1 Kód

```
1  #ifndef _MSC_VER
2  #define _snprintf snprintf
3  #else
4  #include "stdafx.h"
5  #endif
6  #include <stdio.h>
7  #include <string.h>
8  #include <stdlib.h>
9
10 static void usage()
11 {
12         printf("usage: bek1 name\n");
13         printf("\n");
14         printf("bek1 prints your name and does a few other things.\n");
15         printf("Watch out for errors, there is a lot of them!\n");
16         exit(EXIT_SUCCESS);
17 }
18
19 static int CalcMagicNum(int nHi, int nLo)
20 {
21         int nResult;
22
23         if (nHi > 9)
24                 nResult = CalcMagicNum(nHi / 10, nHi % 10) + nLo;
25         else
26                 nResult = nHi + nLo;
27
28         if (nResult > 9)
29                 return CalcMagicNum(1, nResult % 10);
30
31         return nResult;
32 }
33
34 static int CalcMagicStr(const char* psz)
35 {
36         int nResult = 0;
37         char c = psz[0];
38
39         if (c != 0)
40         {
41                 c -= '0';
42                 nResult = c + CalcMagicStr(++psz);
43                 nResult = CalcMagicNum(nResult, 0);
44         }
45
46         return nResult;
47 }
48
49 static void encrypt(char* psz)
50 {
51         while (*psz)
52         {
```

```
53                  char c = psz[0];
54                  if (c >= 'A' && c <= 'Z')
55                          c = 'Z' - c + 'A';
56                  else if (c >= 'a' && c <= 'z')
57                          c = 'z' - c + 'a';
58                  else if (c >= '0' && c <= '9')
59                          c = '9' - c + '0';
60                  *psz++ = c;
61          }
62  }
63
64  int main(int argc, char* argv[])
65  {
66          const char* pszName;
67          char szName[32];
68          char szDay[16];
69          char szMonth[16];
70          char szYear[16];
71          char szComb[16];
72          char szBuffer[80];
73          char szMagic[16];
74          unsigned short sMagic = 0;
75          FILE* pf = NULL;
76
77          //--------------------------------
78          //  Check for the help argument
79          //--------------------------------
80          if (!strcmp(argv[1], "/?") || !strcmp(argv[1], "-?"))
81                  usage();
82
83          //------------------------------------------------
84          //  Check whether we already have user information
85          //------------------------------------------------
86          pf = fopen(argv[1], "rt");
87          if (pf != NULL)
88          {
89                  printf("Your name is: ");
90                  fgets(szName, sizeof(szName), pf);
91                  encrypt(szName);
92                  printf(szName);
93                  fgets(szDay, sizeof(szName), pf);
94                  encrypt(szDay);
95                  fgets(szMonth, sizeof(szName), pf);
96                  encrypt(szMonth);
97                  fgets(szYear, sizeof(szName), pf);
98                  encrypt(szYear);
99                  fgets(szMagic, sizeof(szName), pf);
100         encrypt(szMagic);
101         sscanf(szMagic, "%hd", &sMagic);
102         printf("\nYou were born on %s-%s-%s\n", szYear, szMonth, szDay);
103         }
104         else
105         {
106                 //--------------------------------
107                 //  Print the user's name first
108                 //--------------------------------
109                 pszName = argv[1];
110                 printf("Your name is: ");
111                 printf(pszName);
112                 printf("\n");
113
```

```
114                 //--------------------------------
115                 //  Get the user's birth details
116                 //--------------------------------
117                 printf("Enter your day of birth: ");
118                 scanf("%s", szDay);
119                 printf("Enter your month of birth: ");
120                 scanf("%s", szMonth);
121                 printf("Enter your year of birth: ");
122                 scanf("%s", szYear);
123 
124                 //-----------------------------------------
125                 //  Combine all 3 strings into one, securely
126                 //-----------------------------------------
127                 _snprintf(szComb, sizeof(szComb), "%s%s%s", szDay, szMonth, szYear);
128 
129                 //-----------------------------------------
130                 //  Calculate and print the "magic" number
131                 //-----------------------------------------
132                 sMagic = CalcMagicStr(szComb);
133 
134                 //---------------------------------------
135                 //  Write all encrypted data into a file
136                 //---------------------------------------
137                 pf = fopen(argv[1], "wt");
138                 encrypt(argv[1]);
139                 encrypt(szDay);
140                 encrypt(szMonth);
141                 encrypt(szYear);
142                 sprintf(szMagic, "%d", sMagic);
143                 encrypt(szMagic);
144                 fprintf(pf, "%s\n%s\n%s\n%s\n%d", argv[1], szDay, szMonth, szYear, sMagic);
145                 fclose(pf);
146         }
147 
148         printf("Your magic number is: %d\n", sMagic);
149 
150         return EXIT_SUCCESS;
151 }
```

