Bezpieczne programowanie

Zagadnienia

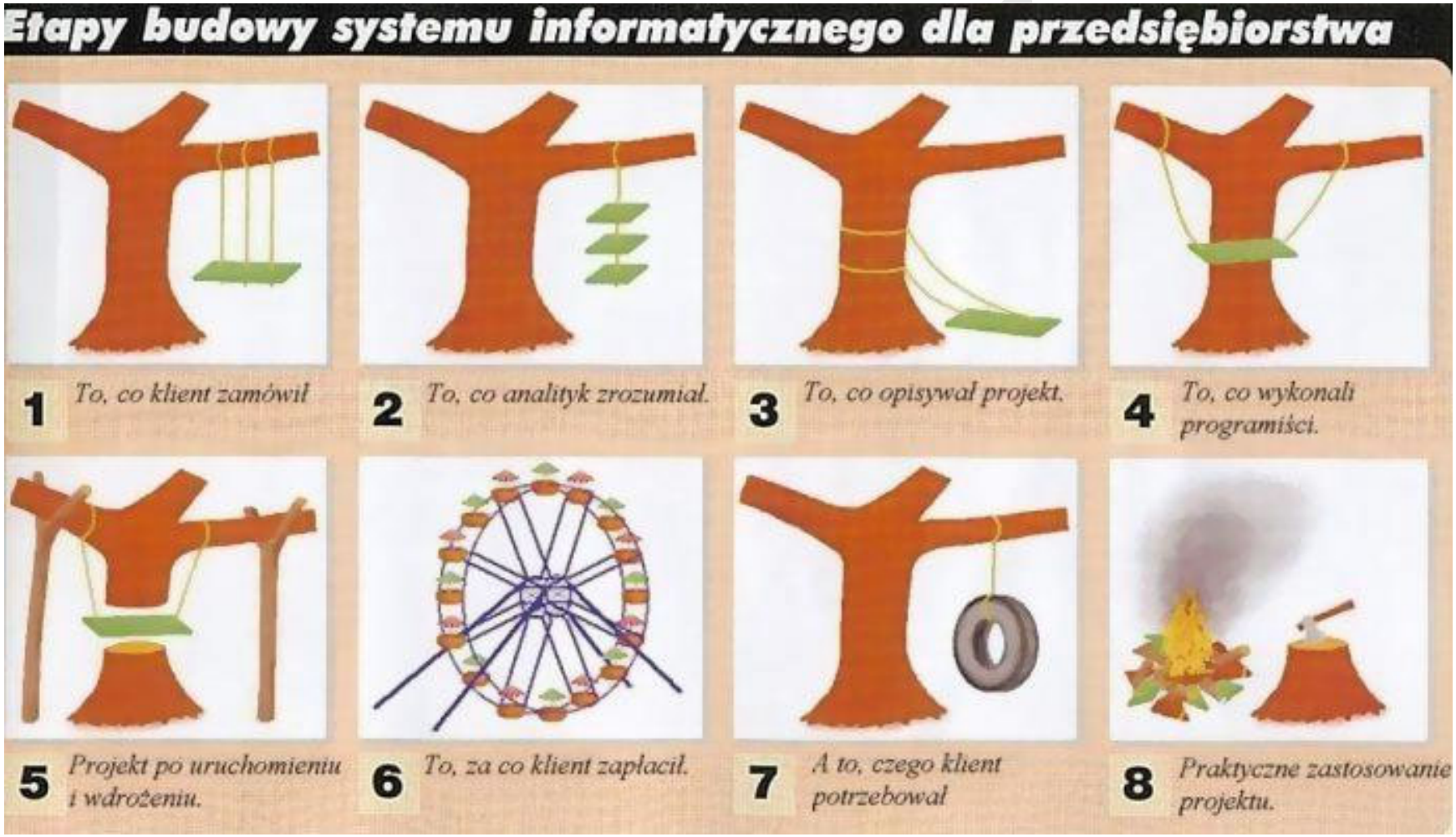
1. **Krytyczne błędy programistyczne**
   - przepełnienie bufora
2. **Ochrona przed błędami**
   - bezpieczna kompilacja
   - bezpieczne biblioteki
3. **Sztuka tworzenia bezpiecznego kodu**

Motto programisty

Programowanie jest trudne

(*E. Dijkstra*)

Złożoność procesu tworzenia oprogramowania



Błędy programistyczne

Uproszczenia

$$\frac{1}{\pi} Sin\pi x = six = 6$$

Błędy programistyczne

Swap x ↔ y

```
swap(inout x, y)
{
    x:=y;
    y:=x;
}
```

Błędy programistyczne

Swap x ↔ y

```
swap(inout x, y)
{
    x:=x-y;
    y:=y+x;
    x:=y-x;
}
```

Błędy programistyczne

Zaniedbania

- nieprawidłowy format danych
- przekroczenie zakresu danych
- przepełnienie bufora
- . . .

