BNB(BNB)
Smartcontract Security
Audit Report

2022. 03

From SCOPE

https://blosafe.com

SCOPE

2022. 03

Document History

Date	Name	History
2022.03	Blosafe	Initial

1. Project outline

1.1. Purpose

The purpose of this inspection is to conduct a security audit on the [BnB] 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	0xB8c77482e45F1F44dE1745F52C74426C631bDD52	ETH Mainnet

1.3. Schedule

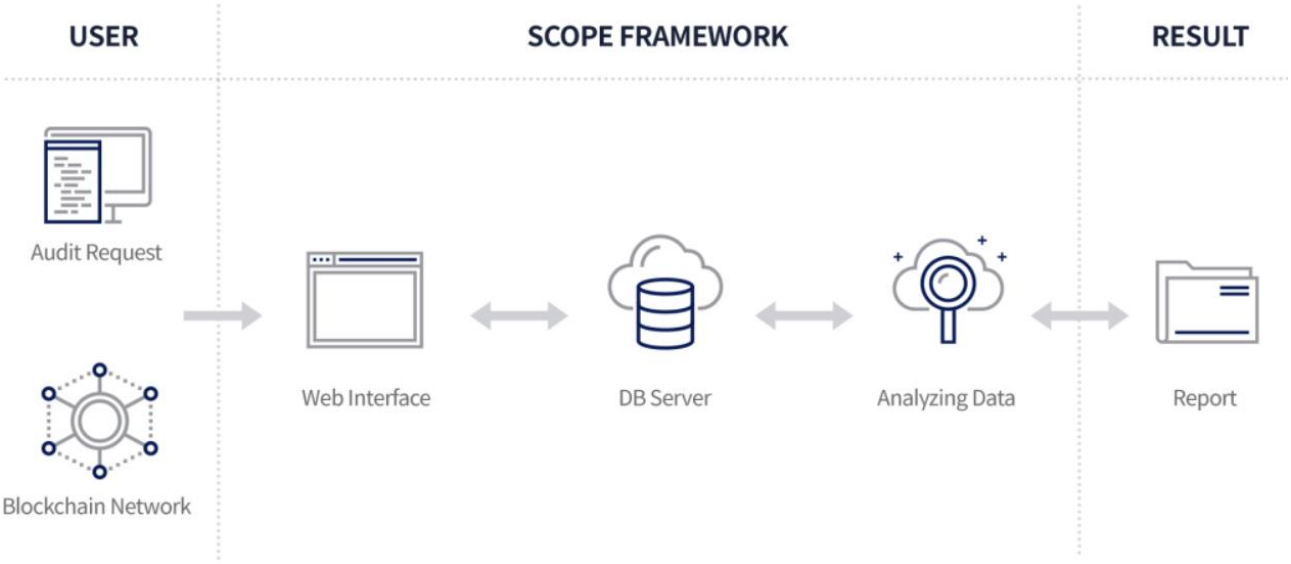
Work	Detail	Timeline	Memo
business consultation	Build Environment	1 day	
Audit	Smartcontract static auditing	2 days	
	Smartcontract Dynamic Auditing	3 days	
Report / review	Report	1 day	
	Review	1 day	

