

Metasecurelabs analysis report

metasecurelabs.io

November 14, 2022

1 Introduction

1.1 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:

```
1 contract A {
2     int[3] ether\textunderscore balances; // storage signed
3     ↪ integer array
4     function bad0() private {
5         // ...
6         ether\textunderscore balances = [\textendash 1, \textendash
7         ↪ 1, \textendash 1];
8         // ...
9     }
10 }
11
12 /*bad0() uses a (storage\textendash allocated) signed integer
13 ↪ array state variable to store the ether balances of three
14 ↪ accounts. 1 is supposed to indicate uninitialized values
15 ↪ but the Solidity bug makes these as 1, which could be
16 ↪ exploited by the accounts.*/
17
18 }
19
20 }
21
22 DASP : Unknown unknowns
23 Found: false
```

1.2 modifier_like_Sol_keyword

SWC_ID:

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

Example:

```
11 contract Contract{
12     modifier public() {
13     }
14
15     function doSomething() public {
16         require(owner == msg.sender);
17         owner = newOwner;
18     }
19 }
20
21 /*public is a modifier meant to look like a Solidity keyword.*/
}
}
DASP : Unknown Unknowns
Found: false
```

1.3 arbitrary_from_in_transferFrom

SWC_ID:

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

Example:

```
22 function a(address from, address to, uint256 amount) public {
23     ERC20.transferFrom(from, to, am);
24 }
25
26 /*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
```

1.4 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:

```

27 Integer overflow/underflow
28 /*
29  * @source:
    ↪ https://capturetheether.com/challenges/math/token\textendash
    ↪ sale/
30  * @author: Steve Marx
31  */
32 pragma solidity \textsuperscript 0.4.21;
33 contract TokenSaleChallenge {
34     mapping(address => uint256) public balanceOf;
35     uint256 constant PRICE\textunderscore PER\textunderscore
    ↪ TOKEN = 1 ether;
36
37     function TokenSaleChallenge(address \textunderscore player)
    ↪ public payable {
38         require(msg.value == 1 ether);
39     }
40
41     function isComplete() public view returns (bool) {
42         return address(this).balance < 1 ether;
43     }
44
45     function buy(uint256 numTokens) public payable {
46         require(msg.value == numTokens * PRICE\textunderscore
    ↪ PER\textunderscore TOKEN);
47
48         balanceOf[msg.sender] += numTokens;
49     }
50 }
51
52 /*Division issues*/
53 contract Division {
54

