

Metasecurelabs analysis report

metasecurelabs.io

November 26, 2022

1 Introduction

1.1 dangerous_enum_conversion

SWC_ID:

Description: out-of-range enum conversion may occur (solc < 0.4.5).

Example:

```
1 | pragma solidity 0.4.2;
2 | contract Test{
3 |     enum E{a}
4 |     function bug(uint a) public returns(E){
5 |         return E(a);
6 |     }
7 | }
  | }
  | }
```

DASP : Unknown Unknowns

Found: false

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:

```
8 | /*In the following example, functions with both public and
   | ↪ external visibility modifiers are used: */
9 |
10 | contract Token {
11 |
12 |     mapping (address => uint256) internal _balances;
```

```

13
14     function transfer_public(address to, uint256 value) public
15         ↪ {
16             require(value <= _ balances[msg.sender]);
17
18             _ balances[msg.sender] { = value;
19             _ balances[to] += value;
20         }
21
22     function transfer_external(address to, uint256 value)
23         ↪ external {
24             require(value <= _ balances[msg.sender]);
25
26             _ balances[msg.sender] { = value;
27             _ balances[to] += value;
28         }
29     }
30
31     /*The second function requires less gas.*/
32 }
33
34 }
35
36 DASP : Unknown unknowns
37 Found: true

```

1.3 access_control

SWC_ID:

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:

```

30 contract TestContract is MultiOwnable {
31
32     function withdrawAll(){

```

```

33 |         msg.sender.transfer(this.balance);
34 |     }
35 | }
    | }
    | }
    DASP : Access control
    Found: true

```

1.4 erc20_event_not_indexed

SWC_ID:

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

Example:

```

36 | contract ERC20Bad {
37 |     // ...
38 |     event Transfer(address from, address to, uint value);
39 |     event Approval(address owner, address spender, uint value);
40 |
41 |     // ...
42 | }
43 |
44 | /*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

```

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:

```

45 | /* In the following example, contracts programmed to receive
   | → ether does not call transfer, send, or call.value function:
   | → */
46 |

```

```

47 | pragma solidity 0.4.25;
48 |
49 | contract BadMarketPlace {
50 |     function deposit() payable {
51 |         require(msg.value > 0);
52 |     }
53 | }
    |
    | }
    | }
    |
    | DASP : Unknown unknowns
    | Found: false

```

1.6 arithmetic

SWC_ID:

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 2^8-1 . 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:

```

54 | Integer overflow/underflow
55 | /*
56 | * @source: https://capturetheether.com/challenges/math/token{
57 |   ↪ sale/
58 | * @author: Steve Marx
59 | */
59 | pragma solidity 0.4.21;
60 | contract TokenSaleChallenge {
61 |     mapping(address => uint256) public balanceOf;
62 |     uint256 constant PRICE_PER_TOKEN = 1 ether;
63 |
64 |     function TokenSaleChallenge(address _player) public payable
65 |     ↪ {
        require(msg.value == 1 ether);
    }
}

```

```

66     }
67
68     function isComplete() public view returns (bool) {
69         return address(this).balance < 1 ether;
70     }
71
72     function buy(uint256 numTokens) public payable {
73         require(msg.value == numTokens * PRICE_ PER_ TOKEN);
74
75         balanceOf[msg.sender] += numTokens;
76     }
77 }
78
79 /*Division issues*/
80 contract Division {
81
82     /*function unsigned_ division(uint32 x, uint32 y) returns
83     ↪ (int r) {
84         //if (y == 0) { throw; }
85         r = x / y;
86     }*/
87
88     function signed_ division(int x, int y) returns (int) {
89         //if ((y == 0) ((x == { 2**255) & & (y == { 1}))) {
90         ↪ throw; }
91         return x / y;
92     }
93
94     /*Type deduction overflow*/
95     contract For_ Test {
96     ...
97     function Test () payable public {
98         if ( msg . value > 0.1 ether ) {
99             uint256 multi = 0;
100             uint256 amountToTransfer = 0;
101             for ( var i = 0; i < 2* msg . value ; i ++ ) {
102                 multi = i *2;
103                 if ( multi < amountToTransfer ) {
104                     break ;
105                     amountToTransfer = multi ;
106                 }
107             }
108             msg.sender.transfer( amountToTransfer );
109         }
110     }

```

```

110 | }
    |
    | }
    | }
    | DASP : Arithmetic
    | Found: true

```

1.7 incorrect_shift_in_assembly

SWC_ID:

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

Example:

```

111 | contract C {
112 |     function f() internal returns (uint a) {
113 |         assembly {
114 |             a := shr(a, 8)
115 |         }
116 |     }
    |
    | }
    |
    | }
    | DASP : Unknown Unknowns
    | Found: false

```

1.8 multiple_constructor_schemes

SWC_ID:

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

Example:

```

117 | contract A {
118 |     uint x;
119 |     constructor() public {
120 |         x = 0;
121 |     }
122 |     function A() public {
123 |         x = 1;
124 |     }
125 |
126 |     function test() public returns(uint) {
127 |         return x;
128 |     }
129 | }

```

```

130 |
131 | /*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

```

1.9 local_variable_shadowing

SWC_ID:

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

Example:

```

132 | pragma solidity 0.4.24;
133 |
134 | contract Bug {
135 |     uint owner;
136 |
137 |     function sensitive_function(address owner) public {
138 |         // ...
139 |         require(owner == msg.sender);
140 |     }
141 |
142 |     function alternate_sensitive_function() public {
143 |         address owner = msg.sender;
144 |         // ...
145 |         require(owner == msg.sender);
146 |     }
147 | }
148 |
149 | /*sensitive_function.owner shadows Bug.owner. As a result, the
    | ↪ use of owner in sensitive_function might be incorrect.*/
    |
    | }
    | }
    | DASP : Unknown unknowns
    | Found: false

```

1.10 redundant_code

SWC_ID:

Description: Redundant statements may have no effect.

Example:

```

150 contract RedundantStatementsContract {
151
152     constructor() public {
153         uint; // Elementary Type Name
154         bool; // Elementary Type Name
155         RedundantStatementsContract; // Identifier
156     }
157
158     function test() public returns (uint) {
159         uint; // Elementary Type Name
160         assert; // Identifier
161         test; // Identifier
162         return 777;
163     }
164 }
165
166 /*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

```

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:

```

167 Here is the example of correct view{ function:
168
169 contract C {
170     function f(uint a, uint b) view returns (uint) {
171         return a * (b + 42) + now;
172     }
173 }
}
}
DASP : Unknown unknowns
Found: false

```

1.12 arbitrary_from_in_transferFrom

SWC_ID:

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

Example:

```
174 function a(address from, address to, uint256 amount) public {
175     ERC20.transferFrom(from, to, am);
176 }
177
178 /*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

1.13 func_modifying_storage_array_by_value

SWC_ID:

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

Example:

```
179 contract Memory {
180     uint[1] public x; // storage
181
182     function f() public {
183         f1(x); // update x
184         f2(x); // do not update x
185     }
186
187     function f1(uint[1] storage arr) internal { // by reference
188         arr[0] = 1;
189     }
190
191     function f2(uint[1] arr) internal { // by value
192         arr[0] = 2;
193     }
194 }
195
196 /*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

```

1.14 dead_code

SWC_ID:

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:

```

197 | pragma solidity 0.5.0;
198 |
199 | contract DepositBox {
200 |     mapping(address => uint) balance;
201 |
202 |     // Accept deposit
203 |     function deposit(uint amount) public payable {
204 |         require(msg.value == amount, 'incorrect amount');
205 |         // Should update user balance
206 |         balance[msg.sender] = amount;
207 |     }
208 | }
    |
    | }
    | }
    | DASP : Unknown unknowns
    | Found: false

```

1.15 short_addresses

SWC_ID:

Description:MISSING

Example:

```

209 | MISSING
    |
    | }
    | }
    | DASP : Unknown unknowns
    | Found: false

```

1.16 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:

```
210 /*In the following example, currentBlockBlockhash function
    ↳ always returns 0:*/
211
212 pragma solidity 0.4.25;
213
214 contract MyContract {
215     function currentBlockHash() public view returns(bytes32) {
216         return blockhash(block.number);
217     }
218 }
    }
    }
DASP : Unknown unknowns
Found: false
```

1.17 msg.value_in_loop

SWC_ID:

Description: It is error-prone to use msg.value inside a loop.

Example:

```
219 contract MsgValueInLoop{
220     mapping (address => uint256) balances;
221
222     function bad(address[] memory receivers) public payable {
223         for (uint256 i=0; i < receivers.length; i++) {
224             balances[receivers[i]] += msg.value;
225         }
226     }
227 }
228
229 /*msg.value should be tracked through a local variable and
    ↳ decrease its amount on every iteration/usage.*/
    }
    }
DASP : Unknown unknowns
Found: false
```

1.18 reentrancy

SWC_ID:

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:

```
230 /*
231 * @source: http://blockchain.unica.it/projects/ethereum{
232 ↪ survey/attacks.htmlsimpledao
233 * @author: {
234 * @vulnerable_at_lines: 19
235 */
236
237 pragma solidity 0.4.2;
238
239 contract SimpleDAO {
240     mapping (address => uint) public credit;
241
242     function donate(address to) payable {
243         credit[to] += msg.value;
244     }
245
246     function withdraw(uint amount) {
247         if (credit[msg.sender]>= amount) {
248             // <yes> <report> REENTRANCY
249             bool res = msg.sender.call.value(amount)();
250             credit[msg.sender]{ =amount;
251         }
252     }
253 }
254
255 }
256
257 }
258
259 }
260
261 }
262
263 }
264
265 }
266
267 }
268
269 }
270
271 }
272
273 }
274
275 }
276
277 }
278
279 }
280
281 }
282
283 }
284
285 }
286
287 }
288
289 }
290
291 }
292
293 }
294
295 }
296
297 }
298
299 }
300
301 }
302
303 }
304
305 }
306
307 }
308
309 }
310
311 }
312
313 }
314
315 }
316
317 }
318
319 }
320
321 }
322
323 }
324
325 }
326
327 }
328
329 }
330
331 }
332
333 }
334
335 }
336
337 }
338
339 }
340
341 }
342
343 }
344
345 }
346
347 }
348
349 }
350
351 }
352
353 }
354
355 }
356
357 }
358
359 }
360
361 }
362
363 }
364
365 }
366
367 }
368
369 }
370
371 }
372
373 }
374
375 }
376
377 }
378
379 }
380
381 }
382
383 }
384
385 }
386
387 }
388
389 }
390
391 }
392
393 }
394
395 }
396
397 }
398
399 }
400
401 }
402
403 }
404
405 }
406
407 }
408
409 }
410
411 }
412
413 }
414
415 }
416
417 }
418
419 }
420
421 }
422
423 }
424
425 }
426
427 }
428
429 }
430
431 }
432
433 }
434
435 }
436
437 }
438
439 }
440
441 }
442
443 }
444
445 }
446
447 }
448
449 }
450
451 }
452
453 }
454
455 }
456
457 }
458
459 }
460
461 }
462
463 }
464
465 }
466
467 }
468
469 }
470
471 }
472
473 }
474
475 }
476
477 }
478
479 }
480
481 }
482
483 }
484
485 }
486
487 }
488
489 }
490
491 }
492
493 }
494
495 }
496
497 }
498
499 }
500
501 }
502
503 }
504
505 }
506
507 }
508
509 }
510
511 }
512
513 }
514
515 }
516
517 }
518
519 }
520
521 }
522
523 }
524
525 }
526
527 }
528
529 }
530
531 }
532
533 }
534
535 }
536
537 }
538
539 }
540
541 }
542
543 }
544
545 }
546
547 }
548
549 }
550
551 }
552
553 }
554
555 }
556
557 }
558
559 }
560
561 }
562
563 }
564
565 }
566
567 }
568
569 }
570
571 }
572
573 }
574
575 }
576
577 }
578
579 }
580
581 }
582
583 }
584
585 }
586
587 }
588
589 }
590
591 }
592
593 }
594
595 }
596
597 }
598
599 }
600
601 }
602
603 }
604
605 }
606
607 }
608
609 }
610
611 }
612
613 }
614
615 }
616
617 }
618
619 }
620
621 }
622
623 }
624
625 }
626
627 }
628
629 }
630
631 }
632
633 }
634
635 }
636
637 }
638
639 }
640
641 }
642
643 }
644
645 }
646
647 }
648
649 }
650
651 }
652
653 }
654
655 }
656
657 }
658
659 }
660
661 }
662
663 }
664
665 }
666
667 }
668
669 }
670
671 }
672
673 }
674
675 }
676
677 }
678
679 }
680
681 }
682
683 }
684
685 }
686
687 }
688
689 }
690
691 }
692
693 }
694
695 }
696
697 }
698
699 }
700
701 }
702
703 }
704
705 }
706
707 }
708
709 }
710
711 }
712
713 }
714
715 }
716
717 }
718
719 }
720
721 }
722
723 }
724
725 }
726
727 }
728
729 }
730
731 }
732
733 }
734
735 }
736
737 }
738
739 }
740
741 }
742
743 }
744
745 }
746
747 }
748
749 }
750
751 }
752
753 }
754
755 }
756
757 }
758
759 }
760
761 }
762
763 }
764
765 }
766
767 }
768
769 }
770
771 }
772
773 }
774
775 }
776
777 }
778
779 }
780
781 }
782
783 }
784
785 }
786
787 }
788
789 }
790
791 }
792
793 }
794
795 }
796
797 }
798
799 }
800
801 }
802
803 }
804
805 }
806
807 }
808
809 }
810
811 }
812
813 }
814
815 }
816
817 }
818
819 }
820
821 }
822
823 }
824
825 }
826
827 }
828
829 }
830
831 }
832
833 }
834
835 }
836
837 }
838
839 }
840
841 }
842
843 }
844
845 }
846
847 }
848
849 }
850
851 }
852
853 }
854
855 }
856
857 }
858
859 }
860
861 }
862
863 }
864
865 }
866
867 }
868
869 }
870
871 }
872
873 }
874
875 }
876
877 }
878
879 }
880
881 }
882
883 }
884
885 }
886
887 }
888
889 }
890
891 }
892
893 }
894
895 }
896
897 }
898
899 }
900
901 }
902
903 }
904
905 }
906
907 }
908
909 }
910
911 }
912
913 }
914
915 }
916
917 }
918
919 }
920
921 }
922
923 }
924
925 }
926
927 }
928
929 }
930
931 }
932
933 }
934
935 }
936
937 }
938
939 }
940
941 }
942
943 }
944
945 }
946
947 }
948
949 }
950
951 }
952
953 }
954
955 }
956
957 }
958
959 }
960
961 }
962
963 }
964
965 }
966
967 }
968
969 }
970
971 }
972
973 }
974
975 }
976
977 }
978
979 }
980
981 }
982
983 }
984
985 }
986
987 }
988
989 }
990
991 }
992
993 }
994
995 }
996
997 }
998
999 }
```

DASP : Reentrancy

Found: true

1.19 visibility

SWC_ID:

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:

```

253  /*In this example, a specific modifier, such as public, is not
    ↪  used when declaring a function: */
254
255  function foo();
256
257  Preferred alternatives:
258
259  function foo() public;
260  function foo() internal;
    }
    }
    DASP : Unknown Unknowns
    Found: true

```

1.20 array_length_manipulation

SWC_ID:

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:

```

261  pragma solidity 0.4.24;
262
263  contract dataStorage {
264      uint[] public data;
265
266      function writeData(uint[] _ data) external {
267          for(uint i = data.length; i < _ data.length; i++) {
268              data.length++;
269              data[i]=_ data[i];
270          }
271      }
272  }
    }
    }
    DASP : Unknown Unknowns
    Found: false

```

1.21 denial_of_service

SWC_ID:

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 an infinite loop. * Recursive external calls may consume a large number of callstacks, which may lead to DoS.

Example:

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

DASP : Denial of Services

Found: true

1.22 uninitialized_state_variable

SWC_ID:

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

Example:

```
274 | contract Uninitialized{  
275 |     address destination;  
276 |  
277 |     function transfer() payable public{  
278 |         destination.transfer(msg.value);  
279 |     }  
280 | }  
  
    | }  
    | }
```

DASP : Unknown unknowns

Found: true

1.23 unchecked_calls

SWC_ID:

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:

```
281 pragma solidity 0.4.25;
282
283 contract ReturnValue {
284
285     checked
286     function callchecked(address callee) public {
287         require(callee.call());
288     }
289
290     function callnotchecked(address callee) public {
291         callee.call();
292     }
293 }
294
295 }
296
297 DASP : Unchecked Low Level Calls
298 Found: true
```

1.24 unused_retval

SWC_ID:

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

Example:

```
294 contract MyConc{
295     using SafeMath for uint;
296     function my_ func(uint a, uint b) public{
297         a.add(b);
298     }
299 }
300
301 /*MyConc calls add of SafeMath, but does not store the result
302 ↪ in a. As a result, the computation has no effect. */
303
304 }
305
306 DASP : Unknown Unknowns
307 Found: true
```

1.25 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:

```

302  /*This code uses incorrect function signature:*/
303
304  pragma solidity 0.5.1;
305  contract Signature {
306      function callFoo(address addr, uint value) public returns
307          ↪ (bool) {
308          bytes memory data = abi.encodeWithSignature("foo(uint)",
309              ↪ value);
310          (bool status, ) = addr.call(data);
311          return status;
312      }
313  }
314
315  /*Use "foo(uint256)" instead.*/
316
317  }
318
319  }
320
321  DASP : Unknown Unknowns
322  Found: false

```

1.26 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:

```

324  pragma solidity 0.4.0;
325
326  contract A{
327      uint num = 5;
328      constructor(uint x) public{
329          num += x;
330      }
331  }
332
333  contract B is A{
334      constructor() A(2) public { /* ... */ }
335  }
336

```



```

327 | contract C is A {
328 |     constructor() A(3) public { /* ... */ }
329 | }
330 |
331 | contract D is B, C {
332 |     constructor() public { /* ... */ }
333 | }
    |
    | }
    | }
    | DASP : Unknown unknowns
    | Found: false

```

1.27 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:

```

334 | modifier myModif(){
335 |     if(..){
336 |         - ;
337 |     }
338 | }
339 | function get() myModif returns(uint){}
340 |
341 | /*If the condition in myModif is false, the execution of get()
    | ↪ will return 0.*/
    |
    | }
    | }
    | DASP : Unknown unknowns
    | Found: false

```

1.28 deprecated_standards

SWC_ID:

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:

```

342 pragma solidity 0.4.24;
343
344 contract BreakThisHash {
345     bytes32 hash;
346     uint birthday;
347     constructor(bytes32 _hash) public payable {
348         hash = _hash;
349         birthday = now;
350     }
351
352     function kill(bytes password) external {
353         if (sha3(password) != hash) {
354             throw;
355         }
356         suicide(msg.sender);
357     }
358
359     function hashAge() public constant returns(uint) {
360         return(now - birthday);
361     }
362 }
363
364 /*Use keccak256, selfdestruct, revert() instead.*/
}
}
DASP : Unknown unknowns
Found: false

```

1.29 state_variable_shadowing

SWC_ID:

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:

```

365 pragma solidity 0.4.25;
366
367 contract Tokensale {
368     uint public hardcap = 10000 ether;
369
370     function Tokensale() {}
371

```

```

372     function fetchCap() public constant returns(uint) {
373         return hardcap;
374     }
375 }
376
377 contract Presale is Tokensale {
378     //uint hardcap = 1000 ether;
379     //If the hardcap variables were both needed we would have to
    ↪ rename one to fix this.
380     function Presale() Tokensale() {
381         hardcap = 1000 ether;
382     }
383 }
    }
    }
    DASP : Unknown Unknowns
    Found: false

```

1.30 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:

```

384 function callme(){
385     if( ! (msg.sender.call()( ) ) ){
386         throw;
387     }
388     counter += 1
389 }
390
391 /*callme() contains a benign reentrancy.*/
    }
    }
    DASP : Unknown unknowns
    Found: false

```

1.31 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:

```

392  /* In the following example, the send function is used:*/
393
394
395  if(!addr.send(42 ether)) {
396      revert();
397  }
398
399  /*Preferred alternative:
400
401  addr.transfer(42 ether);*/
402  }
403  }
404  DASP : Unknown Unknowns
405  Found: false

```

1.32 race_condition

SWC_ID:

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:

```

402  /* In this example, one can front{ run transactions to claim
403  ↪ his/her reward before the owner reduces the reward amount.*/
404
405
406  pragma solidity 0.4.16;
407
408  contract EthTxOrderDependenceMinimal {
409      address public owner;
410      bool public claimed;
411      uint public reward;
412
413      function EthTxOrderDependenceMinimal() public {
414          owner = msg.sender;
415      }
416
417      function setReward() public payable {
418          require (!claimed);
419          require(msg.sender == owner);

```

```

418         owner.transfer(reward);
419         reward = msg.value;
420     }
421
422     function claimReward(uint256 submission) {
423         require (!claimed);
424         require(submission < 10);
425         msg.sender.transfer(reward);
426         claimed = true;
427     }
428 }
}
}
DASP : Front Running
Found: true

```

1.33 uninitialized_func_pointer

SWC_ID:

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

Example:

