Reaktive Sicherheit

Übungsblatt 3

Balduin Binder [s6babind@uni-bonn.de] Charlotte Mädler [s6chmaed@uni-bonn.de]
Bünyamin Sarikaya [s6busari@uni-bonn.de]

Aufgabe 2

1

Der Virtuelle Adressraum hält virtuelle Pages, die auf physische Pages abbilden.Er ist druch die Adressgröße beschränkt (allgemein 64 bit). Er umfasst also: 0x0000 0000 0000 0000 bis 0xFFFF FFFF FFFF FFFF (Nutzerprozesse nur 47 Bit: also bis 0x0000 7FFF FFFF FFFF)

2

rax,rbx,rcx,rdx,rbp,rsp,rsi,rdi,r9,r10,r11,r12,r13,r14,r15

3

- rbp ist der Stackframepointer/Basepointer. Er wird von der akutell ausgeführten Funktion benutzt und zeigt den Anfang des Stackframes.
- rsp ist der Stackpointer. Er zeigt auf das ektuelle Ende des Stacks. So kann man den Stack adressieren. Durch push oder pop wird er geändert.
- rip ist der Instructionpointer und dient als PC (Programm Counter). Er zeigt den Befehl an der als nächstes ausgeführt werden soll. Nach dem lesen wird er inkrementiert (abhängig vom Befehl, Bsp. bei Sprüngen, zur Sprungadresse).

4

16 Bytes, also jede Adresse auf dem Stack sollte restlos durch 16 teilbar sein.

5

Prologue

```
push    ebp ; pusht den Stackframpointer auf den stack
mov ebp, esp; setzt den Stackframpointer auf den aktuellen

sub esp, N  ; Erweitert den Stack um N bytes für lokale Variablen
```

Epilog

```
mov esp, ebp; Tut den Stack pointer Zurück an die Position wo er vor dem
        ; Funktionsaufruf war
pop ebp ; stellt den Stackframe der aufgerufenen Funktion wieder her
ret     ; kehrt zurück zur aufrufenden Funktion
```

6

Die Übliche calling convention nutzt folgende Reihenfolge: rdi, rsi, rdx, rcx, r8, r9. In diesem Fall würde es also wie folgt aussehen: rdi = a, rsi = b, rdx = c und rcx = d

hierbei seht "=" dafür, dass das Register die Werter der entsprechenden Variablen vor dem Aufruf der Funktion beinhalten muss

7

Die ersten 6 werden wie in der Vorherigen Teilaufgabe behandelt. Ab dem 7. Integer Paramter müssen sie in umgekehrter Reihenfolge vorm Aufruf auf den Stack gepusht werden.

Aufgabe 4

1

In den folgenden Beispielen wächst der Stack nach unten.

Stack Vor dem Einlesen von gets()

Adresse	Adresseninhalt	Zeile im Sourcecode
10	'F'	5
9	password[7]	6
8	passwort[6]	6
7	passwort[5]	6
8	passwort[4]	6
7	passwort[3]	6
6	passwort[2]	6
5	passwort[1]	6
4	passwort[0]	6

Stack nach dem Einlesen 'overflowT'

Adresse	Adresseninhalt	Zeile im Sourcecode
10	'T'	8
9	'w'	8
8	'o'	8
7	'l'	8
8	'f'	8
7	'r'	8
6	'e'	8
5	'v'	8
4	'o'	8

2

`Beispielstring: 345yG4M3T`

Die Eingabe muss aus 8 chars gefolgt von einem "T" bestehen. z.B. "aaaaaaaaT", "bbbbbbbbT","abcdefghT". Dadurch, dass gets() nicht prüft wie groß die Eingabe ist können wir an die Stelle im Speicher schreiben, die gar nicht mehr für das Passwort gedacht ist. Direkt hinter dem `password` Array ist die Speicheradresse, in der

die `passok` Variable gespeichert wird. Das Programm überprüft, ob diese Variable "T" ist. Das Programm setzt diese Variable erst auf "T", wenn der Benutzer das richtige Passwort eingibt. Da wir nun aber wissen, dass `passok` direkt hinter dem Array im Speicher liegt und `gets()` nicht überprüft, wie lang unsere Eingabe ist können wir die Variable einfach überschreiben indem wir der `gets()` Funktion 8 chars direkt gefolgt von einem "T" geben. Hierdurch werden die 8 chars an die entsprechenden Stellen im `password` Array geschrieben und die `passok` Variable mit "T" **überschrieben** und wir bekommen "Willkommen!" angezeigt.