```

```

55     /*function unsigned\textunderscore division(uint32 x, uint32
    ↪ y) returns (int r) {
56         //if (y == 0) { throw; }
57         r = x / y;
58     }*/
59
60     function signed\textunderscore division(int x, int y)
    ↪ returns (int) {
61         //if ((y == 0) ((x == \textendash 2**255) \& \& (y ==
    ↪ \textendash 1))) { throw; }
62         return x / y;
63     }
64
65 }
66
67 /*Type deduction overflow*/
68 contract For\textunderscore Test {
69     ...
70     function Test () payable public {
71         if ( msg . value > 0.1 ether ) {
72             uint256 multi = 0;
73             uint256 amountToTransfer = 0;
74             for ( var i = 0; i < 2* msg . value ; i ++ ) {
75                 multi = i *2;
76                 if ( multi < amountToTransfer ) {
77                     break ;
78                     amountToTransfer = multi ;
79                 }
80                 msg.sender.transfer( amountToTransfer );
81             }
82         }
83     }
84
85 }
86
87 }
88
89 }
90
91 }
92
93 }
94
95 }
96
97 }
98
99 }
100
101 }
102
103 }
104
105 }
106
107 }
108
109 }
110
111 }
112
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 }
469
470 }
471
472 }
473
474 }
475
476 }
477
478 }
479
480 }
480
481 }
481
482 }
482
483 }
483
484 }
484
485 }
485
486 }
486
487 }
487
488 }
488
489 }
489
490 }
490
491 }
491
492 }
492
493 }
493
494 }
494
495 }
495
496 }
496
497 }
497
498 }
498
499 }
499
500 }
500
501 }
501
502 }
502
503 }
503
504 }
504
505 }
505
506 }
506
507 }
507
508 }
508
509 }
509
510 }
510
511 }
511
512 }
512
513 }
513
514 }
514
515 }
515
516 }
516
517 }
517
518 }
518
519 }
519
520 }
520
521 }
521
522 }
522
523 }
523
524 }
524
525 }
525
526 }
526
527 }
527
528 }
528
529 }
529
530 }
530
531 }
531
532 }
532
533 }
533
534 }
534
535 }
535
536 }
536
537 }
537
538 }
538
539 }
539
540 }
540
541 }
541
542 }
542
543 }
543
544 }
544
545 }
545
546 }
546
547 }
547
548 }
548
549 }
549
550 }
550
551 }
551
552 }
552
553 }
553
554 }
554
555 }
555
556 }
556
557 }
557
558 }
558
559 }
559
560 }
560
561 }
561
562 }
562
563 }
563
564 }
564
565 }
565
566 }
566
567 }
567
568 }
568
569 }
569
570 }
570
571 }
571
572 }
572
573 }
573
574 }
574
575 }
575
576 }
576
577 }
577
578 }
578
579 }
579
580 }
580
581 }
581
582 }
582
583 }
583
584 }
584
585 }
585
586 }
586
587 }
587
588 }
588
589 }
589
590 }
590
591 }
591
592 }
592
593 }
593
594 }
594
595 }
595
596 }
596
597 }
597
598 }
598
599 }
599
600 }
600
601 }
601
602 }
602
603 }
603
604 }
604
605 }
605
606 }
606
607 }
607
608 }
608
609 }
609
610 }
610
611 }
611
612 }
612
613 }
613
614 }
614
615 }
615
616 }
616
617 }
617
618 }
618
619 }
619
620 }
620
621 }
621
622 }
622
623 }
623
624 }
624
625 }
625
626 }
626
627 }
627
628 }
628
629 }
629
630 }
630
631 }
631
632 }
632
633 }
633
634 }
634
635 }
635
636 }
636
637 }
637
638 }
638
639 }
639
640 }
640
641 }
641
642 }
642
643 }
643
644 }
644
645 }
645
646 }
646
647 }
647
648 }
648
649 }
649
650 }
650
651 }
651
652 }
652
653 }
653
654 }
654
655 }
655
656 }
656
657 }
657
658 }
658
659 }
659
660 }
660
661 }
661
662 }
662
663 }
663
664 }
664
665 }
665
666 }
666
667 }
667
668 }
668
669 }
669
670 }
670
671 }
671
672 }
672
673 }
673
674 }
674
675 }
675
676 }
676
677 }
677
678 }
678
679 }
679
680 }
680
681 }
681
682 }
682
683 }
683
684 }
684
685 }
685
686 }
686
687 }
687
688 }
688
689 }
689
690 }
690
691 }
691
692 }
692
693 }
693
694 }
694
695 }
695
696 }
696
697 }
697
698 }
698
699 }
699
700 }
700
701 }
701
702 }
702
703 }
703
704 }
704
705 }
705
706 }
706
707 }
707
708 }
708
709 }
709
710 }
710
711 }
711
712 }
712
713 }
713
714 }
714
715 }
715
716 }
716
717 }
717
718 }
718
719 }
719
720 }
720
721 }
721
722 }
722
723 }
723
724 }
724
725 }
725
726 }
726
727 }
727
728 }
728
729 }
729
730 }
730
731 }
731
732 }
732
733 }
733
734 }
734
735 }
735
736 }
736
737 }
737
738 }
738
739 }
739
740 }
740
741 }
741
742 }
742
743 }
743
744 }
744
745 }
745
746 }
746
747 }
747
748 }
748
749 }
749
750 }
750
751 }
751
752 }
752
753 }
753
754 }
754
755 }
755
756 }
756
757 }
757
758 }
758
759 }
759
760 }
760
761 }
761
762 }
762
763 }
763
764 }
764
765 }
765
766 }
766
767 }
767
768 }
768
769 }
769
770 }
770
771 }
771
772 }
772
773 }
773
774 }
774
775 }
775
776 }
776
777 }
777
778 }
778
779 }
779
780 }
780
781 }
781
782 }
782
783 }
783
784 }
784
785 }
785
786 }
786
787 }
787
788 }
788
789 }
789
790 }
790
791 }
791
792 }
792
793 }
793
794 }
794
795 }
795
796 }
796
797 }
797
798 }
798
799 }
799
800 }
800
801 }
801
802 }
802
803 }
803
804 }
804
805 }
805
806 }
806
807 }
807
808 }
808
809 }
809
810 }
810
811 }
811
812 }
812
813 }
813
814 }
814
815 }
815
816 }
816
817 }
817
818 }
818
819 }
819
820 }
820
821 }
821
822 }
822
823 }
823
824 }
824
825 }
825
826 }
826
827 }
827
828 }
828
829 }
829
830 }
830
831 }
831
832 }
832
833 }
833
834 }
834
835 }
835
836 }
836
837 }
837
838 }
838
839 }
839
840 }
840
841 }
841
842 }
842
843 }
843
844 }
844
845 }
845
846 }
846
847 }
847
848 }
848
849 }
849
850 }
850
851 }
851
852 }
852
853 }
853
854 }
854
855 }
855
856 }
856
857 }
857
858 }
858
859 }
859
860 }
860
861 }
861
862 }
862
863 }
863
864 }
864
865 }
865
866 }
866
867 }
867
868 }
868
869 }
869
870 }
870
871 }
871
872 }
872
873 }
873
874 }
874
875 }
875
876 }
876
877 }
877
878 }
878
879 }
879
880 }
880
881 }
881
882 }
882
883 }
883
884 }
884
885 }
885
886 }
886
887 }
887
888 }
888
889 }
889
890 }
890
891 }
891
892 }
892
893 }
893
894 }
894
895 }
895
896 }
896
897 }
897
898 }
898
899 }
899
900 }
900
901 }
901
902 }
902
903 }
903
904 }
904
905 }
905
906 }
906
907 }
907
908 }
908
909 }
909
910 }
910
911 }
911
912 }
912
913 }
913
914 }
914
915 }
915
916 }
916
917 }
917
918 }
918
919 }
919
920 }
920
921 }
921
922 }
922
923 }
923
924 }
924
925 }
925
926 }
926
927 }
927
928 }
928
929 }
929
930 }
930
931 }
931
932 }
932
933 }
933
934 }
934
935 }
935
936 }
936
937 }
937
938 }
938
939 }
939
940 }
940
941 }
941
942 }
942
943 }
943
944 }
944
945 }
945
946 }
946
947 }
947
948 }
948
949 }
949
950 }
950
951 }
951
952 }
952
953 }
953
954 }
954
955 }
955
956 }
956
957 }
957
958 }
958
959 }
959
960 }
960
961 }
961
962 }
962
963 }
963
964 }
964
965 }
965
966 }
966
967 }
967
968 }
968
969 }
969
970 }
970
971 }
971
972 }
972
973 }
973
974 }
974
975 }
975
976 }
976
977 }
977
978 }
978
979 }
979
980 }
980
981 }
981
982 }
982
983 }
983
984 }
984
985 }
985
986 }
986
987 }
987
988 }
988
989 }
989
990 }
990
991 }
991
992 }
992
993 }
993
994 }
994
995 }
995
996 }
996
997 }
997
998 }
998
