Metasecurelabs analysis report

metasecurelabs.io

December 1, 2022

1 Introduction

1.1 dangerous_enum_conversion

SWC_ID: []

Description:out–of–range enum conversion may occur (solc ¡ 0.4.5).

Example:

```
pragma solidity 0.4.2;
contract Test{
  enum E{a}
  function bug(uint a) public returns(E){
        return E(a);
    }
}
```

```
}
  }
```

DASP : Unknown Unknowns

Found: false

Reported by checker:

```
[]
```

1.2 unused_function_should_be_external

SWC_ID: []

Description:A function with public visibility modifier that is not called internally. Changing visibility level to external increases code readability. Moreover, in many cases functions with external visibility modifier spend less gas comparing to functions with public visibility modifier.

Example:

```
/*In the following example, functions with both public and
↪  external visibility modifiers are used: */

```

```
contract Token {

    mapping (address => uint256) internal _ balances;

    function transfer_ public(address to, uint256 value) public
    ↪ {
        require(value <= _ balances[msg.sender]);

        _ balances[msg.sender] { = value;
        _ balances[to] += value;
    }

    function transfer_ external(address to, uint256 value)
    ↪ external {
        require(value <= _ balances[msg.sender]);

        _ balances[msg.sender] { = value;
        _ balances[to] += value;
    }
}

 /*The second function requires less gas.*/

}
   }
```

DASP : Unknown unknowns
Found: true
Reported by checker:

```
[{"checker_ id":9,"lines":[],"tool":"metasecurelabs{
↪ 9"},{"checker_ id":9,"lines":[],"tool":"metasecurelabs{
↪ 9"},{"checker_ id":9,"lines":[],"tool":"metasecurelabs{ 9"}]
```

1.3 access_control

SWC_ID: [105,106,115]

Description:Access Control issues are common in all programs, not just smart contracts. In fact, it's number 5 on the OWASP top 10. One usually accesses a contract's functionality through its public or external functions. While insecure visibility settings give attackers straightforward ways to access a contract's private values or logic, access control bypasses are sometimes more subtle. These vulnerabilities can occur in the following cases: * Contracts use the deprecated tx.origin to validate callers * Handling large authorization logic with lengthy require * Making reckless use of delegatecall in proxy libraries or proxy contracts. Delegate calling into untrusted contracts is very dangerous, as the code at the target address can change any storage values of the caller and has full control over the caller's balance. * Due to missing or insufficient access

controls, malicious parties can withdraw some or all Ether from the contract account. * Due to missing or insufficient access controls, malicious parties can self–destruct the contract.

Example:

```
contract TestContract is MultiOwnable {

  function withdrawAll(){
    msg.sender.transfer(this.balance);
  }
}

}
  }
```

DASP : Access control

Found: true

Reported by checker:

```
[{"checker_ id":4,"lines":[{"code":"      bool res =
↪  msg.sender.call.value(amount)();\n","function_
↪  name":"","line_ no":19}],"tool":"metasecurelabs{ 4"}]
```

1.4 erc20_event_not_indexed

SWC_ID: []

Description:Events defined by the ERC20 specification that should have some parameters as indexed.

Example:

```
contract ERC20Bad {
    // ...
    event Transfer(address from, address to, uint value);
    event Approval(address owner, address spender, uint value);

    // ...
}

 /*Transfer and Approval events should have the 'indexed'
 ↪  keyword on their two first parameters, as defined by the
 ↪  ERC20 specification. Failure to include these keywords will
 ↪  exclude the parameter data in the transaction/block's bloom
 ↪  filter, so external tooling searching for these parameters
 ↪  may overlook them and fail to index logs from this token
 ↪  contract. */

}
  }
```

DASP : Unknown unknowns

Found: false
Reported by checker:

```
[]
```

1.5 locked_money

SWC_ID: []

Description:Contracts programmed to receive ether should implement a way to withdraw it, i.e., call transfer (recommended), send, or call.value at least once..

Example:

```
/* In the following example, contracts programmed to receive
 ↪  ether does not call transfer, send, or call.value function:
 ↪  */

pragma solidity 0.4.25;

contract BadMarketPlace {
    function deposit() payable {
        require(msg.value > 0);
    }
}
```

}
}
DASP : Unknown unknowns
Found: false
Reported by checker:

```
[]
```

1.6 arithmetic

SWC_ID: [101]

Description:This bug type consists of various arithmetic bugs: integer overflow/underflow, division issues, . * Integer overflow/underflow. An overflow/underflow happens when an arithmetic operation reaches the maximum or minimum size of a type. For instance if a number is stored in the uint8 type, it means that the number is stored in a 8 bits unsigned number ranging from 0 to 2 extsuperscript8–1. In computer programming, an integer overflow occurs when an arithmetic operation attempts to create a numeric value that is outside of the range that can be represented with a given number of bits either larger than the maximum or lower than the minimum representable value. * Division issues. Some wrong will happen when integer or float numbers are divided by zero. * Type deduction overflow. In Solidity, when declaring a variable as type

var, the compiler uses type deduction to automatically infer the smallest possible type from the first expression that is assigned to the variable. Thus, the deduced type may not be appropriate, and it can incur overflow bugs later (see the example).

Example:

```
Integer overflow/underflow
/*
* @source: https://capturetheether.com/challenges/math/token{
↪  sale/
* @author: Steve Marx
*/
pragma solidity  0.4.21;
contract TokenSaleChallenge {
    mapping(address => uint256) public balanceOf;
    uint256 constant PRICE_ PER_ TOKEN = 1 ether;

    function TokenSaleChallenge(address _ player) public payable
    ↪  {
        require(msg.value == 1 ether);
    }

    function isComplete() public view returns (bool) {
        return address(this).balance < 1 ether;
    }

    function buy(uint256 numTokens) public payable {
        require(msg.value == numTokens * PRICE_ PER_ TOKEN);

        balanceOf[msg.sender] += numTokens;
    }
}

 /*Division issues*/
contract Division {

    /*function unsigned_ division(uint32 x, uint32 y) returns
    ↪  (int r) {
      //if (y == 0) { throw; }
      r = x / y;
    }*/

    function signed_ division(int x, int y) returns (int) {
      //if ((y == 0)  ((x == { 2**255) & &  (y == { 1))) {
      ↪  throw; }
      return x / y;
```

```
    }

}

 /*Type deduction overflow*/
contract For_ Test {
...
  function Test () payable public {
    if ( msg . value > 0.1 ether ) {
      uint256 multi = 0;
      uint256 amountToTransfer = 0;
      for ( var i = 0; i < 2* msg . value ; i ++) {
        multi = i *2;
        if ( multi < amountToTransfer ) {
          break ;
        amountToTransfer = multi ;
      }
      msg.sender.transfer( amountToTransfer );
    }
  }
}
```

