

Debugowanie niskopoziomowe z Pwndbg

GDB, czyli GNU Debugger [0], to potężne narzędzie do debugowania aplikacji. Pozwala ono na dynamiczną analizę kodu źródłowego i natywnego; procesów użytkownika, jak i jądra systemów. GDB jest jednak nieco „toporny”: niektóre z jego komend są niewygodne w użyciu, a tekstowy interfejs (TUI) mógłby wyświetlać więcej informacji. Sytuację tę poprawia tytułowe Pwndbg – plugin do GDB, z którym zapoznamy się w tym artykule.

I DO CZEGO MI TO PWNDBG?

Pwndbg przydaje się do debugowania niskopoziomowych programów, inżynierii wstecznej czy eksploatacji binarek. Plugin ten dostarcza kolorowy i czytelny interfejs, automatycznie odczytuje wskaźniki¹ i pokazuje, co się pod nimi znajduje. Ponadto udostępnia sporo komend, których brakuje w standardowym GDB. Narzędzie to jest idealne dla uczestników zawodów CTF (Capture The Flag), a także dla badaczy bezpieczeństwa, którzy potrzebują debugować niskopoziomowy kod w swojej codziennej pracy.

I INSTALACJA

Instalacja Pwndbg jest prosta – robimy to, klonując repozytorium i uruchamiając skrypt instalacyjny: `git clone https://github.com/pwndbg/pwndbg.git` && `cd pwndbg` && `./setup.sh`. Instalator ten działa na wielu dystrybucjach Linuxa, w tym m.in. Debianie, Fedorze, Arch Linuxie. Zainstaluje on odpowiednie pakiety systemowe (GDB, interpreter Pythona czy symbole debugowe do libc), a także doda do pliku `~/.gdbinit` jednolinijkowy kod, aby automatycznie ładować plugin Pwndbg podczas uruchamiania GDB.

Ostatnia opublikowana wersja Pwndbg składa się z paczek (.deb, .rpm, .apk) na niektóre dystrybucje, które zawierają wszystkie potrzebne zależności w odpowiednich wersjach², w tym najnowszą wersję GDB [1].

Uruchomienie GDB z Pwndbg powinno wyglądać jak na Rysunku 1, gdzie przywita nas informacja o załadowanym Pwndbg, losowa porada odnośnie GDB/Pwndbg oraz znak zachęty pwndbg>.

```

root@pwndbg: ~/pwndbg
root@pwndbg:~/pwndbg# gdb --quiet
pwndbg: loaded 147 pwndbg commands and 47 shell commands. Type pwndbg [--shell |
--all] [filter] for a list.
pwndbg: created $rebase, $ida GDB functions (can be used with print/break)
----- tip of the day (disable with set show-tips off) -----
Want to NOP some instructions? Use patch <address> 'nop; nop; nop;'
pwndbg>

```

Rysunek 1. Pierwsze uruchomienie Pwndbg

Jeśli nigdy wcześniej nie korzystaliśmy z GDB, warto zapoznać się z następującymi komendami:

