Damian Koper, 241292
SR-TN-17
Wrocław, 22.05.2019r.

Lab 5: Wątki, alokacja pamięci, przepełnienie stosu

Cel ćwiczenia

- Utworzenie dwóch wątków w obrębie jednej przestrzeni adresowej
- Porównanie czasu wykonania operacji wypełniania tablicy "po" wierszach i kolumnach
- Symulacja ataku typu przepełnienie stosu
- Przydział pamięci funkcjami systemowymi

Wątki

Wątki są procesami (na poziomie kernela), które współdzielą wszystkie dane, poza stosem i rejestrami. Z tego powodu przełączanie kontekstu pomiędzy wątkami jest dużo tańsze, niż pomiędzy odseparowanymi procesami.

Ochrona sekcji krytycznej

Wątki, mając dostęp do tych samych obszarów pamięci mogą konkurować między sobą o jej zasoby, nieraz prowadząc do zjawisk zakleszczenia lub zagłodzenia wątku/procesu. Sposobem przeciwdziałania takim zjawiskom jest odpowiednia ochrona sekcji krytycznej - sekcji, w której wszystkie operacja muszą być traktowane jako jedna operacja atomowa.

W zależności od sytuacji istnieją różne sposoby ochrony sekcji krytycznej:

1. Operacje atomowe - operacje, które muszą wykonać się w całości, zanim ktokolwiek inny uzyska dostęp do używanych zasobów. W porzypadku pojedynczych rozkazów możliwe jest użycie prefixu `LOCK`, który blokuję magistralę na czas wykonywania operacji.
2. Mutexy (Mutual Exclusion, wspólne wykluczenie) - Zablokowanie dostępu do zasobu dla wszystkich, poza jednym z wątków, któremu to dostęp został przydzielony na podstawie zasady *uczciwości* lub kolejki *FIFO*.
    1. SpinLock - wątek w pętli oczekuje na dostęp do zasobu na podstawie flagi, która może być ustawiona przez inny wątek. Po wyjściu z sekcji krytycznej czyści flagę. Użyte są tu niepodzielne instrukcje test-and-set np. `LOCK BTS`.
    2. Semafor - chroniona zmienna lub struktura danych stanowiąca sposób kontroli dostępu do sekcji krytycznej.

Atak typu "Przepełnienie stosu"

Atak typu "przepełnienie stosu" bazuje na braku ochrony kluczowych fragmentów pamięci stosu. Może być wykonany poprzez zapisanie do zmiennej lokalnej (zaalokowanej na stosie) danych w ilości przekraczającej założony rozmiar, modyfikując i nadpisując kluczowe elementy ramki stosu np. adres powrotu. Może być również wykonany, kiedy dostęp do tablicy będzie bazował na dostarczonym przez użytkownika indeksie (ala SQL Injection). Modyfikując adres powrotu pozwalamy na zmianę ścieżki wykonania programu, co może doprowadzić do wykonania niedostępnej w danym momencie funkcji, a nawet do wykonania kodu dostarczonego przez użytkownika.

Obecnie atakowi temu przeciwdziała tzw. Stack Protector - wkładanie na stos i weryfikowanie znanej wartości w celu sprawdzenia jego integralności oraz technika ASLR - randomizowanie przestrzeni adresowej dla kluczowych elementów procesu. Dodatkowo linker ustawia flagę mówiącą, czy instrukcje zapisane na stosie mogą być wykonywane. Zachowanie to można włączyć przekazując do linkera flagę `-z execstack`.

Alokacja pamięci

Istnieją dwa główne metody alokacji pamięci o odmiennym działaniu

1. `mmap` - używana najczęściej, tworzy nowe odwzorowanie w wirtualnej przestrzeni adresowej procesu. Fizyczna pamięć nie jest zajmowana, a oznaczana jako gotowa do wykorzystania i przy pierwszym do niej zapisie jest alokowana przez system operacyjny.
2. `brk` - zmienia lokalizację punktu kończącego segment `.data` (`.bss`) programu, co skutkuje zaalokowaniem pamięci dla procesu.

Rozmiar wirtualny prezentuje wielkość jaką proces *"myśli, że zajmuje"* i do zawartości której może mieć dostęp. Wliczają się w to, prócz danych i kody danego procesu, np. załadowane biblioteki współdzielone.

Programy

- Kody programów `threads`, `arrays`, `memoryFiddle`: https://github.com/damiankoper/OiakLab/tree/master/lab_5

Wątki

Program `threads` za pomocą syscall'a `clone` tworzy dwa wątki ustawiając im odpowiednie flagi oraz blok pamięci stanowiący jego nowy stos, gdzie pierwszy adres stanowi wskaźnik na pierwszą instrukcję. Wątek utworzony syscall'em `clone`kontynuuje swoje działanie od momentu owego przerwania. Mając na szczycie stosu adres swojej funkcji, za pomocą funkcji `RET`może on wrócić/wejść do swojej głównej funkcji.

Wątki wypisują do `stdout` liczby od 9 do 0. W programie występuje zmienna `c`, która jest inkrementowana po stworzeniu wątku i dekrementowana po jego zakończeniu. Obie operacje poprzedzone są prefixem `LOCK`. Zmienna ta jest semaforem, informującym główny wątek, który czeka w `SpinnLock'u`, o zakończeniu wątków - dzieci.

