Structuur van Computerprogrammas II: Week 13

Elisa Gonzalez Boix Maarten Vandercammen Robbe De Greef

Op Canvas vind je een zip bestand `code.zip` dat enkele codebestanden bevat die je in dit WPO nodig zal hebben.

Executiemodel en buffer overflows

1. Hoe kan je de waarde van de variabele `a` in de functie `oef1` uit `oefeningen.c` in `code.zip` aanpassen zonder de variabele zelf te herassignen?

2. In de functie `oef2` uit `oefeningen.c` wordt er een simpel login-systeem van een applicatie gedefinieerd. In die functie wordt aan de gebruiker gevraagd om een paswoord in te geven. Deze string wordt dan vergeleken met een paswoord dat gekend is door het systeem en opgeslagen is als de variabele `password`. Probeer nu om tijdens het inloggen `password` te overschrijven zodat je kan inloggen zonder het echte paswoord te kennen.

   Als je een `stack smashing detected` error krijgt, kan je die verhelpen door het programma te hercompileren met de flag `-fno-stack-protector`.

3. In de `new_stackmachine` folder binnenin `code.zip` vind je een uitbreiding op de stackmachine die je vorige week gezien hebt: met deze uitbreiding is het mogelijk om SmiLang functies te schrijven waar je arrays kan definiëren, als argumenten kan meegeven, en kan opvullen met getallen die ingelezen worden uit de console.

   Aangezien het return-adres van SmiLang functies op dezelfde stack wordt bewaard, kan je een zogenaamde *buffer overflow attack* uitvoeren waarbij je dit return-adres overschrijft door de juiste getallen in de console in te geven.

   Er is aan de stackmachine ook een extra SmiLang programma toegevoegd waar twee functies worden gedefinieerd (`printFoo` en `printBar`. Het SmiLang programma dat door de stackmachine wordt uitgevoerd zal een array van lengte 2 aanmaken, deze opvullen met user input en daarna `printFoo` aanroepen. Zorg ervoor, door enkel de juiste user input te geven, dat niet `printFoo` maar wel `printBar` wordt opgeroepen.

   Je kan gebruik maken van het script `run.sh` om de stackmachine uit te voeren, en van de executable `print_to_input` om op een meer praktische manier user input door te geven aan de stackmachine.

De executable `print_to_input` oproepen als `./print_to_input <command_line_args>` (waar `<command_line_args>` een variabel aantal getallen in decimale of hexadecimale vorm zijn), komt overeen met elk van deze command-line argumenten in te geven als user input aan de stackmachine.

Goto

4. (**Dodona**) Herimplementeer de functie `int fac(int n)` uit het eerste WPO. Maak deze keer echter geen gebruik van recursie of van de klassieke control-flow structuren, zoals `for, while` of `do while`, maar maak enkel gebruik van `goto`.

Trampolines

5. (**Dodona**) In week 2 heb je een staartrecursieve functie `ggd` geschreven waarmee je de grootste gemeenschappelijke deler van twee getallen kan berekenen. Pas nu deze functie aan zodat die gebruikt maakt van een trampoline. Je vindt de implementatie van trampolines zoals gezien tijdens het hoorcollege in `oefeningen.c`.

Recap

6. (**Dodona**) In `oefeningen.c` bevindt zich een implementatie van een `Person` struct, dat allerlei informatie bijhoudt over een persoon. In `code.zip` bevindt er zich ook een tekstbestand `person_info` waar informatie van verscheidene personen wordt bijgehouden. Elke 5 volgende regels stellen, in volgorde, het volgende voor: de leeftijd, burgerlijke staat, opleidingsniveau, tewerkstelling (`0` of `1`), en in welk gewest de persoon woont.

   Schrijf een functie `Person **read_n_people(int n, FILE *file)` die een array van `n` personen teruggeeft, waar elk element in de array een pointer is naar een `Person` die de juiste informatie, ingelezen uit `file`, bijhoudt. Zorg er op het einde van het programma ook voor dat al het dynamisch gealloceerd geheugen weer wordt vrijgegeven.

Extra

7. (Deze oefening zou mogelijk moeten zijn op de meeste Linux varianten, maar waarschijnlijk niet op OSX of Windows)

   Voer een buffer overflow attack uit op een gewoon C programma. Lees hiervoor de instructies op `https://dhavalkapil.com/blogs/Buffer-Overflow-Exploit/`. De functies `echo` en `secretfunction` bevinden zich al in `oefeningen.c`.

Als je `gcc` gebruikt als compiler, zal je naast de compiler flag voor het compileren naar een 32-bit architectuur waarschijnlijk ook de volgende flags moeten meegeven om moderne compiler- en linker-level beveiligingen uit te schakelen: `-fno-stack-protector -m32 -z execstack -no-pie`

Om met gcc een programma te compileren voor een 32-bit architectuur, zal je waarschijnlijk de `gcc-multilib` library moeten installeren. Op Ubuntu kan je dat doen via het commando `sudo apt-get install gcc-multilib`