



**PROIDEA**

Fundacja Wspierania Edukacji Informatycznej

Przemysław Skowron

# Bezpieczne programowanie.

(zagrożenia i ochrona)

# START!

Przemysław Skowron  
<[przemyslaw.skowron@proidea.org.pl](mailto:przemyslaw.skowron@proidea.org.pl)>

# Bezpieczne programowanie.

## (zagrożenia i ochrona)

- Plan prelekcji:
  - Fundacja „*PROIDEA*”.
  - Błędy programistów (, a administrator):
    - Słowo wstępu,
    - Buffer overflow (underflow),
    - Format string,
    - Inne.
  - Podsumowanie.

# Bezpieczne programowanie.

## (zagrożenia i ochrona)

- Fundacja „*PROIDEA*”:
  - kto/co/dlaczego?
    - Organizacja typu „*non-profit*”,
    - Promocja wolnych rozwiązań,
    - Edukacja społeczeństwa,
    - Rozwijanie teleinformatyki w Polsce.
  - Działalność:
    - Kursy (Cisco, Linux, Java, Security(!)),
    - Konferencje, warsztaty, laboratoria,
    - Audyty bezpieczeństwa teleinformatycznego.

# Bezpieczne programowanie.

## (zagrożenia i ochrona)

- Słowo wstępu:
  - Języki programowania (ANSI C),
  - Kompilator (gcc),
  - Debugger (gdb, strace, ltrace, objdump, LD\_PRELOAD, grep, flawfinder, fuzzer),
  - Zagrożenia:
    - DoS,
    - przejęcie kontroli:
      - Uprawnienia (SUID).

# Bezpieczne programowanie.

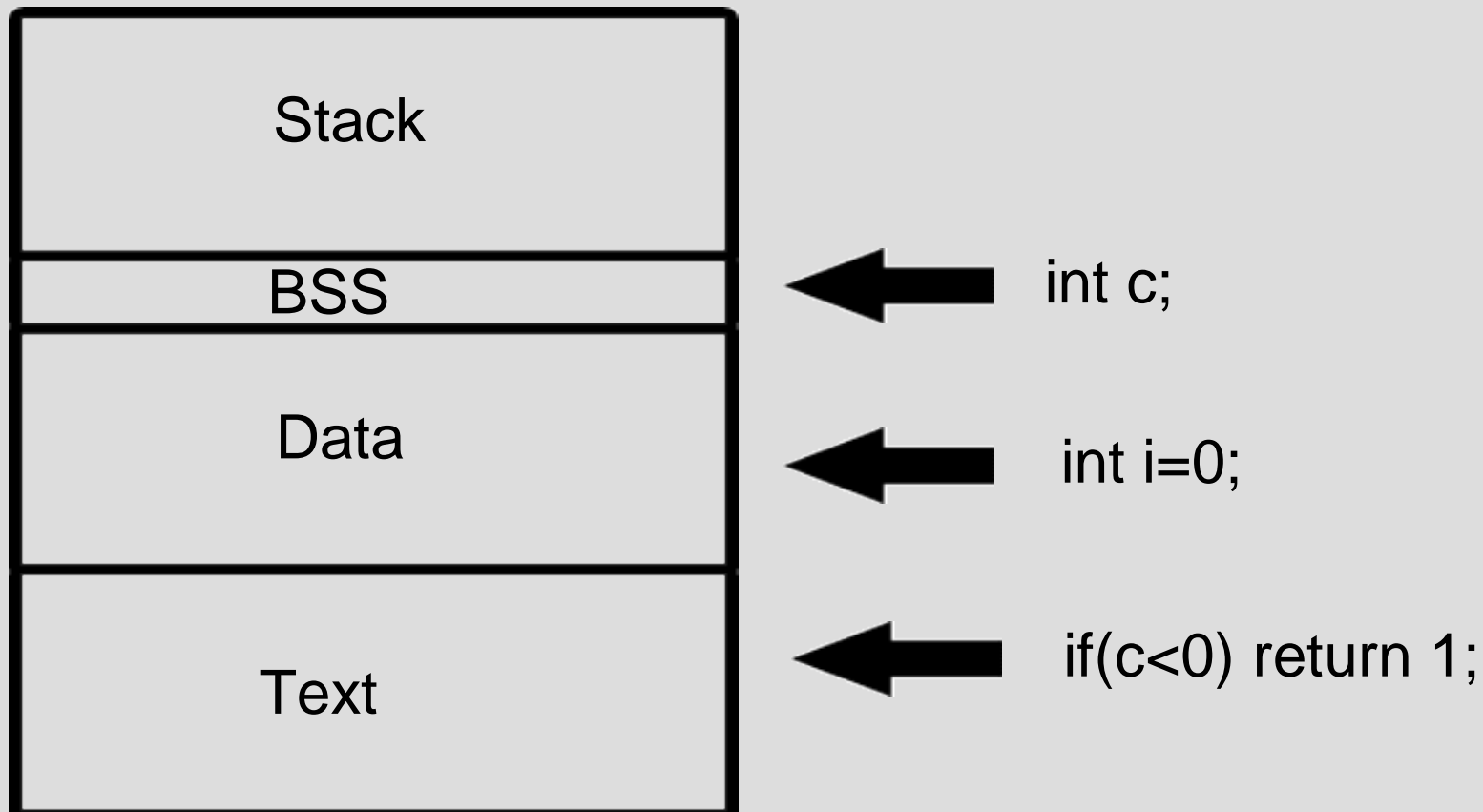
(zagrożenia i ochrona)

