Eemil Aspholm

CT30A3370 Käyttöjärjestelmät ja systeemiohjelmointi

Viikkotehtävä 5.

Tehtävä 1.

A bug hunter’s diary kirjassa kirjan kirjoittaja testaa erilaisia bugeja erilaisista ohjelmista ja kyseisessä kappaleessa hän käsittelee VLC media player ohjelmistoa. Kirjoittaja tutki ohjelmiston lähdekoodia ja lähti tarkastelemaan mahdollisia haavoittuvuuksia ja bugeja, joita lähdekoodi saattaisi sisältää. Noin puolen päivän tutkimisen jälkeen hän löysi haavoittuvuuden, pinopuskurin ylivuodon. Tämä aiheutui, kun ohjelman lähdekoodissa tietyntyyppisellä media tiedostomuodolla (.TIVO) ohjelma kirjoittaa kutsupinon muistiosoitteen aiotun tietorakenteen ulkopuolelle, joka on suuruudeltaan kiinteän kokoinen. (Pinon puskuriin kirjoitettiin enemmän tietoa, kun puskurille oli varattu).

Haavoittuvuuden kirjoittaja löysi luomalla ensin listan VLC mediaplayerin demuxereista (prosesseista, jotka erottavat äänen ja videon ja muun datan streamista, jotta video voidaan toistaa), identifioimalla syötteen, jota demuxerit käsittelevät, ja seuraamalla syötteen viitteitä (missä demuksereita käytetään). Lopputuloksena haavoittuvuuden löytäjä ilmoitti VLC mediaplayerille löydetystä haavoittuvuudesta, joka saatiin kahden yrityksen jälkeen korjattua. Lopulta VLC mediaplayer julkaisi uuden hotfixin ja käyttäjä julkaisi myös postauksen internettiin tietoturva haavoittuvuudesta.

Kuten aiemmassa kappaleessa mainittiin kyseessä, on pinopuskurin ylivuoto hyökkäys. Kyseisessä hyökkäyksessä pinolla on rajallinen koko, ja jos käyttäjän syöte ylittää tuon varatun muistimäärän, eikä ohjelma tarkista, että syöte mahtuu pinoon, tapahtuu pinon ylivuoto. Pinon ylivuoto ei itsessään ole suuri ja vaarallinen ongelma, mutta siitä tulee iso tietoturvariski, jos käyttäjän syöte on tarkoitusperältänsä haitallinen. Mikäli syöte on haitallinen voi se korruptoida pinon ”ruiskuttamalla” suoritettavaa koodia käynnissä olevaan ohjelmaan, ja ottaa tällä tavalla kontrollin prosessista. Tällä tavoin syötteen antajalla on mahdollisuus saada luvattomasti pääsy tietokoneeseen.

Tehtävä 2.

Ohjelmakoodi:

```
1
2 #include <stdlib.h>
3 #include <stdio.h>
4
5 struct LIST{
6         int data;
7         struct LIST* next;
8         };
9
10 void main(){
11         struct LIST* pHead;
12         struct LIST* pNew;
13         pHead = (struct LIST*)malloc(sizeof(struct LIST));
14         for(int i = 0; i<10; i++){
15         if(pHead == NULL){
16         pHead = (struct LIST*)malloc(sizeof(struct LIST));
17         pHead->data = i;
18         }else{
19         pNew = (struct LIST*)malloc(sizeof(struct LIST));
20         pNew->data = i;
21         pHead->next = pHead;
22         }}
23
24 return;}
```