Błędy programistyczne

Ataki poprzez eksploatację błędów

Typowe klasy błędów implementacyjnych:

- buffer overflow (stack overflow, heap overflow)
- out of array
- format string
- double free
- obsługa wejścia/wyjścia
- interakcja z systemem operacyjnym

Błędy programistyczne

Ataki poprzez eksploatację błędów

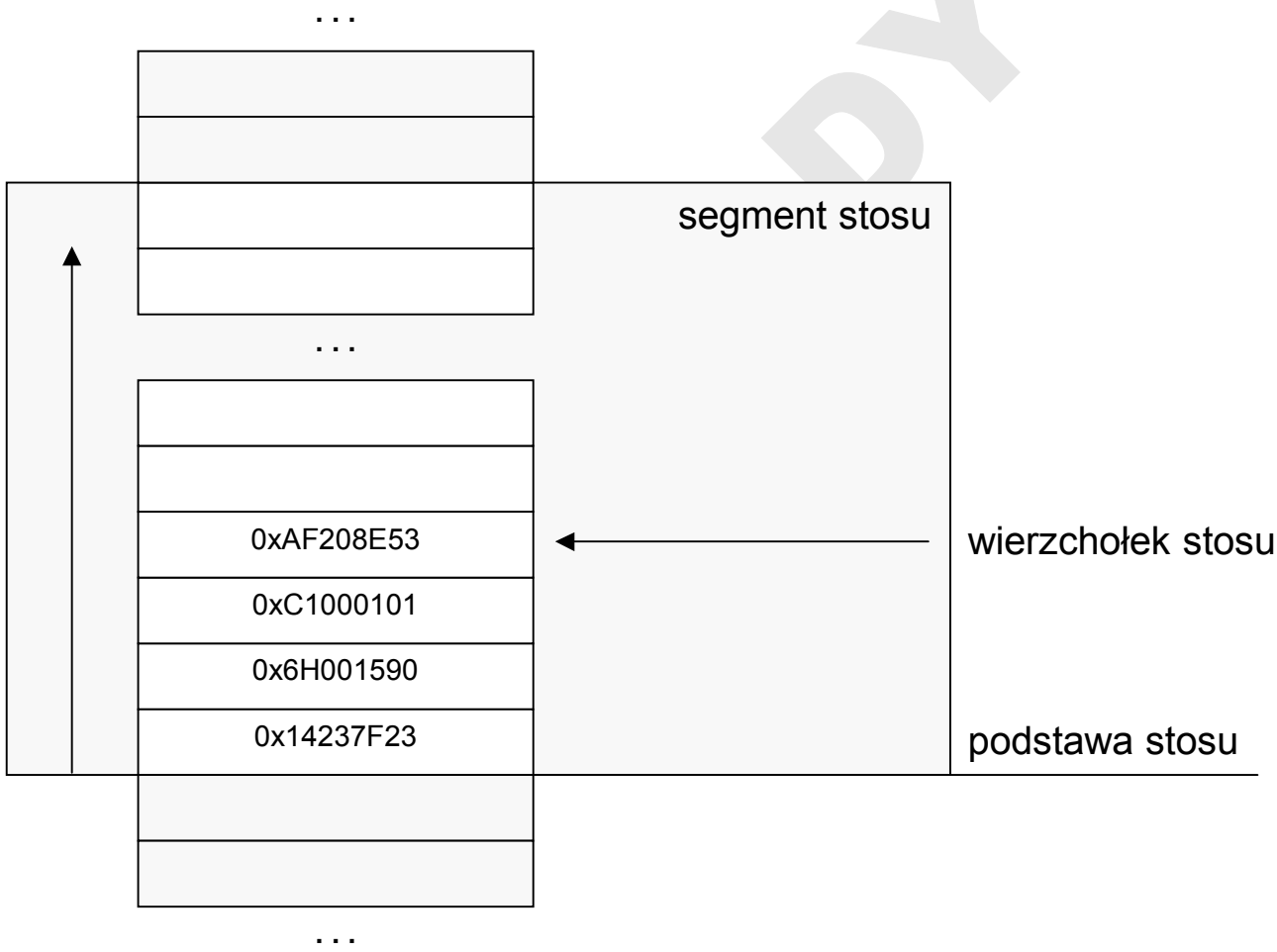
Skutek ataku:

- przejęcie sterowania w procesie
- wykonanie własnych instrukcji / poleceń systemu operacyjnego / bazy danych (tzw. *command injection*)
- obejście chroot
- najczęstsze polecenia: /bin/sh, \windows\system32\cmd.exe (%COMSPEC%)

Przepełnienie bufora

Problem

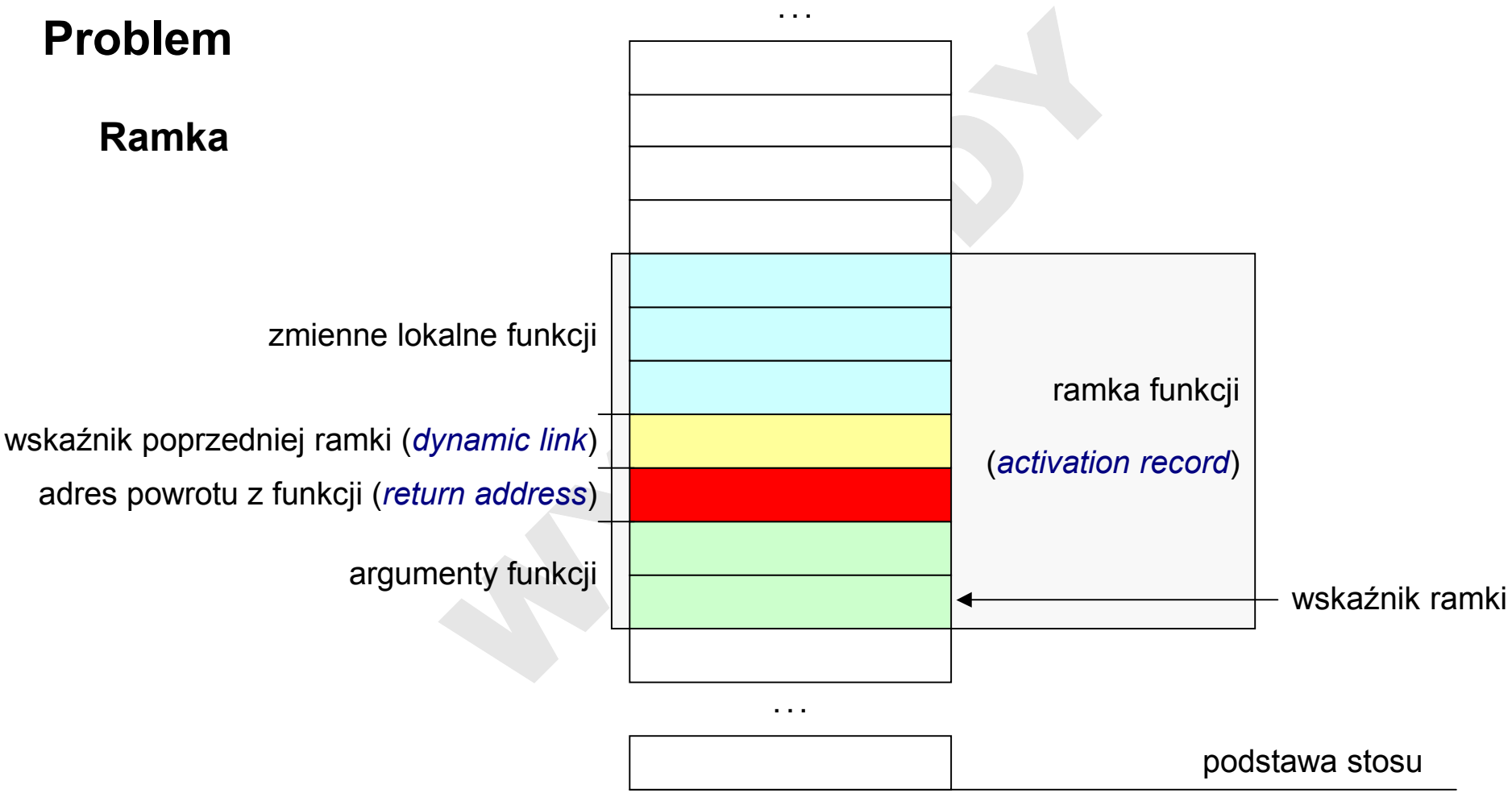
