
Effective Source Code Analysis with Minimization

July 5, 2016

Geet Tapan Telang

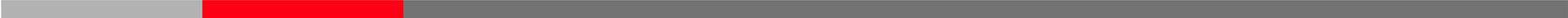
Research Engineer – IT Platform
Hitachi India Pvt. Ltd.

Contents

- 1. Introduction**
- 2. Results**
- 3. Conclusion**

1. Introduction

- Growing demand for OSS/Linux in Safety Critical domain.
- Size of code is approximately 20 million lines of code (Linux OS).
- Validation and analysis makes traditional methods difficult to follow.
- Code coverage and analysis is major part of verification and validation.
- Scoping the target code is a big challenge.



Problem

“#ifdef disasters”

/drivers/dma/dmaengine.c

➤ The #ifdefs makes the code hard to:

- Review
- Debug
- Maintain
- Verify

```
static bool device_has_all_tx_types(struct dma_device *device)
{
    /* A device that satisfies this test has channels that will never cause
     * an async_tx channel switch event as all possible operation types can
     * be handled.
     */
    #ifdef CONFIG_ASYNC_TX_DMA
    if (!dma_has_cap(DMA_INTERRUPT, device->cap_mask))
        return false;
    #endif

    #if defined(CONFIG_ASYNC_MEMCPY) || defined(CONFIG_ASYNC_MEMCPY_MODULE)
    if (!dma_has_cap(DMA_MEMCPY, device->cap_mask))
        return false;
    #endif

    #if defined(CONFIG_ASYNC_XOR) || defined(CONFIG_ASYNC_XOR_MODULE)
    if (!dma_has_cap(DMA_XOR, device->cap_mask))
        return false;
    #endif

    #ifndef CONFIG_ASYNC_TX_DISABLE_XOR_VAL_DMA
    if (!dma_has_cap(DMA_XOR_VAL, device->cap_mask))
        return false;
    #endif
    #endif

    #if defined(CONFIG_ASYNC_PQ) || defined(CONFIG_ASYNC_PQ_MODULE)
    if (!dma_has_cap(DMA_PQ, device->cap_mask))
        return false;
    #endif

    #ifndef CONFIG_ASYNC_TX_DISABLE_PQ_VAL_DMA
    if (!dma_has_cap(DMA_PQ_VAL, device->cap_mask))
        return false;
    #endif
    #endif

    return true;
}
```

/drivers/dma/dmaengine.c

```
static bool device_has_all_tx_types(struct dma_device *device)
{
    /* A device that satisfies this test has channels that will never cause
     * an async_tx channel switch event as all possible operation types can
     * be handled.
     */
    if (!dma_has_cap(DMA_INTERRUPT, device->cap_mask))
        return false;

    if (!dma_has_cap(DMA_PQ, device->cap_mask))
        return false;

    return true;
}
```

If code is free from #ifdef blocks then, analysis shall be more effective.

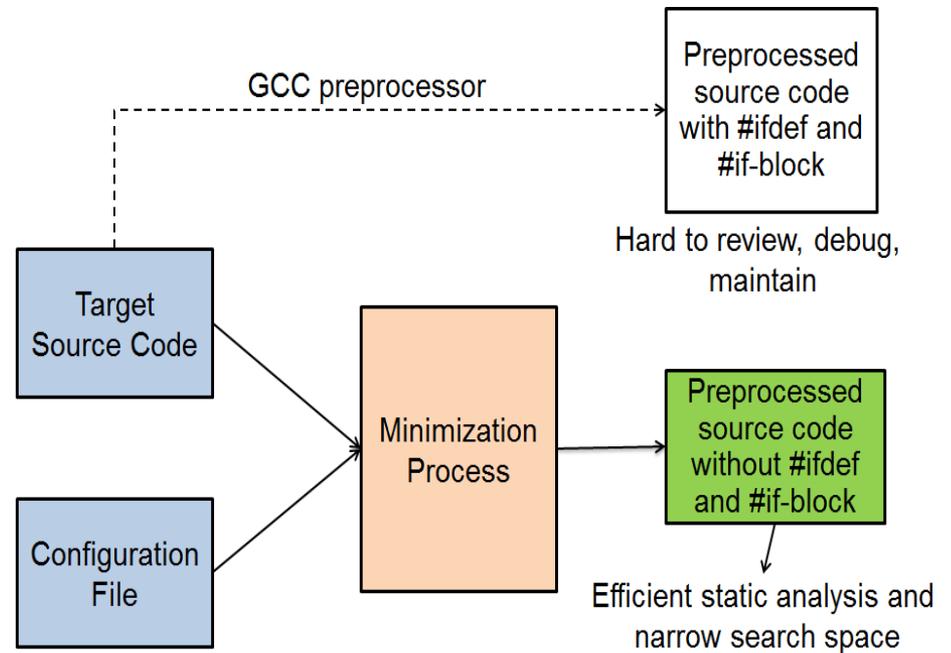
Is there a way ?



