**TÖL309G Tölvutækni og forritun**

**Verkefni 1**

**Hjörvar Sigurðsson**

1. **Þrep 1:**

i. Ég notaði skipunina
```
objdump -d bomb
```
og fann main fallið í textaskjalinu.

ii. Ég rak mig í gegnum main fallið þar til ég fann kall á <phase_1> fallið.

iii. Í phase_1 fallinu má finna kall á fallið <strings_not_equal>. Næstu tvær skipanir á eftir kallinu eru
```
test %eax, %eax
jne 1624 <phase_1+0x1d>
```
Þetta þýðir að ef að skilagildi fallsins <strings_not_equal> != 0, þá stekkur forritið í línu 1624.

iv. Lína 1624 segir eftirfarandi:
```
call   1bdf <explode_bomb>
```

v. Þar með veit ég að í þrepi 1 er borið saman streng úr inntaki við einhvern streng, og ef þeir eru ekki eins, þá springur sprengjan.

vi. Næst athugaði ég hvaða strengir eru í boði. Ég gerði það með því að nota skipunina
```
strings bomb
```
til að fá lista yfir alla strengi forritsins.

vii. Einn strengurinn var augljóslega frábrugðinn öllum hinum. Ég giskaði því á að hann væri lausnarstrengurinn en svo reyndist vera. Sprengjan aftengdist og þrep 2 hófst.

Lausnarstrengur:

**Why make trillions when we could make... billions?**

2. **Þrep 2:**

i. Ég byrjaði að rekja mig í gegnum fallið <phase_2>.

ii. Fyrsta kallið í því á annað fall er
```
call   1c0b <read_six_numbers>
```
Strax taldi ég því líklegt að lausnarstrengurinn væri einhverjar sex tölur.

iii. Ég rak mig í gegnum fallið <read_six_numbers>.

iv. Fallið <read_six_numbers> endar á eftirfarandi tveimur línum:
```
1c4a:       c3                  ret
1c4b:       e8 8f ff ff ff      call   1bdf <explode_bomb>
```
Þar með vissi ég að til þess að sprengjan springi þarf að stökkva í línu 1c4b.

v. Aðeins ofar mátti finna eftirfarandi línur:

```
1c41:  83 f8 05          cmp   $0x5,%eax
1c44:  7e 05             jle   1c4b <read_six_numbers+0x40>
```

Þar með vissi ég að ef að 5 <= %eax, að þá springur sprengjan.
Með því að rekja mig í gegnum <read_six_numbers> með hjálp gdb staðfesti ég það að fallið athugar hvort inntakið er minna en sex tölur, en þá springur sprengjan. Þetta var eftir langan tíma af því að skrifa tölurnar á forminu

```
123456
```

en ekki

```
1 2 3 4 5 6
```

og furða mig á af hverju <read_six_numbers> sprengdi mig sífellt...

vi. Ég fór aftur í <phase_2> fallið.
Næsta lína eftir kallið á <read_six_numbers> er eftirfarandi:

```
164d:  83 3c 24 01       cmpl  $0x1,(%rsp)
1651:  75 0a             jne   165d <phase_2+0x32>
```

Semsagt, ef að efsta gildi á staflanum er ekki 1, þá stekkur forritið yfir í línu 165d, sem er kall á fallið <explode_bomb>.
Þar með vissi ég að fremsta tala í sex tölu strengnum hlýtur að vera 1.

vii. Ég byrjaði núna að prufa að breyta tölunum og rekja mig í gegnum <phase_2> fallið með gdb:

Strengur:
3 1 9 5 8 2 – sprakk við fyrsta tékk
1 3 9 5 8 2 – stóðst fyrsta tékk, sprakk við annað tékk (verið að bera saman síðustu tölu við 3? Prufa.
1 3 9 5 8 3 – sprakk aftur við annað tékk. Prufa að breyta annari tölu.
1 4 9 5 8 3 – stóðst annað tékk. Stóðst líka þriðja tékk. Sýnist að 9 var rétt vegna þess að 4+4+1 = 9.
1 4 9 5 8 3 – sprakk við fjórða tékk. Sýnist forritið athuga hvort að næsta tala í strengnum, lögð saman við 4, jafngildi 16. Prufa það næst.
1 4 9 12 8 3 – sprakk við fjórða tékk.

viii. Þegar hér var komið við sögu sá ég mynstrið í strengnum. Ég sá að forritið var að athuga hvort strengurinn sé 1 4 9 16 25 36, eða það sem jafngildir eftirfarandi kóðabút:

```
func(int[] inntaksstrengur) {
    if (inntaksstrengur.length() <= 5) {
        sprengja();
    else {
        for (int i = 0; i < 6; i++) {
            if (inntaksstrengur[i] != i^2) {
                sprengja();
            else {
                phase3();
            }
```

```
                }
            }
        }
```

Ég prófaði strenginn og forritið samþykkti hann.

Lausnarstrengur:

**1 4 9 16 25 36**

3. **Þrep 3:**

i. Ég rak mig í gegnum fallið <phase_3>.
Ég sá fyrst að fallið gerir nokkrar aðgerðir á gistum, og ber svo saman innihald gisti %eax við töluna 1, og sprengir sprengjuna ef 1 <= %eax.
Hér er fallið að athuga hvort inntakið sé minna en, eða jafnt og, 1, og sprengir sprengjuna ef svo er.

ii. Næst ber það saman vistfangið sem geymt er í gisti $rsp ( ($rsp) í objdump skjalinu) við töluna 7, og sprengir sprengjuna ef 7 > ($rsp) (hér notar það unsigned comparison).

iii. Næst heldur fallið áfram að gera aðgerðir á ýmsum gistum, en svo kemur eftirfarandi:

```
 16e8:  3e ff e0              notrack jmp *%rax
 16eb:  e8 ef 04 00 00        call   1bdf <explode_bomb>
```

Fallið stekkur semsagt einhvert sem tilgreint er inni í vistfangi %rax.

iv. Ég hélt áfram að rekja mig í gegnum fallið með inntakið

```
1 2
```

þar til ég kom að eftirfarandi:

```
16f7:  39 44 24 04           cmp    %eax,0x4(%rsp)
16fb:  75 52                 jne    174f <phase_3+0xaf>
```

Hér leggur fallið 4 við það sem vistfangið sem er geymt í %rsp, og ber það saman við %eax. Ef að breyturnar eru ekki jafnar, þá stekkur fallið í línu 174f, en það er fallið <explode_bomb>.

Þegar inntakið er

```
1 2
```

þá var hér %eax = 815, og vistfangið í %rsp var 0x7fffffffddb0, en í því var geymd talan 1.
Þegar ég legg fjóra við vistangið, þá verður það 0x7fffffffddb4, en í því vistfangi var geymd talan 2.
Þar sem breyturnar voru ekki jafnar fór fallið í línu 174f og sprengjan sprakk.
Þar með gat ég talið líklegt að tölurnar tvær í inntakinu þyrftu að vera sama talan.

v. Ég prufaði aftur með inntakið

1 1

og aftur sprakk sprengjan á sama stað.
Ég taldi að það sem fallið sé að gera hér er að athuga hvort annað gildið í inntakinu (eða 4 bitum á eftir %rsp, en í %rsp er fyrsta gildið í inntakinu) sé 815. Ég prófaði því að hafa annað gildið í inntakinu 815.
Það virkaði og ég komst fram hjá þar sem sprengjan sprakk síðast.
Inntakið var því hér

1 815

Eftir þetta þá kláraðist þrep 3, og lausnarstrengurinn því:

**1 815**

**4. Skjáskot af keyrslu:**