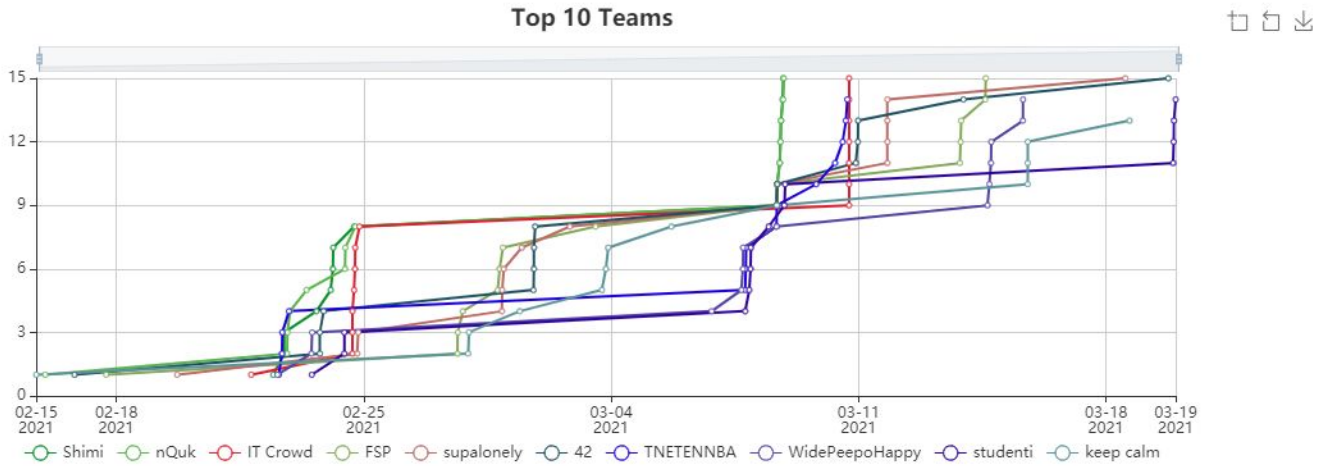
Pamäťové zraniteľnosti

Peter Švec
peter_svec@stuba.sk

Reverzné inžinierstvo výsledky



Place	Team	Score
1	Shimi	15
2	nQuk	15
3	IT Crowd	15

Úvod

- C je nízko úrovňový jazyk (vo veľa veciach verí vývojárovi)
- Čo ak vieme pomocou chyby v programe zapisovať dáta mimo alokovaný priestor?
- Typy pamäťových chýb v rámci zásobníka
- Moderné mitigácie (a techniky ako ich obísť)

Problém 1

Dôvera vývojárovi.

Python:

```
a = [ 1, 2, 3, 4, 5 ]
a[10] = 0x41
```

IndexError: list index out of range

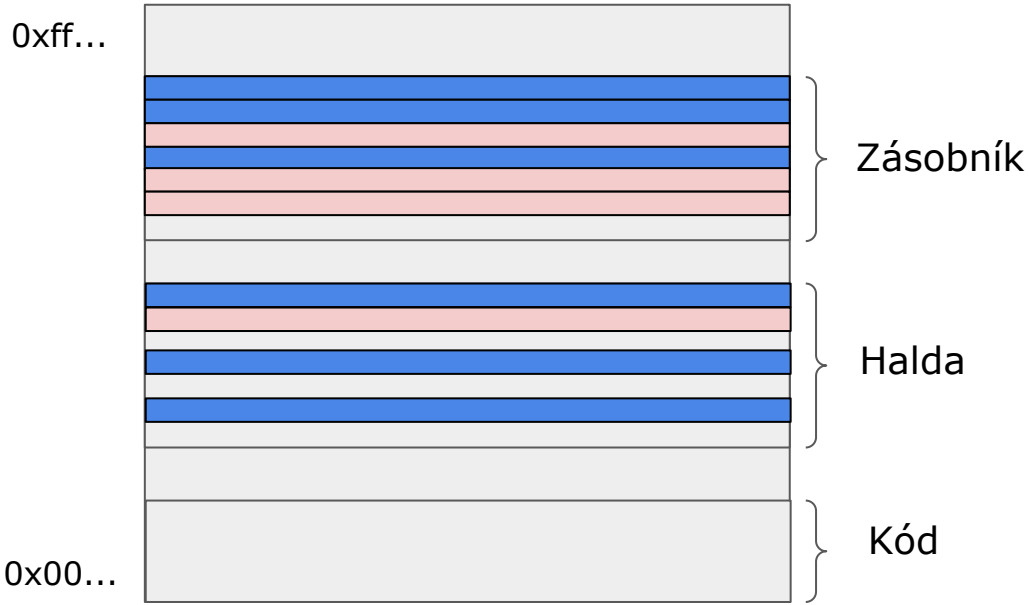
C:

```
int a[5] = { 1, 2, 3, 4, 5 };
a[10] = 0x41;
```

No problem!