999 }
999
1000 }
1000
1001 }
1001
1002 }
1002
1003 }
1003
1004 }
1004
1005 }
1005
1006 }
1006
1007 }
1007
1008 }
1008
1009 }
1009
1010 }
1010
1011 }
1011
1012 }
1012
1013 }
1013
1014 }
1014
1015 }
1015
1016 }
1016
1017 }
1017
1018 }
1018
1019 }
1019
1020 }
1020
1021 }
1021
1022 }
1022
1023 }
1023
1024 }
1024
1025 }
1025
1026 }
1026
1027 }
1027
1028 }
1028
1029 }
1029
1030 }
1030
1031 }
1031
1032 }
1032
1033 }
1033
1034 }
1034
1035 }
1035
1036 }
1036
1037 }
1037
1038 }
1038
1039 }
1039
1040 }
1040
1041 }
1041
1042 }
1042
1043 }
1043
1044 }
1044
1045 }
1045
1046 }
1046
1047 }
1047
1048 }
1048
1049 }
1049
1050 }
1050
1051 }
1051
1052 }
1052
1053 }
1053
1054 }
1054
1055 }
1055
1056 }
1056
1057 }
1057
1058 }
1058
1059 }
1059
1060 }
1060
1061 }
1061
1062 }
1062
1063 }
1063
1064 }
1064
1065 }
1065
1066 }
1066
1067 }
1067
1068 }
1068
1069 }
1069
1070 }
1070
1071 }
1071
1072 }
1072
1073 }
1073
1074 }
1074
1075 }
1075
1076 }
1076
1077 }
1077
1078 }
1078
1079 }
1079
1080 }
1080
1081 }
1081
1082 }
1082
1083 }
1083
1084 }
1084
1085 }
1085
1086 }
1086
1087 }
1087
1088 }
1088
1089 }
1089
1090 }
1090
1091 }
1091
1092 }
1092
1093 }
1093
1094 }
1094
1095 }
1095
1096 }
1096
1097 }
1097
1098 }
1098
1099 }
1099
1100 }
1100
1101 }
1101
1102 }
1102
1103 }
1103
1104 }
1104
1105 }
1105
1106 }
1106
1107 }
1107
1108 }
1108
1109 }
1109
1110 }
1110
1111 }
1111
1112 }
1112
1113 }
1113
1114 }
1114
1115 }
1115
1116 }
1116
1117 }
1117
1118 }
1118
1119 }
1119
1120 }
1120
1121 }
1121
1122 }
1122
1123 }
1123
1124 }
1124
1125 }
1125
1126 }
1126
1127 }
1127
1128 }
1128
1129 }
1129
1130 }
1130
1131 }
1131
1132 }
1132
1133 }
1133
1134 }
1134
1135 }
1135
1136 }
1136
1137 }
1137
1138 }
1138
1139 }
1139
1140 }
1140
1141 }
1141
1142 }
1142
1143 }
1143
1144 }
1144
1145 }
1145
1146 }
1146
1147 }
1147
1148 }
1148
1149 }
1149
1150 }
1150
1151 }
1151
1152 }
1152
1153 }
1153
1154 }
1154
1155 }
1155
1156 }
1156
1157 }
1157
1158 }
1158
1159 }
1159
1160 }
1160
1161 }
1161
1162 }
1162
1163 }
1163
1164 }
1164
1165 }
1165
1166 }
1166
1167 }
1167
1168 }
1168
1169 }
1169
1170 }
1170
1171 }
1171
1172 }
1172
1173 }
1173
1174 }
1174
1175 }
1175
1176 }
1176
1177 }
1177
1178 }
1178
1179 }
1179
1180 }
1180
1181 }
1181
1182 }
1182
1183 }
1183
1184 }
1184
1185 }
1185
1186 }
1186
1187 }
1187
1188 }
1188
1189 }
1189
1190 }
1190
1191 }
1191
1192 }
1192
1193 }
1193
1194 }
1194
1195 }
1195
1196 }
1196
1197 }
1197
1198 }
1198
1199 }
1199
1200 }
1200
1201 }
1201
1202 }
1202
1203 }
1203
1204 }
1204
1205 }
1205
1206 }
1206
1207 }
1207
1208 }
1208
1209 }
1209
1210 }
1210
1211 }
1211
1212 }
1212
1213 }
1213
1214 }
1214
1215 }
1215
1216 }
1216
1217 }
1217
1218 }
1218
1219 }
1219
1220 }
1220
1221 }
1221
1222 }
1222
1223 }
1223
1224 }
1224
1225 }
1225
1226 }
1226
1227 }
1227
1228 }
1228
1229 }
1229
1230 }
1230
1231 }
1231
1232 }
1232
1233 }
1233
1234 }
1234
1235 }
1235
1236 }
1236
1237 }
1237
1238 }
1238
1239 }
1239
1240 }
1240
1241 }
1241
1242 }
1242
1243 }
1243
1244 }
1244
1245 }
1245
1246 }
1246
1247 }
1247
1248 }
1248
1249 }
1249
1250 }
1250
1251 }
1251
1252 }
1252
1253 }
1253
1254 }
1254
1255 }
1255
1256 }
1256
1257 }
1257
1258 }
1258
1259 }
1259
1260 }
1260
1261 }
1261
1262 }
1262
1263 }
1263
1264 }
1264
1265 }
1265
1266 }
1266
1267 }
1267
1268 }
1268
1269 }
1269
1270 }
1270
1271 }
1271
1272 }
1272
1273 }
1273
1274 }
1274
1275 }
1275
1276 }
1276
1277 }
1277
1278 }
1278
1279 }
1279
1280 }
1280
1281 }
1281
1282 }
1282
1283 }
1283
1284 }
1284
1285 }
1285
1286 }
1286
1287 }
1287
1288 }
1288
1289 }
1289
1290 }
1290
1291 }
1291
1292 }
1292
1293 }
1293
1294 }
1294
1295 }
1295
1296 }
1296
1297 }
1297
1298 }
1298
1299 }
1299
1300 }
1300
1301 }
1301
1302 }
1302
1303 }
1303
1304 }
1304
1305 }
1305
1306 }
1306
1307 }
1307
1308 }
1308
1309 }
1309
1310 }
1310
1311 }
1311
1312 }
1312
1313 }
1313
1314 }
1314
1315 }
1315
1316 }
1316
1317 }
1317
1318 }
1318
1319 }
1319
1320 }
1320
1321 }
1321
1322 }
1322
1323 }
1323
1324 }
1324
1325 }
1325
1326 }
1326
1327 }
1327
1328 }
1328
1329 }
1329
1330 }
1330
1331 }
1331
1332 }
1332
1333 }
1333
1334 }
1334
1335 }
1335
1336 }
1336
1337 }
1337
1338 }
1338
1339 }
1339
1340 }
1340
1341 }
1341
1342 }
1342
1343 }
1343
1344 }
1344
1345 }
1345
1346 }
1346
1347 }
1347
1348 }
1348
1349 }
1349
1350 }
1350
1351 }
1351
1352 }
1352
1353 }
1353
1354 }
1354
1355 }
1355
1356 }
1356
1357 }
1357
1358 }
1358
1359 }
1359
1360 }
1360
1361 }
1361
1362 }
1362
1363 }
1363
1364 }
1364
1365 }
1365
1366 }
1366
1367 }
1367
1368 }
1368
1369 }
1369
1370 }
1370
1371 }
1371
1372 }
1372
1373 }
1373
1374 }
1374
1375 }
1375
1376 }
1376
1377 }
1377
1378 }
1378
1379 }
1379
1380 }
1380
1381 }
1381
1382 }
1382
1383 }
1383
1384 }
1384
1385 }
1385
1386 }
1386
1387 }
1387
1388 }
1388
1389 }
1389
1390 }
1390
1391 }
1391
1392 }
1392
1393 }
1393
1394 }
1394
1395 }
1395
1396 }
1396
1397 }
1397
1398 }
1398
1399 }
1399
1400 }
1400
1401 }
1401
1402 }
1402
1403 }
1403
1404 }
1404
1405 }
1405
1406 }
1406
1407 }
1407
1408 }
1408
1409 }
1409
1410 }
1410
1411 }
1411
1412 }
1412
1413 }
1413
1414 }
1414
1415 }
1415
1416 }
1416
1417 }
1417
1418 }
1418
1419 }
1419
1420 }
1420
1421 }
1421
1422 }
1422
1423 }
1423
1424 }
1424
1425 }
1425
1426 }
1426
1427 }
1427
1428 }
1428
1429 }
1429
1430 }
1430
1431 }
1431
1432 }
1432
1433 }
1433
1434 }
1434
1435 }
1435
1436 }
1436
1437 }
1437
1438 }
1438
1439 }
1439
1440 }
1440
1441 }
1441
1442 }
1442
1443 }
1443
1444 }
1444
1445 }
1445
1446 }
1446
1447 }
1447
1448 }
1448
1449 }
1449
1450 }
1450
1451 }
1451
1452 }
1452
1453 }
1453
1454 }
1454
1455 }
1455
1456 }
1456
1457 }
1457
1458 }
1458
1459 }
1459
1460 }
1460
1461 }
1461
1462 }
1462
1463 }
1463
1464 }
1464
1465 }
1465
1466 }
1466
1467 }
1467
1468 }
1468
1469 }
1469
1470 }
1470
1471 }
1471
1472 }
1472
1473 }
1473
1474 }
1474
1475 }
1475
1476 }
1476
1477 }
1477
1478 }
1478
1479 }
1479
1480 }
1480
1481 }
1481
1482 }
1482
1483 }
1483
1484 }
1484
1485 }
1485
1486 }
1486
1487 }
1487
1488 }
1488
1489 }
1489
1490 }
1490
1491 }
1491
1492 }
1492
1493 }
1493
1494 }
1494
1495 }
1495
1496 }
1496
1497 }
1497
1498 }
1498
1499 }
1499
1500 }
1500
1501 }
1501
1502 }
1502
1503 }
1503
1504 }
1504
15
```

```

85     uint[2][3] bad\textunderscore arr = [[1, 2], [3, 4], [5,
      ↪ 6]];
86
87     /* Array of arrays passed to abi.encode is vulnerable */
88     function bad() public {
89         bytes memory b = abi.encode(bad\textunderscore arr);
90     }
91 }
92
93 /*abi.encode(bad\textunderscore 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