1.1

```
72  char szBuffer[80]
```
Tento buffer se alokuje, ale není nikde v programu používán a volán. Zabírá tedy zbytečně místo.

1.2

```
51  while (*psz)
```
Neošetřená hodnota *psz, pokud se např. stane, že *psz=nullptr, nastává segfault.

1.3

```
41  c -= '0';
```
Funkce CalcMagicStr bere jako argument const char* psz, dále do c dosazuje psz[0], tedy první charakter. Od c se poté odečítá ’0’(v dec 48). Neověřuje se zde více, zdali hodnoty v c jsou validní a při zadání určitých hodnot dochází k přetečení hodnoty. Výsledek magického čísla je poté ukládán jako unsigned short(max. hodnota 65535). Např. pokud zadáme ’!’(v dec 33, menší než 48), dostáváme následující:

```
Your name is: random2
Enter your day of birth: !
Enter your month of birth: 2
Enter your year of birth: 2004
Your magic number is: 65529
```

1.4

```
42  nResult = c + CalcMagicStr(++psz);
```
Tato rekurze by mohla přeplnit zásobník, např. při velmi dlouhém vstupu. Zde ale může záviset i na způsobu kompilace, při standardní kompilaci s gcc nás poté při zadání dlouhého řetězce stack protector nejspíše zastaví. Pokud ale kompilujeme s -fno-stack-protector, může docházet k přeplnění zásobníku.

1.5

```
80  if (!strcmp(argv[1], "/?") || !strcmp(argv[1], "-?"))
```
Pokud nezadáme žádný argument na příkazové řádce, program zde padá na segfaultu, snažíme se přistupovat na nedefinovaný argument, viz.:

```
==385== Invalid read of size 1
==385==    at 0x4EE7C50: __strcmp_sse2_unaligned (strcmp-sse2-unaligned.S:24)
==385==    by 0x108C49: main (bek1.cpp:80)
==385==  Address 0x0 is not stack'd, malloc'd or (recently) free'd
==385==
==385==
==385== Process terminating with default action of signal 11 (SIGSEGV)
==385==  Access not within mapped region at address 0x0
==385==    at 0x4EE7C50: __strcmp_sse2_unaligned (strcmp-sse2-unaligned.S:24)
==385==    by 0x108C49: main (bek1.cpp:80)
==385==  If you believe this happened as a result of a stack
==385==  overflow in your program's main thread (unlikely but
==385==  possible), you can try to increase the size of the
==385==  main thread stack using the --main-stacksize= flag.
==385==  The main thread stack size used in this run was 8388608.
==385==
==385== HEAP SUMMARY:
==385==     in use at exit: 0 bytes in 0 blocks
==385==   total heap usage: 1 allocs, 1 frees, 4,096 bytes allocated
==385==
==385== All heap blocks were freed -- no leaks are possible
==385==
==385== For counts of detected and suppressed errors, rerun with: -v
==385== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
Segmentation fault (core dumped)
```

1.6

Ohledně vstupních a výstupních operacích bych kontroloval flagy jako ferror nebo feof, např. pokud má soubor, ze kterého čteme, nějakou určitou strukturu, po přečtení nějakého počtu dat můžeme očekávat EOF, nebo pokud nepřečteme dostatečné množství dat a narazíme na EOF, můžeme vypsat alespoň chybovou hlášku. Stejně, pokud nastane při čtení nebo zápisu při streamu nějaký error, uživatel programu by se měl dozvědět, že nastává chyba někde při I/O operaci. To samé při používání sscanf, je dobré kontrolovat, zdali se načetly všechny očekávané argumenty apod..