Stos



Przepełnienie bufora

Problem

Ramka



Przepełnienie bufora

Problem

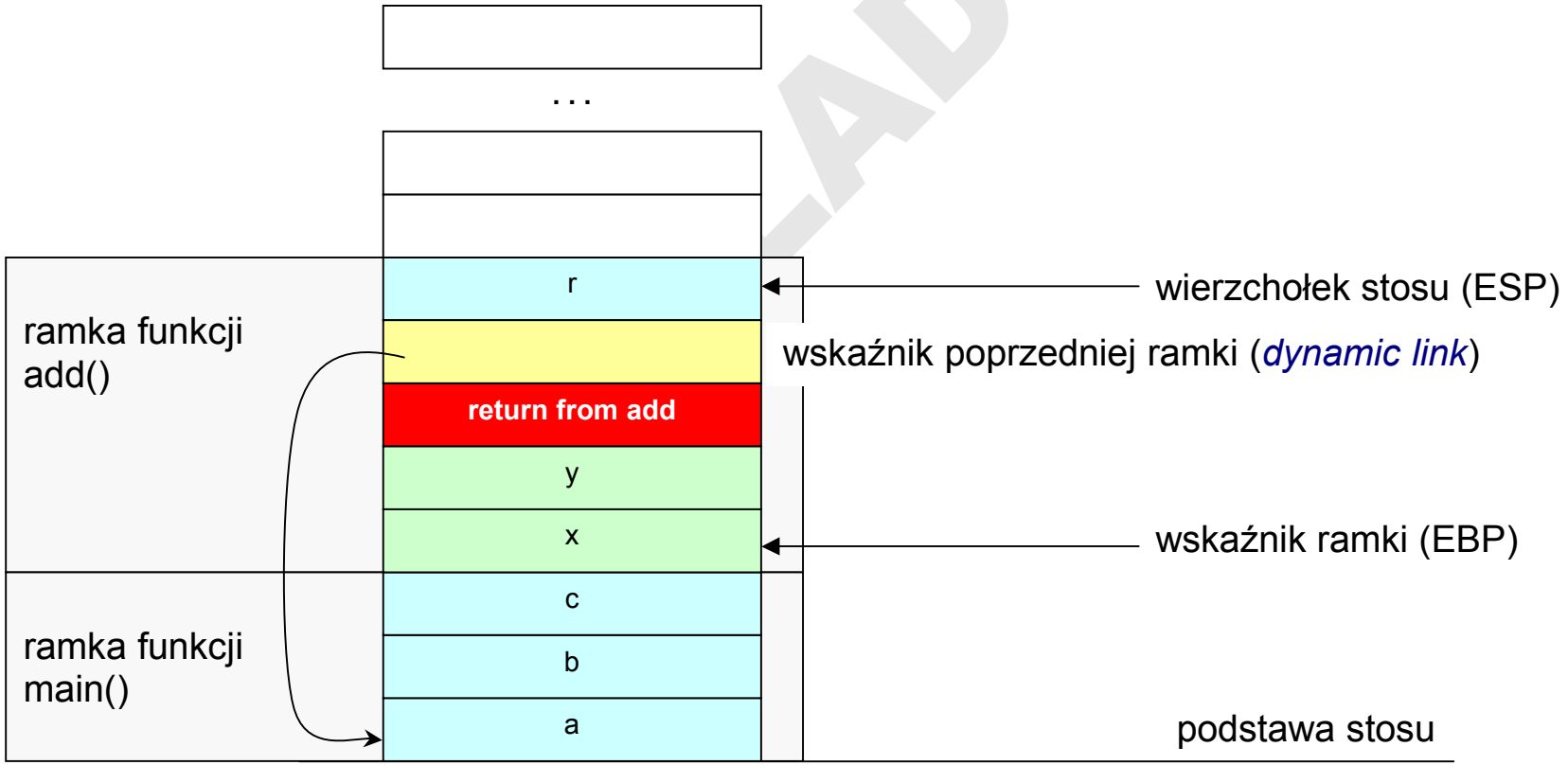
Przykład

```
int add(int x, int y)
{
    int r;
    r=x+y;
    return r;
}

...

int g;
int main(int argc,int **argv)
{
    int a,b,c;
    ...
    c=add(a,b);
    ...
}
```