Approach

- The minimization approach tweaks integrated MakeFile options to produce compilable stripped code.
- Signifies efficient way to get a set of stripped kernel source code based on a .config file.

- Generate source tree where;
 - Unused `#ifdef`, `#if` blocks have been removed
 - `#include` and `#define` lines are preserved
 - Only used source files exist
 - Produces the same binary file as the original tree



```
C:\Users\khashimoto\Desktop\hoger\uname.c
131 if (toprint == 0) { /* no opts => -s (sysname) */
132     toprint = 1;
133 }
134
135 uname(&uname_info.name); /* never fails */
136
137 #if defined(__sparc) && defined(__linux__)
138     if (fake_sparc && (fake_sparc[0] | 0x20) == 'y') {
139         strcpy(uname_info.name.machine, "sparc");
140     }
141 #endif
142
143 strcpy(uname_info.processor, unknown_str);
144 strcpy(uname_info.platform, unknown_str);
145 strcpy(uname_info.os, CONFIG_UNAME_OSNAME);
146
147 #if 0
148 /* Fedora does something like this */
149 strcpy(uname_info.processor, uname_info.name.machine);
150 strcpy(uname_info.platform, uname_info.name.machine);
151 if (uname_info.platform[0] == 'i'
152     && uname_info.platform[1]
153     && uname_info.platform[2] == '8'
154     && uname_info.platform[3] == '6'
155 ) {
156     uname_info.platform[1] = '3';
157 }
158 #endif
159
160 delta = utsname_offset;
161 fmt = "%s" + 1;
```

```
C:\Users\khashimoto\Desktop\hoger\uname.c.minimized
126 if (toprint == 0) { /* no opts => -s (sysname) */
127     toprint = 1;
128 }
129
130 uname(&uname_info.name); /* never fails */
131
132
133
134
135
136
137
138
139
140
141
142
143 strcpy(uname_info.processor, unknown_str);
144 strcpy(uname_info.platform, unknown_str);
145 strcpy(uname_info.os, CONFIG_UNAME_OSNAME);
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
```

minimize

This code transformation is what we term as Minimization.