Bez prefixu `LOCK` mogłoby dojść do sytuacji, gdzie oba wątki, dekrementując zmienną w tym samym czasie, ustawiają jej wartość na 1, co skutkowałoby zawieszeniem wątku głównego.

Jako, że wątki są traktowane jak procesy - trzeba zakończyć je używając przerwania systemowego `exit`.

Wynikiem działania programu jest ciąg cyfr w różnej kolejności, zależnej od kolejności pracy wątków np.:

```
987968574635241321
```

Przepełnienie stosu

Tę część zadania udało się zrealizować tylko w części. Nie udało się wykonać dostarczonego z zewnątrz kodu. Udało się natomiast doprowadzić do przepełniania bufora, nadpisania adresu powrodu funkcji `func` i

wykonania funkcji `topSecret`, która nie jest nigdzie wywoływana w programie. W celu uruchomienia programu z zamiarem osiągnięcia celu, należało wyłączyć opisane wyżej zabezpieczenia.

```
int func()
{
    char str[4];
    FILE *f = fopen("helloFromHello.out", "rb");
    if (f == NULL)
    {
        printf("Nie ma takiego pliku\n %d", errno);
    }
    else
    {
        fread(str, 1, 24, f);
    }
    return 0;
}
```

Disasemblując program wyliczone zostało miejsce przechowywania na stosie adresu powrotu dla funkcji `func` i na tej podstawie plik `helloFromHello.out` został odpowiednia spreparowany:

```
  Offset: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000: 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61
00000010: 61 61 61 61 0D 56 55 56
```

Cztery ostatnie bajty to zapisany od końca (*little endian*) adres funkcji `topSecret`, który nadpisuje adres powrotu do funkcji `main`. Skutkuje to wczytaniem pliku do pamięci, wykonaniem funkcji `topSecret` i odebraniem sygnału `SIGSEGV`.

Wypełnianie macierzy - pomiar czasu

Program `arrays` wypełnia macierz kwadratową liczbą `0`. Tabela przedstawia średnie zmierzone czasy (w cyklach procesora) wypełniania macierzy (w sekcji `.data`) iterując po wierszach i po kolumnach:

	Wiersze	**Kolumny**
AVG	8426024.032	10808873.792
STDV	0883212.547	01830653.867

Jak widać czas wypełniania po kolumnach jest o rząd wielkości większy, niż dla wypełniania po wierszach, co może wynikach z potrzeby kalkulacji adresu kolumny i większej ilości porównań.

Pamięć

Program `memoryFiddle` alokuje `0xef7fffff` bajtów za pomocą syscall'a `mmap` (`old_mmap`). `mmap` jako argument przyjmuje strukturę, gdzie podany jest rozmiar, różnego rodzaju parametry i flagi dla alokowanej pamięci:

```
mmap_arg_struct:
    .long 0
    .long SIZE
    .long PROT_READ | PROT_WRITE
    .long 34
    .long -1
    .long 0
```

Po wykonaniu tej części kodu rozmiar wirtualnej pamięci jest równy ok. `3.7GB`.

Część pliku `/proc/[procid]/smaps`

```
085d4000-f7dd4000 rwxp 00000000 00:00 0
Size:            3923968 kB
KernelPageSize:        4 kB
MMUPageSize:           4 kB
```

Następnie w pętli zaalokowana pamięć wypełniana jest wartością `-1`. Wraz z kolejnymi iteracjami wielkość `Resident Memory` wzrasta, co świadczy o tym, że adresom wirtualnym przypisywany jest adres fizyczny w pamięci.

Maksymalny rozmiar możliwy do zaalokowania dla architektury 32b wynosi teoretycznie `2^32-1 ~4GB`. Alokując jednak tak dużą pamięć nadpiszemy kluczowe dla procesu struktury - dane, stos itd.

Sprawdzając możliwą ilość zasobów do zaalokowania przerwaniem `getrlimit` z flagą `RLIMIT_AS`, otrzymamy w wyniku `0x7fffffff`, co jest równe w przybliżeniu `2GB`, co nie zgadza się z praktyką.

```
mov $SYS_getrlimit, %eax
mov $RLIMIT_AS, %ebx
mov $rlimitSoft, %ecx
int $0x80
```

Komenda `ulimit -v` (uruchomiona w shellu 64b) daje wynik `unlimited`.

Wnioski

Pomimo zastosowania semafora, wątek główny w programie `threads` kończył sie nieraz przed wypisaniem wszystkich cyfr. Nie udało się zweryfikować przyczyny tego problemu.

Używając różnych metod uzyskania wartości limitu możliwych do przydzielenia zasobów, uzyskano różne wartości. Górnym ograniczeniem jednak jest rozmiar słowa adresowego, który w tym przypadku ma 32b.

Literatura

1. Raw Linux Threads via System Calls - https://nullprogram.com/blog/2015/05/15/
2. Buffer Overflow - https://dhavalkapil.com/blogs/Buffer-Overflow-Exploit/

3. Pamięć - https://softwareengineering.stackexchange.com/questions/207386/how-are-the-size-of-the-stack-and-heap-limited-by-the-os