BLOSAFE
BLOCKCHAIN SCOPE

Wrapped Bitcoin (WBTC)
Smartcontract Security
Audit Report

2022. 04

From SCOPE

https://blosafe.com

BLOSAFE
BLOCKCHAIN SCOPE

SCOPE

2022. 04

Document History

Date	Name	History
2022.04	Blosafe	Initial

1. Project outline

1.1. Purpose

The purpose of this inspection is to conduct a security audit on the [WBTC] Smartcontract to discover potential hacking weaknesses, analyze the cause, and respond

1.2. Target

The subjects of this inspection are as follows.

No	Category	Addr	Memo
1	Smartcontract	0x2260FAC5E5542a773Aa44fBCfeDf7C193bc2C599	ETH Mainnet

1.3. Schedule

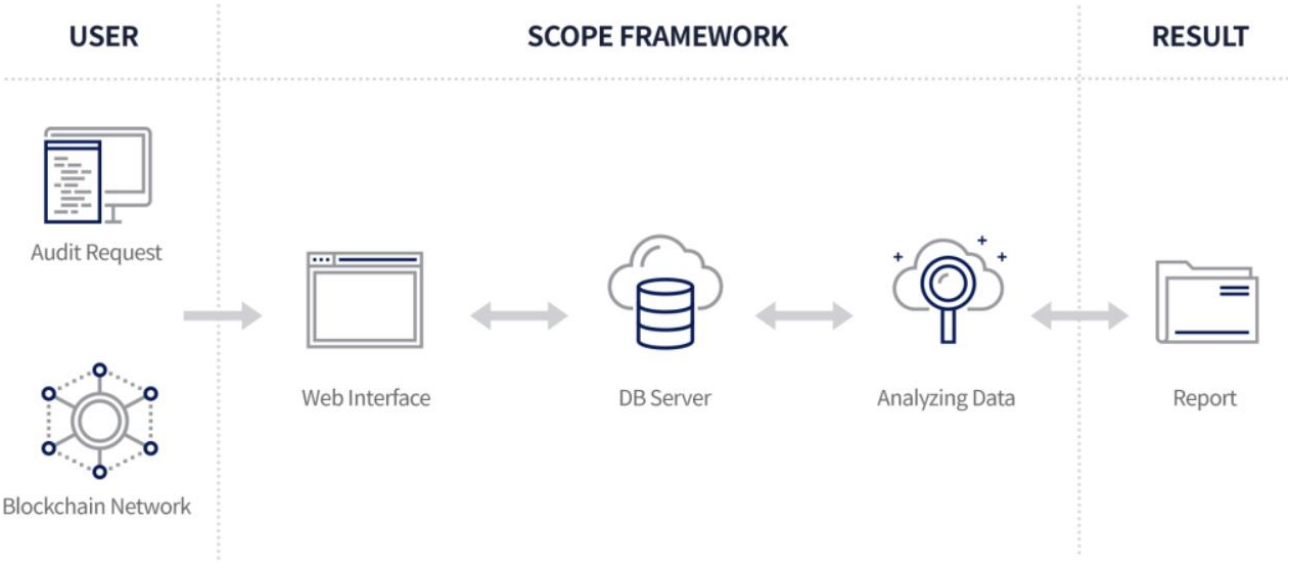
Work	Detail	Timeline	Memo
business consultation	Build Environment	1 day	
Audit	Smartcontract static auditing	2 days	
	Smartcontract Dynamic Auditing	3 days	
Report / review	Report	1 day	
	Review	1 day	

1.4. Environment

업무 구분	Name	Platform	Memo
Audit	Scope Audit	SaaS	

2. Process

2.1. Process Detail



2.2. Check List

No	Detector	What it Detects	Impact	external-function
1	`abiencoderv2-array`	Storage abiencoderv2 array	High	High
2	`array-by-reference`	Modifying storage array by value	High	High
3	`incorrect-shift`	The order of parameters in a shift instruction is incorrect.	High	High
4	`multiple-constructors`	Multiple constructor schemes	High	High
5	`name-reused`	Contract's name reused	High	High
6	`public-mappings-nested`	Public mappings with nested variables	High	High
7	`rtlo`	Right-To-Left-Override control character is used	High	High
8	`shadowing-state`	State variables shadowing	High	High
9	`suicidal`	Functions allowing anyone to destruct the contract	High	High