Problém 2

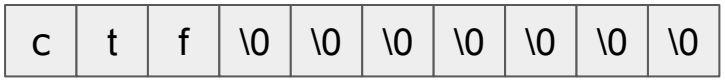
Mixovanie **dát** (aj používateľského vstupu) a **“Control flow”** údajov (návratové adresy, smerníky na funkcie)



Problém 3

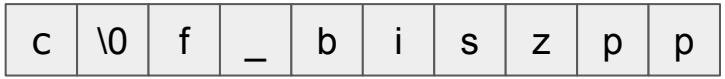
Mixovanie dát a metadát.

char data[10] = “ctf”;

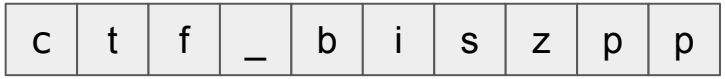


strlen(data) = 3

read(0, data, sizeof(data));



strlen(data) = 1



strlen(data) = ??

Problém 4

Inicializácia a dealokácia zásobníka.

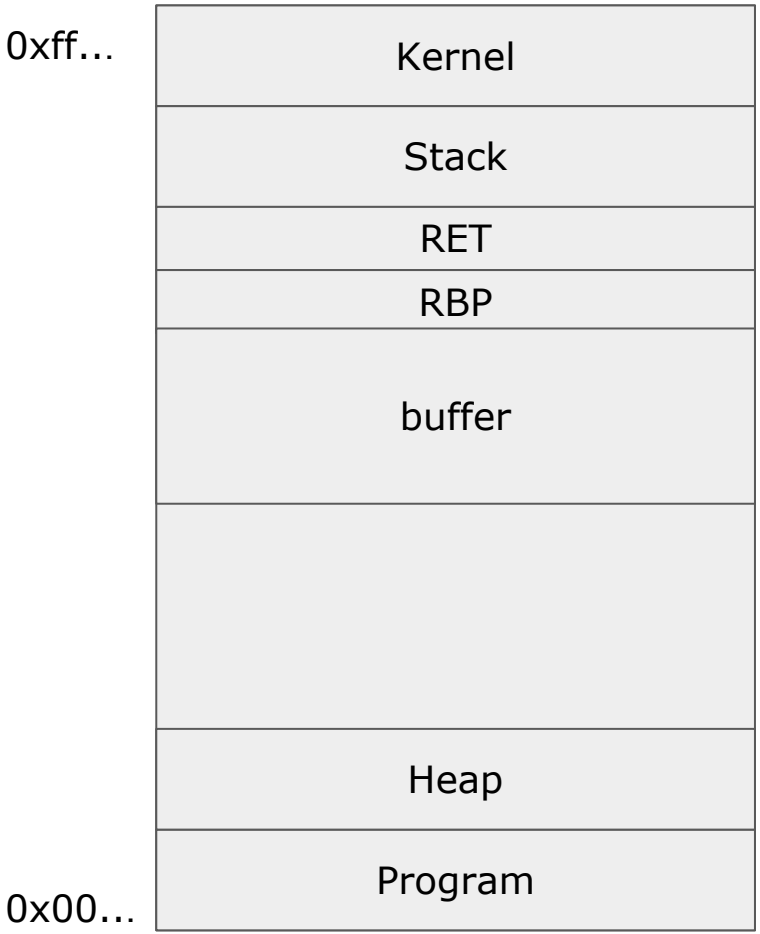
```
void foo()
{
	char data[20];
	// aka je hodnota poľa data??
}
```

Inicializácia a aj dealokácia je iba posun registrov. Dáta treba vyčistiť ručne.

