Buffer Overflow (Přetečení vyrovnávací paměti)

Buffer Overflow je kritická chyba v zabezpečení softwaru, která vzniká, pokud program nedokáže správně kontrolovat délku vstupních dat. Tato chyba umožňuje útočníkovi přepsat části paměti procesu, což může vést k pádu aplikace (DoS) nebo, v horším případě, ke **spuštění libovolného škodlivého kódu** (Remote Code Execution) s oprávněním dané aplikace.

Problém se týká především jazyků jako **C** a **C++**, které neprovádějí automatickou kontrolu hranic polí (array bounds checking) a umožňují přímou manipulaci s pamětí.

Jak Buffer Overflow funguje?

V operační paměti (RAM) jsou data uložena v přesně definovaných strukturách. Jednou z nejdůležitějších je **Zásobník (Stack)**. Zásobník obsahuje kromě proměnných také tzv. **Návratovou adresu (Return Address)**, která říká procesoru, kam se má vrátit po dokončení aktuální funkce.

```
1. **Očekávaný stav:** Program vyhradí 10 bajtů pro jméno uživatele.
2. **Útok:** Útočník pošle 50 bajtů dat.
3. **Přetečení:** Program zapíše prvních 10 bajtů do vyhrazeného prostoru a zbylých 40 bajtů "přeteče" do sousedních buněk.
4. **Přepsání adresy:** Útočník záměrně zformátuje data tak, aby přepsal návratovou adresu svou vlastní adresou, která míří na vložený škodlivý kód (shellcode).
```

Typy přetečení

Typ	Popis
Stack Overflow	Přetečení na zásobníku. Nejčastější cesta k ovládnutí programu.
Heap Overflow	Přetečení v haldě (dynamicky alokovaná paměť). Složitější na zneužití, ale velmi nebezpečné.
Integer Overflow	Aritmetická chyba, kdy výsledek operace přesáhne rozsah číselného typu, což následně vede k nesprávné alokaci bufferu.

Ukázka v jazyce C (Zranitelný kód)

Následující kód je klasickým příkladem chyby, protože funkce `gets()` nekontroluje délku vstupu:

```
void login() {
    char buffer[8]; // Vyhrazeno pouze 8 bajtů
    printf("Zadejte heslo: ");
    gets(buffer);   // NEBEZPEČNÉ: načte neomezeně dat
}
```

Pokud uživatel zadá více než 8 znaků, dojde k přetečení paměti.

Ochranné mechanismy

Moderní operační systémy a kompilátory obsahují několik vrstev ochrany, které zneužití těchto chyb ztěžují:

- **ASLR (Address Space Layout Randomization):** Náhodně mění umístění klíčových dat v paměti při každém spuštění programu. Útočník pak neví, kde se nachází jeho kód.
- **DEP / NX bit (Data Execution Prevention):** Označuje části paměti s daty jako „nespustitelné“. I když útočník kód do paměti vloží, procesor jej odmítne vykonat.
- **Stack Canaries:** Do paměti před návratovou adresu se vloží náhodná hodnota („kanárek“). Pokud je při ukončení funkce tato hodnota jiná (byla přepsána), program okamžitě skončí dříve, než se stihne spustit škodlivý kód.

Prevence při vývoji

Nejúčinnější obranou je psát bezpečný kód:

- Používat moderní jazyky (Rust, Go, Java, Python), které paměť spravují automaticky.
- V C/C++ nahradit nebezpečné funkce (`gets`, `strcpy`, `sprintf`) bezpečnými variantami (`fgets`, `strncpy`, `snprintf`), které vyžadují limit délky.
- Provádět pravidelné audity kódu a statickou analýzu.

Související pojmy: Malware, Exploit, Zásobník (Stack), Halda (Heap), ASLR, Shellcode, C++, Kybernetická bezpečnost.