Shellcode

S9.1. The essence of a shellcode (32-bit) is to prepare four registers, `eax`, `ebx`, `ecx`, and `edx`, before invoking the `execve()` system call. Please describe what values these four registers should contain.

S9.2. In the stack-based approach, we need to store command string in the memory, and then save the string's address in `ebx`. Please write a code snippet (32-bit) to store the string "aaaabbbbccccdddd" in the memory, and then save its address to `ebx`.

S9.3. In the stack-based approach, we need to store the argument array `argv[]` in the memory, and then store the array's address in `ecx`. Please write a code snippet (32-bit) to construct the following `argv[]` array in the memory, and then assign its address to `ecx`.

```
argv[0] = 0x11111111
argv[1] = 0x22222222
argv[2] = 0x33333333
argv[3] = 0x00000000
```

S9.4. Compared to the stack approach, what is the main difference of the code segment approach in writing shellcode?

S9.5. The following shellcode is incomplete. You need to replace all the ⋆'s with actual numbers. Please also add a brief comment to each line marked by a circled number to explain its purposes. You cannot just describe the meaning of each instruction (such as saying "`pop eax`" is to take out a value from the stack and store it to `eax`). You need to explain its purpose, i.e., why the instruction is needed there.

```
section .text
  global _start
    _start:
        BITS 32
        jmp short two
    one:
        pop ebx              ①
        xor eax, eax
        mov [ebx+⋆],  al     ②
        mov [ebx+⋆],  ebx    ③
        mov [ebx+⋆], eax     ④
        lea ecx, [ebx+⋆]     ⑤
        xor edx, edx
        mov al,  0x0b
        int 0x80
     two:
        call one
        db '/bin/shabcde'    ⑥
        db 'AAAA'           ⑦
        db 'BBBB'           ⑧
```

S9.6. Please replace the question marks in the following 32-bit shellcode with concrete numbers. Please also briefly explain how you get the numbers.

```
section .text
  global _start
    _start:
        BITS 32
        jmp short two
    one:
        pop ebx
        xor eax, eax
        mov [ebx+?],  al
        mov [ebx+?],  ebx
        mov [ebx+?],  eax
        lea ecx, [ebx+?]
        xor edx, edx
        mov al,  0x0b
        int 0x80
     two:
        call one
        db 'AAAA'
        db 'BBBB'
        db '/bin/sh*'
```

S9.7. The following shellcode is incomplete. Its goal is to execute the following command: `"/bin/rm -rf *"` (without the quotations). Part of the code is given, with some helpful information. Please complete this code. Your code should be well commented or you will lost points.

```
  _start:
      jmp short two
  one:
      ... add code here ...

      mov al,  0x0b   ; invoke execve() system call
      int 0x80
  two:
      call one
      db 'abcd/bin/rmab-rf****'
      db 'AAAABBBBCCCCDDDDEEEEFFFFGGGG'
```

S9.8. Why does shellcode in general not allow zeros in the code?

S9.9. Please list three typical solutions to get rid of zeros in shellcode.

S9.10. We would like to store a string ”ab” on the stack, but we are not allowed to include any zero in the code (the end of the string has a binary zero). (1) Please complete the code for a little endian machine. (2) Please complete the code for a big endian machine.

```
mov ecx, "ab⋆⋆"
... (missing code) ...
push ecx
```

S9.11. ★
Please store `0xAA00BB00` in the `eax` register. You cannot have any binary zero in the final machine code.