Εντοπισμός σφαλμάτων στη διαχείριση μνήμης

Για να εντοπιστούν λάθη στη διαχείριση μνήμης συνίσταται η χρήση κατάλληλων εργαλείων. Θα παρουσιάσουμε δύο εργαλεία, το AddressSanitizer και το Valgrind, μέσα από σύντομα παραδείγματα που επιδεικνύουν τη λειτουργία τους.

Valgrind

Ανήκει σε μια κατηγορία εργαλείων που λέγονται memory profilers. Πρόκειται για εργαλεία που παίρνουν ως είσοδο εκτελέσιμο κώδικα, εισάγουν δικές τους εντολές σε αυτόν οι οποίες παρατηρούν και αναλύουν τη χρήση μνήμης και κατόπιν τον εκτελούν, καταγράφοντας τα αποτελέσματα της ανάλυσης. Για περισσότερες πληροφορίες δείτε στο https://valgrind.org/.

Εγκατάσταση

Το valgrind είναι διαθέσιμο για linux, για mac os μέχρι την έκδοση 10.12 και στο WSL. Μπορείτε να το εγκαταστήσετε μέσω της εφαρμογής εγκατάστασης λογισμικού της διανομής linux που έχετε. Εάν έχετε το VM που σας δώσαμε ή WSL, τότε γράψτε σε ένα τερματικό τις εντολές:

```
sudo apt update
sudo apt install valgrind
```

και επιλέξτε y σε ότι ερώτηση σας κάνει.

Μεταγλώττιση κώδικα και χρήση

Μεταγλωττίστε το πρόγραμμα σας με την επιλογή -g.

Εκτελέστε το πρόγραμμά σας γράφοντας valgrind και ακολούθως την εντολή εκτέλεσης, για παράδειγμα:

```
valgrind ./project1 250 250 < in7 > /dev/null
```

Για να ελαχιστοποιηθεί ο "θόρυβος" στην οθόνη, ανακατευθύνεται η συμβατική έξοδος του προγράμματος στο `/dev/null`, ένα ειδικό αρχείο στο οποίο ότι γράφεται εξαφανίζεται/απορρίπτεται.

AddressSanitizer

Πρόκειται για εργαλείο που παρέχεται μέσω συγκεκριμένων μεταγλωττιστών όπως gcc και clang/llvm και ανιχνεύει σφάλματα όπως stack overflow, προσπέλαση αποδεσμευμένης μνήμης, memory leaks κ.α. Για περισσότερες πληροφορίες δείτε στο https://github.com/google/sanitizers/wiki/AddressSanitizer.

Μεταγλώττιση κώδικα και χρήση

Μεταγλωττίστε το πρόγραμμά σας με τις επιλογές `-fsanitize=address -static-libasan`
Σε macOS παραλείψτε το -static-libasan. Μετά, εκτελέστε κανονικά το πρόγραμμά σας.

Παράδειγμα 1

```
 1 #include<stdio.h>
 2 #include<stdlib.h>
 3
 4 int main (int argc, char *argv[]) {
 5    int **p, i;
 6    p = (int**)malloc(sizeof(int)*10);
 7    for (i=0; i<10; i++) {
 8           p[i] = (int*)malloc(sizeof(int));
 9    }
10    for (i=0; i<10; i++) {
11           printf("%p\n", p[i]);
12    }
13    free(p);
14    return 0;
15 }
```

Έξοδος valgrind:

```
==49942== Invalid write of size 8
==49942==    at 0x108720: main (test.c:8)
==49942==  Address 0x522d068 is 0 bytes after a block of size 40 alloc'd
==49942==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==49942==    by 0x1086F3: main (test.c:6)
==49942==
==49942== Invalid read of size 8
==49942==    at 0x10874A: main (test.c:11)
==49942==  Address 0x522d068 is 0 bytes after a block of size 40 alloc'd
==49942==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==49942==    by 0x1086F3: main (test.c:6)
==49942==
==49942== HEAP SUMMARY:
==49942==     in use at exit: 40 bytes in 10 blocks
==49942==   total heap usage: 12 allocs, 2 frees, 4,176 bytes allocated
==49942==
==49942== LEAK SUMMARY:
==49942==    definitely lost: 40 bytes in 10 blocks
==49942==    indirectly lost: 0 bytes in 0 blocks
==49942==      possibly lost: 0 bytes in 0 blocks
==49942==    still reachable: 0 bytes in 0 blocks
==49942==         suppressed: 0 bytes in 0 blocks
==49942==
==49942== For counts of detected and suppressed errors, rerun with: -v
==49942== ERROR SUMMARY: 11 errors from 3 contexts (suppressed: 0 from 0)
```

Ερμηνεία:

Το μήνυμα "`Invalid write of size 8`" σημαίνει ότι προσπαθούμε να γράψουμε σε μνήμη την οποία δεν έχουμε δεσμεύσει. Προκύπτει στη γραμμή 8 του κώδικά μας και η παράνομη εγγραφή πάει να γίνει στο πρώτο byte αμέσως μετά από τα bytes που δεσμεύτηκαν στη γραμμή 6. Επιπλέον, το μήνυμα μας ενημερώνει ότι δεσμεύτηκαν 40 bytes από τη συγκεκριμένη malloc. Σε αυτό το σημείο παρατηρούμε ότι το 40 είναι αρκετά μικρότερο από όσο θα έπρεπε: δεσμεύσαμε 10 θέσεις για ακεραίους (μεγέθους 4 bytes ο καθένας), ενώ χρειαζόμασταν 10 θέσεις για δείκτες (μεγέθους 8 bytes ο καθένας).

Το μήνυμα "`Invalid read of size 8`" σημαίνει ότι προσπαθούμε να διαβάσουμε δεδομένα μνήμη την οποία δεν έχουμε δεσμεύσει και οφείλεται στο ίδιο λάθος με παραπάνω.

Ακολουθούν πληροφορίες για το κατά πόσο αποδεσμεύτηκε η δυναμικά δεσμευμένη μνήμη του προγράμματος (HEAP SUMMARY). Το μήνυμα "`in use at exit: 40 bytes in 10 blocks`" σημαίνει ότι στο τέλος του προγράμματος υπήρχαν 40 bytes που δεν είχαν γίνει free. Παρατηρούμε πως το valgrind προτείνει να το τρέξουμε με επιλογή `--leak-check=full` για να πάρουμε περισσότερες πληροφορίες. Αν το κάνουμε, θα δούμε επιπλέον:

```
==49942== 40 bytes in 10 blocks are definitely lost in loss record 1 of 1
==49942==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==49942==    by 0x10871F: main (test.c:8)
```

Η μνήμη που ξεχάσαμε να αποδεσμεύσουμε αντιστοιχεί στη δέσμευση που κάναμε στη γραμμή 8.

Σημείωση: Ίσως παρατηρήσετε ότι φαίνεται να υπάρχει μια ασυνέπεια στο μέτρημα των δεσμεύσεων μνήμης. Το valgrind καταγράφει "`12 allocs, 2 frees, 4,176 bytes allocated`" ενώ ο κώδικάς μας δεσμεύει 11 πράγματα (1 στη γραμμή 6 και 10 στη γραμμή 8) και σίγουρα όχι 4176 bytes. Αυτό οφείλεται στο ότι το valgrind καταγράφει και ότι δεσμεύσεις/αποδεσμεύσεις γίνονται κατά την εκτέλεση συναρτήσεων βιβλιοθήκης. Στη συγκεκριμένη περίπτωση, η επιπλέον μνήμη προήλθε από buffers που δεσμεύτηκαν/αποδεσμεύτηκαν κατά τη λειτουργία της printf. Δοκιμάστε να βάλετε την printf σε σχόλια και να ξανατρέξετε valgrind στον κώδικα για να δείτε τη διαφορά.

Έξοδος sanitizer:

```
==54415==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x604000000078 at pc
0x55748f71aeed bp 0x7fff277149c0 sp 0x7fff277149b0
WRITE of size 8 at 0x604000000078 thread T0
    #0 0x55748f71aeec in main /srv/homes/vdoufexi/tests/test.c:8
    #1 0x7f8fb105bb96 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x21b96)
    #2 0x55748f61c8b9 in _start (/srv/homes/vdoufexi/tests/a.out+0x78b9)

0x604000000078 is located 0 bytes to the right of 40-byte region
[0x604000000050,0x604000000078)
allocated by thread T0 here:
    #0 0x55748f6da690 in malloc (/srv/homes/vdoufexi/tests/a.out+0xc5690)
    #1 0x55748f71ae9d in main /srv/homes/vdoufexi/tests/test.c:6
    #2 0x7f8fb105bb96 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x21b96)

SUMMARY: AddressSanitizer: heap-buffer-overflow /srv/homes/vdoufexi/tests/test.c:8 in main
```

Ερμηνεία:
Τα μηνύματα είναι παρεμφερή με αυτά του valgrind. Και εδώ γίνεται αναφορά σε παράνομη εγγραφή (`WRITE of size 8`) αμέσως μετά το τέλος (`0 bytes to the right`) τμήματος δεσμευμένης μνήμης, κι ενημερωνόμαστε ότι (α) η παράνομη εγγραφή προέκυψε στη γραμμή 8 του προγράμματος και η μνήμη που υπερβήκαμε είχε δεσμευτεί στη γραμμή 6 του προγράμματος.

