Pwn-ing Linux x86/x64
Zadania kwalifikacyjne

Jakub Szewczyk — kubasz51@gmail.com
Grzegorz Uriasz — gorbak25@gmail.com

2 czerwca 2017

Ostateczną wersję rozwiązań zadań proszę przesłać mailowo do nas w formacie pdf, w terminie podanym na stronie WWW. Zachęcamy do wcześniejszego przesłania nam rozwiązań zadań na nasze adresy mailowe – będziemy wtedy w stanie powiedzieć, co jest źle i podpowiedzieć jak dopracować finalną wersję. Nie wymagamy wykonania dużej ilości zadań – większa ich ilość wymagana będzie dopiero przy dużej ilości uczestników. Tak więc wyślij nam to co udało ci się wykonać. Jeżeli coś nie jest jasne, to napisz do nas – z chęcią nakierujemy na literaturę bądź wytłumaczymy. Kody źródłowe programów załączonych w tych zadaniach kwalifikacyjnych są dostępne pod poniższym linkiem i qr-codem.

https://goo.gl/9tYrH1

1 Korzystanie z Windowsa

Nie polecamy korzystania z Windowsa, ponieważ na naszych warsztatach skupiamy się na exploitacji programów działających na Linuxie. Dlatego zachęcamy do zainstalowania dystrybucji Linuxa na swoim komputerze w konfiguracji dual boot(wybór systemu przy starcie komputera), np. Linux Mint, Antergos, Ubuntu. Inną opcją jest zainstalowanie modułu WSL na Windowsie, znanego też jako Bash on Windows - symuluje on działające jądro Linuxa na Windowsie 10, jest oficjalnym modułem Microsoftu i powinien wystarczyć do naszych zadań. Instrukcje jak go zainstalować są tu: `https://msdn.microsoft.com/en-us/commandline/wsl/install_guide`. Polecamy wcześniej zaktualizować system do wydania Creator's update, WSL został tam znacznie ulepszony w stosunku do Anniversary Update.
Potrzebne będzie zainstalowanie pakietów w systemach Ubuntu, Mint, bądź WSL komendą: `sudo apt install libssl-dev gcc-multilib yasm`

2 Dlaczego chcesz uczestniczyć w tych warsztatach? - Obowiązkowo

Odpowiedz na to pytanie ściśle i zwięźle. Nie jest to zadanie typu "Gdybyś był owocem, to jakim? Uzasadnij twoje stanowisko na minimalnie dwudziestu stronach A4" - oczekujemy tylko krótkiego opisu.

3 Rozgrzewka - reprezentacja binarna liczb zmiennoprzecinkowych

Napisz program, który wykonuje dwie operacje (język programowania dowolny, prosimy o kod źródłowy):

1. Wczytuje liczbę zmiennoprzecinkową, a następnie wypisuje jej reprezentację bitową w pamięci komputera jako liczbę zmiennoprzecinkową podwójnej precyzji(double) oraz pojedynczej precyzji(float).

2. Wczytuje reprezentacje binarną liczby zmiennoprzecinkowej podwójnej precyzji w formacie litle-endian(tak jak w poprzednim podpunkcie na PC-tach się wypisze) oraz dla niej wypisuje odpowiadającą jej wartość liczbową w systemie dziesiętnym.

Przykładowe wywołanie programu może przebiegać następująco:

```
grzegorz@grzegorz-arch ~/WWW13-KWA ./1                    ✓ 10832 01:11:29
Enter a number: 2.71828
Float: 10110010000111111011010000000010
Double: 0000100111101111010101011010100110010000111111011010000000000010
Enter a number reprezentation: 0010111101101011110000100010101011011111100001001001000000000010
This number is: 3.1415926535800000607
grzegorz@grzegorz-arch ~/WWW13-KWA
```

4 Assembler – Instalacja i Przetestowanie

Polecamy assembler yasm (`http://yasm.tortall.net/Download.html`), ale zadziała także nasm, fasm, lub jakikolwiek inny, jeżeli znasz jego składnię. Zainstaluj assembler, zassembluj podany poniżej program i go uruchom na Linuxie bądź WSLu, pokaż screenshoty. Nie przejmuj się, jeżeli nie rozumiesz tego kodu – chodzi tylko o sprawdzenie narzędzia, podstawy assemblera wyjaśnimy na początku warsztatów.

