Buffer Overflow on x86-64 With Hello World Shellcode

Etash Tyagi

1 Requirements

- **Machine:** Linux x86-64
- **ASLR:** Turned off using '**sudo sysctl -w kernel.randomize_va_space=0**'
- **Victim:** Must have stack smashing protection turned off.
- **Programs:**
  - *python3:* **sudo apt-get install python3**
  - *gdb:* **sudo apt-get install libc6-dbg gdb valgrind**
  - *nasm:* **sudo apt-get install nasm**
- Make all files executable if not already, using **sudo chmod +x ***.

2 Files

- **victim-exec-stack:** This is a binary executable which can be used to test the shellcode by buffer overflow.
- **Makefile** Used to compile, run and exploit [see comments for more info]. Commands:
  - *make compile_shellcode:* used to compile shellcode
  - *make run:* used to run shellcode
  - *make exploit:* used to exploit **victim-exec-stack** using **bo_generator.py**
- **get_rbp.sh $1 $2:** Shell script to get RBP register of $1 binary and $2 function using gdb.
- **get_shellcode.sh $1:** Shell script to get hex shellcode of the given binary.
- **bo_generator.py $1 $2 $3 $4:** Used to output buffer overflow payload for binary $1, function $2, with shellcode $3 to file $4. [see comments for more details]

3 Working

Following diagram demonstrates how this attack works, please see **bo_generator.py**'s comments to understand better. NOP has been included to counter stack space taken by gdb environment variables.

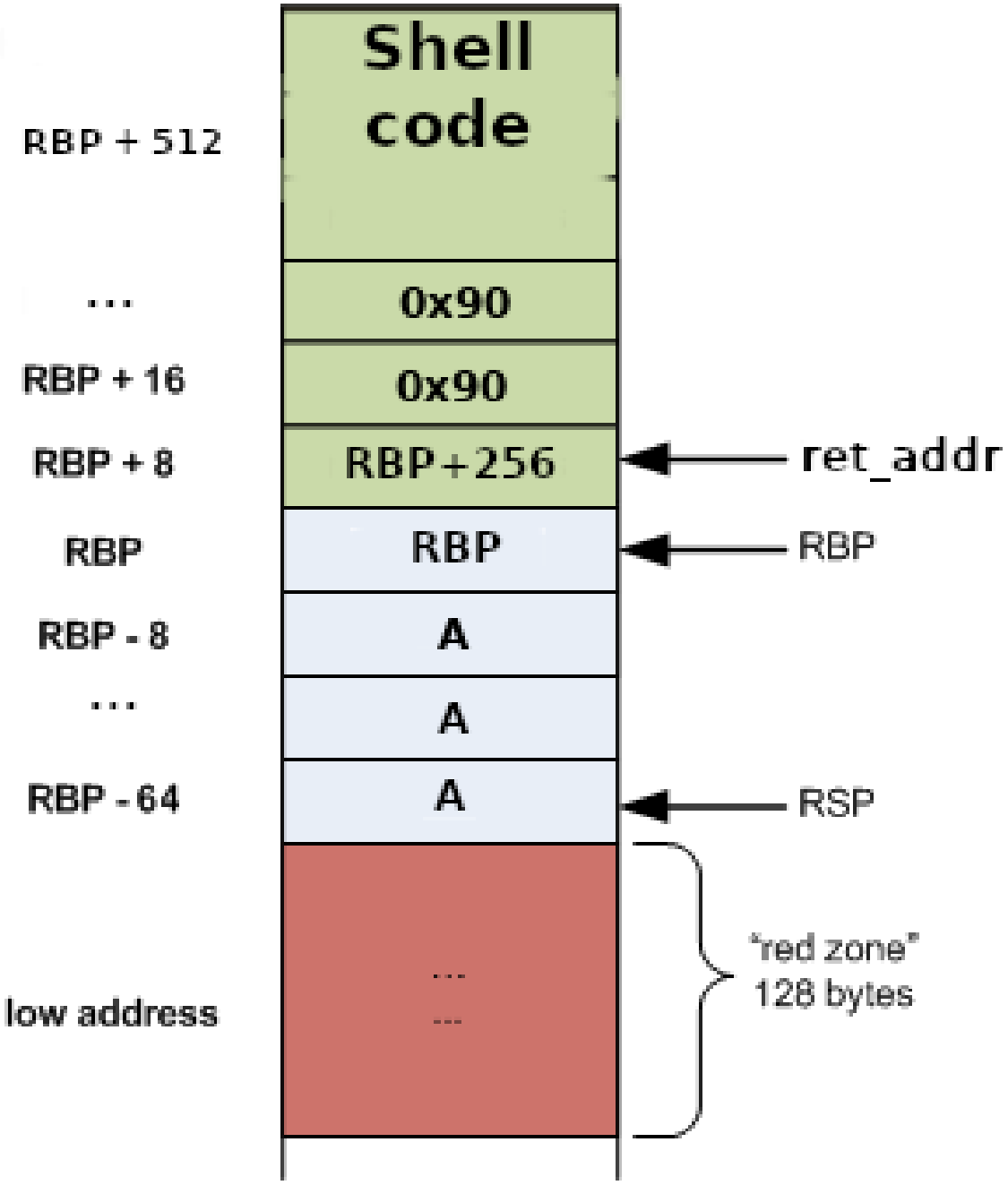

Figure 1: Buffer diagram

4 Demonstration

Figure 2: Without buffer overflow

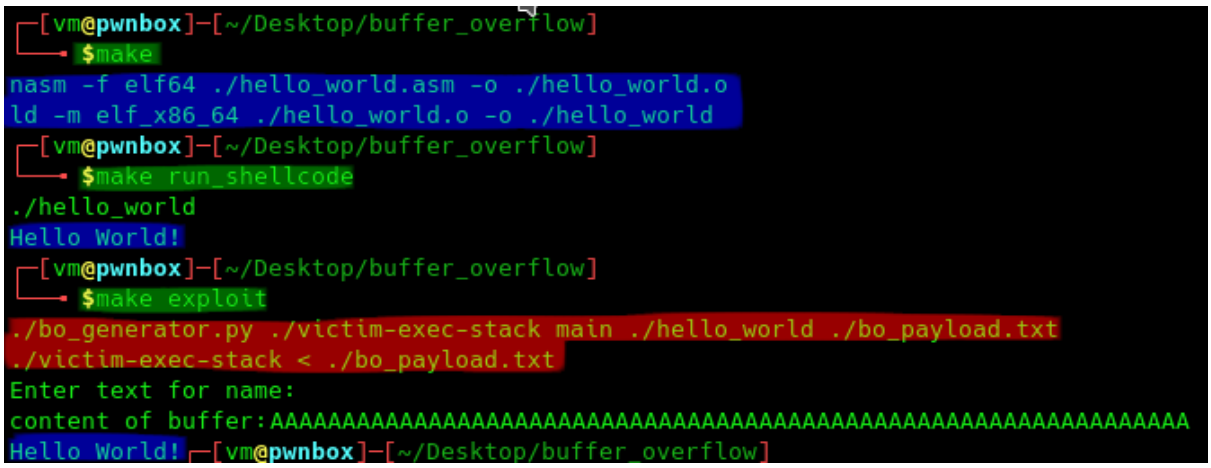

Figure 3: With buffer overflow