

# 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:**

```
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 | }  
8 |  
9 | }  
10| }
```

**DASP :** Unknown Unknowns

**Found:** false

**Reported by checker:**

8 | []

### 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:**

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

```

11 contract Token {
12
13     mapping (address => uint256) internal _ balances;
14
15     function transfer_public(address to, uint256 value) public
16     ↪ {
17         require(value <= _ balances[msg.sender]);
18
19         _ balances[msg.sender] { = value;
20         _ balances[to] += value;
21     }
22
23     function transfer_external(address to, uint256 value)
24     ↪ external {
25         require(value <= _ balances[msg.sender]);
26
27         _ balances[msg.sender] { = value;
28         _ balances[to] += value;
29     }
30 }
31
32 /*The second function requires less gas.*/
33
34 }
35
36 DASP : Unknown unknowns
37 Found: true
38 Reported by checker:
39
40 [{"checker_ id":9,"lines":[],"tool":"metasecurelabs{
41 ↪ 9"}, {"checker_ id":9,"lines":[],"tool":"metasecurelabs{
42 ↪ 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:**

```

32 contract TestContract is MultiOwnable {
33
34     function withdrawAll(){
35         msg.sender.transfer(this.balance);
36     }
37 }
    }
    }
DASP : Access control
Found: true
Reported by checker:
38 [{"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:**

```

39 contract ERC20Bad {
40     // ...
41     event Transfer(address from, address to, uint value);
42     event Approval(address owner, address spender, uint value);
43
44     // ...
45 }
46
47 /*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:**

48 | []

## 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:**

```
49 | /* In the following example, contracts programmed to receive
    | ↪ ether does not call transfer, send, or call.value function:
    | ↪ */
50 |
51 | pragma solidity 0.4.25;
52 |
53 | contract BadMarketPlace {
54 |     function deposit() payable {
55 |         require(msg.value > 0);
56 |     }
57 | }
    |
    | }
    | }
```

**DASP :** Unknown unknowns

**Found:** false

**Reported by checker:**

58 | []

## 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<sup>extsuperscript8</sup>-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:**

```
59 Integer overflow/underflow
60 /*
61  * @source: https://capturetheether.com/challenges/math/token{
62   ↪ sale/
63  * @author: Steve Marx
64  */
65 pragma solidity 0.4.21;
66 contract TokenSaleChallenge {
67     mapping(address => uint256) public balanceOf;
68     uint256 constant PRICE_PER_TOKEN = 1 ether;
69
70     function TokenSaleChallenge(address _player) public payable
71     ↪ {
72         require(msg.value == 1 ether);
73     }
74
75     function isComplete() public view returns (bool) {
76         return address(this).balance < 1 ether;
77     }
78
79     function buy(uint256 numTokens) public payable {
80         require(msg.value == numTokens * PRICE_PER_TOKEN);
81
82         balanceOf[msg.sender] += numTokens;
83     }
84 }
85
86 /*Division issues*/
87 contract Division {
88
89     /*function unsigned_division(uint32 x, uint32 y) returns
90     ↪ (int r) {
91         //if (y == 0) { throw; }
92         r = x / y;
93     }*/
94
95     function signed_division(int x, int y) returns (int) {
96         //if ((y == 0) ((x == { 2**255) && (y == { 1}))) {
97         ↪ throw; }
98         return x / y;
```