1.7

```
90                  fgets(szName, sizeof(szName), pf);
91                  encrypt(szName);
92                  printf(szName);
93                  fgets(szDay, sizeof(szName), pf);
94                  encrypt(szDay);
95                  fgets(szMonth, sizeof(szName), pf);
96                  encrypt(szMonth);
97                  fgets(szYear, sizeof(szName), pf);
98                  encrypt(szYear);
99                  fgets(szMagic, sizeof(szName), pf);
```

Od fgets na řádce 93 se nemění sizeof argument, např. `fgets(szDay, sizeof(szName), pf);` načítá 32 bytů do szDay o velikosti 16 bytů, stejně tak u dalších fgets.

1.8

Při zápisu do souboru magické číslo nešifruje pomocí funkce encrypt jako szDay, szMonth apod., resp. šifruje, ale do souboru se zapisuje sMagic. Při čtení ze souboru se volá na szMagic enrypt(), dostáváme tedy jiné magické číslo při čtení ze souboru. Dochází ke konfliktu, kdy při prvním výpočetu magického čísla se zadáním uživatelem na příkazové řádce vypisuje jiné magické číslo než při dalším čtení již ze souboru, viz.:

```
j@DESKTOP-F7I559N: ./a.out abc
Your name is: abc
Enter your day of birth: 12
Enter your month of birth: 4
Enter your year of birth: 2004
Your magic number is: 4
j@DESKTOP-F7I559N: ./a.out abc
Your name is: abc

You were born on 2004
-4
-12

Your magic number is: 5
j@DESKTOP-F7I559N:
```

1.9

V mainu několikrát načítáme vstup s paramterem %s, měli bychom ale dělat nějaké kontroly, zdali jsou data zadaná uživatelem vůbec validní pro činnost programu a funkcí. Validitu argumentů funkcí jako CalcMagicNum bychom pro modularitu a přenositelnost kódu také měli kontrolovat přímo v nich a ne v mainu.

1.10

```
110                 printf("Your name is: ");
111                 printf(pszName);
112                 printf("\n");
```

Tento způsob výpisu mi nepřijde zcela korektní, pokud používáme funkci printf, měli bychom definovat výstupní formáty. Útočník tohoto může zneužít a úmyslně program shodit nebo např. vypisovat hodnoty na zásobníku, viz.:

```
j@DESKTOP-F7I559N: ./a.out "%s %s %s %s"
Your name is: Your name is:  (null) (null) (null)
Enter your day of birth: 2
Enter your month of birth: 4
Enter your year of birth: 2004
Your magic number is: 3
j@DESKTOP-F7I559N: ./a.out "%s %s %s %s %s %s %s %s %s %s %s %s %s"
Segmentation fault (core dumped)
j@DESKTOP-F7I559N: ./a.out "%p %p %p %p"
Your name is: 0x7fffe93ea490 (nil) (nil) (nil)
Enter your day of birth: 2
Enter your month of birth: 4
Enter your year of birth: 2004
Your magic number is: 3
```

1.11

```
117                 printf("Enter your day of birth: ");
118                 scanf("%s", szDay);
119                 printf("Enter your month of birth: ");
120                 scanf("%s", szMonth);
121                 printf("Enter your year of birth: ");
122                 scanf("%s", szYear);
```

Zde pomocí scanf čteme string do bufferu s omezenou velikostí, nikde se ale nekontroluje, zdali se zadaný vstup do bufferu vejde.

1.12

```
87  if (pf != NULL)
```

Pro tuto větev v programu chybí ukončení streamu ze souboru, je potřeba dodat fclose(pf).

1.13

```
132  sMagic = CalcMagicStr(szComb);
```

sMagic je typu unsigned short, kdežto CalcMagicStr vrací int, oba tyto typy mají různé rozsahy, hodnota se při dosazení nemusí vejít, problém se zápornými hodnotami.

1.14

```
101  sscanf(szMagic, "%hd", &sMagic);
```

Podobná chyba jako předchozí, očekává se short int*, ale zapisuje se do short unsigned int*, problém se zápornými hodnotami.