- Buffer overflow (underflow):
  - Organizacja pamięci,
  - Stos,
  - SIGSEGV (RET),
  - Podatne funkcje,
  - Przykłady BO,
  - Obrona BO.

# Bezpieczne programowanie.

(zagrożenia i ochrona)

## Organizacja pamięci

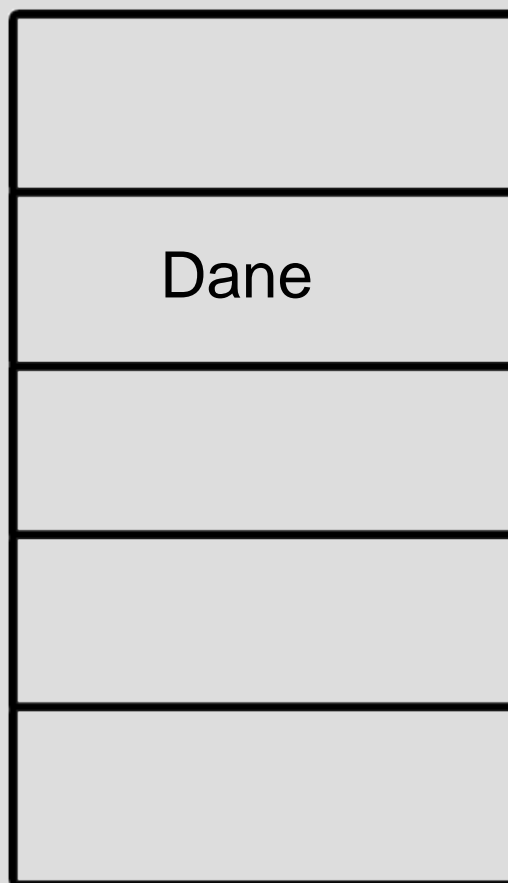


# Bezpieczne programowanie.

(zagrożenia i ochrona)