```

95     }
96
97 }
98
99 /*Type deduction overflow*/
100 contract For_Test {
101     ...
102     function Test () payable public {
103         if ( msg . value > 0.1 ether ) {
104             uint256 multi = 0;
105             uint256 amountToTransfer = 0;
106             for ( var i = 0; i < 2* msg . value ; i ++ ) {
107                 multi = i *2;
108                 if ( multi < amountToTransfer ) {
109                     break ;
110                     amountToTransfer = multi ;
111                 }
112                 msg.sender.transfer( amountToTransfer );
113             }
114         }
115     }
116 }
117
118 }
119
120 }
121
122 }
123
124 }
125
126 }
127
128 }
129
130 }
131
132 }
133
134 }
135
136 }
137
138 }
139
140 }
141
142 }
143
144 }
145
146 }
147
148 }
149
150 }
151
152 }
153
154 }
155
156 }
157
158 }
159
160 }
161
162 }
163
164 }
165
166 }
167
168 }
169
170 }
171
172 }
173
174 }
175
176 }
177
178 }
179
180 }
181
182 }
183
184 }
185
186 }
187
188 }
189
190 }
191
192 }
193
194 }
195
196 }
197
198 }
199
200 }
201
202 }
203
204 }
205
206 }
207
208 }
209
210 }
211
212 }
213
214 }
215
216 }
217
218 }
219
220 }
221
222 }
223
224 }
225
226 }
227
228 }
229
230 }
231
232 }
233
234 }
235
236 }
237
238 }
239
240 }
241
242 }
243
244 }
245
246 }
247
248 }
249
250 }
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
1000 }

```

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

```

116 [{"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:**

```
117 | contract C {  
118 |     function f() internal returns (uint a) {  
119 |         assembly {  
120 |             a := shr(a, 8)  
121 |         }  
122 |     }  
123 | }  
124 |  
125 | }
```

**DASP :** Unknown Unknowns

**Found:** false

**Reported by checker:**

```
123 | []
```

## 1.8 multiple\_constructor\_schemes

**SWC\_ID:** []

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

**Example:**

```
124 | contract A {  
125 |     uint x;  
126 |     constructor() public {  
127 |         x = 0;  
128 |     }  
129 |     function A() public {  
130 |         x = 1;  
131 |     }  
132 |  
133 |     function test() public returns(uint) {  
134 |         return x;  
135 |     }  
136 | }  
137 |  
138 | /*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:**

139 | []

## 1.9 local\_variable\_shadowing

**SWC\_ID:** []

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

**Example:**

```
140 | pragma solidity 0.4.24;
141 |
142 | contract Bug {
143 |     uint owner;
144 |
145 |     function sensitive_function(address owner) public {
146 |         // ...
147 |         require(owner == msg.sender);
148 |     }
149 |
150 |     function alternate_sensitive_function() public {
151 |         address owner = msg.sender;
152 |         // ...
153 |         require(owner == msg.sender);
154 |     }
155 | }
156 |
157 | /*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:
```

158 | []

## 1.10 redundant\_code

**SWC\_ID:** []

**Description:** Redundant statements may have no effect.

**Example:**

```
159 | contract RedundantStatementsContract {
160 |
161 |     constructor() public {
162 |         uint; // Elementary Type Name
```



```

163     bool; // Elementary Type Name
164     RedundantStatementsContract; // Identifier
165 }
166
167     function test() public returns (uint) {
168         uint; // Elementary Type Name
169         assert; // Identifier
170         test; // Identifier
171         return 777;
172     }
173 }
174
175 /*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:
176 []

```

## 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:**

```

177 Here is the example of correct view{ function:
178
179 contract C {
180     function f(uint a, uint b) view returns (uint) {
181         return a * (b + 42) + now;
182     }
183 }
184
185 }
186
187 DASP : Unknown unknowns
188 Found: false
189 Reported by checker:
190
191 []

```

## 1.12 arbitrary\_from\_in\_transferFrom

**SWC\_ID:** []

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

**Example:**

```
185 function a(address from, address to, uint256 amount) public {
186     ERC20.transferFrom(from, to, am);
187 }
188
189 /*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:**

```
190 | []
```

## 1.13 func\_modifying\_storage\_array\_by\_value

**SWC\_ID:** []

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

**Example:**

```
191 contract Memory {
192     uint[1] public x; // storage
193
194     function f() public {
195         f1(x); // update x
196         f2(x); // do not update x
197     }
198
199     function f1(uint[1] storage arr) internal { // by reference
200         arr[0] = 1;
201     }
202
203     function f2(uint[1] arr) internal { // by value
204         arr[0] = 2;
205     }
206 }
```

```

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:**

