Sistemski poziv *ptrace* i njegova uloga u radu debagera

Seminarski rad u okviru kursa
Verifikacija softvera
Matematički fakultet

Nikola Dimitrijević, 1086/2017
nikoladim95@gmail.com

6. decembar 2018

Sažetak

ptrace je sistemski poziv u Unix i njemu sličnim operativnim sistemima. Ime je skraćenica od "proces tragač" (eng. *process trace*). Korišćenjem *ptrace* jedan proces može da kontroliše drugi. Time upravljački proces ima mogućnost da upravlja unutrašnjim stanjem ciljanog procesa. *ptrace* najviše koriste debageri kako bi mogli da zaustavljaju program, da posmatraju memoriju i menjaju je.

Sadržaj

1 Uvod

Sistemski poziv *ptrace* pruža mogućnost da jedan proces posmatra i kontroliše izvršavanje drugog procesa kao i da čita i menja memoriju i registre traženog procesa. Za proces koji upravlja drugim se koristi termin tragač (eng. *tracer*), a proces kojim on upravlja se naziva traženi (eng. *tracee*) proces.

Koristi se za implementaciju debagera i praćenja sistemskih poziva. Prvo je potrebno da traženi proces bude povezan na tragača. Povezivanje i sve prateće komande se zasebno rade po niti procesa. *ptrace* komande se uvek šalju određenom traženom procesu u sledećem formatu:

```
ptrace(enum zahtev, pid_t proces, void* adresa, void* podaci)
```

Argumenti adresa i podaci se koriste ili ignorišu u zavisnosti od vrste zahteva.

2 Primene

Debageri poput *gdb* koriste *ptrace*, kao i alati *ltrace*, *strace* i alati za računanje test pokrivenosti koda. Povezivanjem na drugi proces korišćenjem *ptrace* roditeljski proces dobija veliku kontrolu nad ciljanim procesom. Tu spada:

- desktriptori datoteka
- memorija
- registri
- izvršavanje instrukciju po instrukciju
- posmatranje i presretanje sistemskih poziva
- uvid u povratne vrednosti sistemskih poziva
- manipulisanje upravljačem signala
- primanje i slanje signala umesto procesa

Menjanjem memorije programa se, pored menjanja podataka programa, može menjati i segmenat koda. Na taj način kontroler proces može da ubaci tačke prekida (eng. *brakepoints*) i da izmeni instrukcije programa prilikom njegovog izvršavanja.

Pomenuti alat *ltrace* služi za prikaz poziva koje program šalje deljenim bibliotekama. To radi tako što se zakači na sistem za dinamičko punjenje pa tako može da vidi funkcije, parametre i povratne vrednosti poziva biblioteka [5].

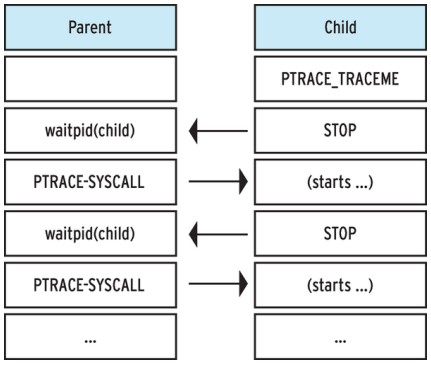
Alat *strace* se koristi za debagovanje i dijagnostiku na Linuks operativnom sistemu. Može da vidi i barata sa interakcijama između procesa i Linuks jezgra, što uključuje sistemske pozive, dostavljanja signala i menjanje stanja procesa [7]. Oba alata pokreću zadati program do kraja izvršavanja.

Debagerima je *ptrace* poput švajcarskog nožića, ali kako zapravo debageri postavljaju tačke prekida? Oni zamene instrukciju koja bi trebalo da se izvrši sa instrukcijom zamke (eng. *trap instruction*) [2]. Kada se traženi proces zaustavi onda program tragač, debager, može da ga proučava. Kada je potrebno da traženi program nastavi izvršavanje onda će se umesto tačke prekida vratiti prvobitna instrukcija na svoje mesto.