1.15

Vzhledem k tomu, že je celý program napsán v C, nevidím důvod, proč používat koncovku cpp. Pokud kód neplánujeme dále nějak rozšiřovat s C++ knihovnami, přijde mi to zbytečné, ale zde záleží i na záměrech vývojáře. Takto je dle mě nesmyslně při kompilaci volán c++ kompilátor.

1.16

Možná je to pro tento příklad příliš, ale chybělo by mi třeba použití locale.h a jeho funkcí jako set_locale. Tímto je narušena kompatibilita s jinými jazyky používající jiné, než pro nás běžné znaky pro vstupní a výstupní funkce.

1.17

Přijde mi zvláštní, že se jméno zadané uživatelem používá jako název souboru, kde je zcela čitelné, ve kterém je to samé jméno šifrováno pomocí funkce encrypt. Dále funkce encrypt uvažuje pouze malá a velká písmena anglické abecedy s čísly, ale pokud já zadám např. "Jiří", znaky s diakritikou se nešifrují a výpisem souboru se dá zadané jméno již snadno domyslet, viz..

```
j@DESKTOP-F7I559N: cat Jiří
Qrří
87
7
8008
7j@DESKTOP-F7I559N:
```

1.18

```
28  if (nResult > 9)
```

Pro validní vstupy, tzn. ty, které má program očekávat (na vyzvání validní číselnou hodnotu dne v měsíci apod.) a ne nutně ty, které v současnosti přijímá, se mi nedaří dostat do této větve. Pokud omezíme nežádoucí vstupy (do dne v měsíci neakceptovat jiné, než numerické znaky 0-9), nebude tato větev třeba, nebudou přicházet vyšší číselné hodnoty při převodu znaku abecedy, tedy těch v ASCII s vyšší hodnotou, na int.

1.19

```
29  return CalcMagicNum(1, nResult % 10);
43  nResult = CalcMagicNum(nResult, 0);
```

Nepřijde mi vhodný tento návrh ohledně předávání argumentů této funkci, lze ji upravit na práci pouze s jedním argumentem. Na řádce 29 vždy předáváme 1 jako první argument, na řádce 43 vždy 0 jako druhý argument. Předávají se proměnné nabývající různých hodnot apod., ne takto předdefinované hodnoty zapsané přímo explicitně.

1.20

Funkci CalcMagicNum lze napsat efektivněji a bez využití rekurze.

1.21

```
49 static void encrypt(char* psz)
```

Funkce encrypt je volána jak pro šifrování, tak i pro dešifrování. Označil bych to za špatné pojmenování funkce a nahradil bych to kombinací funkcí encrypt/decrypt a nebo bych tuto funkci pouze přejmenoval, aby název zachycoval i její dešifrovací účel.

1.22

```
64 int main(int argc, char* argv[])
```

Tímto způsobem nelze zadávat jména s více slovy a mezerami. Pro jméno a příjmení (s mezerou) se přijmení přijme jako druhý argument a bez upozornění uživateli se v programu dále s druhým argumentem nijak nepracuje.

1.23

V mainu je příliš kódu, nevhodný návrh pro modularitu a přenositelnost. Tento kód by měl být přesunut do funkcí.

1.24

Chybějící dokumentace a komentáře, např. pro funkce.

1.25

Při prvním výpočtu pro nové jméno se do výstupního souboru zapisuje celý první argument jako jméno, tedy jméno zapisováno do souboru není téměř nijak velikostně omezeno. Dále při čtení již ze souboru se jako jméno načítá pouze prvních 32 bytů v souboru. Tímto se celkově rozbijí načítání ze souboru, pokud jsme původně zadali velmi dlouhé jméno, viz..

```
j@DESKTOP-F7I559N: ./a.out mmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmm
Your name is: mmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmm
Enter your day of birth: 20
Enter your month of birth: 4
Enter your year of birth: 1992
Your magic number is: 9
j@DESKTOP-F7I559N: ./a.out mmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmm
Your name is: mmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmm
You were born on 4
-20
-mmmmmmmmm

Your magic number is: 1992
j@DESKTOP-F7I559N:
```

1.26

```
92  printf(szName);
```

Opět chybí výstupní formát, stejná chyba jako v sekci 1.10.