```
223 | contract Bug{
224 |     function buy() public payable{
225 |         uint minted = msg.value * (1000 / address(this).balance);
226 |         // ...
227 |     }
228 | }
229 |
230 | /*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:
```

231 | []

### 1.16 short\_addresses

**SWC\_ID:** []

**Description:** MISSING

**Example:**

```
232 | MISSING
    |
    | }
    | }
    | DASP : Unknown unknowns
    | Found: false
    | Reported by checker:
```

233 | []

### 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:**

```

234 | /*In the following example, currentBlockBlockhash function
    | ↪ always returns 0:*/
235 |
236 | pragma solidity 0.4.25;
237 |
238 | contract MyContract {
239 |     function currentBlockHash() public view returns(bytes32) {
240 |         return blockhash(block.number);
241 |     }
242 | }
    |
    | }
    | }
    | DASP : Unknown unknowns
    | Found: false
    | Reported by checker:
243 | []

```

### 1.18 msg.value\_in\_loop

**SWC\_ID:** []

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

**Example:**

```

244 | contract MsgValueInLoop{
245 |     mapping (address => uint256) balances;
246 |
247 |     function bad(address[] memory receivers) public payable {
248 |         for (uint256 i=0; i < receivers.length; i++) {
249 |             balances[receivers[i]] += msg.value;
250 |         }
251 |     }
252 | }
253 |
254 | /*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:
255 | []

```

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:**

```

256  /*
257  * @source: http://blockchain.unica.it/projects/ethereum{
    ↳ survey/attacks.htmlsimpledao
258  * @author: {
259  * @vulnerable_at_lines: 19
260  */
261
262  pragma solidity 0.4.2;
263
264  contract SimpleDAO {
265      mapping (address => uint) public credit;
266
267      function donate(address to) payable {
268          credit[to] += msg.value;
269      }
270
271      function withdraw(uint amount) {
272          if (credit[msg.sender] >= amount) {
273              // <yes> <report> REENTRANCY
274              bool res = msg.sender.call.value(amount)();
275              credit[msg.sender]{ =amount;
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
1000  }

```

**DASP : Reentrancy**  
**Found: true**  
**Reported by checker:**

```

279 [{"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:**

```

280 /*In this example, a specific modifier, such as public, is not
    ↳ used when declaring a function: */
281
282 function foo();
283
284 Preferred alternatives:
285
286 function foo() public;
287 function foo() internal;
    }
    }

```

**DASP** : Unknown Unknowns

**Found**: true

**Reported by checker**:

```
288 [{"checker_
    → id":10,"lines":[{"code":"functiondonate(addrresto)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(addrresto)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**:

```
289 pragma solidity 0.4.24;
290
291 contract dataStorage {
292     uint[] public data;
293
294     function writeData(uint[] _ data) external {
295         for(uint i = data.length; i < _ data.length; i++) {
296             data.length++;
297             data[i]=_ data[i];
298         }
299     }
300 }
}
```

**DASP** : Unknown Unknowns

**Found**: false

**Reported by checker**:

```
301 | []
```

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

**Example:**

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

**DASP** : Denial of Services

**Found**: true

**Reported by checker**:

```
303 | [{"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**: `UNINITIALIZED_STATE_VARIABLE`

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

**Example:**

```
304 | contract Uninitialized{
305 |     address destination;
306 |
307 |     function transfer() payable public{
308 |         destination.transfer(msg.value);
309 |     }
310 | }
```

**DASP** : Unknown unknowns

**Found**: true

**Reported by checker**:

```

311 [{"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:**

```

312 pragma solidity 0.4.25;
313
314 contract ReturnValue {
315
316     checked
317     function callchecked(address callee) public {
318         require(callee.call());
319     }
320
321     function callnotchecked(address callee) public {
322         callee.call();
323     }
324 }

```

**DASP :** Unchecked Low Level Calls

**Found:** true

**Reported by checker:**

```

325 [{"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:**

```

326 contract MyConc{
327     using SafeMath for uint;
328     function my_func(uint a, uint b) public{
329         a.add(b);
330     }
331 }
332
333 /*MyConc calls add of SafeMath, but does not store the result
    ↪ in a. As a result, the computation has no effect. */
334
335 }