3 Upotreba i implementacija

ptrace ima mnogo vrsta zahteva, man stranica je velika, a opis svih njegovih detalja bi mogao da ispuni osrednju knjigu. Zato će ovde biti opisane glavne vrste *ptrace* zahteva.

3.1 Povezivanje



Slika 1: Dijagram toka povezivanja procesa.

Tragač može da se poveže na nit korišćenjem poziva:

```
ptrace(PTRACE_ATTACH, pid, 0, 0);
```

ili

```
ptrace(PTRACE_SEIZE, pid, 0, PTRACE_O_FLAGS);
```

PTRACE_ATTACH šalje signal SIGSTOP ciljanoj niti [6]. Novije verzije Linux-a imaju opciju PTRACE_SEIZE umesto PTRACE_ATTACH. PTRACE_SEIZE ne zaustavlja prikačeni proces.

Ako je potrebno zaustavljanje nakon priključenja može se pozvati ptrace sa opcijom PTRACE_INTERRUPT.

Jedini ptrace poziv koji traženi proces koristi je PTRACE_TRACEME, sve ostale pozive koristi samo tragač. PTRACE_TRACEME govori roditelju procesu da želi da bude praćen. Dobra praksa je da se nakon PTRACE_TRACEME pozove

```
raise(SIGSTOP);
```

i tako dopusti roditeljskom procesu, koji je sada tragač, da posmatra dostavljanje signala. Dakle roditeljski proces može da započne traganje, ali i dete proces može da to zatraži od roditelja. Ne bi trebalo da dete proces pozive PTRACE_TRACEME ako roditeljski proces to ne očekuje. Nakon pozivanja TRACEME program nastavlja sa izvršavanjem, ali svaki sledeći signal dostavljen tom procesu će uslediti njegovom zaustavljanju i njegov roditeljski proces će biti obavešten putem wait(). Jedino signal SIGKILL neće biti prvo prosleđen roditelju nego će se proces zaustaviti svakako. Za određeni proces samo jedan proces može da mu bude tragač u bilo kom trenutku.

U slučaju da je nad procesom pozvan *ptrace* sa PTRACE_ATTACH onda traženi proces nastavlja sa izvršavanjem sve dok ne uđe u sistemski poziv, u tom trenutku ga zaustavlja Linux jezgro. Traženom procesu to

izgleda kao da je zaustavljen jer je primio SIGTRAP signal. Sada tragač može da radi šta mu je volja. Tok povezivanja prikazan je na slici 1.

3.2 Vrste zahteva

U sekciji 3.1 su uvedeni ATTACH, TRACEME i INTERRUPT i SEIZE tako da ovde neće biti ponovljen opis.

PTRACE_SINGLESTEP zahtev je jedan od najinteresantnijih. Takav zahtev govori operativnom sistemu: *molim te ponovo pokreni dete proces, ali zaustavi ga čim završi izvršavanje sledeće instrukcije.*

PTRACE_PEEKTEXT, PTRACE_PEEKDATA zahtevi čitaju memoriju traženog procesa sa date adrese. Moguće je čitati podatke ali i izvršni kod programa.

PTRACE_POKETEXT, PTRACE_POKEDATA zahtevi kopiraju date podatke na zadatu adresu u memoriju traženog procesa. Ova dva zahteva su ekvivalentna pošto Linux nema odvojene segmente za kod i podatke [6]. Isto važi i za prethodna dva zahteva.

PTRACE_GETREGS, PTRACE_GETFPREGS zahtevi kopiraju registre opšte namene, odnosno registre za operacije u pokretnom zarezu traženog procesa.

PTRACE_SETREGS, PTRACE_SETFPREGS zahtevi postavljaju vrednosti registara traženog procesa.

PTRACE_GETSIGINFO daje informaciju o signalu zbog kog se traženi program zaustavio.

PTRACE_CONT zahtevom se nastavlja izvršavanje traženog procesa i po želji se dopušta ili zabranjuje primanje signala.