```

1.6 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:

```

94 pragma solidity \textsuperscript 0.5.0;
95
96 contract DepositBox {
97     mapping(address => uint) balance;
98
99     // Accept deposit
100     function deposit(uint amount) public payable {
101         require(msg.value == amount, 'incorrect amount');
102         // Should update user balance
103         balance[msg.sender] = amount;
104     }
105 }
}
}
DASP : Unknown unknowns
Found: false

```

1.7 func_modifying_storage_array_by_value

SWC_ID:

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

Example:

```
106 contract Memory {
107     uint[1] public x; // storage
108
109     function f() public {
110         f1(x); // update x
111         f2(x); // do not update x
112     }
113
114     function f1(uint[1] storage arr) internal { // by reference
115         arr[0] = 1;
116     }
117
118     function f2(uint[1] arr) internal { // by value
119         arr[0] = 2;
120     }
121 }
122
123 /*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

1.8 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:

```
124 if (address(this).balance == 42 ether ) {
125     /* ... */
126 }
127 secure alternative:
```

```

128 |
129 | if (address(this).balance >= 42 ether ) {
130 |     /* ... */
131 | }
    |
    | }
    | }
    | DASP : Unknown unknowns
    | Found: false

```

1.9 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:

```

132 | pragma solidity 0.4.25;
133 |
134 | contract Crowdsale {
135 |
136 |     address public owner;
137 |
138 |     uint rate;
139 |     uint cap;
140 |
141 |     constructor() {
142 |         owner = msg.sender;
143 |     }
144 |
145 |     function setRate(\textunderscore rate) public onlyOwner {
146 |         rate = \textunderscore rate;
147 |     }
148 |
149 |     function setCap(\textunderscore cap) public {
150 |         require (msg.sender == owner);
151 |         cap = \textunderscore cap;
152 |     }
153 | }
    |
    | }
    | }
    | DASP : Unknown unknowns
    | Found: false

```

1.10 `erc20_event_not_indexed`

SWC ID:

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

Example:

```
154 contract ERC20Bad {
155     // ...
156     event Transfer(address from, address to, uint value);
157     event Approval(address owner, address spender, uint value);
158
159     // ...
160 }
161
162 /*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