Przepełnienie bufora

Problem

Łańcuch ramek



Przepełnienie bufora

Problem

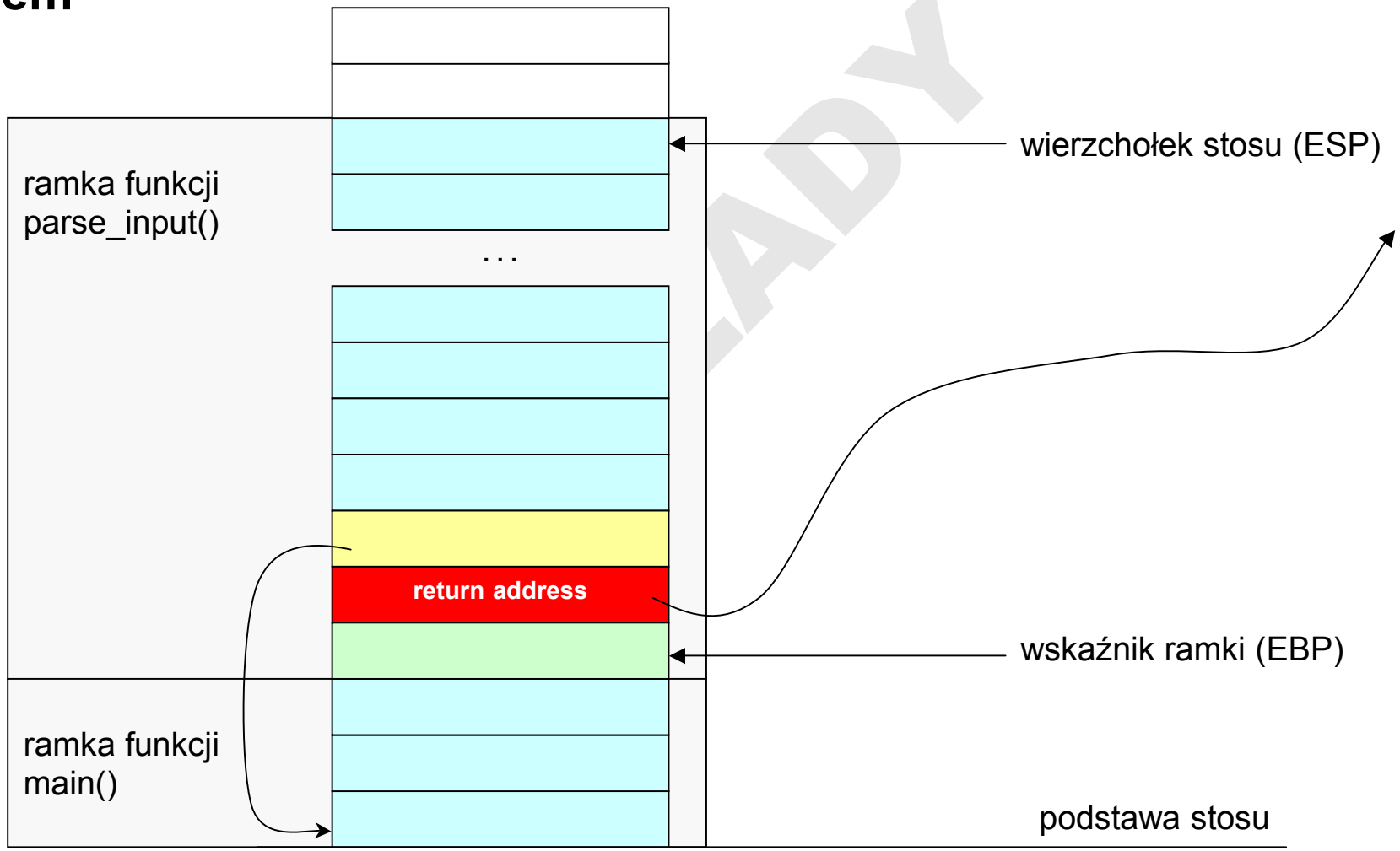
Przykład

dostatecznie duży przydział pamięci na oczekiwane dane wejściowe

```
int parse_input(char **input)
{
     char buffer[1024];
     strcpy(buffer,input);
...
}

...

int main(int argc,char **argv)
{
     if(argc>1)
     {
          parse_input(buffer,argv[1]);
     }
...
}
```