- » **file <plik>** – ładuje dany program, aby później go debugować.
- » **run [<argumenty...>]** – startuje program z podanymi argumentami.
- » **starti [<argumenty...>]** – startuje program z podanymi argumentami, ale zatrzymuje jego działanie na pierwszej wykonywanej instrukcji. W przypadku programów linkowanych dynamicznie będzie to pierwsza instrukcja linkera³.
- » **break <nazwa funkcji lub *adres>** – ustawienia punktu, w którym ma zatrzymać się nasz program, tak zwany breakpoint.
- » Działający program możemy też przerwać, wysyłając do GDB odpowiedni sygnał poprzez wciśnięcie CTRL+C na klawiaturze.
- » **info breakpoint** – listuje wszystkie breakpointy.
- » **stepli oraz nexti** (lub **si** i **ni**) – inaczej „step/next instruction”, pozwalają przeskoczyć do następnej wykonywanej instrukcji, wchodząc do wywoływanych funkcji (**si**) lub przeskakując je (**ni**).
- » **step oraz next** (lub **s** i **n**) – podobnie jak **stepli** oraz **nexti**, tylko zamiast przeskakiwać instrukcje, przeskakują linie kodu źródłowego, pod warunkiem że mamy symbole debugowe.
- » **continue** (lub skrótowno **c**) – kontynuuje program (po zatrzymaniu się na jakimś breakpointie).
- » **print <wyrażenie>** – wypisze wynik danego wyrażenia, np.: **print zmienna** wypisze daną zmienną, **print \$rip** wypisze wartość rejestru RIP (na x86-64), a **print *(uint64_t*)\$rsp** wyświetli wartość pamięci pod adresem z rejestru RSP jako liczbę o typie `uint64_t`.
- » **x/<format> <adres>** – wyświetli daną pamięć w danym formacie, np. **x/10i \$rip** wyświetli 10 instrukcji spod adresu rejestru RIP, a **x/4xg \$rsp** wyświetli 4 wartości w formacie heksadecymalnym (literka **x**) o wielkości 8 bajtów (literka **g**). Informację o różnych formatach możemy wyświetlić, wpisując komendę **help x**.
- » **backtrace** (lub **bt**) – wyświetla stacktrace programu, czyli listę kolejnych funkcji, w której znajduje się debugowany wątek. Informacja ta pochodzi z GDB, który oblicza ją na podstawie ramek stosu.
- » **up [<n>], down [<n>]** – przełącza nas pomiędzy kolejnymi ramkami stosu, które są wyświetlane przez **backtrace**. Analiza programu z symbolami debugowymi umożliwia inspekcję zmiennych lokalnych czy argumentów funkcji.
- » GDB możemy konfigurować, ustawiając różne parametry. Li-

1. Wskaźnik, czyli zmienne przechowujące adres. W niektórych językach programowania referencje pełnią rolę wskaźników.

2. Takie samowystarczalne pakiety są rzadko spotykane, gdyż nie mają żadnych zależności, które znajdują się w paczkach danej dystrybucji. Ma to dwie zalety: 1) pozwala na pełną instalację Pwndbg na systemach bez Internetu oraz 2) zapewnia, że wszystkie zależności mają odpowiednie wersje, niezależnie od używanego systemu.

3. Konkretniej, to „interpreter”, który ładuje program i jego biblioteki. Ścieżkę interpretera dla danego programu dynamicznie linkowanego możemy wyświetlić za pomocą shellowej komendy: `readelf -a <program> | grep interpreter`.

```

root@pwndbg: ~
pwndbg> ctx
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
[ REGISTERS / show-flags off / show-compact-regs off ]
*RAX 0x0
RBX 0x0
RCX 0x555555557dc0 (__do_global_dtors_aux_fini_array_entry) -> 0x55555555100 (__do_global_dtors_aux) <- endbr64
*RDX 0x14
*RDI 0x0
*RSI 0x555555556004 <- 'Hello world'
R8 0x7ffff7fa2f10 (initial+16) <- 0x4
R9 0x7ffff7fc9040 (_dl_fini) <- endbr64
R10 0x7ffff7fc3908 <- 0xd0012000000e
R11 0x7ffff7fde680 (_dl_audit_preinit) <- endbr64
R12 0x7ffff7ffe4b8 -> 0x7ffff7ffe723 <- '/root/a.out'
R13 0x55555555149 (main) <- endbr64
R14 0x555555557dc0 (__do_global_dtors_aux_fini_array_entry) -> 0x55555555100 (__do_global_dtors_aux) <- endbr64
R15 0x7ffff7fd040 (_rtld_global) -> 0x7ffff7ffe2e0 -> 0x555555554000 <- 0x10102464c457f
RBP 0x7ffff7fe3a0 <- 0x1
RSP 0x7ffff7fe3a0 <- 0x1
*RIP 0x5555555516a (main+33) <- call 0x55555555050
[ DISASM / x86-64 / set emulate on ]
0x55555555151 <main+8> mov edx, 0x14
0x55555555156 <main+13> lea rax, [rip + 0xea7]
0x5555555515d <main+20> mov rsi, rax
0x55555555160 <main+23> mov edi, 0
0x55555555165 <main+28> mov eax, 0
> 0x5555555516a <main+33> call write@plt <write@plt>
fd: 0x0 (/dev/pts/0)
buf: 0x555555556004 <- 'Hello world'
n: 0x14
0x5555555516f <main+38> mov eax, 0
0x55555555174 <main+43> pop rbp
0x55555555175 <main+44> ret
0x55555555176 add byte ptr [rax], al
0x55555555178 <_fini> endbr64
[ STACK ]
00:0000 | rbp rsp 0x7ffff7fe3a0 <- 0x1
01:0008 | 0x7ffff7fe3a8 -> 0x7ffff7db1d90 (__libc_start_call_main+128) <- mov edi, eax
02:0010 | 0x7ffff7fe3b0 <- 0x0
03:0018 | 0x7ffff7fe3b8 -> 0x55555555149 (main) <- endbr64
04:0020 | 0x7ffff7fe3c0 <- 0x1fffffe4a0
05:0028 | 0x7ffff7fe3c8 -> 0x7ffff7ffe4b8 -> 0x7ffff7ffe723 <- '/root/a.out'
06:0030 | 0x7ffff7fe3d0 <- 0x0
07:0038 | 0x7ffff7fe3d8 <- 0x375ae7637563d7a2
[ BACKTRACE ]
> 0 0x5555555516a main+33
1 0x7ffff7db1d90 __libc_start_call_main+128
2 0x7ffff7db1e40 __libc_start_main+128
3 0x55555555085 _start+37
pwndbg>