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

SCOPE

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

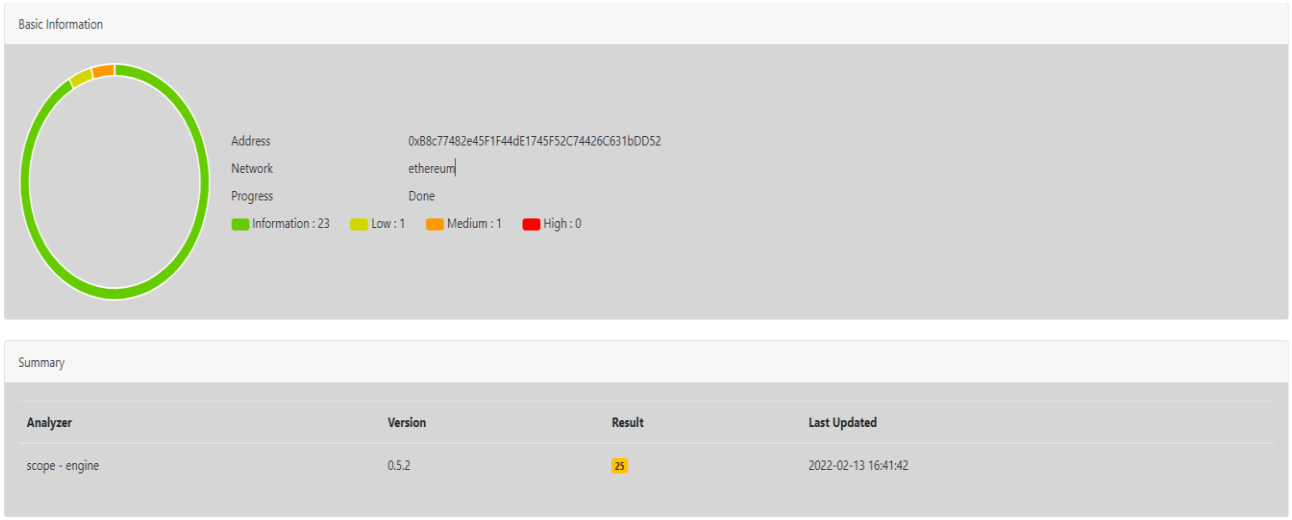
SCOPE

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

SCOPE

Summary of results

2.3. Result



[Passed]

[BNB] As a result of the Smartcontract security audit, a total of 32 vulnerabilities were found, among which 30 vulnerabilities of 'high', 2 of 'medium' vulnerabilities, 0 of 'low' vulnerabilities, and 'information' ratings were found.

3. Detailed results

3.1. Smartcontract

```
/**
 *Submitted for verification at Etherscan.io on 2017-07-06
*/

pragma solidity ^0.4.8;

/**
 * Math operations with safety checks
 */
contract SafeMath {
  function safeMul(uint256 a, uint256 b) internal returns (uint256) {
    uint256 c = a * b;
    assert(a == 0 || c / a == b);
    return c;
  }

  function safeDiv(uint256 a, uint256 b) internal returns (uint256) {
    assert(b > 0);
    uint256 c = a / b;
    assert(a == b * c + a % b);
    return c;
  }
```

```
  function safeSub(uint256 a, uint256 b) internal returns (uint256) {
    assert(b <= a);
    return a - b;
  }

  function safeAdd(uint256 a, uint256 b) internal returns (uint256) {
    uint256 c = a + b;
    assert(c>=a && c>=b);
    return c;
  }

  function assert(bool assertion) internal {
    if (!assertion) {
      throw;
    }
  }
}
contract BNB is SafeMath{
    string public name;
    string public symbol;
    uint8 public decimals;
    uint256 public totalSupply;
        address public owner;
```

```
/* This creates an array with all balances */
mapping (address => uint256) public balanceOf;
    mapping (address => uint256) public freezeOf;
mapping (address => mapping (address => uint256)) public allowance;

/* This generates a public event on the blockchain that will notify clients */
event Transfer(address indexed from, address indexed to, uint256 value);

/* This notifies clients about the amount burnt */
event Burn(address indexed from, uint256 value);

    /* This notifies clients about the amount frozen */
event Freeze(address indexed from, uint256 value);

    /* This notifies clients about the amount unfrozen */
event Unfreeze(address indexed from, uint256 value);

/* Initializes contract with initial supply tokens to the creator of the contract */
function BNB(
    uint256 initialSupply,
    string tokenName,
    uint8 decimalUnits,
    string tokenSymbol
    ) {
    balanceOf[msg.sender] = initialSupply;              // Give the creator all initial tokens
```


