Shellcode writeup

Ethical Hacking 2022/23, University of Padua

Eleonora Losiouk, Alessandro Brighente, Gabriele Orazi, Francesco Marchiori

1 Task 1

In order to execute a `.s` file you have to run this commands (as reported in the readme):

```
$ nasm -f elf32 yourfile.s -o yourfile.o
$ ld -m elf_i386 yourfile.o -o yourfile
$ ./yourfile
```

1.1 Task 1.a

Just follow the readme all along using the provided commands.

1.2 Task 1.b

- **line 5-6**: pushing the string terminator using the xor technique (first bullet point of the hints). This works since exclusive or always result in a 0 byte! It should be also faster since less operations under the hood are involved.
- **line 20-21**: pushing 0 with the previous `eax` register and then update the lsb `al` with the `b` value (second bullet point of the hints)

In order to use `/bin/bash` instad of `/bin/sh`, avoiding the usage of multiple `/` it is possible to take advantage of the third bullet point of the hints, which basically use a shift-left-right technique to fill the 4-byte string with the necessary zeros. In the example, a simple `h` was missing since `/bin/bash` is 9 chars and we have to push in the stack 4 chars per time. What has been done is to use the string `h###` and replace the `#` with zeros. You can find the solution in script `mysh1b.s`. Here the snippet used:

```
    mov ebx, "h###"
    shl ebx, 24
    shr ebx, 24
    push ebx
```

1.3 Task 1.c

Solution is in the file `mysh1c.s`. We are using `eax`, `ebx`, `ecx` and `edx` to construct piece by piece the entire `argv`. Same techniques of before are applied. The shift technique is used to fill up the gaps. We create one argument per time using an "app" register each and then we store the pointer in the same register. At the end, we compose the `argv` using the pointers saved before.

1.4 Task 1.d

Solution is in the file `myenv.s`. We're taking advantage of some usable registers in order to create the `env[]` array. `esi` and `edi` are used in addition of `eax`, `ebx`, `ecx` and `edx`.

At the end of the string preparation, registers have to point to: - `ebx` to the `/usr/bin/env` string - `ecx` to the `argv[]` array - `edx` to the `env[]` array

The two new registers are necessary because of the preparation of the `env[]` array, which contains 3 elements (strings) plus the terminator `0` (always contained in `eax`).

The `cccc=1234` element can be composed like it was done before, using `#` to fill up the string until 4 chars and then shift left right to obtain zeros.

Here a reference to check how `execve` works.

```
#include <unistd.h>

int execve(const char *pathname, char *const argv[],
            char *const envp[]);
```

If you try to reuse the `ebx` register, you will get a `Segmentation fault` since the register needs to cointain `/usr/bin/env`. It doesn't matter that you already load the pointer to the string into `argv[]`.

Since this is an exeptional case, you can reuse `edx`. In fact, in the solution script the register is usede to create the `aaa=1234` string and then to contain the pointer to `env[]` array.

---

2 Task 2

1. Here an explanation of the code line per line:

```
    pop ebx                 ; extract in ebx from the stack the pointer
                            ; to the last     line of the string, namely the
                            ; instruction where to return once the function is ended
    xor eax, eax            ; fill eax with 0x00
    mov [ebx+7], al         ; string = '/bin/sh0OAAABBBB'
    mov [ebx+8], ebx        ; string = '/bin/sh0/binBBBB'
    mov [ebx+12], eax       ; string = '/bin/sh0/bin0000'
    lea ecx, [ebx+8]        ; ecx = ebx + 8 = '/bin'
    xor edx, edx            ; fill edx with 0x00, no env[] variables
    mov al,  0x0b           ; invoke execve (1)
    int 0x80                ; invoke execve (2)
two:
    call one                ; jump to the one function, pushing the pointer of the
                            ; next line into the stack
    db '/bin/sh*AAAABBBB'   ; place the string in this position in the executable
```

2. You can find the solution in the `task2.s` file.