10	uninitialized-state	Uninitialized state variables	High	High
11	uninitialized-storage	Uninitialized storage variables	High	High
12	unprotected-upgrade	Unprotected upgradeable contract	High	High
13	arbitrary-send	Functions that send Ether to arbitrary destinations	High	Medium
14	controlled-array-length	Tainted array length assignment	High	Medium
15	controlled-delegatecall	Controlled delegatecall destination	High	Medium
16	delegatecall-loop	Payable functions using delegatecall inside a loop	High	Medium
17	msg-value-loop	msg.value inside a loop	High	Medium
18	reentrancy-eth	Reentrancy vulnerabilities (theft of ethers)	High	Medium
19	storage-array	Signed storage integer array compiler bug	High	Medium
20	unchecked-transfer	Unchecked tokens transfer	High	Medium
21	weak-prng	Weak PRNG	High	Medium
22	enum-conversion	Detect dangerous enum conversion	Medium	High
23	erc20-interface	Incorrect ERC20 interfaces	Medium	High
24	erc721-interface	Incorrect ERC721 interfaces	Medium	High
25	incorrect-equality	Dangerous strict equalities	Medium	High
26	locked-ether	Contracts that lock ether	Medium	High
27	mapping-deletion	Deletion on mapping containing a structure	Medium	High
28	shadowing-abstract	State variables shadowing from abstract contracts	Medium	High
29	tautology	Tautology or contradiction	Medium	High
30	write-after-write	Unused write	Medium	High
31	boolean-cst	Misuse of Boolean constant	Medium	Medium

SCOPE

32	constant-function-asm	Constant functions using assembly code	Medium	Medium
33	constant-function-state	Constant functions changing the state	Medium	Medium
34	divide-before-multiply	Imprecise arithmetic operations order	Medium	Medium
35	reentrancy-no-eth	Reentrancy vulnerabilities (no theft of ethers)	Medium	Medium
36	reused-constructor	Reused base constructor	Medium	Medium
37	tx-origin	Dangerous usage of tx.origin	Medium	Medium
38	unchecked-lowlevel	Unchecked low-level calls	Medium	Medium
39	unchecked-send	Unchecked send	Medium	Medium
40	uninitialized-local	Uninitialized local variables	Medium	Medium
41	unused-return	Unused return values	Medium	Medium
42	incorrect-modifier	Modifiers that can return the default value	Low	High
43	shadowing-builtin	Built-in symbol shadowing	Low	High
44	shadowing-local	Local variables shadowing	Low	High
45	uninitialized-fptr-cst	Uninitialized function pointer calls in constructors	Low	High
46	variable-scope	Local variables used prior their declaration	Low	High
47	void-cst	Constructor called not implemented	Low	High
48	calls-loop	Multiple calls in a loop	Low	Medium
49	events-access	Missing Events Access Control	Low	Medium
50	events-maths	Missing Events Arithmetic	Low	Medium
51	incorrect-unary	Dangerous unary expressions	Low	Medium
52	missing-zero-check	Missing Zero Address Validation	Low	Medium
53	reentrancy-benign	Benign reentrancy vulnerabilities	Low	Medium

SCOPE

54	reentrancy-events	Reentrancy vulnerabilities leading to out-of-order Events	Low	Medium
55	timestamp	Dangerous usage of block.timestamp	Low	Medium
56	assembly	Assembly usage	Informational	High
57	assert-state-change	Assert state change	Informational	High
58	boolean-equal	Comparison to boolean constant	Informational	High
59	deprecated-standards	Deprecated Solidity Standards	Informational	High
60	erc20-indexed	Un-indexed ERC20 event parameters	Informational	High
61	function-init-state	Function initializing state variables	Informational	High
62	low-level-calls	Low level calls	Informational	High
63	missing-inheritance	Missing inheritance	Informational	High
64	naming-convention	Conformity to Solidity naming conventions	Informational	High
65	pragma	If different pragma directives are used	Informational	High
66	redundant-statements	Redundant statements	Informational	High
67	solc-version	Incorrect Solidity version	Informational	High
68	unimplemented-functions	Unimplemented functions	Informational	High
69	unused-state	Unused state variables	Informational	High
70	costly-loop	Costly operations in a loop	Informational	Medium
71	dead-code	Functions that are not used	Informational	Medium
72	reentrancy-unlimited-gas	Reentrancy vulnerabilities through send and transfer	Informational	Medium
73	similar-names	Variable names are too similar	Informational	Medium
74	too-many-digits	Conformance to numeric notation best practices	Informational	Medium
75	constable-states	State variables that could be declared constant	Optimization	High

SCOPE

76	external-function	Public function that could be declared external	Optimization	High

SCOPE

Summary of results

2.3. Result

[Passed]

[WBTC] As a result of the Smartcontract security audit, a total of 2 vulnerabilities were found, among which 0 vulnerabilities of 'high', 0 of 'medium' vulnerabilities, 2 of 'low' vulnerabilities, and 54 of 'information' ratings were found.