}
}
DASP : Arithmetic
Found: true
Reported by checker:

```
[{"checker_ id":1,"lines":[{"code":"    credit[to] +=
↪  msg.value;\n","function_ name":"","line_
↪  no":13}],"tool":"metasecurelabs{ 1"},{"checker_
↪  id":5,"lines":[{"code":"    credit[to] +=
↪  msg.value;\n","function_ name":"","line_
↪  no":13}],"tool":"metasecurelabs{ 5"},{"checker_
↪  id":6,"lines":[{"code":"    credit[to] +=
↪  msg.value;\n","function_ name":"","line_
↪  no":13}],"tool":"metasecurelabs{ 6"},{"checker_
↪  id":6,"lines":[{"code":"      credit[msg.sender]{
↪  =amount;\n","function_ name":"","line_
↪  no":20}],"tool":"metasecurelabs{ 6"},{"checker_
↪  id":7,"lines":[{"code":"    credit[to] +=
↪  msg.value;\n","function_ name":"","line_
↪  no":13}],"tool":"metasecurelabs{ 7"}]
```

1.7 incorrect_shift_in_assembly

SWC_ID: []

Description:The values in a shift operation could be reversed (in a wrong order)

Example:

```
contract C {
  function f() internal returns (uint a) {
    assembly {
        a := shr(a, 8)
    }
}
```

}

}

DASP : Unknown Unknowns

Found: false

Reported by checker:

```
[]
```

1.8 multiple_constructor_schemes

SWC_ID: []

Description:Multiple constructor definitions in the same contract (using new and old schemes).

Example:

```
contract A {
    uint x;
    constructor() public {
        x = 0;
    }
    function A() public {
        x = 1;
    }

    function test() public returns(uint) {
        return x;
    }
}

 /*In Solidity 0.4.22, a contract with both constructor schemes
 ↪  will compile. The first constructor will take precedence
 ↪  over the second, which may be unintended.*/
```

}

}

DASP : Unknown unknowns

Found: false

Reported by checker:

```
[]
```

1.9 local_variable_shadowing

SWC_ID: []

Description:Something wrong may happen when local variables shadowing state variables or other local variables.

Example:

```
pragma solidity  0.4.24;

contract Bug {
    uint owner;

    function sensitive_ function(address owner) public {
        // ...
        require(owner == msg.sender);
    }

    function alternate_ sensitive_ function() public {
        address owner = msg.sender;
        // ...
        require(owner == msg.sender);
    }
}

 /*sensitive_ function.owner shadows Bug.owner. As a result, the
  ↪  use of owner in sensitive_ function might be incorrect.*/

}
   }
```

DASP : Unknown unknowns

Found: false

Reported by checker:

```
[]
```

1.10 redundant_code

SWC_ID: []

Description:Redundant statements may have no effect.

Example:

```
contract RedundantStatementsContract {

    constructor() public {
        uint; // Elementary Type Name
```

```
        bool; // Elementary Type Name
        RedundantStatementsContract; // Identifier
    }

    function test() public returns (uint) {
        uint; // Elementary Type Name
        assert; // Identifier
        test; // Identifier
        return 777;
    }
}

 /*Each commented line references types/identifiers, but
  ↪  performs no action with them, so no code will be generated
  ↪  for such statements and they can be removed.*/
```

}
}
DASP : Unknown unknowns
Found: false
Reported by checker:

```
[]
```

1.11 should_be_view

SWC_ID: []
Description:In Solidity, functions that do not read from the state or modify it can be declared as view.
Example:

```
Here is the example of correct view{ function:

contract C {
    function f(uint a, uint b) view returns (uint) {
        return a * (b + 42) + now;
    }
}
```

}
}
DASP : Unknown unknowns
Found: false
Reported by checker:

```
[]
```

1.12 arbitrary_from_in_transferFrom

SWC_ID: []

Description:Something wrong happens when msg.sender is not used as 'from' in transferFrom.

Example:

```
function a(address from, address to, uint256 amount) public {
  erc20.transferFrom(from, to, am);
}

 /*Alice approves this contract to spend her ERC20 tokens. Bob
  ↪  can call a and specify Alice's address as the from
  ↪  parameter in transferFrom, allowing him to transfer Alice's
  ↪  tokens to himself.*/

}
  }
```

DASP : Unknown unknowns

Found: false

Reported by checker:

```
[]
```

1.13 func_modifying_storage_array_by_value

SWC_ID: []

Description:Arrays passed to a function that expects reference to a storage array.

Example:

```
contract Memory {
    uint[1] public x; // storage

    function f() public {
        f1(x); // update x
        f2(x); // do not update x
    }

    function f1(uint[1] storage arr) internal { // by reference
        arr[0] = 1;
    }

    function f2(uint[1] arr) internal { // by value
        arr[0] = 2;
    }
}
```

```
207
208   /*Bob calls f(). Bob assumes that at the end of the call x[0]
      ↪  is 2, but it is 1. As a result, Bob's usage of the contract
      ↪  is incorrect. */
```

}
}
DASP : Unknown Unknowns
Found: false
Reported by checker:

```
209 []
```

1.14 dead_code

SWC_ID: [135]

Description:In Solidity, it's possible to write code that does not produce the intended effects. Currently, the solidity compiler will not return a warning for effect–free code. This can lead to the introduction of "dead" code that does not properly performing an intended action.

For example, it's easy to miss the trailing parentheses in msg.sender.call.value(xx)("");, which could lead to a function proceeding without transferring funds to msg.sender. Also, internal functions could be 'dead' when they are not invoked.

Example:

```
210 pragma solidity  0.5.0;
211
212 contract DepositBox {
213     mapping(address => uint) balance;
214
215     // Accept deposit
216     function deposit(uint amount) public payable {
217         require(msg.value == amount, 'incorrect amount');
218         // Should update user balance
219         balance[msg.sender] = amount;
220     }
221 }
```

}
}
DASP : Unknown unknowns
Found: false
Reported by checker:

```
222 []
```

1.15 incorrect_using_balance_and_msg_value

SWC_ID: []

Description:this.balance will include the value sent by msg.value, which might lead to incorrect computation.

Example:

```
contract Bug{
  function buy() public payable{
    uint minted = msg.value * (1000 / address(this).balance);
    // ...
  }
}

 /*buy is meant to compute a price that changes a ratio over the
  ↪  contract's balance. .balance will include msg.value and
  ↪  lead to an incorrect price computation.*/
```

}

}

DASP : Unknown unknowns

Found: false

Reported by checker:

```
[]
```

1.16 short_addresses

SWC_ID: []

Description:MISSING

Example:

```
MISSING
```

}

}

DASP : Unknown unknowns

Found: false

Reported by checker:

```
[]
```

1.17 blockhash_current

SWC_ID: []

Description:blockhash function returns a non–zero value only for 256 last blocks. Besides, it always returns 0 for the current block, i.e. blockhash(block.number) always equals to 0.