Kun kääntäjällä suoritetaan ohjelma, jossa varataan muistia malloc käskyllä, mutta joka ei vapauta varattua muistia ennen ohjelman lopetusta ei tapahdu virheilmoitusta, tai muuta ohjelman suorittamista haittaavaa ajon aikana. Gdb käskyä ei voi käyttää ongelmien löytämiseen, sillä se ei pysty ilmoittamaan muistivuotoja ohjelmassa. Valgrind puolestaan ilmoittaa muistivuodoista ohjelmassa seuraavan kuvan avulla:

```
==2518== HEAP SUMMARY:
==2518==     in use at exit: 176 bytes in 11 blocks
==2518==   total heap usage: 11 allocs, 0 frees, 176 bytes allocated
==2518==
==2518== 160 bytes in 10 blocks are definitely lost in loss record 2 of 2
==2518==    at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreloa
d_memcheck-amd64-linux.so)
==2518==    by 0x109195: main (in /home/eemil/Desktop/program.exe)
==2518==
==2518== LEAK SUMMARY:
==2518==    definitely lost: 160 bytes in 10 blocks
==2518==    indirectly lost: 0 bytes in 0 blocks
==2518==      possibly lost: 0 bytes in 0 blocks
==2518==    still reachable: 16 bytes in 1 blocks
==2518==         suppressed: 0 bytes in 0 blocks
==2518== Reachable blocks (those to which a pointer was found) are not shown.
==2518== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==2518==
==2518== For lists of detected and suppressed errors, rerun with: -s
==2518== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

```
#include <stdlib.h>
#include <stdio.h>


void main(){
        int length = 100;
        int *data = (int*) malloc(length*sizeof(int));
        data[100] = 0;

return;}
```

Ohjelmaa nro 2 suoritettaessa ei myöskään tule ongelmia (suoritus / kääntäminen). Kuitenkin valgrind ilmoittaa virheellisestä kirjoituksesta ja muistivuodosta ohjelmassa.

```
==2583== Invalid write of size 4
==2583==    at 0x10917B: main (in /home/eemil/Desktop/program.exe)
==2583==  Address 0x4a521d0 is 0 bytes after a block of size 400 alloc'd
==2583==    at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreloa
d_memcheck-amd64-linux.so)
==2583==    by 0x10916C: main (in /home/eemil/Desktop/program.exe)
==2583==
==2583==
==2583== HEAP SUMMARY:
==2583==     in use at exit: 400 bytes in 1 blocks
==2583==   total heap usage: 1 allocs, 0 frees, 400 bytes allocated
==2583==
==2583== 400 bytes in 1 blocks are definitely lost in loss record 1 of 1
==2583==    at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreloa
d_memcheck-amd64-linux.so)
==2583==    by 0x10916C: main (in /home/eemil/Desktop/program.exe)
==2583==
==2583== LEAK SUMMARY:
==2583==    definitely lost: 400 bytes in 1 blocks
==2583==    indirectly lost: 0 bytes in 0 blocks
==2583==      possibly lost: 0 bytes in 0 blocks
```

Tehtävä 3.

Ohjelmakoodi:

```
#include <stdlib.h>
#include <stdio.h>


void main(){
        int length = 100;
        int *data = (int*) malloc(length*sizeof(int));
        free(data);
        printf("%d", data[5]);
return;}
```

Ohjelmaa suorittuu ja kääntyy normaalisti ilman ongelmia, lisäksi ohjelma tulostaa terminaaliin numeron 0, vaikka muistipaikat ovat vapautettu ennen tulostamista. Valgrind ilmoittaa puolestaan ettei muistivuotoja ole havaittu, mutta syntaksista löytyy silti lukuvirhe. Alla kuva valgrindin tulosteesta:

```
==2631== Invalid read of size 4
==2631==    at 0x1091C5: main (in /home/eemil/Desktop/program.exe)
==2631==  Address 0x4a52054 is 20 bytes inside a block of size 400 free'd
==2631==    at 0x483CA3F: free (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_
memcheck-amd64-linux.so)
==2631==    by 0x1091BC: main (in /home/eemil/Desktop/program.exe)
==2631==  Block was alloc'd at
==2631==    at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreloa
d_memcheck-amd64-linux.so)
==2631==    by 0x1091AC: main (in /home/eemil/Desktop/program.exe)
==2631==
0==2631==
==2631== HEAP SUMMARY:
==2631==     in use at exit: 0 bytes in 0 blocks
==2631==   total heap usage: 2 allocs, 2 frees, 1,424 bytes allocated
==2631==
==2631== All heap blocks were freed -- no leaks are possible
==2631==
==2631== For lists of detected and suppressed errors, rerun with: -s
==2631== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Muokattaessa ohjelmaa vapauttamaan aluksi osoitin listan kuudenteen jäseneen, ohjelma kääntyy, mutta suorittaessa tulee vastaan ongelma. Suorittaessa tapahtuu suoritusaikainen virhe. Tällaisen löytämiseen ohjelmistokoodista tarvita gdb:tä erikoisempia työkaluja, sillä se ilmoittaa ohjelmaa käännettäessä terminaaliin, mikäli kääntämiskäskyn oheen lisättäisiin erilaisia virhetestauksia.

```
eemil@eemil-VirtualBox:~/Desktop$ ./program.exe
free(): invalid pointer
Aborted (core dumped)
```