```

429 contract Bug{
430     function buy() public payable{
431         uint minted = msg.value * (1000 / address(this).balance);
432         // ...
433     }
434 }
435
436 /*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

```

1.34 modifier_like_Sol_keyword

SWC_ID:

Description: A contract may contain modifier that looks similar to Solidity keyword

Example:

```

437 contract Contract{
438     modifier public() {
439     }
440
441     function doSomething() public {
442         require(owner == msg.sender);
443         owner = newOwner;
444     }
445 }
446
447 /*public is a modifier meant to look like a Solidity keyword.*/
}
}
DASP : Unknown Unknowns
Found: false

```

1.35 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:

```

448 contract Token{
449     function ownerOf(uint256 _ tokenId) external view returns
450         ↪ (bool);
451     //...
452 }
453
454 }
}
DASP : Unknown unknowns
Found: false

```

1.36 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:

```

452 contract Token{
453     function transfer(address to, uint value) external;
454     //...

```

```

455 }
456
457 /*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

```

1.37 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:

```

458 struct BalancesStruct{
459     address owner;
460     mapping(address => uint) balances;
461 }
462 mapping(address => BalancesStruct) public stackBalance;
463
464 function remove() internal{
465     delete stackBalance[msg.sender];
466 }
}
}
DASP : Unknown unknowns
Found: false

```

1.38 use_after_delete

SWC_ID:

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

Example:

```

467 mapping(address => uint) public balances;
468 function f() public {
469     delete balances[msg.sender];
470     msg.sender.transfer(balances[msg.sender]);
471 }

```

```

472 |
473 | /*balances[msg.sender] is deleted before it's sent to the
    | ↳ caller, leading the transfer to always send zero.*/
    |
    | }
    | }
    | DASP : Unknown unknowns
    | Found: false

```

1.39 function_declared_return_but_no_return

SWC_ID:

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

Example:

```

474 | /*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:*/
475 |
476 | pragma solidity 0.4.25;
477 |
478 | contract NewContract {
479 |     uint minimumBuy;
480 |
481 |     function setMinimumBuy(uint256 newMinimumBuy) returns
    |     ↳ (bool){
482 |         minimumBuy = newMinimumBuy;
483 |     }
484 | }
    |
    | }
    | }
    | DASP : Unknown unknowns
    | Found: false

```

1.40 controlled_lowlevel_call

SWC_ID:

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

Example:

```

485 | address token;
486 |
487 | function call_token(bytes data){
488 |     token.call(data);
489 | }

```



```

490 |
491 | /*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

```

1.41 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:

```

492 | /*In the following contract, the address is specified in the
    | ↳ source code:*/
    |
493 |
494 | pragma solidity 0.4.24;
495 | contract C {
496 |     function f(uint a, uint b) pure returns (address) {
497 |         address public multisig =
    | ↳ 0xf64B584972FE6055a770477670208d737Fff282f;
498 |         return multisig;
499 |     }
500 | }
501 |
502 | /*Do not forget to check the contract at the address
    | ↳ 0xf64B584972FE6055a770477670208d737Fff282f for
    | ↳ vulnerabilities.*/
    |
    | }
    | }
    | DASP : Unknown unknowns
    | Found: false

```

1.42 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:

```

503  /*In the following example, amount variable is divided by
    ↪ DELIMITER and then multiplied by BONUS. Thus, a rounding
    ↪ error appears (consider amount = 9000):*/
504
505  pragma solidity 0.4.25;
506
507  contract MyContract {
508
509      uint constant BONUS = 500;
510      uint constant DELIMITER = 10000;
511
512      function calculateBonus(uint amount) returns (uint) {
513          return amount/DELIMITER*BONUS;
514      }
515  }
516
517  }
518
519  }
520
521  }
522
523  }
524
525  }
526
527  }
528
529  }
530
531  }
532
533  }
534
535  }
536
537  }
538
539  }
540
541  }
542
543  }
544
545  }
546
547  }
548
549  }
550
551  }
552
553  }
554
555  }
556
557  }
558
559  }
560
561  }
562
563  }
564
565  }
566
567  }
568
569  }
570
571  }
572
573  }
574
575  }
576
577  }
578
579  }
580
581  }
582
583  }
584
585  }
586
587  }
588
589  }
590
591  }
592
593  }
594
595  }
596
597  }
598
599  }
600
601  }
602
603  }
604
605  }
606
607  }
608
609  }
610
611  }
612
613  }
614
615  }
616
617  }
618
619  }
620
621  }
622
623  }
624
625  }
626
627  }
628
629  }
630
631  }
632
633  }
634
635  }
636
637  }
638
639  }
640
641  }
642
643  }
644
645  }
646
647  }
648
649  }
650
651  }
652
653  }
654
655  }
656
657  }
658
659  }
660
661  }
662
663  }
664
665  }
666
667  }
668
669  }
670
671  }
672
673  }
674
675  }
676
677  }
678
679  }
680
681  }
682
683  }
684
685  }
686
687  }
688
689  }
690
691  }
692
693  }
694
695  }
696
697  }
698
699  }
700
701  }
702
703  }
704
705  }
706
707  }
708
709  }
710
711  }
712
713  }
714
715  }
716
717  }
718
719  }
720
721  }
722
723  }
724
725  }
726
727  }
728
729  }
730
731  }
732
733  }
734
735  }
736
737  }
738
739  }
740
741  }
742
743  }
744
745  }
746
747  }
748
749  }
750
751  }
752
753  }
754
755  }
756
757  }
758
759  }
760
761  }
762
763  }
764
765  }
766
767  }
768
769  }
770
771  }
772
773  }
774
775  }
776
777  }
778
779  }
780
781  }
782
783  }
784
785  }
786
787  }
788
789  }
790
791  }
792
793  }
794
795  }
796
797  }
798
799  }
800
801  }
802
803  }
804
805  }
806
807  }
808
809  }
810
811  }
812
813  }
814
815  }
816
817  }
818
819  }
820
821  }
822
823  }
824
825  }
826
827  }
828
829  }
830
831  }
832
833  }
834
835  }
836
837  }
838
839  }
840
841  }
842
843  }
844
845  }
846
847  }
848
849  }
850
851  }
852
853  }
854
855  }
856
857  }
858
859  }
860
861  }
862
863  }
864
865  }
866
867  }
868
869  }
870
871  }
872
873  }
874
875  }
876
877  }
878
879  }
880
881  }
882
883  }
884
885  }
886
887  }
888
889  }
890
891  }
892
893  }
894
895  }
896
897  }
898
899  }
900
901  }
902
903  }
904
905  }
906
907  }
908
909  }
910
911  }
912
913  }
914
915  }
916
917  }
918
919  }
920
921  }
922
923  }
924
925  }
926
927  }
928
929  }
930
931  }
932
933  }
934
935  }
936
937  }
938
939  }
940
941  }
942
943  }
944
945  }
946
947  }
948
949  }
950
951  }
952
953  }
954
955  }
956
957  }
958
959  }
960
961  }
962
963  }
964
965  }
966
967  }
968
969  }
970
971  }
972
973  }
974
975  }
976
977  }
978
979  }
980
981  }
982
983  }
984
985  }
986
987  }
988
989  }
990
991  }
992
993  }
994
995  }
996
997  }
998
999  }
1000
1001  }
1002
1003  }
1004
1005  }
1006
1007  }
1008
1009  }
1010
1011  }
1012
1013  }
1014
1015  }
1016
1017  }
1018
1019  }
1020
1021  }
1022
1023  }
1024
1025  }
1026
1027  }
1028
1029  }
1030
1031  }
1032
1033  }
1034
1035  }
1036
1037  }
1038
1039  }
1040
1041  }
1042
1043  }
1044
1045  }
1046
1047  }
1048
1049  }
1050
1051  }
1052
1053  }
1054
1055  }
1056
1057  }
1058
1059  }
1060
1061  }
1062
1063  }
1064
1065  }
1066
1067  }
1068
1069  }
1070
1071  }
1072
1073  }
1074
1075  }
1076
1077  }
1078
1079  }
1080
1081  }
1082
1083  }
1084
1085  }
1086
1087  }
1088
1089  }
1090
1091  }
1092
1093  }
1094
1095  }
1096
1097  }
1098
1099  }
1100
1101  }
1102
1103  }
1104
1105  }
1106
1107  }
1108
1109  }
1110
1111  }
1112
1113  }
1114
1115  }
1116
1117  }
1118
1119  }
1120
1121  }
1122
1123  }
1124
1125  }
1126
1127  }
1128
1129  }
1130
1131  }
1132
1133  }
1134
1135  }
1136
1137  }
1138
1139  }
1140
1141  }
1142
1143  }
1144
1145  }
1146
1147  }
1148
1149  }
1150
1151  }
1152
1153  }
1154
1155  }
1156
1157  }
1158
1159  }
1160
1161  }
1162
1163  }
1164
1165  }
1166
1167  }
1168
1169  }
1170
1171  }
1172
1173  }
1174
1175  }
1176
1177  }
1178
1179  }
1180
1181  }
1182
1183  }
1184
1185  }
1186
1187  }
1188
1189  }
1190
1191  }
1192
1193  }
1194
1195  }
1196
1197  }
1198
1199  }
1200
1201  }
1202
1203  }
1204
1205  }
1206
1207  }
1208
1209  }
1210
1211  }
1212
1213  }
1214
1215  }
1216
1217  }
1218
1219  }
1220
1221  }
1222
1223  }
1224
1225  }
1226
1227  }
1228
1229  }
1230
1231  }
1232
1233  }
1234
1235  }
1236
1237  }
1238
1239  }
1240
1241  }
1242
1243  }
1244
1245  }
1246
1247  }
1248
1249  }
1250
1251  }
1252
1253  }
1254
1255  }
1256
1257  }
1258
1259  }
1260
1261  }
1262
1263  }
1264
1265  }
1266
1267  }
1268
1269  }
1270
1271  }
1272
1273  }
1274
1275  }
1276
1277  }
1278
1279  }
1280
1281  }
1282
1283  }
1284
1285  }
1286
1287  }
1288
1289  }
1290
1291  }
1292
1293  }
1294
1295  }
1296
1297  }
1298
1299  }
1300
1301  }
1302
1303  }
1304
1305  }
1306
1307  }
1308
1309  }
1310
1311  }
1312
1313  }
1314
1315  }
1316
1317  }
1318
1319  }
1320
1321  }
1322
1323  }
1324
1325  }
1326
1327  }
1328
1329  }
1330
1331  }
1332
1333  }
1334
1335  }
1336
1337  }
1338
1339  }
1340
1341  }
1342
1343  }
1344
1345  }
1346
1347  }
1348
1349  }
1350
1351  }
1352
1353  }
1354
1355  }
1356
1357  }
1358
1359  }
1360
1361  }
1362
1363  }
1364
1365  }
1366
1367  }
1368
1369  }
1370
1371  }
1372
1373  }
1374
1375  }
1376
1377  }
1378
1379  }
1380
1381  }
1382
1383  }
1384
1385  }
1386
1387  }
1388
1389  }
1390
1391  }
1392
1393  }
1394
1395  }
1396
1397  }
1398
1399  }
1400
1401  }
1402
1403  }
1404
1405  }
1406
1407  }
1408
1409  }
1410
1411  }
1412
1413  }
1414
1415  }
1416
1417  }
1418
1419  }
1420
1421  }
1422
1423  }
1424
1425  }
1426
1427  }
1428
1429  }
1430
1431  }
1432
1433  }
1434
1435  }
1436
1437  }
1438
1439  }
1440
1441  }
1442
1443  }
1444
1445  }
1446
1447  }
1448
1449  }
1450
1451  }
1452
1453  }
1454
1455  }
1456
1457  }
1458
1459  }
1460
1461  }
1462
1463  }
1464
1465  }
1466
1467  }
1468
1469  }
1470
1471  }
1472
1473  }
1474
1475  }
1476
1477  }
1478
1479  }
1480
1481  }
1482
1483  }
1484
1485  }
1486
1487  }
1488
1489  }
1490
1491  }
1492
1493  }
1494
1495  }
1496
1497  }
1498
1499  }
1500
1501  }
1502
1503  }
1504
1505  }
1506
1507  }
1508
1509  }
1510
1511  }
1512
1513  }
1514
1515  }
1516
1517  }
1518
1519  }
1520
1521  }
1522
1523  }
1524
1525  }
1526
1527  }
1528
1529  }
1530
1531  }
1532
1533  }
1534
1535  }
1536
1537  }
1538
1539  }
1540
1541  }
1542
1543  }
1544
1545  }
1546
1547  }
1548
1549  }
1550
1551  }
1552
1553  }
1554
1555  }
1556
1557  }
1558
1559  }
1560
1561  }
1562
1563  }
1564
1565  }
1566
1567  }
1568
1569  }
1570
1571  }
1572
1573  }
1574