4.1 Wersja 32-bitowa - nie działa pod WSL

```
;; hello.asm
segment .data
    msg     db      "Hello WWW13!", 0Ah

segment .text
        global  _start
_start:
 ; write
    mov     eax, 4
    mov     ebx, 1
    mov     ecx, msg
    mov     edx, 13
    int     80h
 ; exit
    mov     eax, 1
    xor     ebx, ebx
    int     0x80
```

Komenda do zassemblowania: `yasm -f elf32 -o hello.o hello.asm`
Komenda do zlinkowania: `ld hello.o -o hello`
Komenda do uruchomienia: `./hello`

4.2 Wersja 64-bitowa - działa pod WSL

```
;; hello64.asm
segment .data
    msg     db      "Hello WWW13!", 0Ah

segment .text
        global  _start
_start:
```

```
; write
   mov     rax, 1
   mov     rdi, 1
   mov     rsi, msg
   mov     rdx, 13
   syscall
; exit
   mov     rax, 60
   xor     rdi, 0
   syscall
```

Komenda do zassemblowania: `yasm -f elf64 -o hello64.o hello64.asm`
Komenda do zlinkowania: `ld hello64.o -o hello64`
Komenda do uruchomienia: `./hello64`

5 SSH – Instalacja i Przetestowanie

Zainstaluj SSH (zobacz opis warsztatów) i prześlij nam zrzut ekranu, co się wyświetli jak połączysz się z github.com na koncie git (komenda: `ssh git@github.com`, należy zaakceptować certyfikat githuba). Jeżeli wyświetli się permission denied, należy sprawdzić czy wygenerowało się swój klucz oraz czy dodało się go do konta na githubie (które warto założyć, bo nawet jeżeli na tych warsztatach się nie przyda, to później na pewno).

6 Operacje logiczne

Odpowiedz na pytania i rozwiąż poniższe równania.

6.1 W jaki sposób można odejmować używając dodawania?

Dodawanie na bramkach logicznych jest o wiele łatwiejsze do zrealizowania niż odejmowanie. A więc w jaki sposób używając operacji logicznych typu and, xor, or ... uzyskać odejmowanie dwóch liczb binarnych mając układ dodający dwie liczby binarne do siebie?
Porada: Poczytaj o reprezentacjach binarnych liczb ze znakiem.

6.2 Operacje bitowe

Masz do dyspozycji 8-bitową zmienną liczbową bez znaku o nazwie Adam. Zaproponuj działania dzięki których ustawisz i-ty bit Adama na 1 oraz j-ty bit Adama na 0. W jaki sposób sprawdzić czy n-ty bit Adama jest jedynką?

Oblicz na kartce(bez pomocy komputera) poniższe dwa działania:
((24 and 101) or (132 xor 241)) nor 5 = ?
(123 xor 153) xor (246 and 235) = ?

7 Błąd w kodzie C

Poniższy kod zawiera wywołanie funkcji system która otwiera powłokę systemową. Na pierwszy rzut oka dotarcie do tej linijki kodu jest niemożliwe ze względu na sprawdzenie, czy pole casual w strukturze state jest różne od 1. Otóż poniższy kod w języku C zawiera błąd który pozwala przejąć Tobie kontrolę nad polem casual. Twoim zadaniem jest przeanalizowanie kodu, znalezienie wspomnianego błędu i opisanie nam go w rozwiązaniu. W rozwiązaniu podaj przykładowe wejście które pozwala na wywołanie funkcji system oraz sposób naprawy tego błędu.