3.3 Ispod haube

Kako je kod javno dostupan [4], može se pogledati implementacija poziva *ptrace*.

```
if (request == PTRACE_ATTACH || request == PTRACE_SEIZE) {
        ret = ptrace_attach(child, request, addr, data);
        /*
        * Some architectures need to do book-keeping after
        * a ptrace attach.
        */
        if (!ret)
                arch_ptrace_attach(child);
        goto out_put_task_struct;
}
```

Nakon provere da li je zahtev PTRACE_ATTACH se poziva ta funkcija. Ona prvo postavlja oznake koje će posle biti zabeležene u jezgru kao reprezentacija traženog procesa. Proverava da traženi zadatak (eng. *task*) nije nit jezgra. Takođe proverava da se ne pokušava povezivanje procesa sa samim sobom. Sada je traženi proces zaustavljen. Na kraju *ptrace* zove arch_ptrace_attach funkciju koja zavisi od procesora, tako da je implementaicja usko vezana za konkretnu arhitekturu.

Postavljena je oznaka da je proces tražen, sad je pitanje kada se ta oznaka proverava prilikom sistemskih poziva. Svaki put kada program napravi sistemski poziv, postoji kod zavisan od arhitekture procesora koji se izvrši na strani jezgra pre nego što se izvrši sistemski poziv.

U x86 asemblerskoj implementaciji se nalazi provera pomenute postavljene oznake[1]. Tako se vidi da se pri svakom sistemskom pozivu vrši ta

provera. U slučaju da je oznaka postavljena poziva se nekoliko funkcija i konačno se okida signal SIGTRAP. Tragač je tada obavešten o primljenom signalu, a traženi proces je zaustavljen. Sada je tragač u stanju da ispituje unutrašnjost traženog procesa.

3.4 Povratna vrednost

Pri uspehu, PTRACE_PEEK zahtevi vraćaju tražene podatke, dok ostali zahtevi vraćaju nulu. Svi zahtevi vraćaju -1 pri neuspehu i *errno* bude postavljen na odgovarajuću vrednost. Tip povratne vrednosti je long.

3.5 Bezbednost

Operativni sistemi pružaju usluge programima preko standardnog API-a za pristup hardveru i drugi sistemima niskog nivoa poput sistema datoteka. Kada proces hoće da zove sistemski poziv prvo postavlja svoje argumente u registre i zove softverski prekid. Ovaj prekid govori jezgru da proveri argumente i potom da izvrši sistemski poziv.

Dinamičko ubacivanje koda se koristi za omogućavanje debagovanja, ali se isto može koristiti i za zlonamerne aktivnosti ako napadač ima privilegije da pokrene *ptrace*. Postoji način da se dobiju veće privilegije i pristupi zabranjenim delovima tako što se *ptrace* koristi zajedno sa bagom u sinhronizaciji zbog čega kernel napravi nit na nebezbedan način [3]. Ovaj način napda je popravljen 2003.

4 Zaključak

ptrace je neverovatno koristan sistemski poziv za debagere, tragače i druge sistemske programe koji treba da kupe korisne informacije iz programa. Spoznavanje svih detalja implementacije je prilično teško pošto zavisi od arhitekture procesora, podeljena je u više datoteka od kojih su neke pisane u jeziku C, a neke u asembleru. Srećom nije neophodno znati sve deliće koda da bi se razumelo šta *ptrace* radi, a to znanje demistifikuje rad debagera i pruža uvid u različite aspekte operativnog sistema.

Literatura

[1] How does strace work? https://blog.packagecloud.io/eng/2016/02/29/how-does-strace-work/.

[2] Linux journal. https://www.linuxjournal.com/article/6210.

[3] Linux local privilege escalation. https://www.exploit-db.com/exploits/3/.

[4] linux/kernel/ptrace.c. https://github.com/torvalds/linux/blob/v3.13/kernel/ptrace.c#L1036.

[5] ltrace man page. https://linux.die.net/man/1/ltrace.

[6] ptrace man page. http://man7.org/linux/man-pages/man2/ptrace.2.html.

[7] strace man page. https://linux.die.net/man/1/strace.