Example:

```
/*In the following example, currentBlockBlockhash function always returns 0:*/

pragma solidity 0.4.25;

contract MyContract {
    function currentBlockHash() public view returns(bytes32) {
        return blockhash(block.number);
    }
}
```

}
}
DASP : Unknown unknowns
Found: false
Reported by checker:

```
[]
```

1.18 msg.value_in_loop

SWC_ID: []
Description:It is error–prone to use msg.value inside a loop.
Example:

```
contract MsgValueInLoop{
    mapping (address => uint256) balances;

    function bad(address[] memory receivers) public payable {
        for (uint256 i=0; i < receivers.length; i++) {
            balances[receivers[i]] += msg.value;
        }
    }
}

 /*msg.value should be tracked through a local variable and decrease its amount on every iteration/usage.*/
```

}
}
DASP : Unknown unknowns
Found: false
Reported by checker:

```
[]
```

1.19 reentrancy

SWC_ID: [107]

Description:One of the major dangers of calling external contracts is that they can take over the control flow. In the reentrancy attack (a.k.a. recursive call attack), a malicious contract calls back into the calling contract before the first invocation of the function is finished. This may cause the different invocations of the function to interact in undesirable ways.

Example:

```
/*
* @source: http://blockchain.unica.it/projects/ethereum{
 ↪  survey/attacks.htmlsimpledao
* @author: {
* @vulnerable_ at_ lines: 19
*/

pragma solidity  0.4.2;

contract SimpleDAO {
  mapping (address => uint) public credit;

  function donate(address to) payable {
    credit[to] += msg.value;
  }

  function withdraw(uint amount) {
    if (credit[msg.sender]>= amount) {
      // <yes> <report> REENTRANCY
      bool res = msg.sender.call.value(amount)();
      credit[msg.sender]{ =amount;
    }
  }
}

}
  }
```

DASP : Reentrancy

Found: true

Reported by checker:

```
[{"checker_ id":1,"lines":[{"code":"      bool res =
↪  msg.sender.call.value(amount)();\n","function_
↪  name":"","line_ no":19}],"tool":"metasecurelabs{
↪  1"},{"checker_ id":5,"lines":[{"code":"
↪  credit[msg.sender]{ =amount;\n","function_ name":"","line_
↪  no":20}],"tool":"metasecurelabs{ 5"},{"checker_
↪  id":4,"lines":[{"code":"      bool res =
↪  msg.sender.call.value(amount)();\n","function_
↪  name":"","line_ no":19}],"tool":"metasecurelabs{
↪  4"},{"checker_ id":6,"lines":[{"code":"      bool res =
↪  msg.sender.call.value(amount)();\n","function_
↪  name":"","line_ no":19}],"tool":"metasecurelabs{
↪  6"},{"checker_ id":7,"lines":[{"code":"      bool res =
↪  msg.sender.call.value(amount)();\n","function_
↪  name":"","line_ no":19}],"tool":"metasecurelabs{
↪  7"},{"checker_ id":8,"lines":[{"code":"      // <yes>
↪  <report> REENTRANCY\n","function_ name":"","line_
↪  no":18}],"tool":"metasecurelabs{ 8"},{"checker_
↪  id":9,"lines":[{"code":"      credit[msg.sender]{
↪  =amount;\n","function_ name":"","line_ no":20},{"code":"
↪  bool res = msg.sender.call.value(amount)();\n","function_
↪  name":"","line_ no":19}],"tool":"metasecurelabs{ 9"}]
```

1.20 visibility

SWC_ID: [100,108]

Description:The default function visibility level in contracts is public, in interfaces – external, and the state variable default visibility level is internal. In contracts, the fallback function can be external or public. In interfaces, all the functions should be declared as external. Explicitly define function visibility to prevent confusion. Additionally, the visibility of state variables could be a problem. labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.

Example:

```
/*In this example, a specific modifier, such as public, is not
↪  used when declaring a function: */

function foo();

Preferred alternatives:

function foo() public;
function foo() internal;
```

}
}

DASP : Unknown Unknowns
Found: true
Reported by checker:

```
[{"checker_
↪  id":10,"lines":[{"code":"functiondonate(addressto)payable{credit[to]+=msg.value;}","fu
↪  name":"","line_ no":12}],"tool":"metasecurelabs{
↪  10"},{"checker_
↪  id":10,"lines":[{"code":"functionwithdraw(uintamount){if(credit[msg.sender]>=amount){b
↪  =amount;}}","function_ name":"","line_
↪  no":16}],"tool":"metasecurelabs{ 10"},{"checker_
↪  id":10,"lines":[{"code":"functionqueryCredit(addressto)returns(uint){returncredit[to];
↪  name":"","line_ no":24}],"tool":"metasecurelabs{ 10"}]
```

1.21 array_length_manipulation

SWC_ID: [124]

Description:The length of the dynamic array is changed directly. In the following case, the appearance of gigantic arrays is possible and it can lead to a storage overlap attack (collisions with other data in storage).

Example:

```
pragma solidity 0.4.24;

contract dataStorage {
    uint[] public data;

    function writeData(uint[] _ data) external {
        for(uint i = data.length; i < _ data.length; i++) {
            data.length++;
            data[i]=_ data[i];
        }
    }
}
```

}
}
DASP : Unknown Unknowns
Found: false
Reported by checker:

```
[]
```

1.22 denial_of_service

SWC_ID: [113,128]

Description:Denial of service (DoS) is deadly in the world of Ethereum: while other types of applications can eventually recover, smart contracts can

be taken offline forever by just one of these attacks. DoS can happen in the following cases: * External calls can fail accidentally or deliberately, which can cause a DoS condition in the contract. Particularly, DoS would happen if there is a loop where external calls are not isolated. * A large number of loops may consume gas, so it is possible that the function exceeds the block gas limit, and transactions calling it will never be confirmed. * An inappropriate type inference in the loop (e.g., literal –¿ uint8) may cause a infinite loop. * Recursive external calls may consume a large number of callstacks, which may lead to DoS.

Example:

```
for (var i = 0; i < array.length; i++) { /* ... */

}
   }
```

DASP : Denial of Services

Found: true

Reported by checker:

```
[{"checker_ id":6,"lines":[{"code":"      bool res =
↪  msg.sender.call.value(amount)();\n","function_
↪  name":"","line_ no":19}],"tool":"metasecurelabs{
↪  6"},{"checker_ id":7,"lines":[{"code":"      bool res =
↪  msg.sender.call.value(amount)();\n","function_
↪  name":"","line_ no":19}],"tool":"metasecurelabs{ 7"}]
```

1.23 uninitialized_state_variable

SWC_ID: []

Description:Some unexpected error may happen when state variables are not uninitialized.