```

**DASP :** Unknown Unknowns

**Found:** true

**Reported by checker:**

```

334 [{"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:**

```

335 | /*This code uses incorrect function signature:*/
336 |
337 | pragma solidity 0.5.1;
338 | contract Signature {
339 |     function callFoo(address addr, uint value) public returns
340 |         ↪ (bool) {
341 |         bytes memory data = abi.encodeWithSignature("foo(uint)",
342 |         ↪ value);
343 |         (bool status, ) = addr.call(data);
344 |         return status;
345 |     }
346 | }
347 |
348 | /*Use "foo(uint256)" instead.*/
349 |
350 | }
351 |
352 | DASP : Unknown Unknowns
353 | Found: false
354 | Reported by checker:
355 |
356 | []

```

## 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:**

```

348 | pragma solidity 0.4.0;
349 |
350 | contract A{
351 |     uint num = 5;
352 |     constructor(uint x) public{
353 |         num += x;
354 |     }
355 | }
356 |
357 | contract B is A{
358 |     constructor() A(2) public { /* ... */ }
359 | }
360 |
361 | contract C is A {
362 |     constructor() A(3) public { /* ... */ }
363 | }
364 |

```

```

365 | contract D is B, C {
366 |     constructor() public { /* ... */ }
367 | }
    |
    | }
    | }
    | DASP : Unknown unknowns
    | Found: false
    | Reported by checker:
368 | []

```

## 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:**

```

369 | modifier myModif(){
370 |     if(..){
371 |         - ;
372 |     }
373 | }
374 | function get() myModif returns(uint){}
375 |
376 | /*If the condition in myModif is false, the execution of get()
    | ↪ will return 0.*/
    |
    | }
    | }
    | DASP : Unknown unknowns
    | Found: false
    | Reported by checker:
377 | []

```

## 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:**

```

378 | pragma solidity 0.4.24;
379 |
380 | contract BreakThisHash {
381 |     bytes32 hash;
382 |     uint birthday;
383 |     constructor(bytes32 _hash) public payable {
384 |         hash = _hash;
385 |         birthday = now;
386 |     }
387 |
388 |     function kill(bytes password) external {
389 |         if (sha3(password) != hash) {
390 |             throw;
391 |         }
392 |         suicide(msg.sender);
393 |     }
394 |
395 |     function hashAge() public constant returns(uint) {
396 |         return(now { birthday});
397 |     }
398 | }
399 |
400 | /*Use keccak256, selfdestruct, revert() instead.*/
    |
    | }
    | }
    | DASP : Unknown unknowns
    | Found: false
    | Reported by checker:
401 | []

```

### 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:**

```

402 | pragma solidity 0.4.25;
403 |
404 | contract Tokensale {
405 |     uint public hardcap = 10000 ether;

```

```

406         function Tokensale() {}
407
408         function fetchCap() public constant returns(uint) {
409             return hardcap;
410         }
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
417     ↪ rename one to fix this.
418     function Presale() Tokensale() {
419         hardcap = 1000 ether;
420     }
421 }
422 }
423
424 DASP : Unknown Unknowns
425 Found: false
426 Reported by checker:
427
428 []

```

### 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.*/
430 }
431 }
432
433 DASP : Unknown unknowns
434 Found: false
435 Reported by checker:
436
437 []

```

### 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:**

```
431 | /* In the following example, the send function is used:*/
432 |
433 |
434 | if(!addr.send(42 ether)) {
435 |     revert();
436 | }
437 |
438 | /*Preferred alternative:
439 |
440 | addr.transfer(42 ether);*/
    | }
    | }
    | DASP : Unknown Unknowns
    | Found: false
    | Reported by checker:
441 | []
```