3. Detailed results

3.1. Smartcontract

```
/**
 *Submitted for verification at Etherscan.io on 2018-11-24
*/

pragma solidity 0.4.24;

// File: openzeppelin-solidity/contracts/token/ERC20/ERC20Basic.sol

/**
 * @title ERC20Basic
 * @dev Simpler version of ERC20 interface
 * See https://github.com/ethereum/EIPs/issues/179
 */
contract ERC20Basic {
  function totalSupply() public view returns (uint256);
  function balanceOf(address _who) public view returns (uint256);
  function transfer(address _to, uint256 _value) public returns (bool);
  event Transfer(address indexed from, address indexed to, uint256 value);
}

// File: openzeppelin-solidity/contracts/math/SafeMath.sol
```

```
/**
 * @title SafeMath
 * @dev Math operations with safety checks that throw on error
 */
library SafeMath {

  /**
  * @dev Multiplies two numbers, throws on overflow.
  */
  function mul(uint256 _a, uint256 _b) internal pure returns (uint256 c) {
    // Gas optimization: this is cheaper than asserting 'a' not being zero, but the
    // benefit is lost if 'b' is also tested.
    // See: https://github.com/OpenZeppelin/openzeppelin-solidity/pull/522
    if (_a == 0) {
      return 0;
    }

    c = _a * _b;
    assert(c / _a == _b);
    return c;
  }

  /**
  * @dev Integer division of two numbers, truncating the quotient.
  */
```

```
  function div(uint256 _a, uint256 _b) internal pure returns (uint256) {
    // assert(_b > 0); // Solidity automatically throws when dividing by 0
    // uint256 c = _a / _b;
    // assert(_a == _b * c + _a % _b); // There is no case in which this doesn't hold
    return _a / _b;
  }

  /**
  * @dev Subtracts two numbers, throws on overflow (i.e. if subtrahend is greater than minuend).
  */
  function sub(uint256 _a, uint256 _b) internal pure returns (uint256) {
    assert(_b <= _a);
    return _a - _b;
  }

  /**
  * @dev Adds two numbers, throws on overflow.
  */
  function add(uint256 _a, uint256 _b) internal pure returns (uint256 c) {
    c = _a + _b;
    assert(c >= _a);
    return c;
  }
}
```

```
// File: openzeppelin-solidity/contracts/token/ERC20/BasicToken.sol


/**
 * @title Basic token
 * @dev Basic version of StandardToken, with no allowances.
 */
contract BasicToken is ERC20Basic {
  using SafeMath for uint256;

  mapping(address => uint256) internal balances;

  uint256 internal totalSupply_;

  /**
  * @dev Total number of tokens in existence
  */
  function totalSupply() public view returns (uint256) {
    return totalSupply_;
  }

  /**
  * @dev Transfer token for a specified address
  * @param _to The address to transfer to.
  * @param _value The amount to be transferred.
  */
```


SCOPE

```
    function transfer(address _to, uint256 _value) public returns (bool) {
        require(_value <= balances[msg.sender]);
        require(_to != address(0));

        balances[msg.sender] = balances[msg.sender].sub(_value);
        balances[_to] = balances[_to].add(_value);
        emit Transfer(msg.sender, _to, _value);
        return true;
    }

    /**
    * @dev Gets the balance of the specified address.
    * @param _owner The address to query the the balance of.
    * @return An uint256 representing the amount owned by the passed address.
    */
    function balanceOf(address _owner) public view returns (uint256) {
        return balances[_owner];
    }

}

// File: openzeppelin-solidity/contracts/token/ERC20/ERC20.sol

/**
 * @title ERC20 interface
```

```
 * @dev see https://github.com/ethereum/EIPs/issues/20
 */
contract ERC20 is ERC20Basic {
  function allowance(address _owner, address _spender)
    public view returns (uint256);

  function transferFrom(address _from, address _to, uint256 _value)
    public returns (bool);

  function approve(address _spender, uint256 _value) public returns (bool);
  event Approval(
    address indexed owner,
    address indexed spender,
    uint256 value
  );
}

// File: openzeppelin-solidity/contracts/token/ERC20/StandardToken.sol

/**
 * @title Standard ERC20 token
 *
 * @dev Implementation of the basic standard token.
 * https://github.com/ethereum/EIPs/issues/20
```

```
 * Based on code by FirstBlood:
https://github.com/Firstbloodio/token/blob/master/smart_contract/FirstBloodToken.sol
 */
contract StandardToken is ERC20, BasicToken {

  mapping (address => mapping (address => uint256)) internal allowed;


  /**
   * @dev Transfer tokens from one address to another
   * @param _from address The address which you want to send tokens from
   * @param _to address The address which you want to transfer to
   * @param _value uint256 the amount of tokens to be transferred
   */
  function transferFrom(
    address _from,
    address _to,
    uint256 _value
  )
    public
    returns (bool)
  {
    require(_value <= balances[_from]);
    require(_value <= allowed[_from][msg.sender]);
    require(_to != address(0));
```