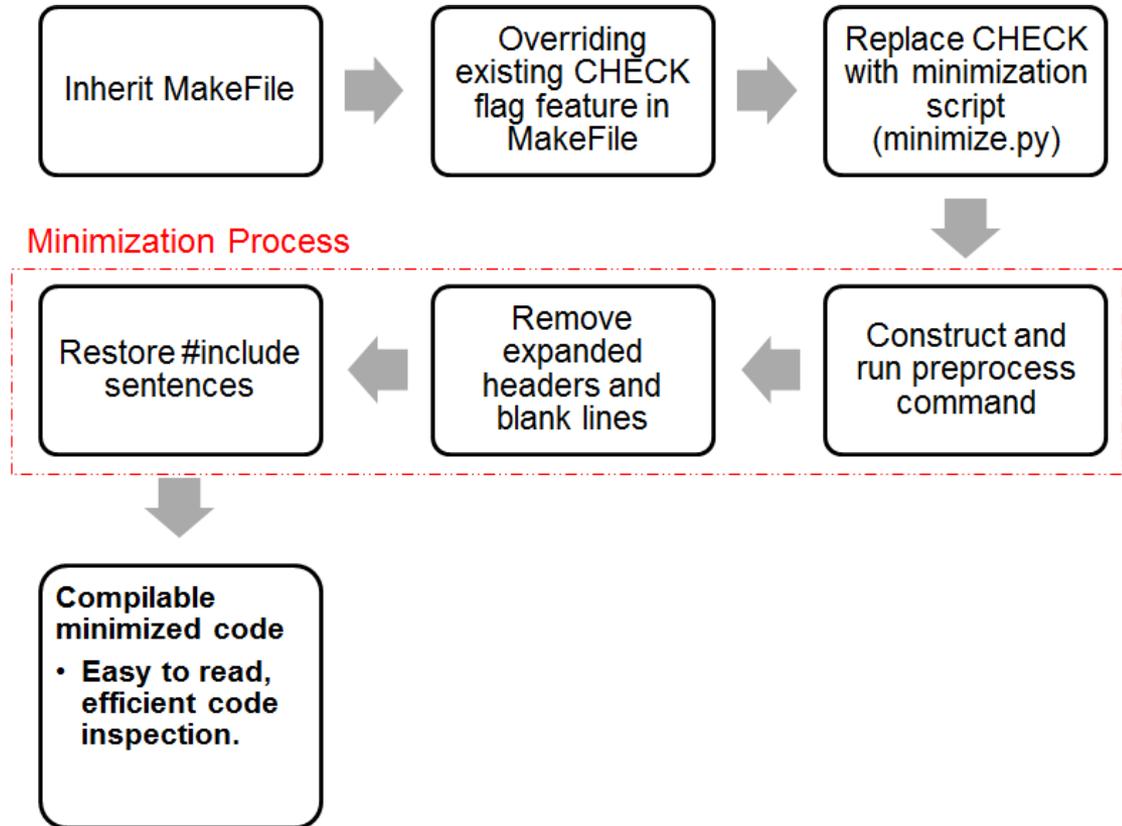
Original idea of using GREP (Approach-I)

- Requires complete build in advance.
- Text parsing has to be acquired from build log.
- Source code modification to remove redundant code.

Too much user Involvement!!!

Minimize.py script (Approach-II)

- MakeFile integration
 - Override existing CHECK flag feature
- Minimizing procedure
 - Preprocess, expanded header restoration
- Binary verification
 - Compare “minimized binary” and the original



- Override existing CHECK feature in kernel MakeFile

```
kotaro@kotaro-OptiPlex-7020:~/Minimization/linux-4.3.3$ make help | grep CHECK
make C=1    [targets] Check all c source with $CHECK (sparse by default)
make C=2    [targets] Force check of all c source with $CHECK
```

- Makefile of the root directory:

```
CHECK          = sparse
CHECKFLAGS     := -D__linux__ -Dlinux -D__STDC__ -Dunix -D__unix__ \
                 -Wbitwise -Wno-return-void $(CF)
```

- Minimization script(minimize.py) usage:

Replace **CHECK** with **minimize.py** so make can process minimization

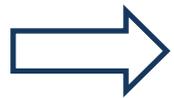
```
$ make C=1 CHECK=minimize.py CF="-mindir ../minimized-tree/"
```

In make process, "minimize.py" will receive the same option as the compile flags of each source file, plus \$CHECKFLAGS variable.

ON THE FLY GENERATION (no post processing)!!!

1. Preprocess the source files

```
gcc -E -fdirectives-only
```



`#ifdef` block disappears, `#include` gets expanded,
but `#define` macros are preserved, also removes empty lines