Przepełnienie bufora

Problem



Przepełnienie bufora

Atak metodą Levy'ego (Elias Levy vel. α1)

Co jeśli do bufora wprowadzony zostanie sprytnie dobrany ciąg bajtów?

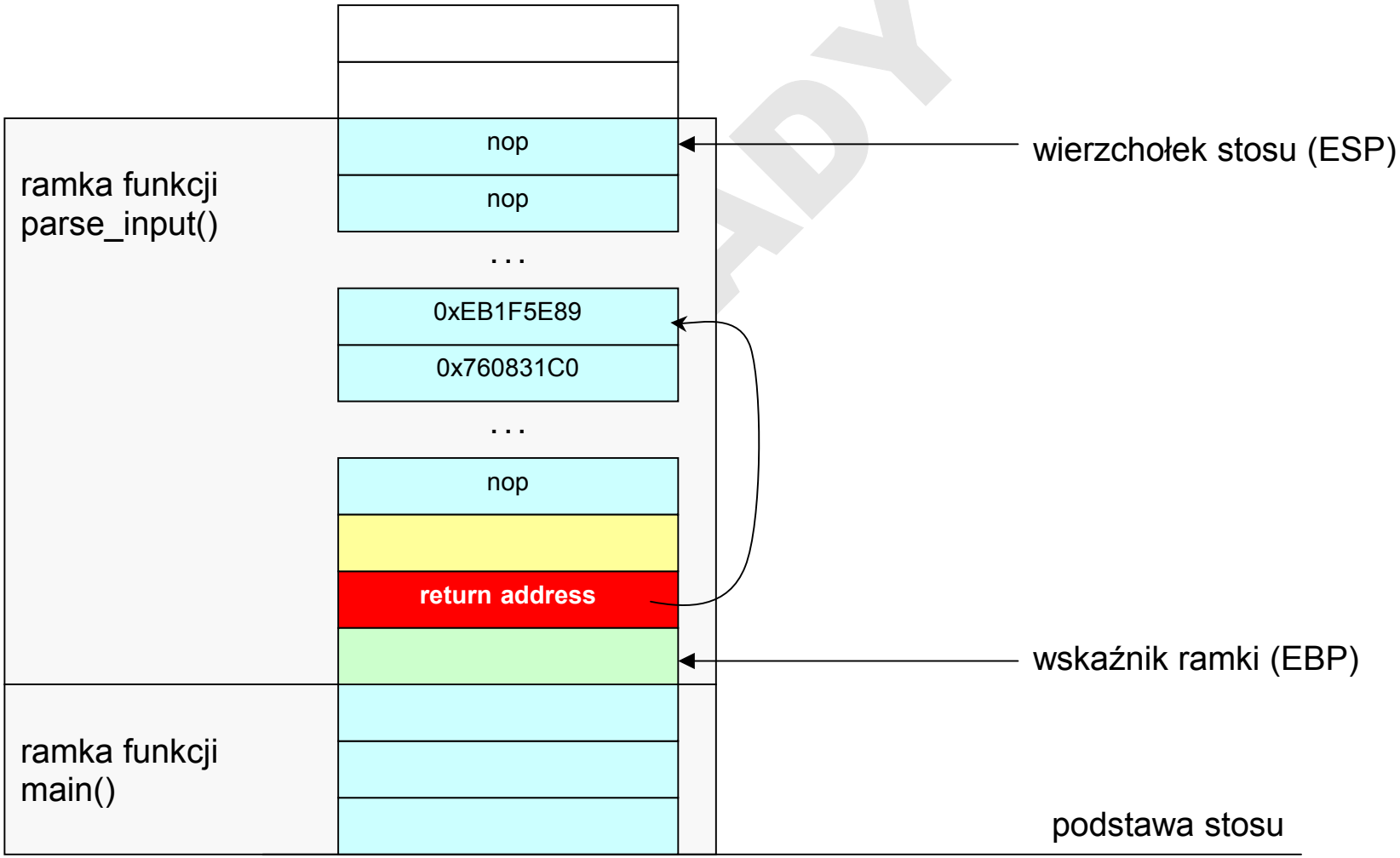
- dłuższy niż 1024 B (dokł. 1032 B)
- zawierający w początkowej części kod maszynowy instrukcji:

```
jmp 0x1f
opl %esi
ovl %esi,0x8(%esi)
orl %eax,%eax
ovb %eax,0x7(%esi)
ovl %eax,0xc(%esi)
ovb $0xb,%al
ovl %esi,%ebx
eal 0x8(%esi),%ecx
eal 0xc(%esi),%edx
nt $0x80
orl %ebx,%ebx
ovl %ebx,%eax
nc %eax
nt $0x80
all -0x24
string \"/bin/sh\"
```