SCOPE

```
    balances[_from] = balances[_from].sub(_value);
    balances[_to] = balances[_to].add(_value);
    allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
    emit Transfer(_from, _to, _value);
    return true;
  }

  /**
   * @dev Approve the passed address to spend the specified amount of tokens on behalf of msg.sender.
   * Beware that changing an allowance with this method brings the risk that someone may use both the old
   * and the new allowance by unfortunate transaction ordering. One possible solution to mitigate this
   * race condition is to first reduce the spender's allowance to 0 and set the desired value afterwards:
   * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
   * @param _spender The address which will spend the funds.
   * @param _value The amount of tokens to be spent.
   */
  function approve(address _spender, uint256 _value) public returns (bool) {
    allowed[msg.sender][_spender] = _value;
    emit Approval(msg.sender, _spender, _value);
    return true;
  }

  /**
```

```
 * @dev Function to check the amount of tokens that an owner allowed to a spender.
 * @param _owner address The address which owns the funds.
 * @param _spender address The address which will spend the funds.
 * @return A uint256 specifying the amount of tokens still available for the spender.
 */
function allowance(
  address _owner,
  address _spender
 )
  public
  view
  returns (uint256)
{
  return allowed[_owner][_spender];
}

/**
 * @dev Increase the amount of tokens that an owner allowed to a spender.
 * approve should be called when allowed[_spender] == 0. To increment
 * allowed value is better to use this function to avoid 2 calls (and wait until
 * the first transaction is mined)
 * From MonolithDAO Token.sol
 * @param _spender The address which will spend the funds.
 * @param _addedValue The amount of tokens to increase the allowance by.
 */
```

```
function increaseApproval(
    address _spender,
    uint256 _addedValue
)
    public
    returns (bool)
{
    allowed[msg.sender][_spender] = (
      allowed[msg.sender][_spender].add(_addedValue));
    emit Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
    return true;
}

/**
 * @dev Decrease the amount of tokens that an owner allowed to a spender.
 * approve should be called when allowed[_spender] == 0. To decrement
 * allowed value is better to use this function to avoid 2 calls (and wait until
 * the first transaction is mined)
 * From MonolithDAO Token.sol
 * @param _spender The address which will spend the funds.
 * @param _subtractedValue The amount of tokens to decrease the allowance by.
 */
function decreaseApproval(
    address _spender,
    uint256 _subtractedValue
```

```
  )
    public
    returns (bool)
  {
    uint256 oldValue = allowed[msg.sender][_spender];
    if (_subtractedValue >= oldValue) {
      allowed[msg.sender][_spender] = 0;
    } else {
      allowed[msg.sender][_spender] = oldValue.sub(_subtractedValue);
    }
    emit Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
    return true;
  }

}

// File: openzeppelin-solidity/contracts/token/ERC20/DetailedERC20.sol

/**
 * @title DetailedERC20 token
 * @dev The decimals are only for visualization purposes.
 * All the operations are done using the smallest and indivisible token unit,
 * just as on Ethereum all the operations are done in wei.
 */
contract DetailedERC20 is ERC20 {
```

```
    string public name;
    string public symbol;
    uint8 public decimals;

    constructor(string _name, string _symbol, uint8 _decimals) public {
        name = _name;
        symbol = _symbol;
        decimals = _decimals;
    }
}

// File: openzeppelin-solidity/contracts/ownership/Ownable.sol

/**
 * @title Ownable
 * @dev The Ownable contract has an owner address, and provides basic authorization control
 * functions, this simplifies the implementation of "user permissions".
 */
contract Ownable {
    address public owner;


    event OwnershipRenounced(address indexed previousOwner);
    event OwnershipTransferred(
        address indexed previousOwner,
```

```
    address indexed newOwner
);



/**
 * @dev The Ownable constructor sets the original `owner` of the contract to the sender
 * account.
 */
constructor() public {
    owner = msg.sender;
}


/**
 * @dev Throws if called by any account other than the owner.
 */
modifier onlyOwner() {
    require(msg.sender == owner);
    _;
}


/**
 * @dev Allows the current owner to relinquish control of the contract.
 * @notice Renouncing to ownership will leave the contract without an owner.
 * It will not be possible to call the functions with the `onlyOwner`
 * modifier anymore.
```

```
    */
    function renounceOwnership() public onlyOwner {
        emit OwnershipRenounced(owner);
        owner = address(0);
    }

    /**
     * @dev Allows the current owner to transfer control of the contract to a newOwner.
     * @param _newOwner The address to transfer ownership to.
     */
    function transferOwnership(address _newOwner) public onlyOwner {
        _transferOwnership(_newOwner);
    }

    /**
     * @dev Transfers control of the contract to a newOwner.
     * @param _newOwner The address to transfer ownership to.
     */
    function _transferOwnership(address _newOwner) internal {
        require(_newOwner != address(0));
        emit OwnershipTransferred(owner, _newOwner);
        owner = _newOwner;
    }
}
```

```
// File: openzeppelin-solidity/contracts/token/ERC20/MintableToken.sol


/**
 * @title Mintable token
 * @dev Simple ERC20 Token example, with mintable token creation
 * Based on code by TokenMarketNet: https://github.com/TokenMarketNet/ico/blob/master/contracts/MintableToken.sol
 */
contract MintableToken is StandardToken, Ownable {
  event Mint(address indexed to, uint256 amount);
  event MintFinished();

  bool public mintingFinished = false;


  modifier canMint() {
    require(!mintingFinished);
    _;
  }

  modifier hasMintPermission() {
    require(msg.sender == owner);
    _;
  }
```

```
/**
 * @dev Function to mint tokens
 * @param _to The address that will receive the minted tokens.
 * @param _amount The amount of tokens to mint.
 * @return A boolean that indicates if the operation was successful.
 */
function mint(
  address _to,
  uint256 _amount
)
  public
  hasMintPermission
  canMint
  returns (bool)
{
  totalSupply_ = totalSupply_.add(_amount);
  balances[_to] = balances[_to].add(_amount);
  emit Mint(_to, _amount);
  emit Transfer(address(0), _to, _amount);
  return true;
}

/**
 * @dev Function to stop minting new tokens.
 * @return True if the operation was successful.
```

```
    */
  function finishMinting() public onlyOwner canMint returns (bool) {
    mintingFinished = true;
    emit MintFinished();
    return true;
  }
}

// File: openzeppelin-solidity/contracts/token/ERC20/BurnableToken.sol

/**
 * @title Burnable Token
 * @dev Token that can be irreversibly burned (destroyed).
 */
contract BurnableToken is BasicToken {

  event Burn(address indexed burner, uint256 value);

  /**
   * @dev Burns a specific amount of tokens.
   * @param _value The amount of token to be burned.
   */
  function burn(uint256 _value) public {
    _burn(msg.sender, _value);
  }
```

```
  function _burn(address _who, uint256 _value) internal {
    require(_value <= balances[_who]);
    // no need to require value <= totalSupply, since that would imply the
    // sender's balance is greater than the totalSupply, which *should* be an assertion failure

    balances[_who] = balances[_who].sub(_value);
    totalSupply_ = totalSupply_.sub(_value);
    emit Burn(_who, _value);
    emit Transfer(_who, address(0), _value);
  }
}

// File: openzeppelin-solidity/contracts/lifecycle/Pausable.sol

/**
 * @title Pausable
 * @dev Base contract which allows children to implement an emergency stop mechanism.
 */
contract Pausable is Ownable {
  event Pause();
  event Unpause();

  bool public paused = false;
```

```
/**
 * @dev Modifier to make a function callable only when the contract is not paused.
 */
modifier whenNotPaused() {
  require(!paused);
  _;
}

/**
 * @dev Modifier to make a function callable only when the contract is paused.
 */
modifier whenPaused() {
  require(paused);
  _;
}

/**
 * @dev called by the owner to pause, triggers stopped state
 */
function pause() public onlyOwner whenNotPaused {
  paused = true;
  emit Pause();
}
```

```
  /**
   * @dev called by the owner to unpause, returns to normal state
   */
  function unpause() public onlyOwner whenPaused {
    paused = false;
    emit Unpause();
  }
}

// File: openzeppelin-solidity/contracts/token/ERC20/PausableToken.sol

/**
 * @title Pausable token
 * @dev StandardToken modified with pausable transfers.
 **/
contract PausableToken is StandardToken, Pausable {

  function transfer(
    address _to,
    uint256 _value
  )
    public
    whenNotPaused
    returns (bool)
  {
```

```
    return super.transfer(_to, _value);
}

function transferFrom(
    address _from,
    address _to,
    uint256 _value
)
    public
    whenNotPaused
    returns (bool)
{
    return super.transferFrom(_from, _to, _value);
}

function approve(
    address _spender,
    uint256 _value
)
    public
    whenNotPaused
    returns (bool)
{
    return super.approve(_spender, _value);
}
```

```
  function increaseApproval(
    address _spender,
    uint _addedValue
  )
    public
    whenNotPaused
    returns (bool success)
  {
    return super.increaseApproval(_spender, _addedValue);
  }

  function decreaseApproval(
    address _spender,
    uint _subtractedValue
  )
    public
    whenNotPaused
    returns (bool success)
  {
    return super.decreaseApproval(_spender, _subtractedValue);
  }
}

// File: openzeppelin-solidity/contracts/ownership/Claimable.sol
```

```
/**
 * @title Claimable
 * @dev Extension for the Ownable contract, where the ownership needs to be claimed.
 * This allows the new owner to accept the transfer.
 */
contract Claimable is Ownable {
  address public pendingOwner;

  /**
   * @dev Modifier throws if called by any account other than the pendingOwner.
   */
  modifier onlyPendingOwner() {
    require(msg.sender == pendingOwner);
    _;
  }

  /**
   * @dev Allows the current owner to set the pendingOwner address.
   * @param newOwner The address to transfer ownership to.
   */
  function transferOwnership(address newOwner) public onlyOwner {
    pendingOwner = newOwner;
  }
```

```
  /**
   * @dev Allows the pendingOwner address to finalize the transfer.
   */
  function claimOwnership() public onlyPendingOwner {
    emit OwnershipTransferred(owner, pendingOwner);
    owner = pendingOwner;
    pendingOwner = address(0);
  }
}

// File: openzeppelin-solidity/contracts/token/ERC20/SafeERC20.sol

/**
 * @title SafeERC20
 * @dev Wrappers around ERC20 operations that throw on failure.
 * To use this library you can add a `using SafeERC20 for ERC20;` statement to your contract,
 * which allows you to call the safe operations as `token.safeTransfer(...)`, etc.
 */
library SafeERC20 {
  function safeTransfer(
    ERC20Basic _token,
    address _to,
    uint256 _value
  )
    internal
```