Vektori on tietorakenne, jonka alkioiden määrää voi muuttaa ohjelman suorituksen, vahvuutena vektorilla voidaan katsoa olevan etenkin alkioiden poistaminen tietorakenteen lopusta, satunnainen alkioiden lukeminen tietorakenteesta, sekä alkioiden lisääminen rakenteen loppuun. Linkitetyt lista puolestaan ovat hyviä alkioiden lisäämiseen tietorakenteen alkuun tai loppuun vakioajassa, mutta esimerkiksi linkitetyn listan keskeltä alkion poistaminen ei ole kovinkaan tehokasta (ensin täytyy löytää haluttu alkio ja tämän jälkeen suorittaa halutut operandit). Yhteenvetona vektorit ovat parempi vaihtoehto dynaamiselle tietorakenteelle, mikäli säilöttävänä ei ole miljoonia olioita.

Alla vector.c ohjelman suorittaminen Valgrindin avulla:

```
eemil@eemil-VirtualBox:~$ valgrind --leak-check=full ./vector.exe
==2003== Memcheck, a memory error detector
==2003== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==2003== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==2003== Command: ./vector.exe
==2003==
Start:
Vector has 0 spaces and 0 items stored
Insert number to the vector:9
Vector has 1 spaces and 1 items stored
The value at index 0 is 9
Insert number to the vector:90
Vector has 2 spaces and 2 items stored
The value at index 1 is 90
==2003==
==2003== HEAP SUMMARY:
==2003==     in use at exit: 0 bytes in 0 blocks
==2003==   total heap usage: 6 allocs, 6 frees, 2,076 bytes allocated
==2003==
==2003== All heap blocks were freed -- no leaks are possible
==2003==
==2003== For lists of detected and suppressed errors, rerun with: -s
==2003== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Ohjelmakoodi, jossa lähteenä käytetty:

https://github.com/martalist/ostep/blob/master/chapter_13/hw13/vector.c

```c
#include <stdio.h>
#include <stdlib.h>

//Source https://github.com/martalist/ostep/blob/master/chapter_13/hw13/vector.c

typedef struct vect {
    int *data;
    int size;
    int capacity;
} Vector;

void freeMem(Vector *v)
{
    free(v->data);
    free(v);
}

Vector *insert(Vector *v, int numb)
{
    if (v->capacity >= v->size) {
        v->data = realloc(v->data, (v->size + 1) * sizeof(int));
        v->size++;
    }
    *(v->data + v->capacity++) = numb;
    return v;
}

int main()
{
    int x;
    Vector *vector = (Vector *) malloc(sizeof(Vector));
    if (vector == NULL){
        fprintf(stderr, "Failed to allocate memory\n");
        return EXIT_FAILURE;
    }
    vector->size = 0;
    vector->capacity = 0;SS
    vector->data = (int *) calloc(0, sizeof(int));
    if (vector->data == NULL){
        fprintf(stderr, "Failed to allocate memory\n");
        return EXIT_FAILURE;
    }

    printf("Start:\n");
    printf("Vector has %d spaces and %d items stored\n", vector->size, vector->capacity);

    printf("Insert number to the vector:");
    printf("Insert number to the vector:");
    scanf("%d", &x);
    vector = insert(vector, x);
    printf("Vector has %d spaces and %d items stored\n", vector->size,vector->capacity);
    printf("The value at index %d is %d\n", 0, vector->data[0]);
    printf("Insert number to the vector:");
    scanf("%d", &x);
    vector = insert(vector, x);
    printf("Vector has %d spaces and %d items stored\n", vector->size,vector->capacity);
    printf("The value at index %d is %d\n", 1, vector->data[1]);
    freeMem(vector);
    return 0;
}
```