1575  }
1576
1577  }
1578
1579  }
1580
1581  }
1582
1583  }
1584
1585  }
1586
1587  }
1588
1589  }
1590
1591  }
1592
1593  }
1594
1595  }
1596
1597  }
1598
1599  }
1600
1601  }
1602
1603  }
1604
1605  }
1606
1607  }
1608
1609  }
1610
1611  }
1612
1613  }
1614
1615  }
1616
1617  }
1618
1619  }
1620
1621  }
1622
1623  }
1624
1625  }
1626
1627  }
1628
1629  }
1630
1631  }
1632
1633  }
1634
1635  }
1636
1637  }
1638
1639  }
1640
1641  }
1642
1643  }
1644
1645  }
1646
1647  }
1648
1649  }
1650
1651  }
1652
1653  }
1654
1655  }
1656
1657  }
1658
1659  }
1660
1661  }
1662
1663  }
1664
1665  }
1666
1667  }
1668
1669  }
1670
1671  }
1672
1673  }
1674
1675  }
1676
1677  }
1678
1679  }
1680
1681  }
1682
1683  }
1684
1685  }
1686
1687  }
1688
1689  }
1690
1691  }
1692
1693  }
1694
1695  }
1696
1697  }
1698
1699  }
1700
1701  }
1702
1703  }
1704
1705  }
1706
1707  }
1708
1709  }
1710
1711  }
1712
1713  }
1714
1715  }
1716
1717  }
1718
1719  }
1720
1721  }
1722
1723  }
1724
1725  }
1726
1727  }
1728
1729  }
1730
1731  }
1732
1733  }
1734
1735  }
1736
1737  }
1738
1739  }
1740
1741  }
1742
1743  }
1744
1745  }
1746
1747  }
1748
1749  }
1750
1751  }
1752
1753  }
1754
1755  }
1756
1757  }
1758
1759  }
1760
1761  }
1762
1763  }
1764
1765  }
1766
1767  }
1768
1769  }
1770
1771  }
1772
1773  }
1774
1775  }
1776
1777  }
1778
1779  }
1780
1781  }
1782
1783  }
1784
1785  }
1786
1787  }
1788
1789  }
1790
1791  }
1792
1793  }
1794
1795  }
1796
1797  }
1798
1799  }
1800
1801  }
1802
1803  }
1804
1805  }
1806
1807  }
1808
1809  }
1810
1811  }
1812
1813  }
1814
1815  }
1816
1817  }
1818
1819  }
1820
1821  }
1822
1823  }
1824
1825  }
1826
1827  }
1828
1829  }
1830
1831  }
1832
1833  }
1834
1835  }
1836
1837  }
1838
1839  }
1840
1841  }
1842
1843  }
1844
1845  }
1846
1847  }
1848
1849  }
1850
1851  }
1852
1853  }
1854
1855  }
1856
1857  }
1858
1859  }
1860
1861  }
1862
1863  }
1864
1865  }
1866
1867  }
1868
1869  }
1870
1871  }
1872
1873  }
1874
1875  }
1876
1877  }
1878
1879  }
1880
1881  }
1882
1883  }
1884
1885  }
1886
1887  }
1888
1889  }
1890
1891  }
1892
1893  }
1894
1895  }
1896
1897  }
1898
1899  }
1900
1901  }
1902
1903  }
1904
1905  }
1906
1907  }
1908
1909  }
1910
1911  }
1912
1913  }
1914
1915  }
1916
1917  }
1918
1919  }
1920
1921  }
1922
1923  }
1924
1925  }
1926
1927  }
1928
1929  }
1930
1931  }
1932
1933  }
1934
1935  }
1936
1937  }
1938
1939  }
1940
1941  }
1942
1943  }
1944
1945  }
1946
1947  }
1948
1949  }
1950
1951  }
1952
1953  }
1954
1955  }
1956
1957  }
1958
1959  }
1960
1961  }
1962
1963  }
1964
1965  }
1966
1967  }
1968
1969  }
1970
1971  }
1972
1973  }
1974
1975  }
1976
1977  }
1978
1979  }
1980
1981  }
1982
1983  }
1984
1985  }
1986
1987  }
1988
1989  }
1990
1991  }
1992
1993  }
1994
1995  }
1996
1997  }
1998
1999  }
2000
2001  }
2002
2003  }
2004
2005  }
2006
2007  }
2008
2009  }
2010
2011  }
2012
2013  }
2014
2015  }
2016
2017  }
2018
2019  }
2020
2021  }
2022
2023  }
2024
2025  }
2026
2027  }
2028
2029  }
2030
2031  }
2032
2033  }
2034
2035  }
2036
2037  }
2038
2039  }
2040
2041  }
2042
2043  }
2044
2045  }
2046
2047  }
2048
2049  }
2050
2051  }
2052
2053  }
2054
2055  }
2056
2057  }
2058
2059  }
2060
2061  }
2062
2063  }
2064
2065  }
2066
2067  }
2068
2069  }
2070
2071  }
2072
2073  }
2074
2075  }
2076
2077  }
2078
2079  }
2080
2081  }
2082
2083  }
2084
2085  }
2086
2087  }
2088
2089  }
2090
2091  }
2092
2093  }
2094
2095  }
2096
2097  }
2098
2099  }
2100
2101  }
2102
2103  }
2104
2105  }
2106
2107  }
2108
2109  }
2110
2111  }
2112
2113  }
2114
2115  }
2116
2117  }
2118
2119  }
2120
2121  }
2122
2123  }
2124
2125  }
2126
2127  }
2128
2129  }
2130
2131  }
2132
2133  }
2134
2135  }
2136
2137  }
2138
2139  }
2140
2141  }
2142
2143  }
2144
2145  }
2146
2147  }
2148
2149  }
2150
2151  }
2152
2153  }
2154
2155  }
2156
2157  }
2158
2159  }
2160
2161  }
2162
2163  }
2164
2165  }
2166
2167  }
2168
2169  }
2170
2171  }
2172
2173  }
2174
2175  }
2176
2177  }
2178
2179  }
2180
2181  }
2182
2183  }
2184
2185  }
2186
2187  }
2188
2189  }
2190
2191  }
2192
2193  }
2194
2195  }
2196
2197  }
2198
2199  }
2200
2201  }
2202
2203  }
2204
2205  }
2206
2207  }
2208
2209  }
2210
2211  }
2212
2213  }
2214
2215  }
2216
2217  }
2218
2219  }
2220
2221  }
2222
2223  }
2224
2225  }
2226
2227  }
2228
2229  }
2230
2231  }
2232
2233  }
2234
2235  }
2236
2237  }
2238
2239  }
2240
2241  }
2242
2243  }
2244
2245  }
2246
2247  }
2248
2249  }
2250
2251  }
2252
2253  }
2254
2255  }
2256
2257  }
2258
2259  }
2260
2261  }
2262
2263  }
2264
2265  }
2266
2267  }
2268
2269  }
2270
2271  }
2272
2273  }
2274
2275  }
2276
2277  }
2278
2279  }
2280
2281  }
2282
2283  }
2284
2285  }
2286
2287  }
2288
2289  }
2290
2291  }
2292
2293  }
2294
2295  }
2296
2297  }
2298
2299  }
2300
2301  }
2302
2303  }
2304
2305  }
2306
2307  }
2308
2309  }
2310
2311  }
2312
2313  }
2314
2315  }
2316
2317  }
2318
2319  }
2320
2321  }
2322
2323  }
2324
2325  }
2326
2327  }
2328
2329  }
2330
2331  }
2332
2333  }
2334
2335  }
2336
2337  }
2338
2339  }
2340
2341  }
2342
2343  }
2344
2345  }
2346
2347  }
2348
2349  }
2350
2351  }
2352
2353  }
2354
2355  }
2356
2357  }
2358
2359  }
2360
2361  }
2362
2363  }
2364
2365  }
2366
2367  }
2368
2369  }
2370
2371  }
2372
2373  }
2374
2375  }
2376
2377  }
2378
2379  }
2380
2381  }
2382
2383  }
2384
2385  }
2386
2387  }
2388
2389  }
2390
2391  }
2392
2393  }
2394
2395  }
2396
2397  }
2398
2399  }
2400
2401  }
2402
2403  }
2404
2405  }
2406
2407  }
2408
2409  }
2410
2411  }
2412
2413  }
2414
2415  }
2416
2417  }
2418
2419  }
2420
2421  }
2422
2423  }
2424
2425  }
2426
2427  }
2428
2429  }
2430
2431  }
2432
2433  }
2434
2435  }
2436
2437  }
2438
2439  }
2440
2441  }
2442
2443  }
2444
2445  }
2446
2447  }
2448
2449  }
2450
2451  }
2452
2453  }
2454
2455  }
2456
2457  }
2458
2459  }
2459
2460  }
2461
2462  }
2463
2464  }
2465
2466  }
2467
2468  }
2468
2469  }
2469
2470  }
2470
2471  }
2471
2472  }
2472
2473  }
2473
2474  }
2474
2475  }
2475
2476  }
2476
2477  }
2477
2478  }
2478
2479  }
2479
2480  }
2480
2481  }
2481
2482  }
2482
2483  }
2483
2484  }
2484
2485  }
2485
2486  }
2486
2487  }
2487
2488  }
2488
2489  }
2489
2490  }
2490
2491  }
2491
2492  }
2492
2493  }
2493
2494  }
2494
2495  }
2495
2496  }
2496
2497  }
2497
2498  }
2498
2499  }
2499
2500  }
2500
2501  }
2501
2502  }
2502
2503  }
2503
2504  }
2504
2505  }
2505
2506  }
2506
2507  }
2507
2508  }
2508
2509  }
2509
2510  }
2510
2511  }
2511
2512  }
2512
2513  }
2513
2514  }
2514
2515  }
2515
2516  }
2516
2517  }
2517
2518  }
2518
2519  }
2519
2520  }
2520
2521  }
2521
2522  }
2522
2523  }
2523
2524  }
2524
2525  }
2525
2526  }
2526
2527  }
2527
2528  }
2528
2529  }
2529
2530  }
2530
2531  }
2531
2532  }
2532
2533  }
2533
2534  }
2534
2535  }
2535
2536  }
2536
2537  }
2537
2538  }
2538
2539  }
2539
2540  }
2540
2541  }
2541
2542  }
2542
2543  }
2543
2544  }
2544
2545  }
2545
2546  }
2546
2547  }
254
```