Example:

```
contract Uninitialized{
    address destination;

    function transfer() payable public{
        destination.transfer(msg.value);
    }
}

}
   }
```

DASP : Unknown unknowns

Found: true

Reported by checker:

```
[{"checker_ id":4,"lines":[{"code":"    credit[to] +=
↪  msg.value;\n","function_ name":"","line_
↪  no":13}],"tool":"metasecurelabs{ 4"},{"checker_
↪  id":4,"lines":[{"code":"      credit[msg.sender]{
↪  =amount;\n","function_ name":"","line_
↪  no":20}],"tool":"metasecurelabs{ 4"},{"checker_
↪  id":4,"lines":[{"code":"    return credit[to];\n","function_
↪  name":"","line_ no":25}],"tool":"metasecurelabs{
↪  4"},{"checker_ id":4,"lines":[{"code":"  mapping (address =>
↪  uint) public credit;\n","function_ name":"","line_
↪  no":10}],"tool":"metasecurelabs{ 4"},{"checker_
↪  id":4,"lines":[{"code":"    if (credit[msg.sender]>= amount)
↪  {\n","function_ name":"","line_
↪  no":17}],"tool":"metasecurelabs{ 4"}]
```

1.24 unchecked_calls

SWC_ID: [104]

Description:The return value of a message call is not checked. Execution will resume even if the called contract throws an exception. If the call fails accidentally or an attacker forces the call to fail, this may cause unexpected behaviour in the subsequent program logic.

Example:

```
pragma solidity 0.4.25;

contract ReturnValue {

checked
function callchecked(address callee) public {
    require(callee.call());
  }

  function callnotchecked(address callee) public {
    callee.call();
  }
}
```

```
}
  }
```

DASP : Unchecked Low Level Calls

Found: true

Reported by checker:

```
[{"checker_ id":1,"lines":[{"code":"      bool res =
↪  msg.sender.call.value(amount)();\n","function_
↪  name":"","line_ no":19}],"tool":"metasecurelabs{
↪  1"},{"checker_ id":5,"lines":[{"code":"      bool res =
↪  msg.sender.call.value(amount)();\n","function_
↪  name":"","line_ no":19}],"tool":"metasecurelabs{
↪  5"},{"checker_ id":8,"lines":[{"code":"      // <yes>
↪  <report> REENTRANCY\n","function_ name":"","line_
↪  no":18}],"tool":"metasecurelabs{ 8"}]
```

1.25 unused_retval

SWC_ID: []

Description:The return value of an external call is not stored in a local or state variable.

Example:

```
contract MyConc{
    using SafeMath for uint;
    function my_ func(uint a, uint b) public{
        a.add(b);
    }
}

 /*MyConc calls add of SafeMath, but does not store the result
 ↪  in a. As a result, the computation has no effect. */

}
   }
```

DASP : Unknown Unknowns

Found: true

Reported by checker:

```
[{"checker_ id":4,"lines":[{"code":"      bool res =
↪  msg.sender.call.value(amount)();\n","function_
↪  name":"","line_ no":19}],"tool":"metasecurelabs{ 4"}]
```

1.26 wrong_signature

SWC_ID: []

Description:In Solidity, the function signature is defined as the canonical expression of the basic prototype without data location specifier, i.e. the function name with the parenthesised list of parameter types. Parameter types are split by a single comma – no spaces are used. This means one should use uint256 and int256 instead of uint or int.

Example:

```
 /*This code uses incorrect function signature:*/

pragma solidity  0.5.1;
contract Signature {
    function callFoo(address addr, uint value) public returns
    ↪  (bool) {
        bytes memory data = abi.encodeWithSignature("foo(uint)",
        ↪  value);
        (bool status, ) = addr.call(data);
        return status;
    }
}

/*Use "foo(uint256)" instead.*/
```

}
}
DASP : Unknown Unknowns
Found: false
Reported by checker:

```
[]
```

1.27 constant_state_variable

SWC_ID: []
Description:There is a conflict if the same base constructor is called with arguments from two different locations in the same inheritance hierarchy.
Example:

```
pragma solidity  0.4.0;

contract A{
    uint num = 5;
    constructor(uint x) public{
        num += x;
    }
}

contract B is A{
    constructor() A(2) public { /* ... */ }
}

contract C is A {
    constructor() A(3) public { /* ... */ }
}

```

```
contract D is B, C {
    constructor() public { /* ... */ }
}
```

}
}
DASP : Unknown unknowns
Found: false
Reported by checker:

```
[]
```

1.28 incorrect_modifier

SWC_ID: []
Description:If a modifier does not execute —_— or revert, the execution of the function will return the default value, which can be misleading for the caller.
Example:

```
modidfier myModif(){
    if(..){
      _ ;
    }
}
function get() myModif returns(uint){}

 /*If the condition in myModif is false, the execution of get()
 ↪  will return 0.*/
```

}
}
DASP : Unknown unknowns
Found: false
Reported by checker:

```
[]
```

1.29 deprecated_standards

SWC_ID: [111,118]
Description:Several functions and operators in Solidity are deprecated. Using them leads to reduced code quality. With new major versions of the Solidity compiler, deprecated functions and operators may result in side effects and compile errors. Deprecated Alternative suicide(address) selfdestruct(address) block.blockhash(uint) blockhash(uint) sha3(...) keccak256(...) callcode(...) delegatecall(...) throw revert() msg.gas gasleft constant view var corresponding type name
Example:

```
pragma solidity 0.4.24;

contract BreakThisHash {
    bytes32 hash;
    uint birthday;
    constructor(bytes32 _ hash) public payable {
        hash = _ hash;
        birthday = now;
    }

    function kill(bytes password) external {
        if (sha3(password) != hash) {
            throw;
        }
        suicide(msg.sender);
    }

    function hashAge() public constant returns(uint) {
        return(now {  birthday);
    }
}

 /*Use keccak256, selfdestruct, revert() instead.*/

}
   }
```

DASP : Unknown unknowns
Found: false
Reported by checker:

```
[]
```

1.30 state_variable_shadowing

SWC_ID: [119]

Description:Solidity allows for ambiguous naming of state variables when inheritance is used. Contract A with a variable x could inherit contract B that also has a state variable x defined. This would result in two separate versions of x, one of them being accessed from contract A and the other one from contract B. In more complex contract systems this condition could go unnoticed and subsequently lead to security issues.