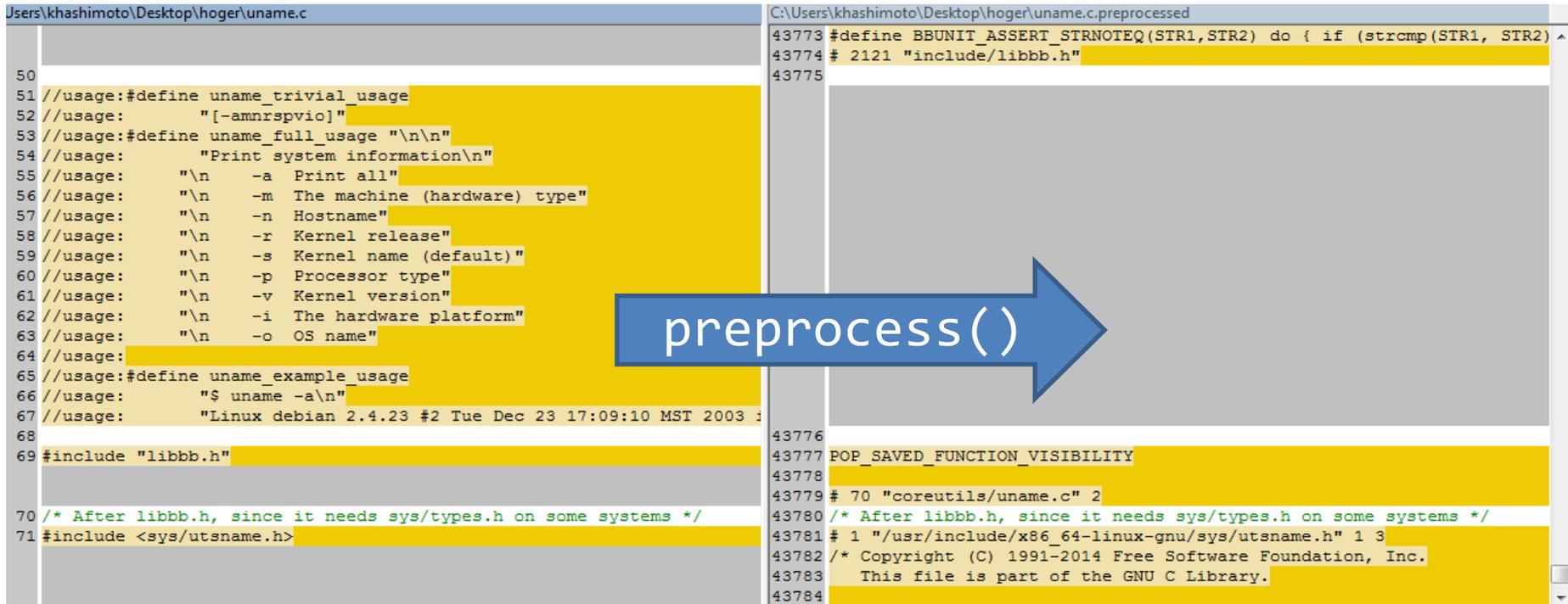
2. Identify & delete the expanded header contents

- Use clues(linemarkers) that exist in the preprocessed file
- Example of linemarkers: `# 30 "/usr/include/sys/stsname.h" 2`

3. Restore `#include` sentences

- Copy relevant `#include` lines from the original source

- preprocess() function in minimize.py
 - Takes gcc options passed via Makefile
 - Appends “-E –fdirectives-only” flags
 - Perform preprocess for the target C file



```
Users\khashimoto\Desktop\hoger\uname.c
50
51 //usage:#define uname_trivial_usage
52 //usage:      "[-amnrspvio]"
53 //usage:#define uname_full_usage "\n\n"
54 //usage:      "Print system information\n"
55 //usage:      "\n  -a  Print all"
56 //usage:      "\n  -m  The machine (hardware) type"
57 //usage:      "\n  -n  Hostname"
58 //usage:      "\n  -r  Kernel release"
59 //usage:      "\n  -s  Kernel name (default)"
60 //usage:      "\n  -p  Processor type"
61 //usage:      "\n  -v  Kernel version"
62 //usage:      "\n  -i  The hardware platform"
63 //usage:      "\n  -o  OS name"
64 //usage:
65 //usage:#define uname_example_usage
66 //usage:      "$ uname -a\n"
67 //usage:      "Linux debian 2.4.23 #2 Tue Dec 23 17:09:10 MST 2003 i
68
69 #include "libbb.h"
70 /* After libbb.h, since it needs sys/types.h on some systems */
71 #include <sys/utsname.h>

C:\Users\khashimoto\Desktop\hoger\uname.c.preprocessed
43773 #define BBUNIT_ASSERT_STRNOTEQ(STR1,STR2) do { if (strcmp(STR1, STR2) ^
43774 # 2121 "include/libbb.h"
43775
43776
43777 POP_SAVED_FUNCTION_VISIBILITY
43778
43779 # 70 "coreutils/uname.c" 2
43780 /* After libbb.h, since it needs sys/types.h on some systems */
43781 # 1 "/usr/include/x86_64-linux-gnu/sys/utsname.h" 1 3
43782 /* Copyright (C) 1991-2014 Free Software Foundation, Inc.
43783    This file is part of the GNU C Library.
43784
```