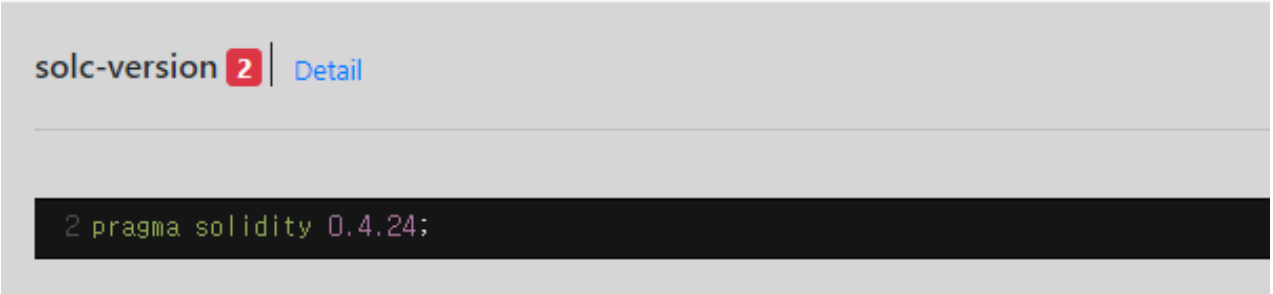
```
    {
        require(_token.transfer(_to, _value));
    }

    function safeTransferFrom(
        ERC20 _token,
        address _from,
        address _to,
        uint256 _value
    )
        internal
    {
        require(_token.transferFrom(_from, _to, _value));
    }

    function safeApprove(
        ERC20 _token,
        address _spender,
        uint256 _value
    )
        internal
    {
        require(_token.approve(_spender, _value));
    }
}
```

```
// File: openzeppelin-solidity/contracts/ownership/CanReclaimToken.sol

/**
 * @title Contracts that should be able to recover tokens
 * @author SylTi
 * @dev This allow a contract to recover any ERC20 token received in a contract by transferring the balance to the contract owner.
 * This will prevent any accidental loss of tokens.
 */
contract CanReclaimToken is Ownable {
  using SafeERC20 for ERC20Basic;

  /**
   * @dev Reclaim all ERC20Basic compatible tokens
   * @param _token ERC20Basic The address of the token contract
   */
  function reclaimToken(ERC20Basic _token) external onlyOwner {
    uint256 balance = _token.balanceOf(this);
    _token.safeTransfer(owner, balance);
  }

}

// File: contracts/utils/OwnableContract.sol
```

```
// empty block is used as this contract just inherits others.
contract OwnableContract is CanReclaimToken, Claimable { } /* solhint-disable-line no-empty-blocks */

// File: contracts/token/WBTC.sol

contract WBTC is StandardToken, DetailedERC20("Wrapped BTC", "WBTC", 8),
    MintableToken, BurnableToken, PausableToken, OwnableContract {

    function burn(uint value) public onlyOwner {
        super.burn(value);
    }

    function finishMinting() public onlyOwner returns (bool) {
        return false;
    }

    function renounceOwnership() public onlyOwner {
        revert("renouncing ownership is blocked");
    }
}
```

