Spis treści

Spis tablic

1 O autorze

Marcin Świnoga
praca przygotowana na zajęcia z Systemów Operacyjnych
Wydział MIM UW
18 XII 2000 Warszawa

2 Wstęp

Dokument ten opisuje struktury danych oraz algorytmy używane przez jądro 2.2-17 systemu operacyunego Linux do obsługi plików wykonywalnych, takich jak pliki binarne, skrypty, czy inne pliki interpretowane.

2.1 Typy plików wykonywalnych

2.1.1 Pliki binarne wspierane przez jądro

Obecnie jądro linuksa może obsługiwać pliki binarne w dwóch formatach:

- w formacie ELF
- w formacie AOUT

Zdecydowana większość nowo powstałych programów tworzona jest jednak w postaci nowszego formatu *ELF*. W dokumencie tym opisano ów format.

2.1.2 Skrypty

W jądrze jest również wsparcie dla skryptów, to znaczy plików tekstowych w specjalnym formacie: pierwsza linia ma szczególne znaczenie w plikach skryptowych - wskazuje ona jaki program ma być uruchomiony, aby można było go zinterpretować[1]. Kolejne linie pliku zawierają instrukcje zrozumiałe dla wskazanego wcześniej programu interpretującego.

2.1.3 Inne pliki wykonywalne

Wyżej wspomniane *skrypty* niosą informację o programie je interpretującym w pierwszej linii tekstu. Wiemy jednak, że niektóre typy plików można poznać na przykład po ich rozszerzeniu. Przykładem mogą tutaj być różnego rodzaju pliki multimedialne. Mamy tutaj charakterystyczne rozszerzenia, takie jak .wav, .mp3, .avi, .mpg. Takie rozszerzenie nie musi być oczywiście jednoznaczne. Istnieje inny/dodatkowy sposób rozpoznania typu pliku: na podstawie nagłówka pliku - w ustalonym miejscu przy początku pliku zapisywany jest ustalony ciąg bitów o ustalonej długości.

Linuks po odpowiedniej kompilacji jądra i konfiguracji systemu rozpoznaje takie "wykonalne"pliku binarne na podstawie rozszerzeń oraz charakterystycznych ciągów bitów.

[1]proces przetwarzania danych wejściowych mający na celu wykonanie określonych tam operacji

3 Struktury danych i funkcje niezależne od typu pliku

3.1 Struktury danych

3.1.1 linux_binfmt

Zdefiniowane są tutaj funkcje używane do wczytywania plików wykonywalnych akceptowanych przez Linuks.

```
struct linux_binfmt {
    struct linux_binfmt * next;
    struct module *module;
    int (*load_binary)(struct linux_binprm *, struct  pt_regs * regs);
    int (*load_shlib)(int fd);
    int (*core_dump)(long signr, struct pt_regs * regs);
};
```

- wszystkie rozpoznawane przez Linuks formaty zorganizowane są w listę, pole **next** wskazuje na następną strukturę z listy
- **module** wskazuje na strukturę zdefiniowaną w plku *include/linux/module* - jest to struktura wspomagająca użycie modułów dynamicznie ładowanych przez jądro
- **load_binary** jest funkcją wywoływaną w celu wczytania pliku
- **load_shlib** to funkcja wywoływana w celu wczytania biblioteki dzielonej
- **core_dump** służy do zapisania obrazu pamięci wykorzystywanej przez program (plik binarny)

3.1.2 linux_binprm

Ta struktura służy do trzymania argumentów używanych podczas wczytywania plików wykonywalnych.

```
struct linux_binprm{
    char buf[128];
    unsigned long page[MAX_ARG_PAGES];
    unsigned long p;
    int sh_bang;
    int java; /* Java binary, prevent recursive invocation */
    struct dentry * dentry;
    int e_uid, e_gid;
    kernel_cap_t cap_inheritable, cap_permitted, cap_effective;
    int argc, envc;
    char * filename; /* Name of binary */
    unsigned long loader, exec;
};
```

3.2 Funkcje

Poniższe funkcje zdefiniowane są w pliku *fs/exec.c*

3.2.1 register_binfmt

Do rejestrowania formatu służy następująca funkcja:

Algorytm 1 register_binfmt

Dane wejściowe: struct linux_binfmt *fmt
Wynik: int

```
jeśli fmt jest juz zarejestrowany to
    zwróć błąd
w przeciwnym przypadku
    dopisz fmt na poczatek globalnej listy formatów
koniec jeśli
```