- stripHeaders() algorithm
 - Find linemakers (starting with “# number “filename””)
 - If *filename* is the target C file:
 - copy the following lines
 - And if *flag* in the linemaker is 2:
 - Mark “TO BE REPLACED” that means “there is #include line”

Flag 2 indicates returning to the file (after having included another file).

```
43768 # 2100 "include/libbb.h"
43769
43770 #define BBUNIT_ASSERT_STREQ(STR1,STR2) do { if (strcmp(STR1, STR2) !=
43771 # 2110 "include/libbb.h"
43772
43773 #define BBUNIT_ASSERT_STRNOTEQ(STR1,STR2) do { if (strcmp(STR1, STR2)
43774 # 2121 "include/libbb.h"
43775
43776
43777 POP_SAVED_FUNCTION_VISIBILITY
43778
43779 # 70 "coreutils/uname.c" 2
43780 /* After libbb.h, since it needs sys/types.h on some systems */
43781 # 1 "/usr/include/x86_64-linux-gnu/sys/utsname.h" 1 3
43782 /* Copyright (C) 1991-2014 Free Software Foundation, Inc.
43783 This file is part of the GNU C Library.
43784
43785 The GNU C Library is free software; you can redistribute it and/or
43786 modify it under the terms of the GNU Lesser General Public
64 //usage:
65 //usage:#define uname_example_usage
66 //usage: "$ uname -a\n"
67 //usage: "Linux debian 2.4.23 #2 Tue Dec 23 17:09:10 MST 2003 i6
68
69 TO BE REPLACED: "include/libbb.h"
70 /* After libbb.h, since it needs sys/types.h on some systems */
71 TO BE REPLACED: "/usr/include/x86_64-linux-gnu/sys/utsname.h"
```

stripHeaders()

- restoreHeaderInclude() function in minimize.py
 - Takes header-stripped preprocessed file
 - Look for “TO BE REPLACED” marks
 - Compare with the original C file, copy original #include lines

restoreHeaderInclude()



```
64 //usage:
65 //usage:#define uname_example_usage
66 //usage:      "$ uname -a\n"
67 //usage:      "Linux debian 2.4.23 #2 Tue Dec 23 17:09:1
68
69 TO BE REPLACED: "include/libbb.h"
70 /* After libbb.h, since it needs sys/types.h on some syst
71 TO BE REPLACED: "/usr/include/x86_64-linux-gnu/sys/utsnam
72
73 typedef struct {
74     struct utsname name;
75     char processor[sizeof(((struct utsname*)NULL)->machin
76     char platform[sizeof(((struct utsname*)NULL)->machin
```

```
64 //usage:
65 //usage:#define uname_example_usage
66 //usage:      "$ uname -a\n"
67 //usage:      "Linux debian 2.4.23 #2 Tue Dec 23 17:09:
68
69 #include "libbb.h"
70 /* After libbb.h, since it needs sys/types.h on some sys
71 #include <sys/utsname.h>
72
73 typedef struct {
74     struct utsname name;
75     char processor[sizeof(((struct utsname*)NULL)->machin
76     char platform[sizeof(((struct utsname*)NULL)->machin
```

- Finally, diff result is only deletions of the unused code.
 - Without changing #include, #define lines.
 - Minimization also removes blank lines which comprised of unused code.

<pre>#if defined(__sparc__) && defined(__linux__) if (fake_sparc && (fake_sparc[0] 0x20) == 'y'){ strcpy(uname_info.name.machine, "sparc"); } #endif strcpy(uname_info.processor, unknown_str); strcpy(uname_info.platform, unknown_str); strcpy(uname_info.os, CONFIG_UNAME_OSNAME); #if 0 /* Fedora does something like this */ strcpy(uname_info.processor, uname_info.name.machine); strcpy(uname_info.platform, uname_info.name.machine); if (uname_info.platform[0] == 'i' && uname_info.platform[1] && uname_info.platform[2] == '8' && uname_info.platform[3] == '6') { uname_info.platform[1] = '3'; } #endif delta = utsname_offset; fmt = "%s" + 1;</pre>		<pre>strcpy(uname_info.processor, unknown_str); strcpy(uname_info.platform, unknown_str); strcpy(uname_info.os, CONFIG_UNAME_OSNAME); delta = utsname_offset; fmt = "%s" + 1;</pre>
---	--	--

2. Results

Linux Kernel Tree

- allnoconfig: 64684 unused lines were removed → 22% of original C code.
- defconfig: 103144 unused lines were removed → 5% of original C code.