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct
{
        char tab[100];
        char casual;
} state;

state s;

int main()
{
        int ans, i;
        printf("How many times to loop: ");
        scanf("%d", &ans);
        if(ans<0 || ans>100)
        {
                printf("Invalid number\n");
                return 0;
        }

        s.casual = 1;

        for(i = 0; i<=ans; i++)
        {
                s.tab[i] = i;
        }

        if(s.casual != 1)
        {
                printf("YOU SHALL NOT CALL ME!\n");
                system("/bin/sh");
                return 0;
        }
        printf("No shell for you\n");

        return 0;
}
```

8 Obfuskacja kodu w C - Niewymagane, ale polecamy spróbować

Zapomniałem hasła do Facebooka! Na szczęście mam zapisane to hasło na komputerze, więc teraz fakt że mam jeden port ethernet na pewno mi nie przeszkodzi (nie wiem czemu ssh na serwerze z moją kopią zapasową wymaga komputera z aż 22 portami ethernet!). Problem w tym, że analizowałem nowego wirusa krążącego po Internecie i przez przypadek go uruchomiłem. Zaszyfrował mi wszystkie pliki! Dobra wiadomość jest taka, że udało mi się uzyskać fragment kodu źródłowego wirusa, który jest odpowiedzialny za sprawdzenie, czy hasło deszyfrujące jest dobre. Ja nie mogę przeanalizować tego kodu, bo mam zaszyfrowanego vim-a, więc potrzebuję twojej pomocy. Na szczęście kod został napisany w języku C, którego składnia jest najpiękniejsza na świecie i kody są najczytelniejsze na świecie, więc nie powinieneś mieć z jego analizą najmnniejszego problemu.

```c
//Obfu5c473d p455w0rd ch3ck3r
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <openssl/md5.h>
char *_____ = "\x66\x65\x64\x05\x06\x4d\x58\x53\x56\x5f\x4d\x00\x31\x17\x0b\
\x6b\x61\x6d\x08\x32\x25\x2c\x3a\x09\x17\x00\x24\x0c\x4d";
const unsigned char* ___ = "\x43\x2f\x45\xb4\x4c\x43\x24\x14\xd2\xf9\x7d\xf0\
\xe5\x74\x38\x18\x36\x04\x17\x76\x71\x22\x2a\x37\x76\x36\x3e\x46\x06\x0b\x77\
\x74\x28\x3b\x27\x77\x55\x4d";
char ____[100]; unsigned char __[MD5_DIGEST_LENGTH+1]; char _;
```

```
void main(){memset(____, 0, sizeof(____));printf("Enter The Password: ");
  scanf("%20s", ____);int _______ = strlen(____);int ______ = strlen(_____);
  for(((1&2)??(&_??))??'=((5&1)??(&_-('c'^'b')??));_<_______;)((_??(____??))??'=
  ((_%______)??(_____??)), (3??(&_-3??))++);MD5((const unsigned char*)____, _______,
   (__??(MD5_DIGEST_LENGTH??)=0,__));int ________ = strlen(_____+______+1)+3;
  for((6??(&_-6??))^=(7??(&_-7??)); _<________;)(____??(_??) = (_____??(_+______+1??)^
  _____??(_%______??))),(4[&_-4])++;________ = strlen(___+MD5_DIGEST_LENGTH);
  for((8??(&_-8??))^=(9??(&_-9??)); _<________;)(____??(_+strlen(____)+1??)=
  ((___+MD5_DIGEST_LENGTH)??(_??))^(_____??(_%______??))),(5??(&_-5??))++;
  for(((1&3)??(&_-1??))??'=(((2&4))??(&_??)); _<MD5_DIGEST_LENGTH;)(_??(___??)??'
  (_??(__??))?(printf(____),exit(0)):(2??(&_-2??))++);printf(____+strlen(____)+1);
  exit(0);}
```

Kompilacja programu odbywa się poleceniem:

```
gcc -o <nazwa pliku wyjsciowego> <kod zrodlowy> -l ssl -l crypto --trigraphs
```

Rozwiązaniem zadania jest:

1. Co zrobiłeś by zrozumieć powyższy kod?
2. W jaki sposób powyższy kod sprawdza hasło?
3. Podaj listę metod zaciemniania zastosowanych w powyższym kodzie.
4. Podaj hasło które po wczytaniu przez program jest indentyfikowane jako poprawne.

To nie jest wirus! Jeżeli w jakimś miejscu nie dajesz rady - napisz do nas - w zamian za przysłanie opisu co zrobiłeś do tej pory odsyłamy hinty.

Powodzenia i do zobaczenia w Wierchomli!