1.11 `unused_retval`

SWC ID:

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

Example:

```
163 contract MyConc{
164     using SafeMath for uint;
165     function my\textunderscore func(uint a, uint b) public{
166         a.add(b);
167     }
168 }
169
170 /*MyConc calls add of SafeMath, but does not store the result
   ↳ in a. As a result, the computation has no effect. */
}
```

DASP : Unknown Unknowns
Found: true

1.12 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:

```
171  /* In the following example, limiter variable is accessed on
172  ↪ every for\textendash loop iteration: */
173
174  pragma solidity 0.4.25;
175
176  contract NewContract {
177      uint limiter = 100;
178
179      function longLoop() {
180          for(uint i = 0; i < limiter; i++) {
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 : Unknown unknowns
Found: false

1.13 uninitialized_state_variable

SWC_ID:

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

Example:

```
184  contract Uninitialized{
185      address destination;
186
187      function transfer() payable public{
188          destination.transfer(msg.value);
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 : Unknown unknowns
Found: true

1.14 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:

```
191 contract C {
192     function f(uint z) public returns (uint) {
193         uint y = x + 9 + z; // 'z' is used pre\textendash
           ↳ declaration
194         uint x = 7;
195
196         if (z % 2 == 0) {
197             uint max = 5;
198             // ...
199         }
200
201         // 'max' was intended to be 5, but it was mistakenly
           ↳ declared in a scope and not assigned (so it is
           ↳ zero).
202         for (uint i = 0; i < max; i++) {
203             x += 1;
204         }
205
206         return x;
207     }
208 }
}
}
DASP : Unknown unknowns
Found: false
```

1.15 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:

```

209  /* In this example, one can front\textendash run transactions to
    ↪ claim his/her reward before the owner reduces the reward
    ↪ amount.*/
210
211  pragma solidity \textsuperscript 0.4.16;
212
213  contract EthTxOrderDependenceMinimal {
214      address public owner;
215      bool public claimed;
216      uint public reward;
217
218      function EthTxOrderDependenceMinimal() public {
219          owner = msg.sender;
220      }
221
222      function setReward() public payable {
223          require (!claimed);
224          require(msg.sender == owner);
225          owner.transfer(reward);
226          reward = msg.value;
227      }
228
229      function claimReward(uint256 submission) {
230          require (!claimed);
231          require(submission < 10);
232          msg.sender.transfer(reward);
233          claimed = true;
234      }
235  }
    }
    }
    DASP : Front Running
    Found: true

```

1.16 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:

```

236  pragma solidity 0.4.25;
237
238  contract ReturnValue {

```

```

239 |
240 | checked
241 | function callchecked(address callee) public {
242 |     require(callee.call());
243 | }
244 |
245 | function callnotchecked(address callee) public {
246 |     callee.call();
247 | }
248 | }
    | }
    | }
    | DASP : Unchecked Low Level Calls
    | Found: true

```

1.17 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:

```

249 | /* In the following example, contracts programmed to receive
    | ↳ ether does not call transfer, send, or call.value function:
    | ↳ */
250 |
251 | pragma solidity 0.4.25;
252 |
253 | contract BadMarketPlace {
254 |     function deposit() payable {
255 |         require(msg.value > 0);
256 |     }
257 | }
    | }
    | }
    | DASP : Unknown unknowns
    | Found: false

```

1.18 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:

```

258 contract Token{
259     function transfer(address to, uint value) external;
260     //...
261 }
262
263 /*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

```

1.19 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:

```

264 /*In the following example, functions with both public and
  ↳ external visibility modifiers are used: */
265
266 contract Token {
267
268     mapping (address => uint256) internal \textunderscore
  ↳ balances;
269
270     function transfer\textunderscore public(address to, uint256
  ↳ value) public {
271         require(value <= \textunderscore balances[msg.sender]);
272
273         \textunderscore balances[msg.sender] \textendash =
  ↳ value;
274         \textunderscore balances[to] += value;
275     }
276
277     function transfer\textunderscore external(address to,
  ↳ uint256 value) external {
278         require(value <= \textunderscore balances[msg.sender]);
279
280         \textunderscore balances[msg.sender] \textendash =
  ↳ value;

```

```

281         \textunderscore balances[to] += value;
282     }
283 }
284
285 /*The second function requires less gas.*/
}
}
DASP : Unknown unknowns
Found: true

```

1.20 uninitialized_func_pointer

SWC_ID:

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

Example:

```

286 contract Bug{
287     function buy() public payable{
288         uint minted = msg.value * (1000 / address(this).balance);
289         // ...
290     }
291 }
292
293 /*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

```

1.21 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:

```

294 /*
295 * @source:
   ↳ http://blockchain.unica.it/projects/ethereum\textendash
   ↳ survey/attacks.htmlsimplifiedao

```

```

296 * @author: \textendash
297 * @vulnerable\textunderscore at\textunderscore lines: 19
298 */
299
300 pragma solidity \textsuperscript 0.4.2;
301
302 contract SimpleDAO {
303     mapping (address => uint) public credit;
304
305     function donate(address to) payable {
306         credit[to] += msg.value;
307     }
308
309     function withdraw(uint amount) {
310         if (credit[msg.sender]>= amount) {
311             // <yes> <report> REENTRANCY
312             bool res = msg.sender.call.value(amount)();
313             credit[msg.sender]\textendash =amount;
314         }
315     }
316 }
}
}
DASP : Reentrancy
Found: true

```

1.22 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:

```

317 /*In this example, a specific modifier, such as public, is not
    ↳ used when declaring a function: */
318
319 function foo();
320
321 Preferred alternatives:
322

```

```

323 | function foo() public;
324 | function foo() internal;
    | }
    | }
    | DASP : Unknown Unknowns
    | Found: true