```

534 |     constructor() public { /* ... */ }
535 | }
    |
    | }
    | }
    | DASP : Unknown unknowns
    | Found: false

```

1.44 unused_state_variables

SWC_ID:

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:

```

536 | pragma solidity >=0.5.0;
537 | pragma experimental ABIEncoderV2;
538 |
539 | import "./base.sol";
540 |
541 | contract DerivedA is Base {
542 |     // i is not used in the current contract
543 |     A i = A(1);
544 |
545 |     int internal j = 500;
546 |
547 |     function call(int a) public {
548 |         assign1(a);
549 |     }
550 |
551 |     function assign3(A memory x) public returns (uint) {
552 |         return g[1] + x.a + uint(j);
553 |     }
554 |
555 |     function ret() public returns (int){
556 |         return this.e();
557 |     }
558 |
559 |     int internal j = 500;
560 |     function call(int a) public {
561 |         assign1(a);
562 |     }
563 |
564 |     function assign3(A memory x) public returns (uint) {

```

```

565         return g[1] + x.a + uint(j);
566     }
567
568     function ret() public returns (int){
569         return this.e();
570     }
571 }
}
}
DASP : Unknown unknowns
Found: false

```

1.45 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:

```

572 /*The following loop is infinite:*/
573
574 do {
575     continue;
576 } while(false);
}
}
DASP : Unknown Unknowns
Found: false

```

1.46 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:

```

577 pragma solidity 0.4.24;
578
579 contract Bug {
580     uint now; // Overshadows current time stamp.
581
582     function assert(bool condition) public {
583         // Overshadows built{ in symbol for providing
584         ↪ assertions.
585     }
586 }

```

```

585     function get_next_expiration(uint earlier_time) private
586         ↪ returns (uint) {
587             return now + 259200; // References overshadowed
588             ↪ timestamp.
589         }
590     }
591 }
592 }
DASP : Unknown unknowns
Found: false

```

1.47 ignore

SWC_ID:

Description:Other trivial bug types.

Example:

```

590 }
591 }
DASP : Unknown Unknowns
Found: true

```

1.48 uninitialized_storage_pointer

SWC_ID:

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

Example:

```

591 contract Uninitialized{
592     address owner = msg.sender;
593
594     struct St{
595         uint a;
596     }
597
598     function func() {
599         St st;
600         st.a = 0x0;
601     }
602 }
603 /*Bob calls func. As a result, owner is overridden to 0.*/

```

```

}
}
DASP : Unknown Unknowns
Found: false

```

1.49 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:

```

604 Here is the example of correct pure{ function:
605
606 pragma solidity 0.4.16;
607
608 contract C {
609     function f(uint a, uint b) pure returns (uint) {
610         return a * (b + 42) + now;
611     }
612 }
613
614 }
615 DASP : Unknown unknowns
616 Found: false

```

1.50 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:

```

613 contract C {
614     function f(uint z) public returns (uint) {
615         uint y = x + 9 + z; // 'z' is used pre{ declaration
616         uint x = 7;
617
618         if (z % 2 == 0) {
619             uint max = 5;
620             // ...
621         }
622
623         // 'max' was intended to be 5, but it was mistakenly
        → declared in a scope and not assigned (so it is
        → zero).

```

```

624         for (uint i = 0; i < max; i++) {
625             x += 1;
626         }
627
628         return x;
629     }
630 }

```

```

}
}
DASP : Unknown unknowns
Found: false

```

1.51 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:

```

631 contract A {
632     uint[2][3] bad_arr = [[1, 2], [3, 4], [5, 6]];
633
634     /* Array of arrays passed to abi.encode is vulnerable */
635     function bad() public {
636         bytes memory b = abi.encode(bad_arr);
637     }
638 }
639
640 /*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

```

1.52 costly_ops_in_loop

SWC_ID:

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

Example:

```

641 contract CostlyOperationsInLoop{
642

```

```

643     uint loop_count = 100;
644     uint state_variable=0;
645
646     function bad() external{
647         for (uint i=0; i < loop_count; i++){
648             state_variable++;
649         }
650     }
651
652     function good() external{
653         uint local_variable = state_variable;
654         for (uint i=0; i < loop_count; i++){
655             local_variable++;
656         }
657         state_variable = local_variable;
658     }
659 }
660 /*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

```

1.53 msg.value_equals_zero

SWC_ID:

Description:The msg.value == 0 condition check is meaningless in most cases.

Example:

```

661 | msg.value == 0
    | }
    | }
DASP : Unknown unknowns
Found: false

```

1.54 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:


```

662 pragma solidity 0.4.25;
663
664 contract Crowdsale {
665
666     address public owner;
667
668     uint rate;
669     uint cap;
670
671     constructor() {
672         owner = msg.sender;
673     }
674
675     function setRate(_ rate) public onlyOwner {
676         rate = _ rate;
677     }
678
679     function setCap(_ cap) public {
680         require (msg.sender == owner);
681         cap = _ cap;
682     }
683 }
}
}
DASP : Unknown unknowns
Found: false

```

1.55 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:

```

684 contract A {
685     int[3] ether_balances; // storage signed integer array
686     function bad0() private {
687         // ...
688         ether_balances = [{ 1, { 1, { 1];
689         // ...
690     }
691 }
692

```