Example:

```
pragma solidity 0.4.25;

contract Tokensale {
    uint public hardcap = 10000 ether;
```

```
406
407     function Tokensale() {}
408
409     function fetchCap() public constant returns(uint) {
410         return hardcap;
411     }
412 }
413
414 contract Presale is Tokensale {
415     //uint hardcap = 1000 ether;
416     //If the hardcap variables were both needed we would have to
        ↪  rename one to fix this.
417     function Presale() Tokensale() {
418         hardcap = 1000 ether;
419     }
420 }
```

}
}
DASP : Unknown Unknowns
Found: false
Reported by checker:

```
421 []
```

1.31 benign_reentrancy

SWC_ID: []

Description:Some re–entrancy bugs have no adverse effect since its exploitation would have the same effect as two consecutive calls.

Example:

```
422 function callme(){
423   if( ! (msg.sender.call()() ) ){
424     throw;
425   }
426   counter += 1
427 }
428
429  /*callme() contains a benign reentrancy.*/
```

}
}
DASP : Unknown unknowns
Found: false
Reported by checker:

```
430 []
```

1.32 using_send

SWC_ID: []

Description:The send function is called inside checks instead of using transfer. The recommended way to perform checked ether payments is addr.transfer(x), which automatically throws an exception if the transfer is unsuccessful.

Example:

```
/* In the following example, the send function is used:*/


if(!addr.send(42 ether)) {
    revert();
}

 /*Preferred alternative:

addr.transfer(42 ether);*/
```

}
}

DASP : Unknown Unknowns

Found: false

Reported by checker:

```
[]
```

1.33 race_condition

SWC_ID: [114]

Description:Since miners always get rewarded via gas fees for running code on behalf of externally owned addresses (EOA), users can specify higher fees to have their transactions mined more quickly. Since the Ethereum blockchain is public, everyone can see the contents of others' pending transactions. This means if a given user is revealing the solution to a puzzle or other valuable secret, a malicious user can steal the solution and copy their transaction with higher fees to preempt the original solution. If developers of smart contracts are not careful, this situation can lead to practical and devastating front–running attacks.

Example:

```
/* In this example, one can front{ run transactions to claim
 ↪  his/her reward before the owner reduces the reward amount.*/

pragma solidity  0.4.16;

contract EthTxOrderDependenceMinimal {
    address public owner;
```

```
    bool public claimed;
    uint public reward;

    function EthTxOrderDependenceMinimal() public {
        owner = msg.sender;
    }

    function setReward() public payable {
        require (!claimed);
        require(msg.sender == owner);
        owner.transfer(reward);
        reward = msg.value;
    }

    function claimReward(uint256 submission) {
        require (!claimed);
        require(submission < 10);
        msg.sender.transfer(reward);
        claimed = true;
    }
}
```

}
}
DASP : Front Running
Found: true
Reported by checker:

```
[{"checker_ id":8,"lines":[{"code":"      // <yes> <report>
↪  REENTRANCY\n","function_ name":"","line_
↪  no":18}],"tool":"metasecurelabs{ 8"},{"checker_
↪  id":8,"lines":[{"code":"      // <yes> <report>
↪  REENTRANCY\n","function_ name":"","line_
↪  no":18}],"tool":"metasecurelabs{ 8"}]
```

1.34 uninitialized_func_pointer

SWC_ID: []

Description:solc versions 0.4.5——— 0.4.26 and 0.5.0——— 0.5.8 contain a compiler bug leading to unexpected behavior when calling uninitialized function pointers in constructors.

Example:

```
contract bad0 {

  constructor() public {
    /* Uninitialized function pointer */
```

```
    function(uint256) internal returns(uint256) a;
    a(10);
  }
}

The call to a(10) will lead to unexpected behavior because
↪  function pointer a is not initialized in the constructor.
```

}
}
DASP : Unknown Unknowns
Found: false
Reported by checker:

```
[]
```

1.35 modifier_like_Sol_keyword

SWC_ID: []
Description:A contract may contain modifier that looks similar to Solidity keyword
Example:

```
contract Contract{
    modifier pub1ic() {
    }

    function doSomething() pub1ic {
        require(owner == msg.sender);
        owner = newOwner;
    }
}

 /*public is a modifier meant to look like a Solidity keyword.*/
```

}
}
DASP : Unknown Unknowns
Found: false
Reported by checker:

```
[]
```

1.36 incorrect_ERC721_interface

SWC_ID: []

Description:Incorrect return values for ERC721 functions. A contract compiled with solidity ¿ 0.4.22 interacting with these functions will fail to execute them, as the return value is missing.

Example:

```
contract Token{
    function ownerOf(uint256 _ tokenId) external view returns
    ↪  (bool);
    //...
}
```

}

}

DASP : Unknown unknowns

Found: false

Reported by checker:

```
[]
```

1.37 incorrect_ERC20_interface

SWC_ID: []

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.

Example:

```
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. */
```

}

}

DASP : Unknown Unknowns

Found: false

Reported by checker:

```
[]
```

1.38 del_structure_containing_mapping

SWC_ID: []

Description:A deletion in a structure containing a mapping will not delete the mapping (see the Solidity documentation). The remaining data may be used to compromise the contract.

Example:

```
struct BalancesStruct{
    address owner;
    mapping(address => uint) balances;
}
mapping(address => BalancesStruct) public stackBalance;

function remove() internal{
      delete stackBalance[msg.sender];
}
```

}
}

DASP : Unknown unknowns

Found: false

Reported by checker:

```
[]
```

1.39 use_after_delete

SWC_ID: []

Description:Using values of variables after they have been explicitly deleted may lead to unexpected behavior or compromise.

Example:

```
mapping(address => uint) public balances;
function f() public {
    delete balances[msg.sender];
    msg.sender.transfer(balances[msg.sender]);
}

 /*balances[msg.sender] is deleted before it's sent to the
 ↪  caller, leading the transfer to always send zero.*/
```

}
}

DASP : Unknown unknowns

Found: false

Reported by checker:

```
[]
```

1.40 function_declared_return_but_no_return

SWC_ID: []

Description:Function doesn't initialize return value. As result default value will be returned.

Example:

```
/*In the following example, the function's signature only denotes the type of the return value, but the function's body does not contain return statement:*/

pragma solidity 0.4.25;

contract NewContract {
    uint minimumBuy;

    function setMinimumBuy(uint256 newMinimumBuy) returns (bool){
        minimumBuy = newMinimumBuy;
    }
}
```

}
}

DASP : Unknown unknowns

Found: false

Reported by checker:

```
[]
```

1.41 controlled_lowlevel_call

SWC_ID: []

Description:Low–level call with a user–controlled data field

Example:

```
address token;

function call_ token(bytes data){
  token.call(data);
}

 /*token` points to an ERC20 token. Bob uses call_ token to call the transfer function of token to withdraw all tokens held by the contract.*/
```

}
}

DASP : Unknown Unknowns
Found: true
Reported by checker:

```
[{"checker_ id":5,"lines":[{"code":"      bool res =
↪  msg.sender.call.value(amount)();\n","function_
↪  name":"","line_ no":19}],"tool":"metasecurelabs{ 5"}]
```

1.42 address_hardcoded

SWC_ID: []

Description:The contract contains unknown address. This address might be used for some malicious activity. Please check hardcoded address and it's usage.