### 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:**

```
442 | /* In this example, one can front{ run transactions to claim
    | ↪ his/her reward before the owner reduces the reward amount.*/
443 |
444 | pragma solidity 0.4.16;
445 |
446 | contract EthTxOrderDependenceMinimal {
447 |     address public owner;
```



```

448     bool public claimed;
449     uint public reward;
450
451     function EthTxOrderDependenceMinimal() public {
452         owner = msg.sender;
453     }
454
455     function setReward() public payable {
456         require (!claimed);
457         require(msg.sender == owner);
458         owner.transfer(reward);
459         reward = msg.value;
460     }
461
462     function claimReward(uint256 submission) {
463         require (!claimed);
464         require(submission < 10);
465         msg.sender.transfer(reward);
466         claimed = true;
467     }
468 }
}
}
DASP : Front Running
Found: true
Reported by checker:
469 [{"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:**

```

470 contract bad0 {
471
472     constructor() public {
473         /* Uninitialized function pointer */

```

```

474     function(uint256) internal returns(uint256) a;
475     a(10);
476 }
477 }
478
479 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:
480 | []

```

### 1.35 modifier\_like\_Sol\_keyword

**SWC\_ID**: []

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

**Example**:

```

481 contract Contract{
482     modifier public() {
483     }
484
485     function doSomething() public {
486         require(owner == msg.sender);
487         owner = newOwner;
488     }
489 }
490
491 /*public is a modifier meant to look like a Solidity keyword.*/
}
}
DASP : Unknown Unknowns
Found: false
Reported by checker:
492 | []

```

### 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:**

```

493 | contract Token{
494 |     function ownerOf(uint256 _ tokenId) external view returns
      |         ↪ (bool);
495 |     //...
496 | }

```

}

}

**DASP :** Unknown unknowns

**Found:** false

**Reported by checker:**

```

497 | []

```

### 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:**

```

498 | contract Token{
499 |     function transfer(address to, uint value) external;
500 |     //...
501 | }

```

502

503

```

/*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:**

```

504 | []

```

### 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:**

```
505 struct BalancesStruct{
506     address owner;
507     mapping(address => uint) balances;
508 }
509 mapping(address => BalancesStruct) public stackBalance;
510
511 function remove() internal{
512     delete stackBalance[msg.sender];
513 }
514 }
```

**DASP :** Unknown unknowns

**Found:** false

**Reported by checker:**

514 | []

### 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:**

```
515 mapping(address => uint) public balances;
516 function f() public {
517     delete balances[msg.sender];
518     msg.sender.transfer(balances[msg.sender]);
519 }
520
521 /*balances[msg.sender] is deleted before it's sent to the
   ↪ caller, leading the transfer to always send zero.*/
522 }
```

**DASP :** Unknown unknowns

**Found:** false

**Reported by checker:**

522 | []

## 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:**

```
523 | /*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:*/
524 |
525 | pragma solidity 0.4.25;
526 |
527 | contract NewContract {
528 |     uint minimumBuy;
529 |
530 |     function setMinimumBuy(uint256 newMinimumBuy) returns
    |     ↪ (bool){
531 |         minimumBuy = newMinimumBuy;
532 |     }
533 | }
```

**DASP :** Unknown unknowns

**Found:** false

**Reported by checker:**

```
534 | []
```

## 1.41 controlled\_lowlevel\_call

**SWC\_ID:** []

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

**Example:**

```
535 | address token;
536 |
537 | function call_token(bytes data){
538 |     token.call(data);
539 | }
540 |
541 | /*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**:

```
542 [{"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**:

```
543 /*In the following contract, the address is specified in the  
    ↳ source code:*/  
  
544  
545 pragma solidity 0.4.24;  
546 contract C {  
547     function f(uint a, uint b) pure returns (address) {  
548         address public multisig =  
            ↳ 0xf64B584972FE6055a770477670208d737Fff282f;  
549         return multisig;  
550     }  
551 }  
  
552  
553 /*Do not forget to check the contract at the address  
    ↳ 0xf64B584972FE6055a770477670208d737Fff282f for  
    ↳ vulnerabilities.*/  
  
}  
  
}
```