3.2. Vulnerability

solc-version



Configuration

- Check: `solc-version`
- Severity: `Informational`
- Confidence: `High`

Description

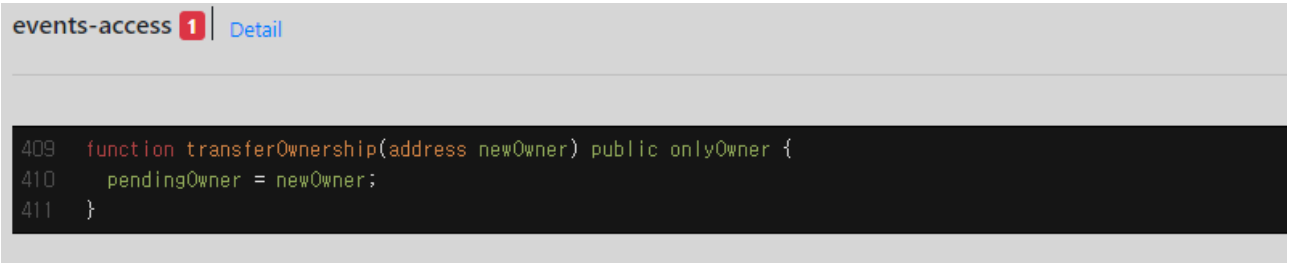
`solc` frequently releases new compiler versions. Using an old version prevents access to new Solidity security checks. We also recommend avoiding complex `pragma` statement.