Παράδειγμα 2

```
 1 #include<stdio.h>
 2 #include<stdlib.h>
 3
 4 int main (int argc, char *argv[]) {
 5    int ** p, i;
 6    p = (int**)malloc(sizeof(int*)*10);
 7    for (i=0; i<10; i++) {
 8           p[i] = (int*)malloc(sizeof(int));
 9    }
10    p = (int**)realloc(p, sizeof(int*)*15);
11    for (i=0; i<15; i++) {
12           if (p[i]) {
13           int x = *p[i];
14           }
15    }
16    return 0;
17 }
```

Έξοδος valgrind:

```
==50478== Conditional jump or move depends on uninitialised value(s)
==50478==    at 0x108715: main (test2.c:12)
<τα υπόλοιπα παραλείπονται γιατί καλύπτονται από το παράδειγμα 1>
```

Ερμηνεία:

Το μήνυμα "Conditional jump or move depends on uninitialised value(s)" αναφέρεται στο ότι κατά τον υπολογισμό της συνθήκης της εντολής if γίνεται χρήση κάποιας ποσότητας που δεν έχει αρχικοποιηθεί. Το λάθος οφείλεται στο ότι η επιπλέον μνήμη που δεσμεύτηκε από τη realloc δεν έχει αρχικοποιηθεί (π.χ. σε NULL). Οι δείκτες από τη θέση p[10] έως και τη θέση p[14] έχουν τυχαίες τιμές, πιθανώς μη-NULL.

Έξοδος sanitizer:

```
==2317==ERROR: AddressSanitizer: SEGV on unknown address 0x000000000000 (pc 0x5576d3a7fa55
bp 0x7ffd1577eb40 sp 0x7ffd1577eb10 T0)
==2317==The signal is caused by a READ memory access.
==2317==Hint: address points to the zero page.
    #0 0x5576d3a7fa54 in main /home/prog1/tests/test2.c:13
    #1 0x7f948b2b9b96 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x21b96)
    #2 0x5576d3a7f859 in _start (/home/prog1/tests/a.out+0x859)

AddressSanitizer can not provide additional info.
```

Ερμηνεία:

Απλά μας ενημερώνει ότι προέκυψε segmentation fault (SEGV) χωρίς να μπορεί να δώσει περισσότερες πληροφορίες.

Παράδειγμα 3

Κώδικας:

```
 1 #include<stdio.h>
 2 #include<string.h>
 3
 4 int main (int argc, char *argv[]) {
 5    char str[6] = {"Hello"};
 6    strcpy(str, &str[1]);
 7    printf("%s\n", str);
 8    return 0;
 9 }
```

Έξοδος valgrind:

```
==50684== Source and destination overlap in strcpy(0x1ffefff162, 0x1ffefff163)
==50684==    at 0x4C32E97: strcpy (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
```

Ερμηνεία:

Η strcpy έχει απροσδιόριστη συμπεριφορά όταν αντιγράφει μέρος ενός string πάνω στον εαυτό του. Αυτό σημαίνει πως μπορεί να λειτουργήσει σωστά, αλλά μπορεί και όχι. Το ίδιο ισχύει και για άλλες συναρτήσεις που ορίζονται στο string.h και αφορούν αντιγραφή δεδομένων (π.χ. strcat, memcpy). Είναι πάντα καλή ιδέα να διαβάζουμε τα man pages των συναρτήσεων που χρησιμοποιούμε, ώστε να βλέπουμε ακριβώς τι περιορισμοί υπάρχουν στη χρήση των συναρτήσεων και να **μην** εξαρτόμαστε μόνο από εμπειρικές παρατηρήσεις ("δούλεψε σωστά τη μία φορά που το δοκίμασα").

Έξοδος sanitizer:

```
==2371==ERROR: AddressSanitizer: strcpy-param-overlap: memory ranges
[0x7ffdf7227710,0x7ffdf7227715) and [0x7ffdf7227711, 0x7ffdf7227716) overlap
    #0 0x7f723a382299  (/usr/lib/x86_64-linux-gnu/libasan.so.4+0x66299)
    #1 0x562ef0212ac9 in main /home/prog1/tests/test3.c:5
    #2 0x7f7239f4cb96 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x21b96)
    #3 0x562ef02128d9 in _start (/home/prog1/tests/a.out+0x8d9)

Address 0x7ffdf7227710 is located in stack of thread T0 at offset 32 in frame
    #0 0x562ef02129c9 in main /home/prog1/tests/test3.c:3

  This frame has 1 object(s):
    [32, 38) 'str' <== Memory access at offset 32 is inside this variable
HINT: this may be a false positive if your program uses some custom stack unwind mechanism
or swapcontext
      (longjmp and C++ exceptions *are* supported)
Address 0x7ffdf7227711 is located in stack of thread T0 at offset 33 in frame
    #0 0x562ef02129c9 in main /home/prog1/tests/test3.c:3
```

Ερμηνεία:

Αναφέρεται ακριβώς το ίδιο πρόβλημα που είδαμε και στο valgrind με παρεμφερή τρόπο.