Example:

```
/*In the following contract, the address is specified in the
↪  source code:*/

pragma solidity 0.4.24;
contract C {
  function f(uint a, uint b) pure returns (address) {
    address public multisig =
    ↪  0xf64B584972FE6055a770477670208d737Fff282f;
    return multisig;
        }
}

 /*Do not forget to check the contract at the address
 ↪  0xf64B584972FE6055a770477670208d737Fff282f for
 ↪  vulnerabilities.*/

}
  }
```

DASP : Unknown unknowns
Found: false
Reported by checker:

```
[]
```

1.43 divide_before_multiply

SWC_ID: []

Description:Solidity operates only with integers. Thus, if the division is done before the multiplication, the rounding errors can increase dramatically. Vulnerability type by SmartDec classification: Precision issues.

Example:

```
/*In the following example, amount variable is divided by
↪  DELIMITER and then multiplied by BONUS. Thus, a rounding
↪  error appears (consider amount = 9000):*/

pragma solidity 0.4.25;

contract MyContract {

    uint constant BONUS = 500;
    uint constant DELIMITER = 10000;

    function calculateBonus(uint amount) returns (uint) {
        return amount/DELIMITER*BONUS;
    }
}
```

}
}
DASP : Unknown Unknowns
Found: false
Reported by checker:

```
[]
```

1.44 reused_base_constructors

SWC_ID: []

Description:There is a conflict if the same base constructor is called with arguments from two different locations in the same inheritance hierarchy.

Example:

```
pragma solidity  0.4.0;

contract A{
    uint num = 5;
    constructor(uint x) public{
        num += x;
    }
}

contract B is A{
    constructor() A(2) public { /* ... */ }
}

contract C is A {
    constructor() A(3) public { /* ... */ }
}
```

```

contract D is B, C {
    constructor() public { /* ... */ }
}
```

}
}
DASP : Unknown unknowns
Found: false
Reported by checker:

```
[]
```

1.45 unused_state_variables

SWC_ID: [131]

Description:Unused variables are allowed in Solidity and they do not pose a direct security issue. It is best practice though to avoid them as they can: * cause an increase in computations (and unnecessary gas consumption) * indicate bugs or malformed data structures and they are generally a sign of poor code quality * cause code noise and decrease readability of the code

Example:

```
pragma solidity >=0.5.0;
pragma experimental ABIEncoderV2;

import "./base.sol";

contract DerivedA is Base {
    // i is not used in the current contract
    A i = A(1);

    int internal j = 500;

    function call(int a) public {
        assign1(a);
    }

    function assign3(A memory x) public returns (uint) {
        return g[1] + x.a + uint(j);
    }

    function ret() public returns (int){
        return this.e();

    }
  int internal j = 500;
```

```
function call(int a) public {
        assign1(a);
    }

    function assign3(A memory x) public returns (uint) {
        return g[1] + x.a + uint(j);
    }

    function ret() public returns (int){
        return this.e();
  }
}
```

}
}
DASP : Unknown unknowns
Found: false
Reported by checker:

```
[]
```

1.46 do_while_continue

SWC_ID: []
Description:Prior to version 0.5.0, Solidity compiler handles continue inside do–while loop incorrectly: it ignores while condition.
Example:

```
/*The following loop is infinite:*/

do {
    continue;
} while(false);
```

}
}
DASP : Unknown Unknowns
Found: false
Reported by checker:

```
[]
```

1.47 builtin_symbol_shadowing

SWC_ID: []
Description:Something wrong may happen when built–in symbols are shadowed by local variables, state variables, functions, modifiers, or events.
Example:

```
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.
    }
}
```

}
}
DASP : Unknown unknowns
Found: false
Reported by checker:

```
[]
```

1.48 ignore

SWC_ID: []
Description:Other trivial bug types.
Example:

```

```

}
}
DASP : Unknown Unknowns
Found: true
Reported by checker:

```
[{"checker_ id":8,"lines":[{"code":"\n","function_
↪  name":"","line_ no":23},{"code":"contract SimpleDAO
↪  {\n","function_ name":"","line_
↪  no":9},{"code":"\n","function_ name":"","line_
↪  no":11},{"code":"\n","function_ name":"","line_
↪  no":15}],"tool":"metasecurelabs{ 8"},{"checker_
↪  id":9,"lines":[{"code":"pragma solidity
↪  0.4.2;\n","function_ name":"","line_
↪  no":7}],"tool":"metasecurelabs{ 9"},{"checker_
↪  id":9,"lines":[],"tool":"metasecurelabs{ 9"},{"checker_
↪  id":9,"lines":[{"code":"      bool res =
↪  msg.sender.call.value(amount)();\n","function_
↪  name":"","line_ no":19}],"tool":"metasecurelabs{
↪  9"},{"checker_
↪  id":10,"lines":[{"code":"call.value(amount)()","function_
↪  name":"","line_ no":19}],"tool":"metasecurelabs{
↪  10"},{"checker_ id":10,"lines":[{"code":" ","function_
↪  name":"","line_ no":7}],"tool":"metasecurelabs{
↪  10"},{"checker_
↪  id":10,"lines":[{"code":"call.value(amount)()","function_
↪  name":"","line_ no":19}],"tool":"metasecurelabs{
↪  10"},{"checker_ id":11,"lines":[{"code":"  mapping (address
↪  => uint) public credit;\n","function_ name":"","line_
↪  no":10}],"tool":"metasecurelabs{ 11"},{"checker_
↪  id":11,"lines":[{"code":"  function donate(address to)
↪  payable {\n","function_ name":"","line_
↪  no":12}],"tool":"metasecurelabs{ 11"},{"checker_
↪  id":11,"lines":[{"code":"    credit[to] +=
↪  msg.value;\n","function_ name":"","line_
↪  no":13}],"tool":"metasecurelabs{ 11"},{"checker_
↪  id":11,"lines":[{"code":"  }\n","function_ name":"","line_
↪  no":14}],"tool":"metasecurelabs{ 11"},{"checker_
↪  id":11,"lines":[{"code":"  function withdraw(uint amount)
↪  {\n","function_ name":"","line_
↪  no":16}],"tool":"metasecurelabs{ 11"},{"checker_
↪  id":11,"lines":[{"code":"    if (credit[msg.sender]>=
↪  amount) {\n","function_ name":"","line_
↪  no":17}],"tool":"metasecurelabs{ 11"},{"checker_
↪  id":11,"lines":[{"code":"      bool res =
↪  msg.sender.call.value(amount)();\n","function_
↪  name":"","line_ no":19}],"tool":"metasecurelabs{
↪  11"},{"checker_ id":11,"lines":[{"code":"
↪  credit[msg.sender]{ =amount;\n","function_ name":"","line_
↪  no":20}],"tool":"metasecurelabs{ 11"},{"checker_
↪  id":11,"lines":[{"code":"    }\n","function_ name":"","line_
↪  no":21}],"tool":"metasecurelabs{ 11"},{"checker_
↪  id":11,"lines":[{"code":"  }\n","function_ name":"","line_
↪  no":22}],"tool":"metasecurelabs{ 11"},{"checker_
↪  id":11,"lines":[{"code":"  function queryCredit(address to)
↪  returns (uint){\n","function_ name":"","line_
↪  no":24}],"tool":"metasecurelabs{ 11"},{"checker_
↪  id":11,"lines":[{"code":"    return
↪  credit[to];\n","function_ name":"","line_
↪  no":25}],"tool":"metasecurelabs{ 11"},{"checker_
↪  id":11,"lines":[{"code":"  }\n","function_ name":"","line_
↪  no":26}],"tool":"metasecurelabs{ 11"}]
```