```

693 | /*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

```

1.56 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:

```

694 | function check(uint a) external returns(bool){
695 |     return (a >= a);
696 | }
    |
    | }
    | }
    | DASP : Unknown unknowns
    | Found: false

```

1.57 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:

```

697 | /* In the following example, limiter variable is accessed on
    | ↪ every for{ loop iteration: /*
698 |
699 | pragma solidity 0.4.25;
700 |
701 | contract NewContract {
702 |     uint limiter = 100;
703 |
704 |     function longLoop() {
705 |         for(uint i = 0; i < limiter; i++) {
706 |             /* ... */

```

```

707     }
708   }
709 }
}
}
DASP : Unknown unknowns
Found: false

```

1.58 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:

```

710 contract DelegatecallInLoop{
711
712     mapping (address => uint256) balances;
713
714     function bad(address[] memory receivers) public payable {
715         for (uint256 i = 0; i < receivers.length; i++) {
716
717             ↪ address(this).delegatecall(abi.encodeWithSignature("addBalance(address)",
718             ↪ receivers[i]));
719
720         }
721     }
722
723     function addBalance(address a) public payable {
724         balances[a] += msg.value;
725     }
726 }
}
}
DASP : Unknown Unknowns
Found: false

```

1.59 right_to_left_char

SWC_ID:

Description: Malicious actors can use the Right-To-Left—— Override uni-code character to force RTL text rendering and confuse users as to the real intent of a contract.

Example:

```

724 /*
725 * @source: https://youtu.be/P_Mtd5Fc_3E

```

```

726 * @author: Shahar Zini
727 */
728 pragma solidity 0.5.0;
729
730 contract GuessTheNumber
731 {
732     uint _secretNumber;
733     address payable _owner;
734     event success(string);
735     event wrongNumber(string);
736
737     function guess(uint n) payable public
738     {
739         require(msg.value == 1 ether);
740
741         uint p = address(this).balance;
742         checkAndTransferPrize(/*The prize/*rebmun desseug*/n ,
743                               ↪ p/*
744                               /*The user who should benefit */ ,msg.sender);
745     }
746
747     function checkAndTransferPrize(uint p, uint n, address
748     ↪ payable guesser) internal returns(bool)
749     {
750         if(n == _secretNumber)
751         {
752             guesser.transfer(p);
753             emit success("You guessed the correct number!");
754         }
755         else
756         {
757             emit wrongNumber("You've made an incorrect guess!");
758         }
759     }
760 }
761
762 }
763
764 }
765
766 }
767
768 }
769
770 }
771
772 }
773
774 }
775
776 }
777
778 }
779
780 }
781
782 }
783
784 }
785
786 }
787
788 }
789
790 }
791
792 }
793
794 }
795
796 }
797
798 }
799
800 }
801
802 }
803
804 }
805
806 }
807
808 }
809
810 }
811
812 }
813
814 }
815
816 }
817
818 }
819
820 }
821
822 }
823
824 }
825
826 }
827
828 }
829
830 }
831
832 }
833
834 }
835
836 }
837
838 }
839
840 }
841
842 }
843
844 }
845
846 }
847
848 }
849
850 }
851
852 }
853
854 }
855
856 }
857
858 }
859
860 }
861
862 }
863
864 }
865
866 }
867
868 }
869
870 }
871
872 }
873
874 }
875
876 }
877
878 }
879
880 }
881
882 }
883
884 }
885
886 }
887
888 }
889
890 }
891
892 }
893
894 }
895
896 }
897
898 }
899
900 }
901
902 }
903
904 }
905
906 }
907
908 }
909
910 }
911
912 }
913
914 }
915
916 }
917
918 }
919
920 }
921
922 }
923
924 }
925
926 }
927
928 }
929
930 }
931
932 }
933
934 }
935
936 }
937
938 }
939
940 }
941
942 }
943
944 }
945
946 }
947
948 }
949
950 }
951
952 }
953
954 }
955
956 }
957
958 }
959
960 }
961
962 }
963
964 }
965
966 }
967
968 }
969
970 }
971
972 }
973
974 }
975
976 }
977
978 }
979
980 }
981
982 }
983
984 }
985
986 }
987
988 }
989
990 }
991
992 }
993
994 }
995
996 }
997
998 }
999
1000 }

```

DASP : Unknown Unknowns
Found: false

1.60 assert_state_change

SWC_ID:

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

Example:

```

759 contract A {
760     uint s_ a;
761
762     function bad() public {
763         assert((s_ a += 1) > 10);
764     }
765 }
766 /*The assert in bad() increments the state variable s_ a while
    ↪ checking for the condition.*/
}
}
DASP : Unknown Unknowns
Found: false

```

1.61 pausable_modifier_absence

SWC_ID:

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

Example:

```

767 function buggyTransfer(address to, uint256 value) external
    ↪ returns (bool){
768     balanceOf[msg.sender] { = value;
769     balanceOf[to] += value;
770     return true;
771 }
772
773 /*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

```

1.62 call_without_data

SWC_ID:

Description:Using low-level call function with no arguments provided.

Example:

```

774  /*In the following example, call function is used for ETH
    ↪ transfer:*/
775  pragma solidity 0.4.24;
776
777  contract MyContract {
778
779      function withdraw() {
780          if (msg.sender.call.value(1)()) {
781              /*...*/
782          }
783      }
784  }
    }
    }
    DASP : Unknown unknowns
    Found: false

```

1.63 time_manipulation

SWC_ID:

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:

```

785  contract TimedCrowdsale
786      event Finished();
787      event notFinished();
788
789      // Sale should finish exactly at January 1, 2019
790      function isSaleFinished() private returns (bool) {
791          return block.timestamp >= 1546300800;
792      }
793
794      function run() public {
795          if (isSaleFinished()) {
796              emit Finished();
797          } else {
798              emit notFinished();
799          }
800      }
801  }

```

```

}
}
DASP : Time Manipulation
Found: false

```

1.64 uninitialized_local_variable

SWC_ID:

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

Example:

```

802 contract Uninitialized is Owner{
803     function withdraw() payable public onlyOwner{
804         address to;
805         to.transfer(this.balance)
806     }
807 }
808
809 /*Bob calls transfer. As a result, all Ether is sent to the
   ↪ address 0x0 and is lost.*/
}
}
DASP : Unknown unknowns
Found: false

```

1.65 strict_balance_equality

SWC_ID:

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:

```

810 if (address(this).balance == 42 ether ) {
811     /* ... */
812 }
813 secure alternative:
814
815 if (address(this).balance >= 42 ether ) {
816     /* ... */
817 }
}

```

```
}  
DASP : Unknown unknowns  
Found: false
```

1.66 byte_array_instead_bytes

SWC_ID:

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

Example:

```
818 /*In the following example, byte array is used:*/  
819  
820 pragma solidity 0.4.24;  
821  
822 contract C {  
823     byte[] someVariable;  
824     ...  
825 }  
826  
827 Alternative:  
828  
829 pragma solidity 0.4.24;  
830  
831 contract C {  
832     bytes someVariable;  
833     ...  
834 }  
  
}  
DASP : Unknown Unknowns  
Found: false
```