## Stos (Push Dane)

Wysokie adresy



Dane

← ESP

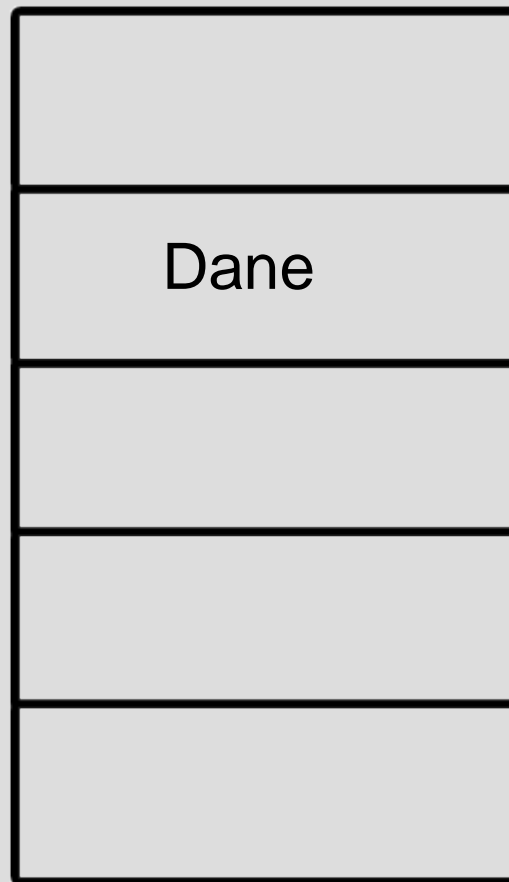
Niskie adresy

# Bezpieczne programowanie.

(zagrożenia i ochrona)

## Stos (POP Dane)

Wysokie adresy



← ESP

Niskie adresy

# Bezpieczne programowanie.

(zagrożenia i ochrona)

## SIGSEGV

- Rejestry:
  - EIP (wskaźnik następnej instrukcji - RET),
  - ESP (wskaźnik stosu),
  - EBP (wskaźnik ramki).

# Bezpieczne programowanie.

(zagrożenia i ochrona)

## SIGSEGV

Wysokie adresy



Niskie adresy

# Bezpieczne programowanie.

(zagrożenia i ochrona)

## SIGSEGV

Wysokie adresy



Niskie adresy

# Bezpieczne programowanie.

(zagrożenia i ochrona)

- Podatne funkcje:

- gets(),
- strcpy(),
- strcat(),
- sprintf(),
- (f)scanf(),
- getwd(),
- realpath(),
- ... .

# Bezpieczne programowanie.

(zagrożenia i ochrona)

## Przykład#1

```
#include <stdio.h>

int main(int argc, char **argv)
{
    char c[8];

    gets(c);
    printf("%s\n", c);

    return 0;
}
```

# Bezpieczne programowanie.

(zagrożenia i ochrona)

## Przykład#1 (cd.)

```
r@mruczek bo $ perl -e 'print "A"x12' | ./1  
AAAAAAAAAAAAAA
```

**Błędna instrukcja**

```
r@mruczek bo $ perl -e 'print "A"x13' | ./1  
AAAAAAAAAAAAAAA
```

**Naruszenie ochrony pamięci**

```
r@mruczek bo $ perl -e 'print "A"x7' | ./1  
AAAAAAA
```

# Bezpieczne programowanie.

(zagrożenia i ochrona)

## Przykład#2

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

int main(int argc, char **argv)
{
    char c[8], b[4];

    strncpy (c, argv[1], 8);
    strncpy (b, c, 4);

    printf("%s\n", b);

    return 0;
}
```

# Bezpieczne programowanie.

(zagrożenia i ochrona)

## Przykład#2 (cd.)

r@mruczek bo \$ ./2 AAA (3x"A")  
AAA

r@mruczek bo \$ ./2 AAAA (4x"A")  
AAAAAAA

r@mruczek bo \$ ./2 AAAAAAAAA (8x"A")  
AAAAAAAAAAAA^ óÿ¿"H@

# Bezpieczne programowanie.

(zagrożenia i ochrona)

## Przykład#2 (cd.)

- Podatne funkcje:
  - strncpy(),
  - ....

# Bezpieczne programowanie.

(zagrożenia i ochrona)

## Przykład#3 (underflow)

```
#include <unistd.h>

int main(void)
{
    char buf[8];
    long addr = 0x12345678;

    bzero(buf, sizeof(buf));           // czyścimy buf
    printf("addr (przed) = %x\n", addr);
    buf[strlen(buf)-1] = '\0';       // „kończymy” buf
    printf("addr (po) = %x\n", addr);

    return 0;
}
```

# Bezpieczne programowanie.

(zagrożenia i ochrona)

## Przykład#3 (cd. underflow)