Recommendation

Deploy with any of the following Solidity versions:

- 0.5.16 - 0.5.17
- 0.6.11 - 0.6.12
- 0.7.5 - 0.7.6 Use a simple pragma version that allows any of these versions. Consider using the latest version of Solidity for testing.

events-access



Configuration

- Check: `events-access`
- Severity: `Low`
- Confidence: `Medium`

Description

Detect missing events for critical access control parameters

Exploit Scenario:

```
contract C {

  modifier onlyAdmin {
    if (msg.sender != owner) throw;
    _;
  }

  function updateOwner(address newOwner) onlyAdmin external {
    owner = newOwner;
  }
}
```

`updateOwner()` has no event, so it is difficult to track off-chain owner changes.

Recommendation

Emit an event for critical parameter changes.

Missing events arithmetic

Configuration

- Check: `events-maths`
- Severity: `Low`
- Confidence: `Medium`

Description

Detect missing events for critical arithmetic parameters.

Exploit Scenario:

```
contract C {
```

```
    modifier onlyOwner {
        if (msg.sender != owner) throw;
        _;
    }

    function setBuyPrice(uint256 newBuyPrice) onlyOwner public {
        buyPrice = newBuyPrice;
    }

    function buy() external {
     ... // buyPrice is used to determine the number of tokens purchased
    }
}
```

`updateOwner()` has no event, so it is difficult to track off-chain changes in the buy price.

Recommendation

Emit an event for critical parameter changes.

Dangerous unary expressions

Configuration

- Check: `incorrect-unary`
- Severity: `Low`
- Confidence: `Medium`

Description

Unary expressions such as `x=+1` probably typos.

Exploit Scenario:

```
contract Bug{
    uint public counter;

    function increase() public returns(uint){
        counter=+1;
        return counter;
    }
}
```

`increase()` uses `=+` instead of `+=`, so `counter` will never exceed 1.

Recommendation

Remove the unary expression.

missing-zero-check

Configuration

- Check: `missing-zero-check`
- Severity: `Low`
- Confidence: `Medium`

Description

Detect missing zero address validation.

Exploit Scenario:

```
contract C {

  modifier onlyAdmin {
    if (msg.sender != owner) throw;
    _;
  }

  function updateOwner(address newOwner) onlyAdmin external {
    owner = newOwner;
  }
}
```

Bob calls `updateOwner` without specifying the `newOwner`, soBob loses ownership of the contract.

Recommendation

Check that the address is not zero.

external-function

external-function 10 Detail

```
61  function totalSupply() public view returns (uint256) {
62    return totalSupply_;
63  }

77  function balanceOf(address _owner) public view returns (uint256) {
78    return balances[_owner];
79  }

86  function allowance(address _owner, address _spender)
87    public view returns (uint256);

219 function renounceOwnership() public onlyOwner {
220   emit OwnershipRenounced(owner);
221   owner = address(0);
222 }

409 function transferOwnership(address newOwner) public onlyOwner {
410   pendingOwner = newOwner;
411 }

257 function mint(
258   address _to,
259   uint256 _amount
260 )
261   public
262   hasMintPermission
263   canMint
264   returns (bool)
265 {
266   totalSupply_ = totalSupply_.add(_amount);
267   balances[_to] = balances[_to].add(_amount);
268   emit Mint(_to, _amount);
269   emit Transfer(address(0), _to, _amount);
270   return true;
271 }

274 function finishMinting() public onlyOwner canMint returns (bool) {
275   mintingFinished = true;
276   emit MintFinished();
277   return true;
278 }

324 function pause() public onlyOwner whenNotPaused {
325   paused = true;
326   emit Pause();
327 }
```

Configuration

- Check: `external-function`
- Severity: `Optimization`
- Confidence: `High`

Description

`public` functions that are never called by the contract should be declared `external` to save gas.

Recommendation

Use the `external` attribute for functions never called from the contract.

dead-code

dead-code 5 Detail

```
function _transferOwnership(address _newOwner) internal {
  require(_newOwner != address(0));
  emit OwnershipTransferred(owner, _newOwner);
  owner = _newOwner;
}
```

```
function safeApprove(
  ERC20 _token,
  address _spender,
  uint256 _value
)
  internal
{
  require(_token.approve(_spender, _value));
}
```

```
function safeTransferFrom(
  ERC20 _token,
  address _from,
  address _to,
  uint256 _value
)
  internal
{
  require(_token.transferFrom(_from, _to, _value));
}
```

```
function div(uint256 _a, uint256 _b) internal pure returns (uint256) {
            return _a / _b;
}
```

```
function mul(uint256 _a, uint256 _b) internal pure returns (uint256 c) {
            if (_a == 0) {
    return 0;
  }

  c = _a * _b;
  assert(c / _a == _b);
  return c;
}
```

Configuration

- Check: `dead-code`
- Severity: `Informational`
- Confidence: `Medium`

Description

Functions that are not sued.

