
       require(proposals[index].value == value, "value mismatched");
   }
}
```

File ReserveRights.sol

```
pragma solidity ^0.4.24;

import "openzeppelin-solidity/contracts/token/ERC20/IERC20.sol";
import "openzeppelin-solidity/contracts/token/ERC20/ERC20Pausable.sol";

contract ReserveRightsToken is ERC20Pausable {
  string public name = "Reserve Rights";
  string public symbol = "RSR";
  uint8 public decimals = 18;

  // Tokens belonging to Reserve team members and early investors are locked until network launch.
  mapping (address => bool) public reserveTeamMemberOrEarlyInvestor;
  event AccountLocked(address indexed lockedAccount);

  // Hard-coded addresses from the previous deployment, which should be locked and contain token allocations.
  address[] previousAddresses = [
    0x8ad9c8ebe26eadab9251b8fc36cd06a1ec399a7f,
```

```
    0xb268c230720d16c69a61cbee24731e3b2a3330a1,
    0x082705fabf49bd30de8f0222821f6d940713b89d,
    0xc3aa4ced5dea58a3d1ca76e507515c79ca1e4436,
    0x66f25f036eb4463d8a45c6594a325f9e89baa6db,
    0x9e454fe7d8e087fcac4ec8c40562de781004477e,
    0x4fcc7ca22680aed155f981eeb13089383d624aa9,
    0x5a66650e5345d76eb8136ea1490cbcce1c08072e,
    0x698a10b5d0972bffea306ba5950bd74d2af3c7ca,
    0xdf437625216cca3d7148e18d09f4aab0d47c763b,
    0x24b4a6847ccb32972de40170c02fda121ddc6a30,
    0x8d29a24f91df381feb4ee7f05405d3fb888c643e,
    0x5a7350d95b9e644dcab4bc642707f43a361bf628,
    0xfc2e9a5cd1bb9b3953ffa7e6ddf0c0447eb95f11,
    0x3ac7a6c3a2ff08613b611485f795d07e785cbb95,
    0x47fc47cbcc5217740905e16c4c953b2f247369d2,
    0xd282337950ac6e936d0f0ebaaff1ffc3de79f3d5,
    0xde59cd3aa43a2bf863723662b31906660c7d12b6,
    0x5f84660cabb98f7b7764cd1ae2553442da91984e,
    0xefbaaf73fc22f70785515c1e2be3d5ba2fb8e9b0,
    0x63c5ffb388d83477a15eb940cfa23991ca0b30f0,
    0x14f018cce044f9d3fb1e1644db6f2fab70f6e3cb,
    0xbe30069d27a250f90c2ee5507bcaca5f868265f7,
    0xcfef27288bedcd587a1ed6e86a996c8c5b01d7c1,
    0x5f57bbccc7ffa4c46864b5ed999a271bc36bb0ce,
    0xbae85de9858375706dde5907c8c9c6ee22b19212,
    0x5cf4bbb0ff093f3c725abec32fba8f34e4e98af1,
    0xcb2d434bf72d3cd43d0c368493971183640ffe99,
    0x02fc8e99401b970c265480140721b28bb3af85ab,
    0xe7ad11517d7254f6a0758cee932bffa328002dd0,
    0x6b39195c164d693d3b6518b70d99877d4f7c87ef,
    0xc59119d8e4d129890036a108aed9d9fe94db1ba9,
    0xd28661e4c75d177d9c1f3c8b821902c1abd103a6,
    0xba385610025b1ea8091ae3e4a2e98913e2691ff7,
    0xcd74834b8f3f71d2e82c6240ae0291c563785356,
    0x657a127639b9e0ccccfbe795a8e394d5ca158526
  ];

  constructor(address previousContract, address reservePrimaryWallet) public {
    IERC20 previousToken = IERC20(previousContract);

    _mint(reservePrimaryWallet, previousToken.balanceOf(reservePrimaryWallet));

    for (uint i = 0; i < previousAddresses.length; i++) {
      reserveTeamMemberOrEarlyInvestor[previousAddresses[i]] = true;
      _mint(previousAddresses[i], previousToken.balanceOf(previousAddresses[i]));
      emit AccountLocked(previousAddresses[i]);
    }
  }

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK "ReserveRightsToken transfer correctness"
    @tag assume_completion
    @post to != 0x0
    @post value <= _balances[msg.sender]
    @post _paused == false
    @post reserveTeamMemberOrEarlyInvestor[msg.sender] == false
```

```
    @post to != msg.sender -> __post._balances[msg.sender] == _balances[msg.sender] - value
    @post to != msg.sender -> __post._balances[to] == _balances[to] + value
    @post to == msg.sender -> __post._balances[msg.sender] == _balances[msg.sender]
   */
  function transfer(address to, uint256 value) public returns (bool) {
    // Tokens belonging to Reserve team members and early investors are locked until network launch.
    require(!reserveTeamMemberOrEarlyInvestor[msg.sender]);
    return super.transfer(to, value);
  }

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK "ReserveRightsToken transferFrom correctness"
    @tag assume_completion
    @post to != 0x0
    @post value <= _balances[from] && value <= _allowed[from][msg.sender]
    @post _paused == false
    @post reserveTeamMemberOrEarlyInvestor[from] == false
    @post to != from -> __post._balances[from] == _balances[from] - value
    @post to != from -> __post._balances[to] == _balances[to] + value
    @post to == from -> __post._balances[from] == _balances[from]
    @post __post._allowed[from][msg.sender] == _allowed[from][msg.sender] - value
   */
  function transferFrom(address from, address to, uint256 value) public returns (bool) {
    // Tokens belonging to Reserve team members and early investors are locked until network launch.
    require(!reserveTeamMemberOrEarlyInvestor[from]);
    return super.transferFrom(from, to, value);
  }

  /// This function is intended to be used only by Reserve team members and investors.
  /// You can call it yourself, but you almost certainly dont want to.
  /// Anyone who calls this function will cause their own tokens to be subject to
  /// a long lockup. Reserve team members and some investors do this to commit
  /// ourselves to not dumping tokens early. If you are not a Reserve team member
  /// or investor, you dont need to limit yourself in this way.
  ///
  /// THIS FUNCTION LOCKS YOUR TOKENS. ONLY USE IT IF YOU KNOW WHAT YOU ARE DOING.
  function lockMyTokensForever(string consent) public returns (bool) {
    require(keccak256(abi.encodePacked(consent)) == keccak256(abi.encodePacked(
      "I understand that I am locking my account forever, or at least until the next token upgrade."
    )));
    reserveTeamMemberOrEarlyInvestor[msg.sender] = true;
    emit AccountLocked(msg.sender);
  }
}
```