**DASP** : Unknown unknowns

**Found**: false

**Reported by checker**:

```
554 | []
```

## 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**:

```

555 | /*In the following example, amount variable is divided by
    | ↪ DELIMITER and then multiplied by BONUS. Thus, a rounding
    | ↪ error appears (consider amount = 9000):*/
556 |
557 | pragma solidity 0.4.25;
558 |
559 | contract MyContract {
560 |
561 |     uint constant BONUS = 500;
562 |     uint constant DELIMITER = 10000;
563 |
564 |     function calculateBonus(uint amount) returns (uint) {
565 |         return amount/DELIMITER*BONUS;
566 |     }
567 | }
    |
    | }
    |
    | }
    |
    | DASP : Unknown Unknowns
    | Found: false
    | Reported by checker:
568 | []

```

## 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:**

```

569 | pragma solidity 0.4.0;
570 |
571 | contract A{
572 |     uint num = 5;
573 |     constructor(uint x) public{
574 |         num += x;
575 |     }
576 | }
577 |
578 | contract B is A{
579 |     constructor() A(2) public { /* ... */ }
580 | }
581 |
582 | contract C is A {
583 |     constructor() A(3) public { /* ... */ }
584 | }

```

```

585 |
586 | contract D is B, C {
587 |     constructor() public { /* ... */ }
588 | }
589 |
590 | }
591 |
592 | DASP : Unknown unknowns
593 | Found: false
594 | Reported by checker:
595 |
596 | []

```

## 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:**

```

590 | pragma solidity >=0.5.0;
591 | pragma experimental ABIEncoderV2;
592 |
593 | import "./base.sol";
594 |
595 | contract DerivedA is Base {
596 |     // i is not used in the current contract
597 |     A i = A(1);
598 |
599 |     int internal j = 500;
600 |
601 |     function call(int a) public {
602 |         assign1(a);
603 |     }
604 |
605 |     function assign3(A memory x) public returns (uint) {
606 |         return g[1] + x.a + uint(j);
607 |     }
608 |
609 |     function ret() public returns (int){
610 |         return this.e();
611 |     }
612 |
613 |     int internal j = 500;

```



```

614 | function call(int a) public {
615 |     assign1(a);
616 | }
617 |
618 |     function assign3(A memory x) public returns (uint) {
619 |         return g[1] + x.a + uint(j);
620 |     }
621 |
622 |     function ret() public returns (int){
623 |         return this.e();
624 |     }
625 | }
    |
    | }
    | }
    | DASP : Unknown unknowns
    | Found: false
    | Reported by checker:
626 | []

```

#### 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:**

```

627 | /*The following loop is infinite:*/
628 |
629 | do {
630 |     continue;
631 | } while(false);
    |
    | }
    | }
    | DASP : Unknown Unknowns
    | Found: false
    | Reported by checker:
632 | []

```

#### 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:**

```

633 | pragma solidity 0.4.24;
634 |
635 | contract Bug {
636 |     uint now; // Overshadows current time stamp.
637 |
638 |     function assert(bool condition) public {
639 |         // Overshadows built{ in symbol for providing
640 |         ↪ assertions.
641 |     }
642 |
643 |     function get_next_expiration(uint earlier_time) private
644 |     ↪ returns (uint) {
645 |         return now + 259200; // References overshadowed
646 |         ↪ timestamp.
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 | }

```

## 1.48 ignore

**SWC\_ID:** []

**Description:**Other trivial bug types.

**Example:**

```

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 |

```

```

[{"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":"        35 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:**

```
649 contract Uninitialized{
650     address owner = msg.sender;
651
652     struct St{
653         uint a;
654     }
655
656     function func() {
657         St st;
658         st.a = 0x0;
659     }
660 }
661 /*Bob calls func. As a result, owner is overridden to 0.*/
}
}
DASP : Unknown Unknowns
Found: false
Reported by checker:
662 []
```

## 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:**

```
663 Here is the example of correct pure{ function:
664
665 pragma solidity 0.4.16;
666
667 contract C {
668     function f(uint a, uint b) pure returns (uint) {
669         return a * (b + 42) + now;
670     }
671 }
}
}
```

**DASP** : Unknown unknowns

**Found**: false

**Reported by checker**:

672 | []

## 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**:

```
673 contract C {
674     function f(uint z) public returns (uint) {
675         uint y = x + 9 + z; // 'z' is used pre{ declaration
676         uint x = 7;
677
678         if (z % 2 == 0) {
679             uint max = 5;
680             // ...
681         }
682
683         // 'max' was intended to be 5, but it was mistakenly
684         ↪ declared in a scope and not assigned (so it is
685         ↪ zero).
686         for (uint i = 0; i < max; i++) {
687             x += 1;
688         }
689         return x;
690     }
691 }
```

**DASP** : Unknown unknowns

**Found**: false

**Reported by checker**:

691 | []

## 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**:

```

692 | contract A {
693 |     uint[2][3] bad_arr = [[1, 2], [3, 4], [5, 6]];
694 |
695 |     /* Array of arrays passed to abi.encode is vulnerable */
696 |     function bad() public {
697 |         bytes memory b = abi.encode(bad_arr);
698 |     }
699 | }
700 |
701 | /*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:
702 | []

```

### 1.53 costly\_ops\_in\_loop

**SWC\_ID:** []

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

**Example:**

```

703 | contract CostlyOperationsInLoop{
704 |
705 |     uint loop_count = 100;
706 |     uint state_variable=0;
707 |
708 |     function bad() external{
709 |         for (uint i=0; i < loop_count; i++){
710 |             state_variable++;
711 |         }
712 |     }
713 |
714 |     function good() external{
715 |         uint local_variable = state_variable;
716 |         for (uint i=0; i < loop_count; i++){
717 |             local_variable++;
718 |         }
719 |         state_variable = local_variable;
720 |     }

```

```

721 | }
722 | /*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:
723 | []

```

### 1.54 msg.value\_equals\_zero

**SWC\_ID:** []

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

**Example:**

```

724 | msg.value == 0
    |
    | }
    | }
    | DASP : Unknown unknowns
    | Found: false
    | Reported by checker:
725 | []

```

### 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:**

```

726 | pragma solidity 0.4.25;
727 |
728 | contract Crowdsale {
729 |
730 |     address public owner;
731 |
732 |     uint rate;
733 |     uint cap;
734 |

```

```

735     constructor() {
736         owner = msg.sender;
737     }
738
739     function setRate(_ rate) public onlyOwner {
740         rate = _ rate;
741     }
742
743     function setCap(_ cap) public {
744         require (msg.sender == owner);
745         cap = _ cap;
746     }
747 }

```

```

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

```

```

748 | []

```

## 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:**

```

749 contract A {
750     int[3] ether_balances; // storage signed integer array
751     function bad0() private {
752         // ...
753         ether_balances = [{ 1, { 1, { 1 } }];
754         // ...
755     }
756 }
757
758 /*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:**

759 | []

## 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:**

```
760 | function check(uint a) external returns(bool){
761 |     return (a >= a);
762 | }
    |
    |
    | }
```

**DASP :** Unknown unknowns

**Found:** false

**Reported by checker:**

763 | []

## 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:**

```
764 | /* In the following example, limiter variable is accessed on
    | ↪ every for{ loop iteration: */
765 |
766 | pragma solidity 0.4.25;
767 |
768 | contract NewContract {
769 |     uint limiter = 100;
770 |
771 |     function longLoop() {
772 |         for(uint i = 0; i < limiter; i++) {
773 |             /* ... */
774 |         }
775 |     }
776 | }
```

```

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

```

777 | []

## 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**:

```

778 contract DelegatecallInLoop{
779
780     mapping (address => uint256) balances;
781
782     function bad(address[] memory receivers) public payable {
783         for (uint256 i = 0; i < receivers.length; i++) {
784
785             ↪ address(this).delegatecall(abi.encodeWithSignature("addBalance(address)",
786             ↪ receivers[i]));
787         }
788     }
789
790     function addBalance(address a) public payable {
791         balances[a] += msg.value;
792     }
793 }

```

```

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

```

792 | []