Exploit Scenario:

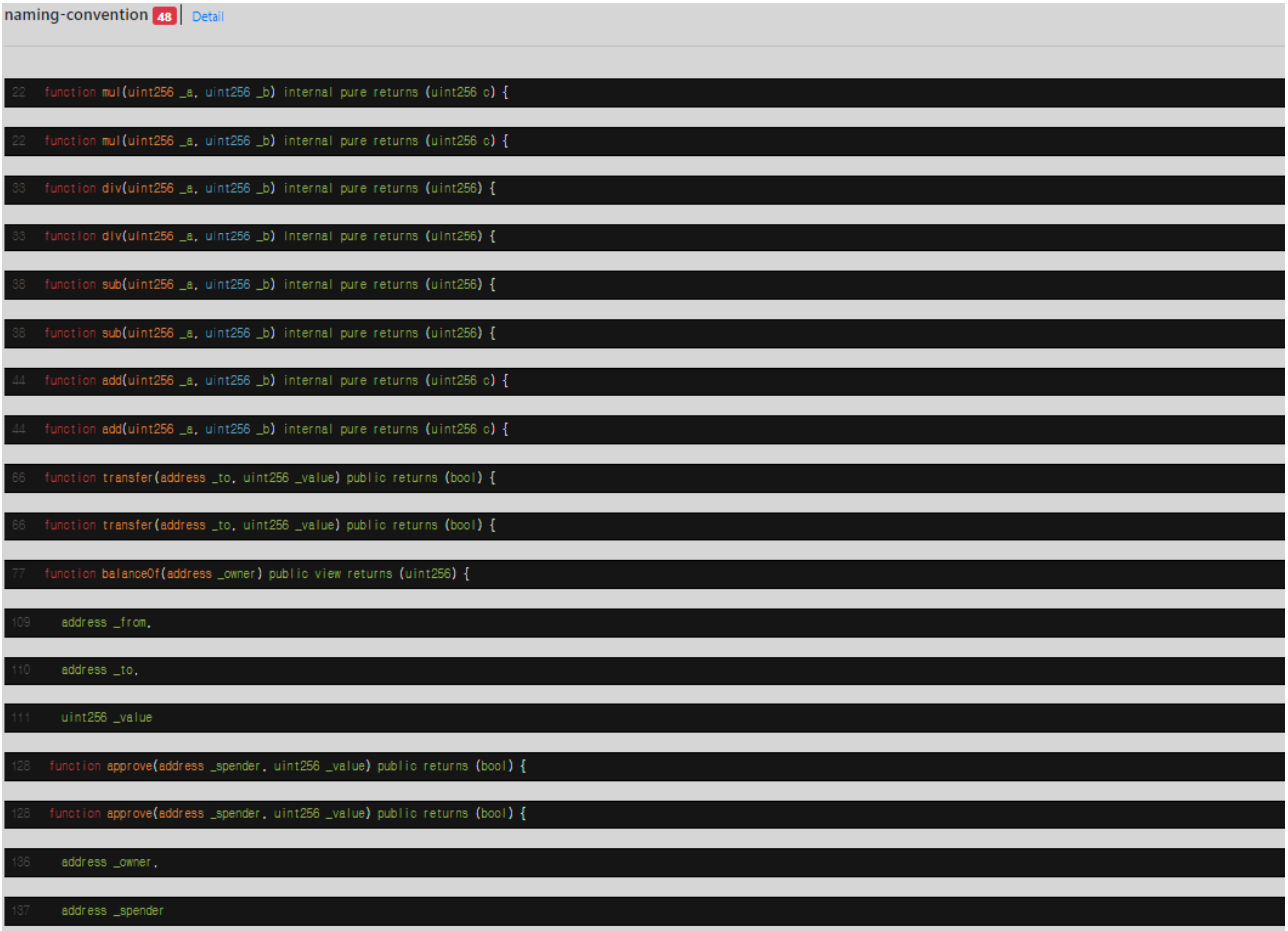
```
contract Contract{
    function dead_code() internal() {}
}
```

`dead_code` is not used in the contract, and make the code's review more difficult.

Recommendation

Remove unused functions.

naming-convention



Configuration

- Check: `naming-convention`
- Severity: `Informational`
- Confidence: `High`

Description

Solidity defines a naming convention that should be followed.

Rule exceptions

- Allow constant variable name/symbol/decimals to be lowercase (`ERC20`).
- Allow `_` at the beginning of the `mixed_case` match for private variables and unused parameters.

Recommendation

Follow the Solidity naming convention.