```

1.23 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:

```

325 | pragma solidity 0.4.25;
326 |
327 | contract Tokensale {
328 |     uint public hardcap = 10000 ether;
329 |
330 |     function Tokensale() {}
331 |
332 |     function fetchCap() public constant returns(uint) {
333 |         return hardcap;
334 |     }
335 | }
336 |
337 | contract Presale is Tokensale {
338 |     //uint hardcap = 1000 ether;
339 |     //If the hardcap variables were both needed we would have to
340 |     ↪ rename one to fix this.
341 |     function Presale() Tokensale() {
342 |         hardcap = 1000 ether;
343 |     }
344 | }
    | }
    | }
    | DASP : Unknown Unknowns
    | Found: false

```

1.24 call_without_data

SWC_ID:

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

Example:

```
344  /*In the following example, call function is used for ETH
    ↪  transfer:*/
345  pragma solidity 0.4.24;
346
347  contract MyContract {
348
349      function withdraw() {
350          if (msg.sender.call.value(1)()) {
351              /*...*/
352          }
353      }
354  }
    }
    }
DASP : Unknown unknowns
Found: false
```

1.25 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:

```
355  modifier myModif(){
356      if(..){
357          \textunderscore ;
358      }
359  }
360  function get() myModif returns(uint){}
361
362  /*If the condition in myModif is false, the execution of get()
    ↪  will return 0.*/
    }
    }
DASP : Unknown unknowns
Found: false
```

1.26 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:

```
363 pragma solidity \textsuperscript 0.4.24;
364
365 contract Bug {
366     uint now; // Overshadows current time stamp.
367
368     function assert(bool condition) public {
369         // Overshadows built\textendash in symbol for providing
370         ↪ assertions.
371     }
372
373     function get\textunderscore next\textunderscore
374     ↪ expiration(uint earlier\textunderscore time) private
375     ↪ returns (uint) {
376         return now + 259200; // References overshadowed
377         ↪ timestamp.
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 }
```

1.27 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:

```
376 /*In the following contract, the address is specified in the
377 ↪ source code:*/
378
379 pragma solidity 0.4.24;
380 contract C {
381     function f(uint a, uint b) pure returns (address) {
382         address public multisig =
383         ↪ 0xf64B584972FE6055a770477670208d737Fff282f;
384         return multisig;
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 : Unknown unknowns
Found: false

```

1.28 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:

```

387  /*This code uses incorrect function signature:*/
388
389  pragma solidity \textsuperscript 0.5.1;
390  contract Signature {
391      function callFoo(address addr, uint value) public returns
392          ↪ (bool) {
393          bytes memory data = abi.encodeWithSignature("foo(uint)",
394          ↪ value);
395          (bool status, ) = addr.call(data);
396          return status;
397      }
398  }
399
400  Use "foo(uint256)" instead.
401
402  }
403  }
404  DASP : Unknown Unknowns
405  Found: false

```

1.29 msg.value_in_loop

SWC_ID:

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

Example:

```

399  contract MsgValueInLoop{
400      mapping (address => uint256) balances;
401
402      function bad(address[] memory receivers) public payable {
403          for (uint256 i=0; i < receivers.length; i++) {
404              balances[receivers[i]] += msg.value;

```

```

405     }
406   }
407 }
408
409 /*msg.value should be tracked through a local variable and
   ↪ decrease its amount on every iteration/usage.*/
}
}
DASP : Unknown unknowns
Found: false

```

1.30 right_to_left_char

SWC ID:

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:

```

410 /*
411  * @source: https://youtu.be/P\textunderscore
   ↪ Mtd5Fc\textunderscore 3E
412  * @author: Shahar Zini
413  */
414 pragma solidity \textsupscrip 0.5.0;
415
416 contract GuessTheNumber
417 {
418     uint \textunderscore secretNumber;
419     address payable \textunderscore owner;
420     event success(string);
421     event wrongNumber(string);
422
423     function guess(uint n) payable public
424     {
425         require(msg.value == 1 ether);
426
427         uint p = address(this).balance;
428         checkAndTransferPrize(/*The prize/*rebmun desseug*/n ,
   ↪ p/*
429                                 /*The user who should benefit */ ,msg.sender);
430     }
431
432     function checkAndTransferPrize(uint p, uint n, address
   ↪ payable guesser) internal returns(bool)
433     {

```

```

434         if(n == \textunderscore secretNumber)
435         {
436             guesser.transfer(p);
437             emit success("You guessed the correct number!");
438         }
439         else
440         {
441             emit wrongNumber("You've made an incorrect guess!");
442         }
443     }
444 }
}
}
DASP : Unknown Unknowns
Found: false

```

1.31 local_variable_shadowing

SWC_ID:

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

Example:

```

445 pragma solidity \textsuperscript 0.4.24;
446
447 contract Bug {
448     uint owner;
449
450     function sensitive\textunderscore function(address owner)
451     ↪ public {
452         // ...
453         require(owner == msg.sender);
454     }
455
456     function alternate\textunderscore sensitive\textunderscore
457     ↪ function() public {
458         address owner = msg.sender;
459         // ...
460         require(owner == msg.sender);
461     }
462 }

```

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

```

}
}
DASP : Unknown unknowns
Found: false

```

1.32 use_after_delete

SWC_ID:

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

Example:

```

463 mapping(address => uint) public balances;
464 function f() public {
465     delete balances[msg.sender];
466     msg.sender.transfer(balances[msg.sender]);
467 }
468
469 /*balances[msg.sender] is deleted before it's sent to the
   ↪ caller, leading the transfer to always send zero.*/
}
}
DASP : Unknown unknowns
Found: false

```

1.33 incorrect_shift_in_assembly

SWC_ID:

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

Example:

```

470 contract C {
471     function f() internal returns (uint a) {
472         assembly {
473             a := shr(a, 8)
474         }
475     }
}
}
DASP : Unknown Unknowns
Found: false

```

1.34 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:

```
476 pragma solidity 0.4.24;
477
478 contract BreakThisHash {
479     bytes32 hash;
480     uint birthday;
481     constructor(bytes32 \textunderscore hash) public payable {
482         hash = \textunderscore hash;
483         birthday = now;
484     }
485
486     function kill(bytes password) external {
487         if (sha3(password) != hash) {
488             throw;
489         }
490         suicide(msg.sender);
491     }
492
493     function hashAge() public constant returns(uint) {
494         return(now \textendash birthday);
495     }
496 }
497
498 /*Use keccak256, selfdestruct, revert() instead.*/
}
}
DASP : Unknown unknowns
Found: false
```