## 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**:

```

793  /*
794  * @source: https://youtu.be/P_ Mtd5Fc_ 3E
795  * @author: Shahar Zini
796  */
797  pragma solidity 0.5.0;
798
799  contract GuessTheNumber
800  {
801      uint _ secretNumber;
802      address payable _ owner;
803      event success(string);
804      event wrongNumber(string);
805
806      function guess(uint n) payable public
807      {
808          require(msg.value == 1 ether);
809
810          uint p = address(this).balance;
811          checkAndTransferPrize(/*The prize/*rebmun desseug*/n ,
            ↪ p/*
812                               /*The user who should benefit */ ,msg.sender);
813      }
814
815      function checkAndTransferPrize(uint p, uint n, address
            ↪ payable guesser) internal returns(bool)
816      {
817          if(n == _ secretNumber)
818          {
819              guesser.transfer(p);
820              emit success("You guessed the correct number!");
821          }
822          else
823          {
824              emit wrongNumber("You've made an incorrect guess!");
825          }
826      }
827  }
828  }

```

**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:**

```
829 contract A {
830     uint s_a;
831
832     function bad() public {
833         assert((s_a += 1) > 10);
834     }
835 }
836 /*The assert in bad() increments the state variable s_a while
    ↳ checking for the condition.*/
}
}
DASP : Unknown Unknowns
Found: false
Reported by checker:
837 []
```

## 1.62 pausable\_modifier\_absence

**SWC\_ID:** []

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

**Example:**

```
838 function buggyTransfer(address to, uint256 value) external
    ↳ returns (bool){
839     balanceOf[msg.sender] { = value;
840     balanceOf[to] += value;
841     return true;
842 }
843
844 /*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 Tx's transferring even in a
    ↳ paused state, which corrupts the contract balance state and
    ↳ affects recovery.*/
}
}
```

**DASP** : Unknown unknowns

**Found**: false

**Reported by checker**:

845 | []

### 1.63 call\_without\_data

**SWC\_ID**: []

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

**Example**:

```
846 | /*In the following example, call function is used for ETH
    | ↪ transfer:*/
847 | pragma solidity 0.4.24;
848 |
849 | contract MyContract {
850 |
851 |     function withdraw() {
852 |         if (msg.sender.call.value(1)()) {
853 |             /*...*/
854 |         }
855 |     }
856 | }
    |
    | }
```

**DASP** : Unknown unknowns

**Found**: false

**Reported by checker**:

857 | []

### 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**:

```
858 | contract TimedCrowdsale
859 |     event Finished();
860 |     event notFinished();
861 |
```

```

862 | // Sale should finish exactly at January 1, 2019
863 | function isSaleFinished() private returns (bool) {
864 |     return block.timestamp >= 1546300800;
865 | }
866 |
867 | function run() public {
868 |     if (isSaleFinished()) {
869 |         emit Finished();
870 |     } else {
871 |         emit notFinished();
872 |     }
873 | }
874 | }
    | }
    | }
    | DASP : Time Manipulation
    | Found: false
    | Reported by checker:
875 | []

```

## 1.65 uninitialized\_local\_variable

**SWC\_ID:** []

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

**Example:**

```

876 | contract Uninitialized is Owner{
877 |     function withdraw() payable public onlyOwner{
878 |         address to;
879 |         to.transfer(this.balance)
880 |     }
881 | }
882 |
883 | /*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:
884 | []

```

## 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:**

```
885 | if (address(this).balance == 42 ether ) {
886 |     /* ... */
887 | }
888 | secure alternative:
889 |
890 | if (address(this).balance >= 42 ether ) {
891 |     /* ... */
892 | }
    | }
    | }
```

**DASP :** Unknown unknowns

**Found:** false

**Reported by checker:**

```
893 | []
```

## 1.67 byte\_array\_instead\_bytes

**SWC ID:** []

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

**Example:**

```
894 | /*In the following example, byte array is used:*/
895 |
896 | pragma solidity 0.4.24;
897 |
898 | contract C {
899 |     byte[] someVariable;
900 |     ...
901 | }
902 |
903 | Alternative:
904 |
905 | pragma solidity 0.4.24;
906 |
907 | contract C {
908 |     bytes someVariable;
```

```

g09 | ...
g10 | }
    }
    }
    DASP : Unknown Unknowns
    Found: false
    Reported by checker:
g11 | []

```