3.2.2 unregister_binfmt

Poniższa funkcja służy do wyrejestrowania formatu:

Algorytm 2 unregister_binfmt

Dane wejściowe: struct linux_binfmt *fmt
Wynik: int

```
jeśli fmt jest zarejestrowany to
    usuń fmt z globalnej listy formatów
w przeciwnym przypadku
    zwróć błąd
koniec jeśli
```

3.2.3 prepare_binprm

Funkcja ta sprawdza, czy plik, którego inode jest już zainicjowany w strukturze ma prawa wykonywania. Jeśli użytkownik może wykonać plik, to wczytywanych jest 128 pierwszych bajtów z pliku. W przeciwnym przypadku zwracany jest błąd.

3.2.4 search_binary_handler

Funkcja wywoływana jest z funkcji *do_execve* opisanej w pracy pt. *Algorytm execve.* Ma ona na celu rozpoznanie formatu pliku.
UWAGA: Poniżej przedstawiony jest algorytm dla komputerów PC!

Algorytm 3 prepare_binprm

Dane wejściowe: struct linux_binprm *bprm
Wynik: int

weź inode pliku ze struktury bprm
jeśli nie ustawiony bit wykonywania **to**
  błąd
w przeciwnym przypadku jeśli ktoś pisze do pliku **to**
  błąd
koniec jeśli
jeśli trzeba i można ustawić euid **to**
  zrób to
koniec jeśli
wyczyść (wpisz zera) bufor bprm->buf
wczytaj pierwsze 128 bajtów z pliku do bprm->buf

Algorytm 4 search_binary_handler

Dane wejściowe: struct linux_binprm *bprm, struct pt_regs *regs
Wynik: int

dla każdego zarejestrowanego formatu binarnego fmt **wykonuj**
  weź funkcję ładującą plik: fn = fmt>load_binary
  wykonaj fn: ret = fn
  jeśli funkcja powiodła się **to**
    program z pliku został wykonany
    zwróć jako wynik wartość funkcji fn: ret
  w przeciwnym przypadku jeśli plik nie był formatu opisanego w fmt **to**
    następny obrót pętli
  w przeciwnym przypadku jeśli zgubiliśmy dentry z bprm **to**
    błąd, zwróć ret
  koniec jeśli{n}ie udało się wczytać pliku
  jeśli włączona obsługa modułów dynamicznie ładowalnych **to**
    wczytaj odpowiedni moduł na podstawie buf z bprm
    powtórz powyższą pętlę
  koniec jeśli
koniec dla każdego

4 Obsługa plików wykonywalnych

Aby uruchomić program (lub wykonać skrypt) użytkownik wpisuje w shell'u jego nazwę i wciska 'ENTER'. Od tego momentu shell, a następnie system operacyjny wykonują szereg czynności, które jeśli się powiądą prowadzą do uruchomienia programu: tworzony jest nowy proces[2] i wywoływana jest funkcja *execve*[3]. W konsekwencji wyoływana jest funkcja *do_execve*, która po zainicjowaniu struktury *linux_binprm*[4] wywołuje funkcję **search_binary_handler**[5]. Ta z kolei funkcja dla każdego z zarejestrowanych formatów plików wykonywalnych próbuje wywołać funkcję wczytującą plik.

Przyjrzyjmy się temu dokładniej:

4.1 Format ELF

Funkcje obsługujące format ELF zdefiniowane są w pliku *fs/binfmt_elf.c*.

4.1.1 Rejestracja formatu

Poniżej widoczna jest zainicjowana struktura *linux_binfmt*.

```
static struct linux_binfmt elf_format = {
    NULL,
    THIS_MODULE,
    load_elf_binary,
    load_elf_library,
    elf_core_dump,
    ELF_EXEC_PAGESIZE
};
```

4.1.2 Wczytanie pliku

Za wczytywanie plików w formacie ELF odpowiedzialna jest funcja **load_elf_binary**, która kontroluje liczbę wywoływanych w danej chwili operacji wczytywania pliku (makra **MOD_INC_USE_COUNT** oraz **MOD_DEC_USE_COUNT** zdefiniowane w pliku *include/ linux/module.h*) wywołuje funkcję **do_load_elf_binary**:

4.1.3 Uwagi

Dokładny opis odwzorowania plików binarnych w pamięci znajduje się w pracy pt. *Odwzorowanie plików w pamięci.*