shell code

Przepełnienie bufora

Atak metodą Levy'ego



Przepełnienie bufora

Atak metodą Levy'ego

Scenariusz ataku:

1. wstawienie kodu (shell code) do pamięci atakowanego procesu
2. doprowadzenie do przepełnienia bufora i nadpisania adresu powrotu na stosie
3. przejęcie kontroli poprzez uruchomienie wstawionego kodu

Problemy:

- określenie z góry wielkości bufora (ile nadpisać?)
- poprawne określenie nowej wartości adresu powrotu (czym nadpisać?)
- czym wypełnić bufor oprócz shellcode ?

Przepełnienie bufora

Atak metodą Levy'ego

Przykład – Morris worm (1988)

- zaatakowany fingerd – demon pracujący z uprawnieniami root
- funkcja gets() pobierała dane do bufora 512 B,
- a robak wysyłał 536 B
- aż do czasów Levy'ego (1996) zainteresowanie przepełnieniem bufora było znikome

Przepełnienie bufora

Rzeczywiste przyczyny

Reprezentacja ciągu znaków:

- w języku C string kończy się znakiem NULL (czasami CR, LFCR)

Brak kontroli zakresu:

- funkcje biblioteczne strcpy(), gets(), ...

Programista:

- dostatecznie duży przydział pamięci na oczekiwane dane wejściowe nie jest dostatecznie duży na nieoczekiwane dane wejściowe

Przepełnienie bufora

Podatność na ataki

Systemy operacyjne

- wszelkie, w tym Linux, OpenBSD, FreeBSD, Solaris, ..., Windows
- Unix ↔ Windows – bardzo podobnie

Aplikacje

- praktycznie wszystko co korzysta z bibliotek C

Przepełnienie bufora

Podatność na ataki

Raport Millera

- Barton P. Miller (1998): http://www.securityfocus.com/library/2087
- badanie powtórne (po 5 latach): obrazuje tempo usprawniania kodu
- producenci oprogramowania ponieśli pewien wysiłek
- i odnieśli sukces ...
- ... dość skromny
- OpenSource ↔ Commercial

Przepełnienie bufora

Ochrona

Eliminacja błędów

- poprawianie kodu źródłowego programów: strcpy() → strncpy(), gets() → fgets()
- zautomatyzowane poszukiwanie błędów
  http://www.blackhat.com/html/bh-asia-00/bh-europe-00-speakers.html#Joey
  http://www.cs.berkeley.edu/~daw/papers/overruns-ndss00.ps
- problem skali: dawno przekroczona masa krytyczna
- brak gwarancji uwolnienia się od przepełnienia bufora:
  nawet w poprawionym kodzie zostają jakieś dziury
  http://www.securityfocus.com/archive/1/9023
- największe sukcesy: OpenBSD, Adamantix

Przepełnienie bufora

Ochrona

Wykluczenie wykonywania kodu na stosie

- segmenty niewykonywalne (wsparcie architektury – bit non-executable NE, NX, nakładki na system, np. PaX, Stack Smashing Protector, OverflowGuard)

  http://www.datasecuritysoftware.com

- metoda bardzo efektywna
- w większości nie ma potrzeby wykonywania instrukcji z segmentu stosu
- jednak czasami jest ...
- ... ponadto istnieją metody obejścia ograniczenia wykonywania stosu – metoda Wojtczuka:

http://www.securityfocus.com/templates/archive.pike?list=1&mid=8470&fromthread=0&date=1998-01-30