Stack Smashing

```
void vuln()
{
	char buffer[80];
	gets(buffer);
}
```

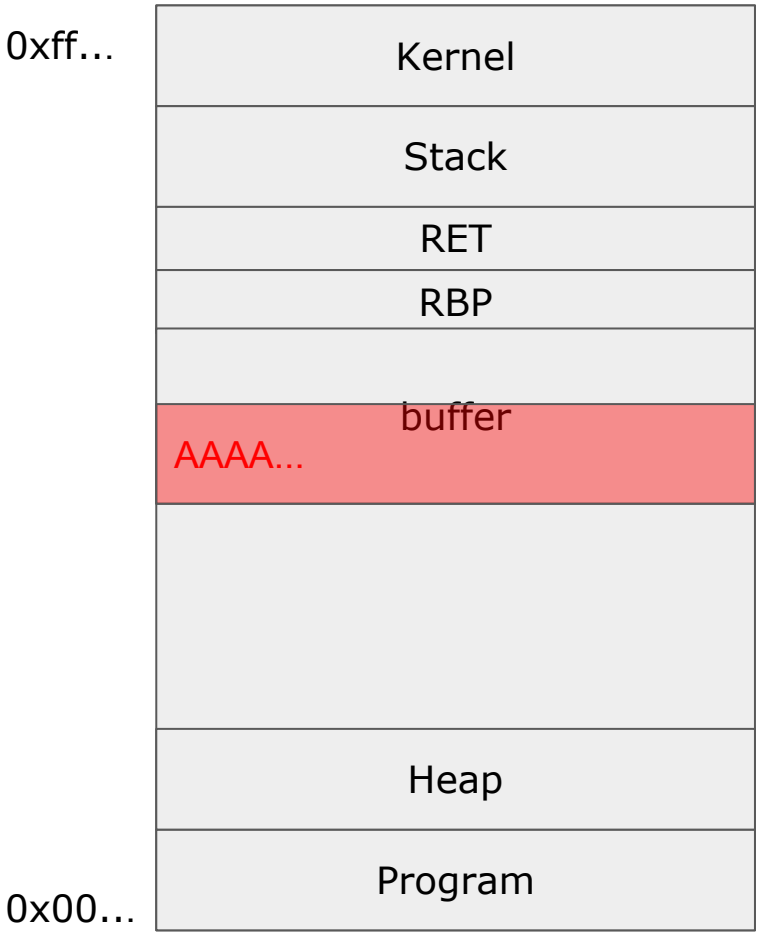
Vstup: 40 * “A”



Stack Smashing

```
void vuln()
{
	char buffer[80];
	gets(buffer);
}
```

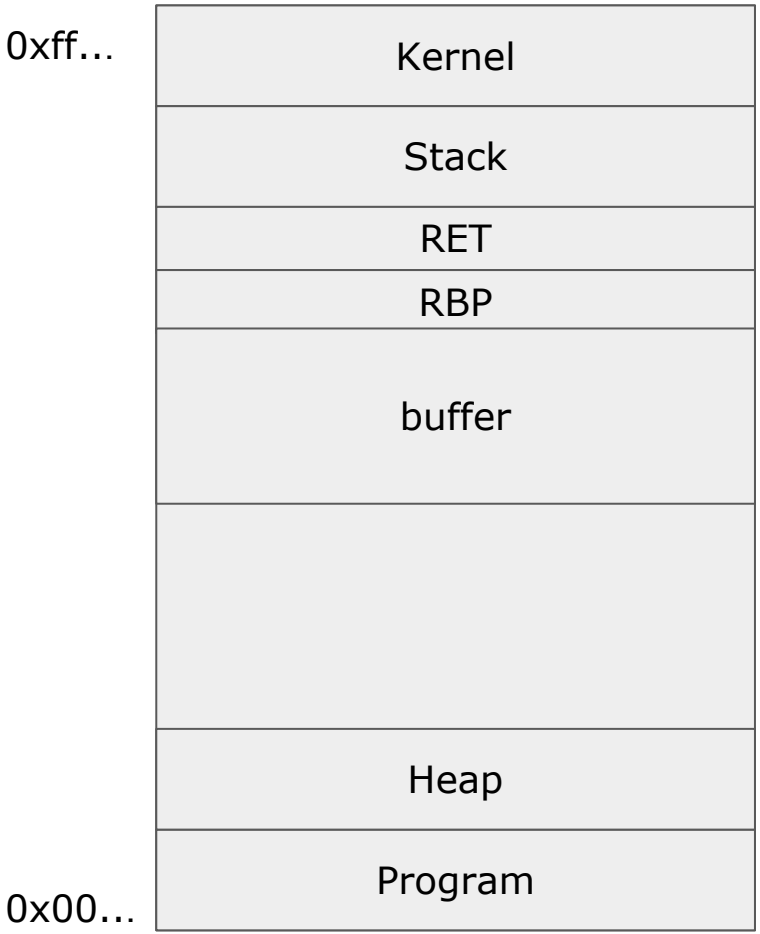
Vstup: 40 * “A”