```
r@mruczek bo $ ./3  
addr (przed) = 12345678  
addr (po) = 345678
```

```
// brak „12”
```

# Bezpieczne programowanie.

(zagrożenia i ochrona)

- Obrona:
  - ProPolice (gcc),
  - libsafe,
  - PaX,
  - Owl (non-executable user stack area),
  - StackGuard,
  - ?

# Bezpieczne programowanie.

(zagrożenia i ochrona)

## ProPolice (IBM)

- Łatka na gcc:
  - `fstack-protector` (`-fno-stack-protector`),
  - `fstack-protector-all` (`-fno-stack-protector-all`),
- Możliwości:
  - Detekcja próby nadpisania adresu powrotu (wyjście z programu),
  - Zmiana kolejności wskaźników na stosie,
  - Inne.
- <http://www.trl.ibm.com/projects/security/ssp/>

# Bezpieczne programowanie.

(zagrożenia i ochrona)

## Libsafe (Avaya Labs Research)

- Biblioteka:
  - LD\_PRELOAD,
  - /etc/ld.so.preload.
- Możliwości:
  - Podmiana podatnych na ataki funkcji (BO),
- <http://www.research.avayalabs.com/project/libsafe/>

# Bezpieczne programowanie.

(zagrożenia i ochrona)

## PaX

- Łatka na kernel:
  - Grsecurity (<http://grsecurity.net>).
- Możliwości:
  - Losowa adresacja przestrzeni procesu (ASLR),
  - Nie wykonywalny obszar stosu dla użytkownika (Owl Linux Patch),
  - Inne.
- <http://pax.grsecurity.net/>

# Bezpieczne programowanie.

(zagrożenia i ochrona)

## StackGuard (Immunix)

- Łatka na kompilator (gcc),
- Możliwości:
  - Detekcja zmiany adresu RET przed powrotem:
    - „*kanarek*”,
    - Pułapka na muchy.
- <http://immunix.org/technology/compiler.php>

# Bezpieczne programowanie.

(zagrożenia i ochrona)

?

# Bezpieczne programowanie.

(zagrożenia i ochrona)

- Format string:
  - Znak formatownia,
  - Podatne funkcje,
  - Formatowanie, a stos,
  - Przykłady FS,
  - Obrona FS.

# Bezpieczne programowanie.

(zagrożenia i ochrona)

## Znak formatownia

- Przykłady:
  - `printf("%d", c);`
  - `printf(c);`
- Rodzaje znaków formatowania:
  - `%c` -znak,
  - `%d` -liczba całkowita,
  - `%f` - liczba rzeczywista,
  - `%s` – ciąg znaków,
  - `%n` – liczba *wydrukowanych* znaków,
  - inne.

# Bezpieczne programowanie.

(zagrożenia i ochrona)

- Podatne funkcje:
  - printf(),
  - fprintf(),
  - sprintf(),
  - snprintf(),
  - vfprintf(),
  - vprintf(),
  - vsprintf(),
  - ... .

# Bezpieczne programowanie.

(zagrożenia i ochrona)

## Formatowanie, a stos

- Znak formatu, zmienna.
- Co gdy nie ma znaku formatu, a jest zmienna?
- Co gdy znak formatu znajduje się w treści zmiennej?
- Co gdy nie ma zmiennej, a jest znak formatu?

# Bezpieczne programowanie.

(zagrożenia i ochrona)

## Formatowanie, a stos

- Znak formatu, zmienna.

```
#include <stdio.h>

int main(int argc, char **argv)
{
    printf("%s\n", argv[1]);

    return 0;
}
```

# Bezpieczne programowanie.

(zagrożenia i ochrona)

## Formatowanie, a stos

- Znak formatu, zmienna.

```
r@mruczek fs $ ./1 AAA  
AAA
```

```
r@mruczek fs $ ./1 %x  
%x
```

# Bezpieczne programowanie.

(zagrożenia i ochrona)

## Formatowanie, a stos

- Co gdy nie ma znaku formatu, a jest zmienna?

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

```
int main(int argc, char **argv)
```

```
{
```

```
    printf(argv[1]);
```

```
    printf("\n");
```

```
    return 0;
```

```
}
```

# Bezpieczne programowanie.

(zagrożenia i ochrona)

## Formatowanie, a stos

- Co gdy nie ma znaku formatu, a jest zmienna?