Przepełnienie bufora

Ochrona

Bezpieczne wersje bibliotek

- kontrola zakresu daje 100% rezultat
- poprawione strcpy() sprawdza czy nie kopiuje poza bezpieczny obszar bieżącej ramki na stosie
- implementacja Snarskiego na FreeBDS
- *LibSafe* (Bell Labs)
- *BOWall protection* (Andriej Koliszak) dla Windows NT

Przepełnienie bufora

Ochrona

Bezpieczne kompilatory

- kontrola zakresu na etapie kompilacji

  (w wielu językach nie ma problemu: Java, Ada, Eiffel, ...)

- pełna kontrola zakresu w C nie jest możliwa:

tu jest łatwa:

```
buffer[i]
```

ale tutaj?:

```
buffer + i
```

Przepełnienie bufora

Ochrona

Metoda z „kanarkiem”

- kompilator alokuje dodatkowy obszar („kanarka”) pomiędzy wskaźnik poprzedniej ramki a adres powrotu
- przed powrotem z funkcji weryfikuje czy wartość „kanarka” nie została zmieniona
- jeśli tak – detekcja ataku – proces ginie
- implementacja – StackGuard (http://immunix.org/stackguard.html)
- StackGuarda można oszukać – ocalić „kanarka” mimo nadpisania adresu powrotu
- ale i przed takimi sztuczkami można się bronić uzależniając początkową wartość „kanarka” od wartości adresu powrotu – detekcja przez porównanie obu wartości
- „sprytne kanarki” są trudne do „przeskoczenia”: 0x 00 0A FF 0D

Funkcje biblioteczne

Biblioteki współdzielone

- współdzielenie bibliotek jest użyteczne również w kontekście bezpieczeństwa – możliwa kontrola ograniczona do jednego punktu
- jednak wymagają tej kontroli – są odseparowane od pliku wykonywalnego:
  - np. nie wystarczy sprawdzanie sumy kontrolnej pliku binarnego polecenia `login` – trzeba również weryfikować funkcje znajdujące się we współdzielonych bibliotekach
  - problem uprawnień bibliotek (suid)
  - kontrola dostępu do bibliotek (zwł. integralności)

Funkcje biblioteczne

Biblioteki współdzielone a przepełnienie bufora

Return into libc

- libc jest niezwykle przydatny do ataku
- jeśli uda się w wykonywalnej przestrzeni adresowej procesu przydatne funkcje (`exec`, `system`, `strcpy`) i spreparować im na stosie odpowiednie parametry
- wówczas w obszar stosu wykonanie w ogóle nie musi wkroczyć – shellcode jest niepotrzebny
- można skorzystać z narzędzi dostępnych w samym libc, np. `dlsym`
  - jeśli exploit zostanie skonsolidowany z tymi samymi współdzielonymi bibliotekami co atakowany program istnieją duże szanse trafienia w `exec` pod adresem podanym przez `dlsym`

Bezpieczne programowanie

http://www.secureprogramming.com/

Podstawowe zagadnienia

- SD^3: Secure by Design - by Default - in Deployment
- filtracja – bezpieczne pobieranie danych wejściowych ("all input is evil!")
- szcz. kontrola interakcji ze światem zewnętrznym („za oknem zawsze czai się wróg!")
- minimalizowanie obszaru ataku
- separacja obszarów kodu i danych
- „zgodność wstecz źródłem nieszczęść"
- usuwaj błędy nawet bez exploit-ów

Bezpieczne programowanie

Przykłady

PHP:

wersja 1:

```
$SQLcmd = "SELECT uid FROM users WHERE name=\"$name\" AND
          pass=\"$pass\";";
```

wersja 2:

```
$SQLcmd = 'SELECT uid FROM users WHERE name="'.$name.'" AND
          pass="'$pass"';';
```

- parameter injection: $pass = " OR 1="1

wersja 1 po ewaluacji:

```
SELECT uid FROM users WHERE name="bond" AND pass="" OR 1="1";
```

wersja 2 po ewaluacji:

```
SELECT uid FROM users WHERE name="bond" AND pass="\" OR 1=\"1";
```