SCOPE

```
        totalSupply = initialSupply;                        // Update total supply
        name = tokenName;                                   // Set the name for display purposes
        symbol = tokenSymbol;                               // Set the symbol for display purposes
        decimals = decimalUnits;                            // Amount of decimals for display purposes
                owner = msg.sender;
    }

    /* Send coins */
    function transfer(address _to, uint256 _value) {
        if (_to == 0x0) throw;                              // Prevent transfer to 0x0 address. Use burn() instead
                if (_value <= 0) throw;
        if (balanceOf[msg.sender] < _value) throw;          // Check if the sender has enough
        if (balanceOf[_to] + _value < balanceOf[_to]) throw; // Check for overflows
        balanceOf[msg.sender] = SafeMath.safeSub(balanceOf[msg.sender], _value); // Subtract from the sender
        balanceOf[_to] = SafeMath.safeAdd(balanceOf[_to], _value);                // Add the same to the recipient
        Transfer(msg.sender, _to, _value);                  // Notify anyone listening that this transfer took place
    }

    /* Allow another contract to spend some tokens in your behalf */
    function approve(address _spender, uint256 _value)
        returns (bool success) {
                if (_value <= 0) throw;
```


SCOPE

```
        allowance[msg.sender][_spender] = _value;
        return true;
    }

    /* A contract attempts to get the coins */
    function transferFrom(address _from, address _to, uint256 _value) returns (bool success) {
        if (_to == 0x0) throw;                                  // Prevent transfer to 0x0 address. Use burn() instead
                if (_value <= 0) throw;
        if (balanceOf[_from] < _value) throw;                   // Check if the sender has enough
        if (balanceOf[_to] + _value < balanceOf[_to]) throw;    // Check for overflows
        if (_value > allowance[_from][msg.sender]) throw;       // Check allowance
        balanceOf[_from] = SafeMath.safeSub(balanceOf[_from], _value);                          // Subtract from the sender
        balanceOf[_to] = SafeMath.safeAdd(balanceOf[_to], _value);                              // Add the same to the recipient
        allowance[_from][msg.sender] = SafeMath.safeSub(allowance[_from][msg.sender], _value);
        Transfer(_from, _to, _value);
        return true;
    }

    function burn(uint256 _value) returns (bool success) {
        if (balanceOf[msg.sender] < _value) throw;              // Check if the sender has enough
                if (_value <= 0) throw;
```


SCOPE

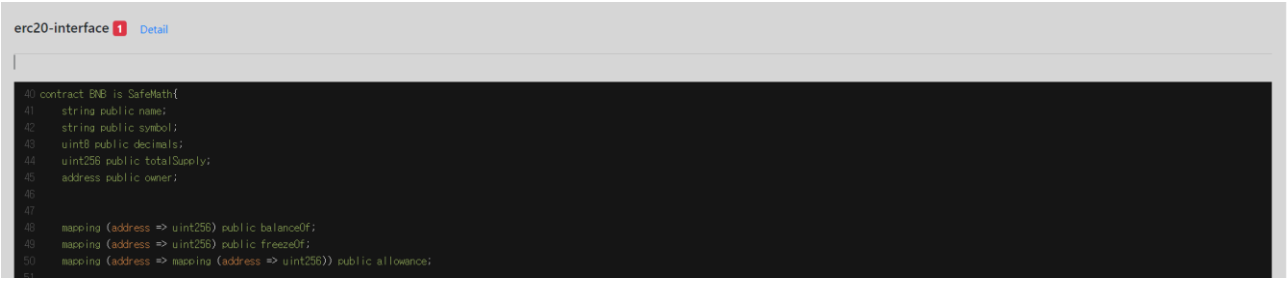
```
        balanceOf[msg.sender] = SafeMath.safeSub(balanceOf[msg.sender], _value);
// Subtract from the sender
        totalSupply = SafeMath.safeSub(totalSupply,_value);                    // Updates
totalSupply
        Burn(msg.sender, _value);
        return true;
    }

        function freeze(uint256 _value) returns (bool success) {
        if (balanceOf[msg.sender] < _value) throw;            // Check if the sender has enough
            if (_value <= 0) throw;
        balanceOf[msg.sender] = SafeMath.safeSub(balanceOf[msg.sender], _value);
// Subtract from the sender
        freezeOf[msg.sender] = SafeMath.safeAdd(freezeOf[msg.sender], _value);
// Updates totalSupply
        Freeze(msg.sender, _value);
        return true;
    }

        function unfreeze(uint256 _value) returns (bool success) {
        if (freezeOf[msg.sender] < _value) throw;            // Check if the sender has enough
            if (_value <= 0) throw;
        freezeOf[msg.sender] = SafeMath.safeSub(freezeOf[msg.sender], _value);
// Subtract from the sender
            balanceOf[msg.sender] = SafeMath.safeAdd(balanceOf[msg.sender], _value);
        Unfreeze(msg.sender, _value);
        return true;
```