```
r@mruczek fs $ ./2 AAA  
AAA
```

- Co gdy znak formatu znajduje się w treści zmiennej?

```
r@mruczek fs $ ./2 %x  
40013760
```

# Bezpieczne programowanie.

(zagrożenia i ochrona)

## Formatowanie, a stos

- Co gdy znak formatu znajduje się w treści zmiennej?

```
r@mruczek fs $ ./2 AAA%x%x%x(...)
AAA40013760(...)414141
```

# Bezpieczne programowanie.

(zagrożenia i ochrona)

## Formatowanie, a stos

- Co gdy nie ma zmiennej, a jest znak formatu?

**Syntax error!**

# Bezpieczne programowanie.

(zagrożenia i ochrona)

- Obrona:
  - Libsafe,
  - FormatGuard,
  - Snort (NI[P|D]S),
  - ?

# Bezpieczne programowanie.

(zagrożenia i ochrona)

## Libsafe (Avaya Labs Research)

- Biblioteka:
  - LD\_PRELOAD,
  - /etc/ld.so.preload.
- Możliwości:
  - Podmiana podatnych na ataki funkcji (%n),
- <http://www.research.avayalabs.com/project/libsafe/>

# Bezpieczne programowanie.

(zagrożenia i ochrona)

## FormatGuard (Immunix)

- Łatka na kompilator (gcc),
- Możliwości:
  - Osłona dla funkcji podatnych na atak *format string*,
  - Licznik znaków formatowania (makra).
- <http://immunix.org/technology/compiler.php>

# Bezpieczne programowanie.

(zagrożenia i ochrona)

## Snort

- System NIDS,
- Możliwości:
  - Dopasowanie sygnatury:
    - Długość,
    - Zawartość.
  - Co dalej? (wyobraźnia)
- <http://www.snort.org>

# Bezpieczne programowanie.

(zagrożenia i ochrona)

?

# Bezpieczne programowanie.

(zagrożenia i ochrona)

- Inne:
  - Heap overflow,
  - Signal race,
  - Integer overflow,
  - Race condition,
  - TMP file,
  - zmienne środowiskowe,
  - pozostałe (?)

# Bezpieczne programowanie.

(zagrożenia i ochrona)

- Heap overflow:
  - Organizacja stery,
  -

# Bezpieczne programowanie.

## (zagrożenia i ochrona)

- Signal race:
  - Sygnał,
  - Obsługa sygnałów,
  - Sytuacja wyścigu:
    - Czas (problem),
    - Przykład:
      - `2*free();`

# Bezpieczne programowanie.

## (zagrożenia i ochrona)

- Integer overflow:
  - Typ danych:
    - int: od -32 768 do 32 768),
  - Przekroczenie zakresu:
    - Ominięcie warunku,
    - Alokacja pamięci łatwej do nadpisania,
    - ?

# Bezpieczne programowanie.

## (zagrożenia i ochrona)

- Race condition:
  - TMP file,
  - Kolejki FIFO:
    - Uprawnienia do plików/katalogów,
    - Blokowanie dostępu do plików/katalogów,

# Bezpieczne programowanie.

(zagrożenia i ochrona)

- Zmienne środowiskowe,
  - Definicja, (\$PATH),
  - Funkcje:
    - getenv(),
    - putenv(),
    - setenv(),
    - ...,
    - system(). [ system("aplikacja"); ]

# Bezpieczne programowanie.

(zagrożenia i ochrona)

- Pozostałe:

?

# Bezpieczne programowanie.

## (zagrożenia i ochrona)

- Podsumowanie:
  - Znajomość składni j. programowania to mało,
  - Audyt kodu źródłowego drogą do bezpieczniejszych systemów/aplikacji,
  - Stosowanie pojedynczych mechanizmów zabezpieczeń dla niedowidzących,
  - Język C tylko wtedy gdy musimy...

# Bezpieczne programowanie.

(zagrożenia i ochrona)

- Bibliografia:

- ,
- ,
- ,
- ,
- ,
- .

# Bezpieczne programowanie.

(zagrożenia i ochrona)

Dziękuję za uwagę.

(<http://www.proidea.org.pl>)