1.35 costly_ops_in_loop

SWC_ID:

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

Example:

```

499 contract CostlyOperationsInLoop{
500
501     uint loop\textunderscore count = 100;
502     uint state\textunderscore variable=0;
503
504     function bad() external{
505         for (uint i=0; i < loop\textunderscore count; i++){
506             state\textunderscore variable++;
507         }
508     }
509
510     function good() external{
511         uint local\textunderscore variable = state\textunderscore
        ↪ variable;
512         for (uint i=0; i < loop\textunderscore count; i++){
513             local\textunderscore variable++;
514         }
515         state\textunderscore variable = local\textunderscore
        ↪ variable;
516     }
517 }
518 /*Incrementing state\textunderscore variable in a loop incurs a
    ↪ lot of gas because of expensive SSTOREs, which might lead
    ↪ to an out\textendash of\textendash gas.*/
}
}
DASP : Unknown Unknowns
Found: false

```

1.36 function_declared_return_but_no_return

SWC_ID:

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

Example:

```

519 /*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:*/
520
521 pragma solidity 0.4.25;
522
523 contract NewContract {
524     uint minimumBuy;
525

```



```

526     function setMinimumBuy(uint256 newMinimumBuy) returns
      ↪ (bool){
527         minimumBuy = newMinimumBuy;
528     }
529 }

```

```

}
}
DASP : Unknown unknowns
Found: false

```

1.37 multiple_constructor_schemes

SWC ID:

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

Example:

```

530 contract A {
531     uint x;
532     constructor() public {
533         x = 0;
534     }
535     function A() public {
536         x = 1;
537     }
538
539     function test() public returns(uint) {
540         return x;
541     }
542 }
543
544 /*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

```

1.38 byte_array_instead_bytes

SWC ID:

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

Example:

```

545 | /*In the following example, byte array is used:*/
546 |
547 | pragma solidity 0.4.24;
548 |
549 | contract C {
550 |     byte[] someVariable;
551 |     ...
552 | }
553 |
554 | Alternative:
555 |
556 | pragma solidity 0.4.24;
557 |
558 | contract C {
559 |     bytes someVariable;
560 |     ...
561 | }
562 |
563 | }
564 | }
565 | DASP : Unknown Unknowns
566 | Found: false

```

1.39 short_addresses

SWC_ID:

Description:MISSING

Example:

```

562 | MISSING
563 |
564 | }
565 | }
566 | DASP : Unknown unknowns
567 | Found: false

```

1.40 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:

```

563 | contract Uninitialized{
564 |     address owner = msg.sender;
565 |
566 |     struct St{

```

```

567         uint a;
568     }
569
570     function func() {
571         St st;
572         st.a = 0x0;
573     }
574 }
575 /*Bob calls func. As a result, owner is overridden to 0.*/
}
}
DASP : Unknown Unknowns
Found: false

```

1.41 pausable_modifier_absence

SWC_ID:

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

Example:

```

576 function buggyTransfer(address to, uint256 value) external
    ↪ returns (bool){
577     balanceOf[msg.sender] \textendash = value;
578     balanceOf[to] += value;
579     return true;
580 }
581
582 /*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

```

1.42 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:

```
583 function check(uint a) external returns(bool){
584     return (a >= a);
585 }
}
}
DASP : Unknown unknowns
Found: false
```

1.43 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:

```
586 function callme(){
587     if( ! (msg.sender.call()( ) ) ){
588         throw;
589     }
590     counter += 1
591 }
592
593 /*callme() contains a benign reentrancy.*/
}
}
DASP : Unknown unknowns
Found: false
```

1.44 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:

```
594 /*In the following example, amount variable is divided by
↪ DELIMITER and then multiplied by BONUS. Thus, a rounding
↪ error appears (consider amount = 9000):*/
595
596 pragma solidity 0.4.25;
597
598 contract MyContract {
599
```

```

600     uint constant BONUS = 500;
601     uint constant DELIMITER = 10000;
602
603     function calculateBonus(uint amount) returns (uint) {
604         return amount/DELIMITER*BONUS;
605     }
606 }
}
}
DASP : Unknown Unknowns
Found: false

```

1.45 `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:

```

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

```

1.46 `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:

```

616 struct BalancesStruct{
617     address owner;
618     mapping(address => uint) balances;
619 }

```

```

620 mapping(address => BalancesStruct) public stackBalance;
621
622 function remove() internal{
623     delete stackBalance[msg.sender];
624 }
}
}
DASP : Unknown unknowns
Found: false

```

1.47 msg.value_equals_zero

SWC ID:

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

Example:

```

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

```

1.48 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:

```

626 pragma solidity >=0.5.0;
627 pragma experimental ABIEncoderV2;
628
629 import "./base.sol";
630
631 contract DerivedA is Base {
632     // i is not used in the current contract
633     A i = A(1);
634
635     int internal j = 500;
636
637     function call(int a) public {

```

```

638         assign1(a);
639     }
640
641     function assign3(A memory x) public returns (uint) {
642         return g[1] + x.a + uint(j);
643     }
644
645     function ret() public returns (int){
646         return this.e();
647     }
648
649     int internal j = 500;
650     function call(int a) public {
651         assign1(a);
652     }
653
654     function assign3(A memory x) public returns (uint) {
655         return g[1] + x.a + uint(j);
656     }
657
658     function ret() public returns (int){
659         return this.e();
660     }
661 }
}
}
DASP : Unknown unknowns
Found: false

```

1.49 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 `-1`, `uint8`) may cause an infinite loop. * Recursive external calls may consume a large number of callstacks, which may lead to DoS.