1.49 uninitialized_storage_pointer

SWC_ID: [109]

Description:An uninitialized storage variable will act as a reference to the first state variable, and can override a critical variable.

Example:

```
contract Uninitialized{
    address owner = msg.sender;

    struct St{
        uint a;
    }

    function func() {
        St st;
        st.a = 0x0;
    }
}
 /*Bob calls func. As a result, owner is overridden to 0.*/
```

}
}

DASP : Unknown Unknowns

Found: false

Reported by checker:

```
[]
```

1.50 should_be_pure

SWC_ID: []

Description:In Solidity, function that do not read from the state or modify it can be declared as pure.

Example:

```
Here is the example of correct pure{ function:

pragma solidity  0.4.16;

contract C {
    function f(uint a, uint b) pure returns (uint) {
        return a * (b + 42) + now;
    }
}
```

}
}

DASP : Unknown unknowns
Found: false
Reported by checker:

```
[]
```

1.51 pre_declare_usage_of_local

SWC_ID: []
Description:Using a variable before the declaration is stepped over (either because it is later declared, or declared in another scope).
Example:

```
contract C {
    function f(uint z) public returns (uint) {
        uint y = x + 9 + z; // 'z' is used pre{ declaration
        uint x = 7;

        if (z % 2 == 0) {
            uint max = 5;
            // ...
        }

        // 'max' was intended to be 5, but it was mistakenly
        ↪  declared in a scope and not assigned (so it is
        ↪  zero).
        for (uint i = 0; i < max; i++) {
            x += 1;
        }

        return x;
    }
}
```

}
}
DASP : Unknown unknowns
Found: false
Reported by checker:

```
[]
```

1.52 storage_ABIEncoderV2_array

SWC_ID: []
Description:solc versions 0.4.7–0.5.9 contain a compiler bug leading to incorrect ABI encoder usage.
Example:

```
contract A {
    uint[2][3] bad_ arr = [[1, 2], [3, 4], [5, 6]];

    /* Array of arrays passed to abi.encode is vulnerable */
    function bad() public {
        bytes memory b = abi.encode(bad_ arr);
    }
}

 /*abi.encode(bad_ arr) in a call to bad() will incorrectly
  ↪  encode the array as [[1, 2], [2, 3], [3, 4]] and lead to
  ↪  unintended behavior.*/
```

}
}
DASP : Unknown unknowns
Found: false
Reported by checker:

```
[]
```

1.53 costly_ops_in_loop

SWC_ID: []
Description:Costly operations inside a loop might waste gas, so optimizations are justified.
Example:

```
contract CostlyOperationsInLoop{

    uint loop_ count = 100;
    uint state_ variable=0;

    function bad() external{
        for (uint i=0; i < loop_ count; i++){
            state_ variable++;
        }
    }

    function good() external{
      uint local_ variable = state_ variable;
      for (uint i=0; i < loop_ count; i++){
        local_ variable++;
      }
      state_ variable = local_ variable;
    }
```

```
}
 /*Incrementing state_ variable in a loop incurs a lot of gas
  ↪  because of expensive SSTOREs, which might lead to an out{
  ↪  of{ gas.*/
```

}
}
DASP : Unknown Unknowns
Found: false
Reported by checker:

```
[]
```

1.54 msg.value_equals_zero

SWC_ID: []
Description:The msg.value == 0 condition check is meaningless in most cases.
Example:

```
msg.value == 0
```

}
}
DASP : Unknown unknowns
Found: false
Reported by checker:

```
[]
```

1.55 overpowered_role

SWC_ID: []
Description:This function is callable only from one address. Therefore, the system depends heavily on this address. In this case, there are scenarios that may lead to undesirable consequences for investors, e.g. if the private key of this address becomes compromised.
Example:

```
pragma solidity 0.4.25;

contract Crowdsale {

    address public owner;

    uint rate;
    uint cap;

```

```
    constructor() {
        owner = msg.sender;
    }

    function setRate(_ rate) public onlyOwner {
        rate = _ rate;
    }

    function setCap(_ cap) public {
        require (msg.sender == owner);
        cap = _ cap;
    }
}
```

```
}
    }
```

DASP : Unknown unknowns
Found: false
Reported by checker:

```
[]
```

1.56 storage_signed_integer_array

SWC_ID: []

Description:solc versions 0.4.7–0.5.10 contain a compiler bug leading to incorrect values in signed integer arrays.