Stack Smashing

```
void vuln()
{
	char buffer[80];
	gets(buffer);
}
```

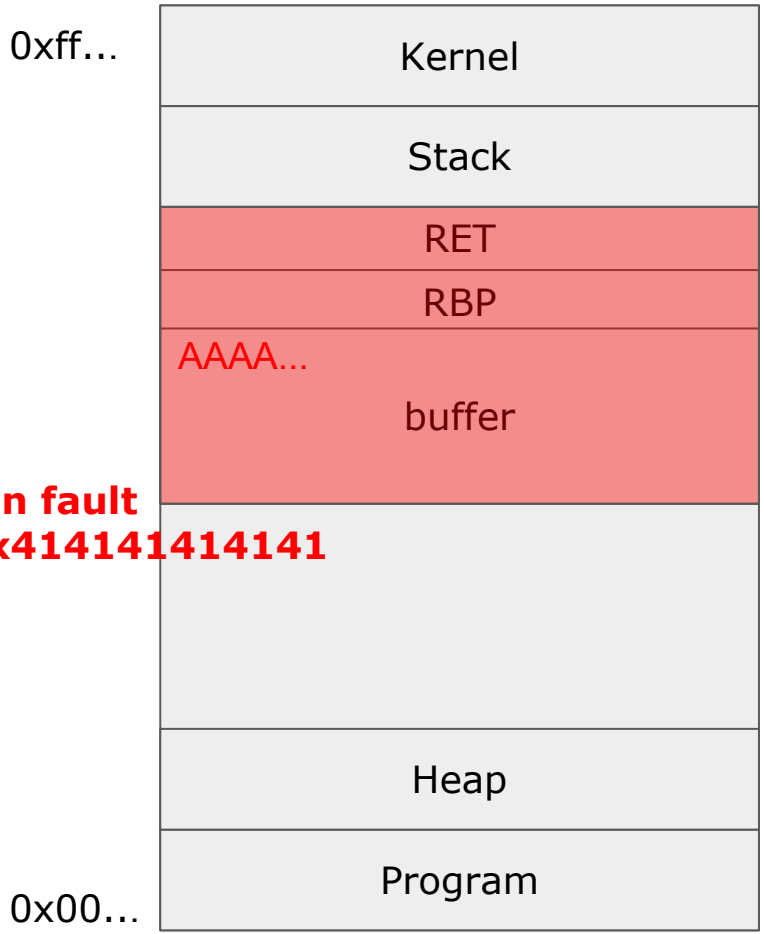
Vstup: 96 * “A”



Stack Smashing

```
void vuln()
{
        char buffer[80];
        gets(buffer);
}
```

Vstup: 94 * “A”



Segmentation fault
Invalid address 0x414141414141

Čo môžeme prepísať?

- Návratovú adresu
- Lokálne premenné
- Hodnotu v pamäti, ktorá ovplyvňuje matematické operácie
- Smerníky (čítanie/zápis)

Buffer overflow

Chyby pri počítaní veľkostí buffra, nebezpečné funkcie (gets, strcpy,...)

```
int main(int argc, char *argv[])
{
	char buffer[32];
	read(0, buffer, 128);
}
```

Signed/Unsigned mixup

Štandardná knižnica používa na definovanie veľkosti typ **unsigned**.

```
int main(int argc, char *argv[])
{
        char buffer[80];
        int size;
        scanf("%i", &size);
        if(size > 80)
                exit(1);
        read(0, buffer, size);
        return 0;
}
```

*Znamienka sa reprezentujú pomocou dvojkoveho doplnku: 0xffffffff = -1, 0xfffffffe = -2, ...