1.27

```
93  fgets(szDay, sizeof(szName), pf);
95  fgets(szMonth, sizeof(szName), pf);
97  fgets(szYear, sizeof(szName), pf);
102  printf("\nYou were born on %s-%s-%s\n", szYear, szMonth, szDay);
```

Nevím, zdali je to záměr v této úloze, ale funkce fgets() čte i odřádkování. Proto se tento výpis může zdát zmatečný, při prvním pohledu na výpis se den a měsíc mohou zdát jako záporná čísla, viz..

```
j@DESKTOP-F7I559N: ./a.out abc
Your name is: abc

You were born on 2004
-4
-12

Your magic number is: 5
```

1.28

```
127  _snprintf(szComb, sizeof(szComb), "%s%s%s", szDay, szMonth, szYear);
```

Zde szComb nemá dostatečnou velikost pro správné zkombinování všech parametrů szDay, szMonth, szYear.

1.29

```
137  pf = fopen(argv[1], "wt")
```

Zde se vůbec nekontroluje, zdali se soubor povedl otevřít před zápisem, který následuje. Kontroloval bych např. alespoň is_open() nebo pf různé od NULL.

1.30

```
86  pf = fopen(argv[1], "rt");
137  pf = fopen(argv[1], "wt")
```

Jelikož nemáme správně ošetřeno, co je a co není validní jméno, dostáváme se zde k velkému konfliktu. Nyní se do jména akceptují i znaky jako např /. Tedy můžeme procházet adresářové struktury a tím si je zmapovat pro nějakou další formu útoku apod.. Můžeme i tímto způsobem, uvážíme-li funkci encrypt, vytvořit na disku nějaký soubor se škodlivým kódem.

```
j@DESKTOP-F7I559N: ./a.out /etc/passwd
Your name is: illg:c:9:9:illg:/illg:/yrm/yzhs
You were born on fhi/hyrm/mloltrm
-wzvnlm:c:8:8:wzvfhi/hyrm/mloltrm
-


Your magic number is: 0
j@DESKTOP-F7I559N:
```

1.31

Funkce encrypt není příliš efektivní pro spolehlivé šifrování. Obecně lze velmi snadno rozšifrovat tyto šifrové texty např. pomocí frekvenční analýzy. Např. znak ”J”bude vždy šifrován jako ”Q”nebo ”1”bude vždy jako ”8”. Dále lze v ŠT vidět přesný počet vstupních slov, kde každé slovo ŠT je stejně dlouhé jako totéž slovo v OT, jelikož mezery na výstupu zůstávají jako na vstupu a znaky různé od malých a velkých písmen anglické abecedy s čísly se vůbec neuvažují, jsou tedy v OT a ŠT stejné na totéž pozici. Pro následující vstup lze také snadno dedukovat, jak šifrovací funkce funguje.

```
j@DESKTOP-F7I559N: ./a.out abcdefABCDEF123456
Your name is: abcdefABCDEF123456
Enter your day of birth: 1
Enter your month of birth: 1
Enter your year of birth: 1
Your magic number is: 3
j@DESKTOP-F7I559N: cat abcdefABCDEF123456
zyxwvuZYXWVU876543
8
8
8
3j@DESKTOP-F7I559N:
```

1.32

```
19 static int CalcMagicNum(int nHi, int nLo)
26  nResult = nHi + nLo;
```

Zde může při neopratrném použití této funkce docházet k přetékání hodnot, se kterými se zde pracuje. Např. pokud předám nHi=3 a nLo=INT_MAX.

1.33

```
60  *psz++ = c;
```

Zde bych tipoval, že v krajních případech může dojít k segfaultu, pro hodně dlouhý vstup, kde bychom přistoupili mimo pamět procesu.

1.34

```
127  _snprintf(szComb, sizeof(szComb), "%s%s%s", szDay, szMonth, szYear);
```

Zde je třeba kontrolovat návratovou hodnotu, zdali není záporná a je menší než sizeof(szComb), tedy zdali se vše z parametrů povedlo bez chyby zkombinovat do szComb.