SCOPE

```
    }


        // transfer balance to owner
        function withdrawEther(uint256 amount) {
                if(msg.sender != owner)throw;
                owner.transfer(amount);
        }


        // can accept ether
        function() payable {
    }
}
```

3.2. Vulnerability

erc20-interface



Configuration

- Check: `erc20-interface`
- Severity: `Medium`
- Confidence: `High`

Description

Incorrect return values for `ERC20` functions. A contract compiled with Solidity > 0.4.22 interacting with these functions will fail to execute them, as the return value is missing.

Exploit Scenario:

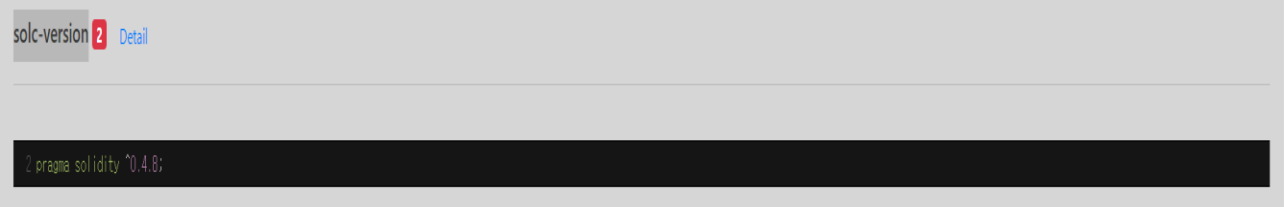
```
contract Token{
    function transfer(address to, uint value) external;
    //...
}
```

`Token.transfer` does not return a boolean. Bob deploys the token. Alice creates a contract that interacts with it but assumes a correct `ERC20` interface implementation. Alice's contract is unable to interact with Bob's contract.

Recommendation

Set the appropriate return values and types for the defined `ERC20` functions.

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.

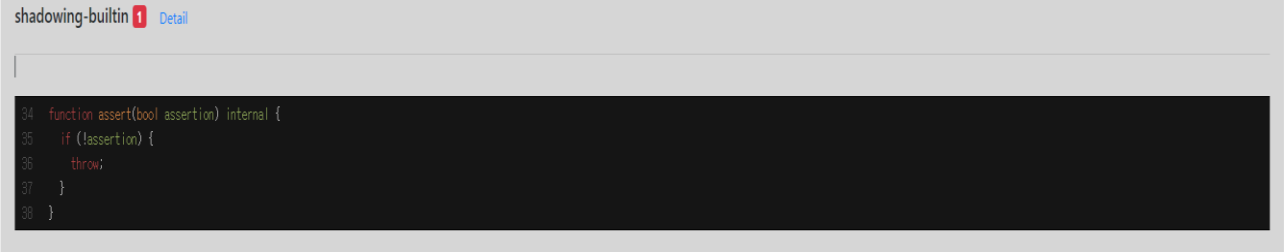
Recommendation

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.

SCOPE

shadowing-builtin



Configuration

- Check: `shadowing-builtin`
- Severity: `Low`
- Confidence: `High`

Description

Detection of shadowing built-in symbols using local variables, state variables, functions, modifiers, or events.