Example:

```

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

```

```

}
}
DASP : Denial of Services
Found: true

```

1.50 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:

```

663 pragma solidity 0.4.24;
664
665 contract dataStorage {
666     uint[] public data;
667
668     function writeData(uint[] \textunderscore data) external {
669         for(uint i = data.length; i < \textunderscore
        ↪ data.length; i++) {
670             data.length++;
671             data[i]=\textunderscore data[i];
672         }
673     }
674 }
}
}
DASP : Unknown Unknowns
Found: false

```

1.51 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:

```

675 pragma solidity \textsuperscript 0.4.0;
676
677 contract A{
678     uint num = 5;
679     constructor(uint x) public{
680         num += x;
681     }
682 }

```



```

683
684 contract B is A{
685     constructor() A(2) public { /* ... */ }
686 }
687
688 contract C is A {
689     constructor() A(3) public { /* ... */ }
690 }
691
692 contract D is B, C {
693     constructor() public { /* ... */ }
694 }
}
}
DASP : Unknown unknowns
Found: false

```

1.52 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:

```

695 contract TestContract is MultiOwnable {
696
697     function withdrawAll(){
698         msg.sender.transfer(this.balance);
699     }
700 }
}
}
DASP : Access control
Found: true

```

1.53 ignore

SWC_ID:

Description:Other trivial bug types.

Example:

```
701 |  
    }  
    }  
    DASP : Unknown Unknowns  
    Found: true
```

1.54 controlled_lowlevel_call

SWC_ID:

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

Example:

```
702 | address token;  
703 |  
704 | function call\textunderscore token(bytes data){  
705 |     token.call(data);  
706 | }  
707 |  
708 | /*token` points to an ERC20 token. Bob uses call\textunderscore  
    ↳ token to call the transfer function of token to withdraw  
    ↳ all tokens held by the contract.*/  
    }  
    }  
    DASP : Unknown Unknowns  
    Found: true
```

1.55 dangerous_enum_conversion

SWC_ID:

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

Example:

```
709 | pragma solidity 0.4.2;  
710 | contract Test{  
711 |     enum E{a}  
712 |     function bug(uint a) public returns(E){  
713 |         return E(a);  
714 |     }  
715 | }
```

```

    }
  }
  DASP : Unknown Unknowns
  Found: false

```

1.56 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:

```

716 Here is the example of correct view\textendash function:
717
718 contract C {
719     function f(uint a, uint b) view returns (uint) {
720         return a * (b + 42) + now;
721     }
722 }
723
724 }
725 DASP : Unknown unknowns
726 Found: false

```

1.57 uninitialized_local_variable

SWC_ID:

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

Example:

```

723 contract Uninitialized is Owner{
724     function withdraw() payable public onlyOwner{
725         address to;
726         to.transfer(this.balance)
727     }
728 }
729
730 /*Bob calls transfer. As a result, all Ether is sent to the
   ↪ address 0x0 and is lost.*/
731
732 }
733 DASP : Unknown unknowns
734 Found: false

```

1.58 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:

```
731 pragma solidity \textsuperscript 0.4.0;
732
733 contract A{
734     uint num = 5;
735     constructor(uint x) public{
736         num += x;
737     }
738 }
739
740 contract B is A{
741     constructor() A(2) public { /* ... */ }
742 }
743
744 contract C is A {
745     constructor() A(3) public { /* ... */ }
746 }
747
748 contract D is B, C {
749     constructor() public { /* ... */ }
750 }
751 }
```

DASP : Unknown unknowns

Found: false

1.59 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:

```
751 /*In the following example, currentBlockBlockhash function
752 ↪ always returns 0:*/
753
754 pragma solidity 0.4.25;
755
756 contract MyContract {
```

```

756     function currentBlockHash() public view returns(bytes32) {
757         return blockhash(block.number);
758     }
759 }
}
}
DASP : Unknown unknowns
Found: false

```

1.60 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:

```

760 contract DelegatecallInLoop{
761
762     mapping (address => uint256) balances;
763
764     function bad(address[] memory receivers) public payable {
765         for (uint256 i = 0; i < receivers.length; i++) {
766             ↪ address(this).delegatecall(abi.encodeWithSignature("addBalance(address)",
767             ↪ receivers[i]));
768         }
769     }
770
771     function addBalance(address a) public payable {
772         balances[a] += msg.value;
773     }
774 }
}
}
DASP : Unknown Unknowns
Found: false

```

1.61 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:

```

774 | /* In the following example, the send function is used:*/
775 |
776 |
777 | if(!addr.send(42 ether)) {
778 |     revert();
779 | }
780 |
781 | /*Preferred alternative:
782 |
783 | addr.transfer(42 ether);*/
    | }
    | }
    | DASP : Unknown Unknowns
    | Found: false

```

1.62 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:

```

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

```

DASP : Time Manipulation

Found: false

1.63 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:

```
801 contract Token{
802     function ownerOf(uint256 \textunderscore tokenId) external
803         ↪ view returns (bool);
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 : Unknown unknowns

Found: false

1.64 redundant_code

SWC ID:

Description: Redundant statements may have no effect.

Example:

```
805 contract RedundantStatementsContract {
806
807     constructor() public {
808         uint; // Elementary Type Name
809         bool; // Elementary Type Name
810         RedundantStatementsContract; // Identifier
811     }
812
813     function test() public returns (uint) {
814         uint; // Elementary Type Name
815         assert; // Identifier
816         test; // Identifier
817         return 777;
818     }
819 }
820
821 /*Each commented line references types/identifiers, but
822 ↪ performs no action with them, so no code will be generated
823 ↪ for such statements and they can be removed.*/
```

```

}
}
DASP : Unknown unknowns
Found: false

```

1.65 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:

```

822 | /*The following loop is infinite:*/
823 |
824 | do {
825 |     continue;
826 | } while(false);
}
}
DASP : Unknown Unknowns
Found: false

```

1.66 assert_state_change

SWC_ID:

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

Example:

```

827 | contract A {
828 |     uint s\textunderscore a;
829 |
830 |     function bad() public {
831 |         assert((s\textunderscore a += 1) > 10);
832 |     }
833 | }
834 | /*The assert in bad() increments the state variable
   ↪ s\textunderscore a while checking for the condition.*/
}
}
DASP : Unknown Unknowns
Found: false

```