```

Rysunek 2. Przykładowy kontekst wyświetlany przez Pwndbg

stę parametrów można wyświetlić za pomocą komendy **show**; wartość konkretnego parametru komendą **show <parametr>**, a ustawiać parametry komendą **set <parametr> <wartość>**. Komendy te przydają się również do konfigurowania Pwndbg.

Oczywiście funkcjonalności GDB jest dużo więcej i Pwndbg dodaje również nowe komendy, które zostaną omówione w dalszej części artykułu. Funkcjonalności wewnątrz GDB można też szukać poleceniem **apropos <fraz>** lub **pwndbg <fraz>** w przypadku komend Pwndbg.

I KONFIGURACJA

Pwndbg definiuje różne parametry GDB, które możemy skonfigurować. Można je wyświetlić za pomocą komend **config**, **theme** oraz **heap_config**. Pozwalają one na ustawienie m.in.:

- » Ile adresów ma być dereferencjonowanych przy funkcjonalności „teleskopu”.

- » Ile linii mają wyświetlać poszczególne wyświetlane sekcje (rejestrów, kodu, deasemblacji itd.). Czy sekcje mają być wyświetlane na standardowe wyjście czy może do innego terminala.
- » W jakich kolorach mają być wyświetlane dane wartości.
- » Oraz wielu innych.

Pwndbg zmienia również następujące parametry GDB:

```

set confirm off
set verbose off
set pagination off
set height 0
set history save on
set follow-fork-mode child
set backtrace past-main on
set step-mode on
set print pretty on
set width {width}
handle SIGALRM nostop print nopass
handle SIGBUS stop print nopass
handle SIGPIPE nostop print nopass
handle SIGSEGV stop print nopass

```

Magazyn programistów i liderów zespołów IT

programista

4/2023 (109)

Cena 28,90 zł (w tym VAT 8%)

.02

.01

DEBUGOWANIE
NISKOPOZIOMOWE
Z Pwndbg

.04

.03

CO PRZYNIESIE C++23?

WŁASNY ASYSTENT AI

DWUKIERUNKOWE TESTY
KONTRAKTOWESEKRETY W ŚRODOWISKU
PROGRAMISTYCZNYM

Kup książkę

JUŻ W SPRZEDAŻY W EMPIKACH

ISSN 2084-9400

04



9 772084 940305