Exploit Scenario:

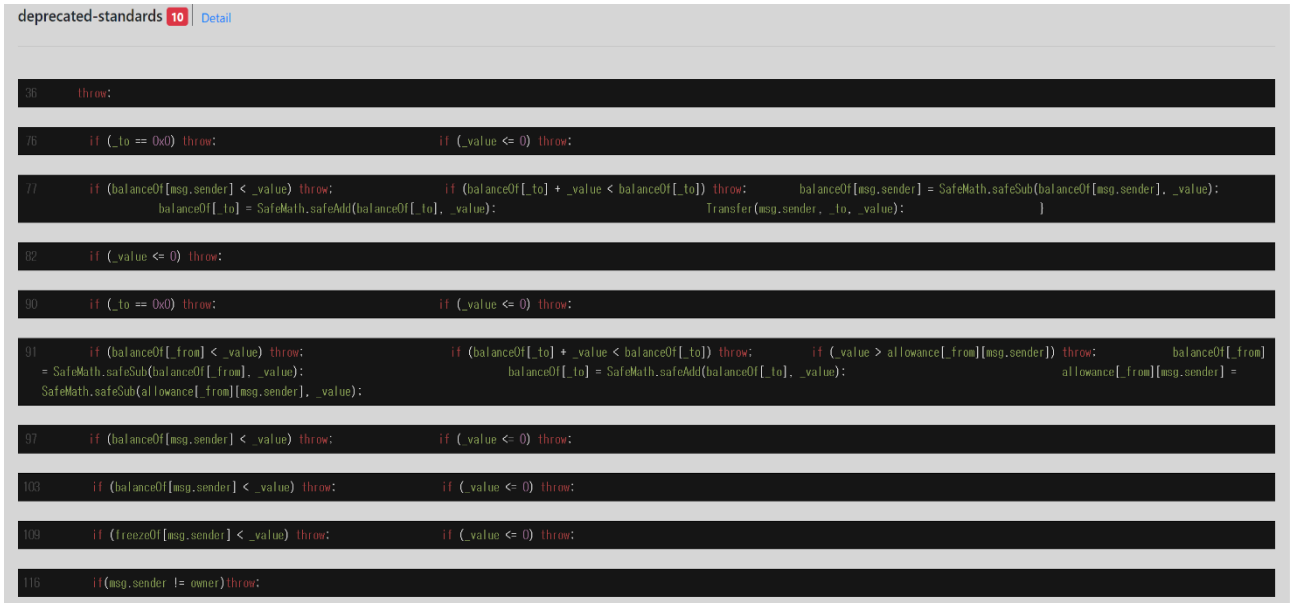
```
pragma solidity ^0.4.24;

contract Bug {
    uint now; // Overshadows current time stamp.

    function assert(bool condition) public {
        // Overshadows built-in symbol for providing assertions.
    }

    function get_next_expiration(uint earlier_time) private returns (uint) {
        return now + 259200; // References overshadowed timestamp.
    }
}
```

`now` is defined as a state variable, and shadows with the built-in symbol `now`. The function `assert` overshadows the built-in `assert` function. Any use of either of these built-in symbols may lead to unexpected results.

Recommendation

Rename the local variables, state variables, functions, modifiers, and events that shadow a builtin symbol.

deprecated-standards



Configuration

- Check: `deprecated-standards`
- Severity: `Informational`
- Confidence: `High`

Description

Detect the usage of deprecated standards.

Exploit Scenario:

```
contract ContractWithDeprecatedReferences {
    // Deprecated: Change block.blockhash() -> blockhash()
    bytes32 globalBlockHash = block.blockhash(0);

    // Deprecated: Change constant -> view
    function functionWithDeprecatedThrow() public constant {
        // Deprecated: Change msg.gas -> gasleft()
        if(msg.gas == msg.value) {
            // Deprecated: Change throw -> revert()
            throw;
        }
    }

    // Deprecated: Change constant -> view
    function functionWithDeprecatedReferences() public constant {
        // Deprecated: Change sha3() -> keccak256()
        bytes32 sha3Result = sha3("test deprecated sha3 usage");
```

```
        // Deprecated: Change callcode() -> delegatecall()
        address(this).callcode();

        // Deprecated: Change suicide() -> selfdestruct()
        suicide(address(0));
    }
}
```

Recommendation

Replace all uses of deprecated symbols.

external-function

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

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.