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.

DO CZEGO MI TO PWNDBG?

Pwndbg przydaje się do debugowania niskopoziomowych programów, inżynierii wstecznej czy eksploitacji binarek. Plugin ten dostarcza kolorowy i czytelny interfejs, automatycznie odczytuje wskaźniki[1] 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.

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 paczki 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 wersjach[2], 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>`.

Rysunek 1. Pierwsze uruchomienie Pwndbg

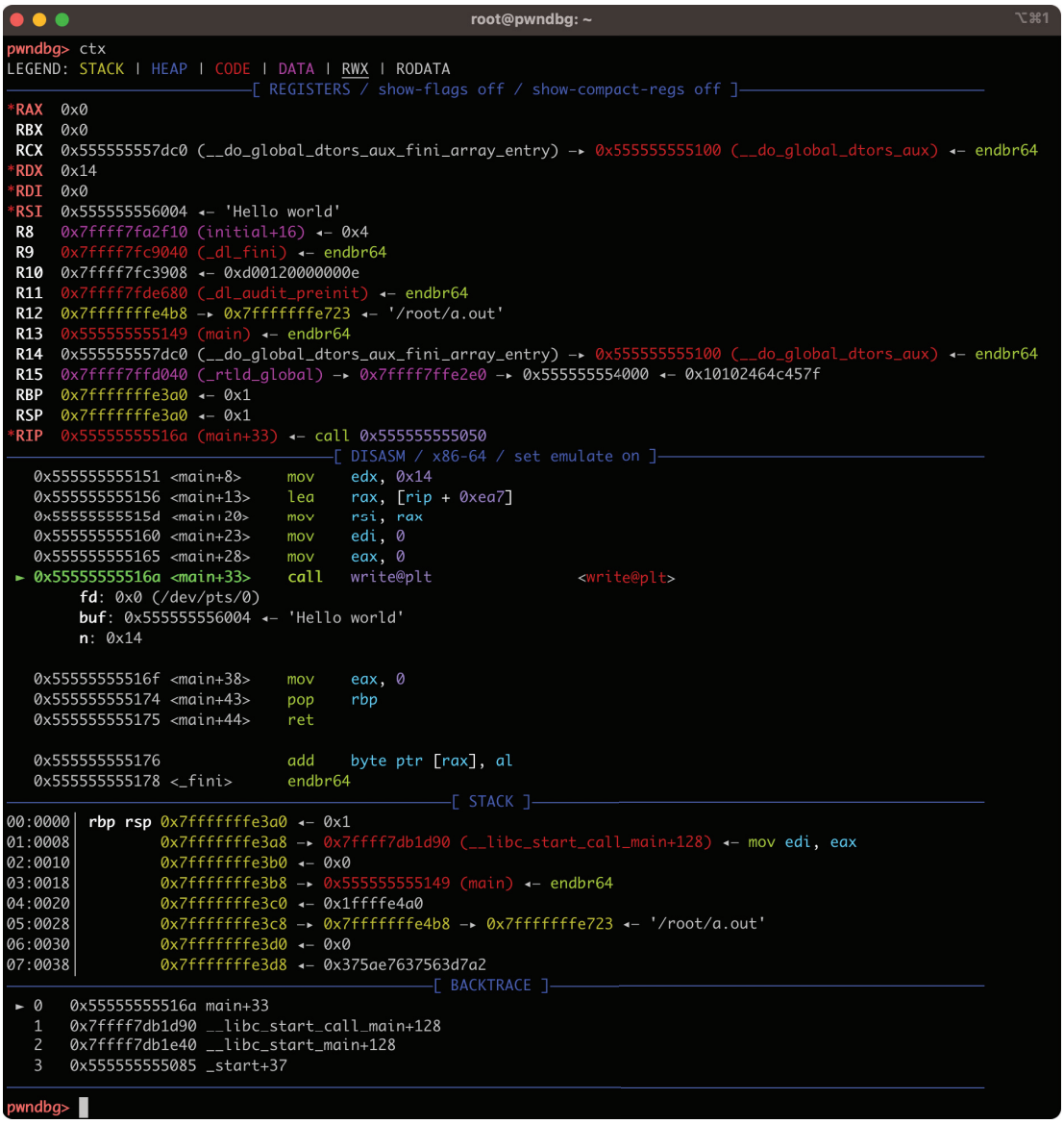
Jeśli nigdy wcześniej nie korzystaliśmy z GDB, warto zapoznać się z następującymi komendami:

- **`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 linkera[3].
- **`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.
- **`stepi` 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 **`stepi`** oraz **`nexti`**, tylko zamiast przeskakiwać instrukcje, przeskakują linie kodu źródłowego, pod warunkiem że mamy symbole debugowe.
- **`continue`** (lub skrótowo **`c`**) – kontynuuje program (po zatrzymaniu się na jakimś breakpoincie).
- **`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 paczki 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 „interpretera", który ładuje program i jego biblioteki. Ścieżkę interpretera dla danego programu dynamicznie linkowanego możemy wyświetlić za pomocą shellowej komendy: `readelf -a <program> | grep interpreter`.



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 <fraza>** lub **pwndbg <fraza>** w przypadku komend Pwndbg.

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
```

CO NOWEGO W JAVA 21? | FREERTOS OD ŚRODKA

Index: 285358

www.programistamag.pl

Magazyn programistów i liderów zespołów IT

programista

4/2023 (109)

Cena 28,90 zł (w tym VAT 8%)

.01

DEBUGOWANIE NISKOPOZIOMOWE Z Pwndbg

.02

.03

.04

CO PRZYNIESIE C++23?

WŁASNY ASYSTENT AI

DWUKIERUNKOWE TESTY KONTRAKTOWE

SEKRETY W ŚRODOWISKU PROGRAMISTYCZNYM

JUŻ W SPRZEDAŻY W EMPIKACH

ISSN 2084-9400

04

9 772084 940305