BusyBox Tree

- allnoconfig: 51 out of 112 compiled C files have been minimized 5945 lines unused lines were removed → 34% of original C code
- defconfig: 296 out of 505 compiled C files have been minimized. 20453 lines unused lines were removed → 11% of original C code

ARCTIC Core source code

- Statistics shows approximately 5.5 times higher chances of eliminating unused `#ifdef` switches.



Evaluation

Complexity Statistics

- To analyze the complexity of “C” program function.
- Linux with PREEMPT_RT patch, Linux Kernel source, BusyBox tree as shown in table below.
- Complexity (a GNU utility) tool has been used.

Disassembled code(“objdump -d”) matches

- Between the binaries built from minimized source and original one.
- Confirmed configuration & target:
 - BusyBox-1.24.1: defconfig, allnoconfig
 - busybox (executable)
 - Linux kernel 4.4.1: allnoconfig
 - vmlinux.o

Minimized code is compilable and produces same binary

Complexity Metrics	Linux Kernel			BusyBox Tree			PREEMPT_RT	
	Original Source	Minimized(x86_defconfig)	Minimized(allnoconfig)	Original Source	Minimized(x86_defconfig)	Minimized(allnoconfig)	Original	Minimized
Average Line Score	23	7	5	22	21	19	10	7
50%-ile score	4	3	2	9	9	5	4	3
Highest Score	1846	194	158	283	283	283	530	194

Measured complexity in terms of average line score, 50%-ile score and highest score.

Benefits

- Verification time and cost improvement
 - Static analysis through Coccinelle
 - Executed a semantic patch for detecting functions have different return type values
 - Statistics
 - Comparison of execution time and minimization was faster.
 - 12[s] and 2.24[s] for original and minimized kernel respectively.
- False positive reduction
 - Wrong indication about presence of particular condition.
 - Statistics
 - Original kernel source: 126
 - Minimized kernel source: 82
- Pruning function call graph
 - Analysis requires every possible call path to establish and trace relationship between program and subroutines.
 - Call graph is a directed graph that represents this relationship.

Extracting Minimal Subtarget Sources

```
$ cd busybox-1.24.1  
$ make init C=2 CHECK=minimize.py CF="-mindir ../min-init"
```

If subtarget is specified in the minimized command,
Only the used source files will be extracted.

```
min-init/  
├── applets  
│   └── applets.c  
├── include  
│   ├── applet_metadata.h  
│   ├── autoconf.h  
│   ├── busybox.h  
│   ├── grp.h  
│   ├── libbb.h  
│   ├── platform.h  
│   ├── pwd.h  
│   ├── shadow.h  
│   └── xatnum.h  
└── init  
    ├── bootchartd.c  
    ├── halt.c  
    ├── init.c  
    ├── mesg.c  
    └── reboot.h
```

Depended *.c files in minimized form.
Actually included *.h files

- ✓ Easy to identify which files are used
- ✓ Helps efficient software walk-through

3. Conclusion

- Improves readability for human.
 - Helps efficient code review / inspection.
- Narrows down “search space”.
 - Gives evidence for unused code.
 - Saves verification cost (time & space).
 - Achieves higher test coverage.
 - Reduces false-positives.
- From analysis stand-point, this provides
 - Reduction in verification time
 - False-positive reduction
- Much more potential for domains like safety and mission critical systems.

- To adapt more config / architecture
 - More than allnoconfig, defconfig / x86, arm
- To adapt more projects
 - For different build system (automake, CMake etc.)
- To prove minimized tree is “equal” to original one
 - How to formally verify equivalence???
- To find out more applications
 - Something that enhances existing tools / techniques
- Available in:
 - <https://github.com/Hitachi-India-Pvt-Ltd-RD/minimization>

END



Effective Source Code Analysis with Minimization

July 5, 2016

Geet Tapan Telang

Hitachi India Pvt. Ltd.