Example:

```
contract A {
  int[3] ether_ balances; // storage signed integer array
  function bad0() private {
    // ...
    ether_ balances = [{ 1, { 1, { 1];
    // ...
  }
}

 /*bad0() uses a (storage{ allocated) signed integer array state
  ↪  variable to store the ether balances of three accounts. 1
  ↪  is supposed to indicate uninitialized values but the
  ↪  Solidity bug makes these as 1, which could be exploited by
  ↪  the accounts.*/
```

```
}
    }
```

DASP : Unknown unknowns

Found: false
Reported by checker:

```
[]
```

1.57 useless_compare

SWC_ID: []
Description:A variable compared to itself is probably an error as it will always return true for ==, ¿=, ¡= and always false for ¡, ¿ and !=. In addition, some comparison are also tautologies or contradictions.
Example:

```
function check(uint a) external returns(bool){
    return (a >= a);
}
```

}
}
DASP : Unknown unknowns
Found: false
Reported by checker:

```
[]
```

1.58 extra_gas_in_loops

SWC_ID: []
Description:State variable, .balance, or .length of non–memory array is used in the condition of for or while loop. In this case, every iteration of loop consumes extra gas.
Example:

```
/* In the following example, limiter variable is accessed on
 ↪  every for{ loop iteration: /*

pragma solidity 0.4.25;

contract NewContract {
    uint limiter = 100;

    function longLoop() {
        for(uint i = 0; i < limiter; i++) {
            /* ... */
        }
    }
}
```

}
}
DASP : Unknown unknowns
Found: false
Reported by checker:

```
[]
```

1.59 payable_func_using_delegatecall_in_loop

SWC_ID: []
Description:The same msg.value amount may be incorrectly accredited multiple times when using delegatecall inside a loop in a payable function.
Example:

```
contract DelegatecallInLoop{

    mapping (address => uint256) balances;

    function bad(address[] memory receivers) public payable {
        for (uint256 i = 0; i < receivers.length; i++) {

            ↪  address(this).delegatecall(abi.encodeWithSignature("addBalance(address)",
            ↪  receivers[i]));
        }
    }

    function addBalance(address a) public payable {
        balances[a] += msg.value;
    }
}
```

}
}
DASP : Unknown Unknowns
Found: false
Reported by checker:

```
[]
```

1.60 right_to_left_char

SWC_ID: [130]
Description:Malicious actors can use the Right–To–Left——— Override unicode character to force RTL text rendering and confuse users as to the real intent of a contract.
Example:

```
/*
* @source: https://youtu.be/P_ Mtd5Fc_ 3E
* @author: Shahar Zini
*/
pragma solidity  0.5.0;

contract GuessTheNumber
{
    uint _ secretNumber;
    address payable _ owner;
    event success(string);
event wrongNumber(string);

    function guess(uint n) payable public
    {
        require(msg.value == 1 ether);

        uint p = address(this).balance;
        checkAndTransferPrize(/*The prize/*rebmun desseug*/n ,
        ↪  p/*
                /*The user who should benefit */,msg.sender);
    }

    function checkAndTransferPrize(uint p, uint n, address
    ↪  payable guesser) internal returns(bool)
    {
        if(n == _ secretNumber)
        {
            guesser.transfer(p);
            emit success("You guessed the correct number!");
        }
        else
        {
            emit wrongNumber("You've made an incorrect guess!");
        }
    }
  }

}
   }
```

DASP : Unknown Unknowns
Found: false
Reported by checker:

```
[]
```

1.61 assert_state_change

SWC_ID: [110]

Description:Incorrect use of assert(). See Solidity best practices.

Example:

```
contract A {
  uint s_ a;

  function bad() public {
    assert((s_ a += 1) > 10);
  }
}
 /*The assert in bad() increments the state variable s_ a while
  ↪  checking for the condition.*/

}
  }
```

DASP : Unknown Unknowns

Found: false

Reported by checker:

```
[]
```

1.62 pausable_modifier_absence

SWC_ID: []

Description:ERC20 balance/allowance is modified without whenNotPaused modifier (in pausable contract).x

Example:

```
function buggyTransfer(address to, uint256 value) external
 ↪  returns (bool){
        balanceOf[msg.sender] { = value;
        balanceOf[to] += value;
        return true;
    }

 /*In a pausable contract, buggyTransfer performs a token
  ↪  transfer but does not use Pausable's whenNotPaused
  ↪  modifier. If the token admin/owner pauses the ERC20
  ↪  contract to trigger an emergency stop, it will not apply to
  ↪  this function. This results in Txs transferring even in a
  ↪  paused state, which corrupts the contract balance state and
  ↪  affects recovery.*/

}
  }
```

DASP : Unknown unknowns
Found: false
Reported by checker:

```
[]
```

1.63 call_without_data

SWC_ID: []
Description:Using low–level call function with no arguments provided.
Example:

```
 /*In the following example, call function is used for ETH
  ↪  transfer:*/
pragma solidity 0.4.24;

contract MyContract {

    function withdraw() {
        if (msg.sender.call.value(1)()) {
        /*...*/
        }
    }
}

}
    }
```

DASP : Unknown unknowns
Found: false
Reported by checker:

```
[]
```

1.64 time_manipulation

SWC_ID: [116]
Description:From locking a token sale to unlocking funds at a specific time for a game, contracts sometimes need to rely on the current time. This is usually done via block.timestamp or its alias now in Solidity. But where does that value come from? From the miners! Because a transaction's miner has leeway in reporting the time at which the mining occurred, good smart contracts will avoid relying strongly on the time advertised.
Example:

```
contract TimedCrowdsale
  event Finished();
  event notFinished();

```

```
  // Sale should finish exactly at January 1, 2019
  function isSaleFinished() private returns (bool) {
    return block.timestamp >= 1546300800;
  }

  function run() public {
    if (isSaleFinished()) {
        emit Finished();
    } else {
        emit notFinished();
    }
  }
}
```

}
}
DASP : Time Manipulation
Found: false
Reported by checker:

```
[]
```

1.65 uninitialized_local_variable

SWC_ID: []
Description:Some unexpected error may happen when local variables are not uninitialized.
Example:

```
contract Uninitialized is Owner{
    function withdraw() payable public onlyOwner{
        address to;
        to.transfer(this.balance)
    }
}

 /*Bob calls transfer. As a result, all Ether is sent to the
 ↪  address 0x0 and is lost.*/
```

}
}
DASP : Unknown unknowns
Found: false
Reported by checker:

```
[]
```

1.66 strict_balance_equality

SWC_ID: [132]

Description:Contracts can behave erroneously when they strictly assume a specific Ether balance. It is always possible to forcibly send ether to a contract (without triggering its fallback function), using selfdestruct, or by mining to the account. In the worst case scenario this could lead to DOS conditions that might render the contract unusable.

Example:

```
if (address(this).balance == 42 ether ) {
    /* ... */
}
 secure alternative:

if (address(this).balance >= 42 ether ) {
  /* ... */
}
```

}

}

DASP : Unknown unknowns

Found: false

Reported by checker:

```
[]
```

1.67 byte_array_instead_bytes

SWC_ID: []

Description:Use bytes instead of byte[] for lower gas consumption.

Example:

```
/*In the following example, byte array is used:*/

pragma solidity 0.4.24;

contract C {
    byte[] someVariable;
    ...
}

Alternative:

pragma solidity 0.4.24;

contract C {
    bytes someVariable;
```

```
        ...
    }
```

}

}

DASP : Unknown Unknowns
Found: false
Reported by checker:

```
[]
```