Integer overflow

Čo sa stane ak k maximálnej hodnote 32 bitového integeru (0xffffffff) pripočítam jednotku?

```
int main(int argc, char *argv[])
{
        unsigned int size;
        scanf("%i", &size);
        char *buff = alloca(size + 1);
        int n = read(0, buff, size);
        buf[n] = '\0';
}
```

Mitigácie voči exploitácii

- **ASLR** (**A**ddress **S**pace **L**ayout **R**andomization)
- **DEP** (**D**ata **E**xecution **P**revention) alebo **NX**
- **Stack Canaries**

ASLR

- Znáhodňovanie adries (offsety však ostávajú rovnaké)
- Plná sila ASLR je iba v kombináciu so zapnutým **PIE** (**P**osition **I**ndependent **E**xecutable) - platí pre sekciu s kódom programu.
- Metódy:
  - **Information Leak** - ak vieme dostať z programu ľubovoľnú adresu, tak sme vyhrali. Offsety ostávajú rovnaké a tým pádom vieme vypočítať base adresu.
  - **Partial Overwrite** - program má stránky zarovnané na 4096 B (0x1000). Spodné tri bajty ostávajú stále rovnaké -> 2^4 brute force.

    1.spustenie => **0x5571aef0b**123

    2.spustenie => **0x55b4a56b6**123

    3.spustenie => **0x562b131d4**123

  - **Brute Force** - podľa situácie - fork service.

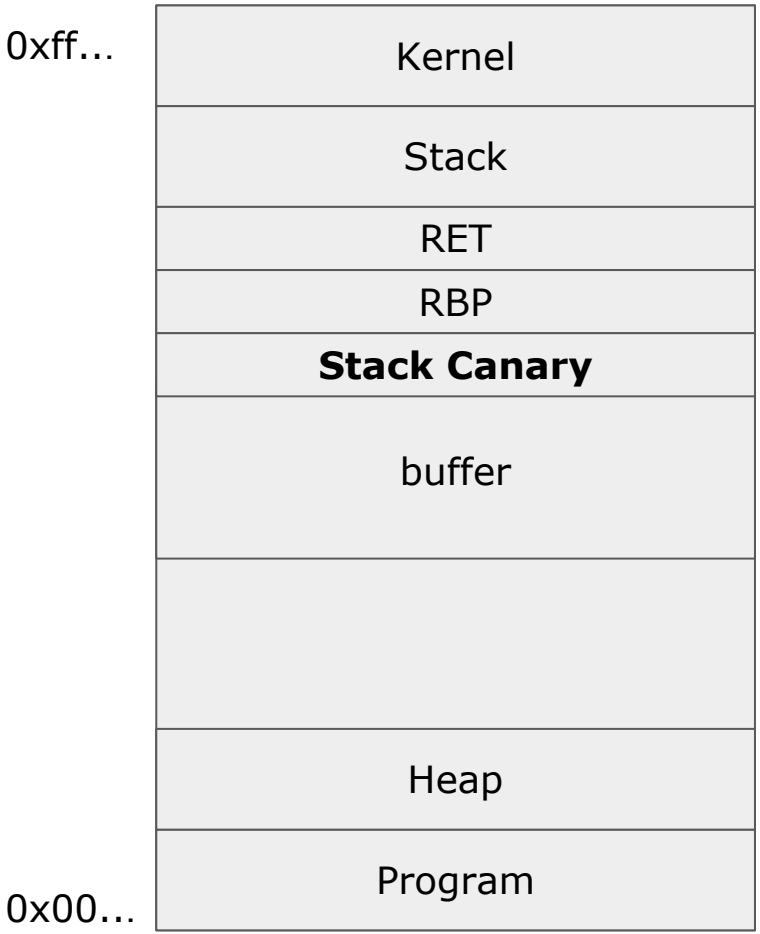
DEP (NX)

- Stránky v pamäti môžu mať rôzne oprávnenia (**R**ead **W**rite **E**xecute)
- Zásobník/Halda pri zapnutom NX nemajú **X** flag, t.j. nevieme spustiť shellcode :(
- Teaser: **R**eturn **O**riented **P**rogramming (ROP) -> ďalší blok

Stack Canary

Náhodná hodnota, ktorá chráni návratovú adresu.

0x**3740b6e89010db**00
0x**d23db590b34909**00
0x**b22e811c4f7a2c**00



Stack Canary

Čo s tým? :/

- **Information Leak**
- **Brute Force** - inteligentná verzia, skúšanie po bajtoch 7 * 255 pokusov. **POZOR**: iba pri fork servise.

Information Leak

- **Buffer overread**

```
char buffer[16] = {};
write(1, buffer, 128);
```

- **Zabudnutý NULL byte**:

```
char name[10] = {0};
char flag[10] = “bispp_flag{...”;
read(0, name, 10);
printf(“Hi %s!\n”, name);
```

pwntools

- Dokumentácia:

  https://python3-pwntools.readthedocs.io/en/latest/
- Tutoriál:

  https://github.com/Gallopsled/pwntools-tutorial