[2]opis funkcji fork znaleźć można w pracy pt. Algorytm fork

[3]opis funkcji execve znaleźć można w pracy pt. Algorytm exec

[4]patrz: sekcja o strukturach danych niezależnych od typu pliku

[5]opis funkcji znajduje się w sekcji poświęconej funkcjom niezależnym od typu pliku

Algorytm 5 do_load_elf_binary

Dane wejściowe: struct linux_binprm *bprm, struct pt_regs *regs
Wynik: int

```
jeśli plik nie jest w wykonywalnym formacie ELF to
    błąd
koniec jeśli
jeśli nie można odczytać architetury komputera, dla którego przeznaczony jest plik to
    błąd
koniec jeśli
jeśli inode pliku nie ma domyślnej funkcji mmap to
    błąd
koniec jeśli
ustal interpreter pliu {plik może być wykonywalny, być biblioteką dzieloną itp.}
jeśli nie można uruchomić interpretera to
    błąd
koniec jeśli
kopiuj plik do pamięci: funkcja mmap {o odwzorowaniu plików w pamięci przeczytasz w pracy
    o tym tytule}
zmień adresy początku, końca itp. bieżącego adresu na załadowany plik
uruchom wątek o podanym adresie
```

4.2 Format Script

Obsługa skryptów ogranicza się do znalezienia, czy też określenia, interpretera. Z tego powodu nie przewiduje się tutaj funkcji do ładowania bibliotek dzielonych, czy też **zrzucania obrazu procesu** (ang. *core dump*).
Definicje formatu znajdują się w pliku *fs/binfmt_script.c*.

4.2.1 Rejestracja formatu

```
struct linux_binfmt script_format = {
#ifndef MODULE
    NULL, 0, load_script, NULL, NULL
#else
    NULL, &__this_module, load_script, NULL, NULL
#endif
};
```

4.2.2 Wczytanie pliku

Funkcja **load_script** działa analogicznie jak **load_elf_binary** [6]: wywołuje ona **do_load_script**:

[6] zobacz: sekcja o formacie ELF

Algorytm 6 do_load_script

Dane wejściowe: struct linux_binprm *bprm, struct pt_regs *regs
Wynik: int

jeśli zły format pierwszej linii lub zapętlił się ciąg wywoływania interpretera **to**
    zwróć błąd
koniec jeśli
zwiększ wartość bprm->sh_bang[7]
odczytaj nazwę interpretera i opcjonalne argumenty
jeśli nie istnieje plik, który ma być interpreterem **to**
    błąd
koniec jeśli
{modyfikujemy strukturę bprm}
jako nazwę pliku do wykonania ustaw nazwę interpretera
ustaw opcjonalne argumenty
dodaj do listy argumentów nazwę skryptu wraz z ewent. argumentami
uzupełnij strukturę bprm dla zmodyfikowanych danych wywołując prepare_binprm[8]
{koniec modyfikacji bprm}
wykonaj search_binary_handler(bprm,regs);

4.2.3 Uwagi

Poprawny format pierwszej linii skryptu jest następujący:
#!/pełna/scieżka/z/nazwą/pliku[argn]*
przy czym długość tej linii nie może być dłuższa (reszta znaków nie będzie brana pod uwagę) niż 127 znaków - linia ta wczytywana jest do tablicy *buf* ze struktury *bprm*.

4.2.4 Zadanie Script

Podaj scenariusz, który prowadzi do zapętlenia się wywołań interpretera.

4.3 Format Misc

Ten sposób obsługi plików wykonywalnych jest bardzo podobny do obsługi *skryptów*.
Definicje tego formatu zawarte są w pliku *fs/binfmt_misc.c*

4.3.1 Rejestracja formatu

```
static struct linux_binfmt misc_format = {
#ifndef MODULE
    NULL, 0, load_misc_binary, NULL, NULL
#else
    NULL, &__this_module, load_misc_binary, NULL, NULL
#endif
};
```

4.3.2 Struktura binfmt_entry

Formaty rozpoznawane przez moduł *binfmt_misc* trzymane są w postaci listy **entries** o elementach typu **binfmt_entry**:

```
struct binfmt_entry {
    struct binfmt_entry *next;
    long id;
    int flags; /* type, status, etc. */
    int offset; /* offset of magic */
    int size; /* size of magic/mask */
    char *magic; /* magic or filename extension */
    char *mask; /* mask, NULL for exact match */
    char *interpreter; /* filename of interpreter */
    char *proc_name;
    struct proc_dir_entry *proc_dir;
};
```

4.3.3 Funkcje pomocnicze

Poniższe funkcje wywoływane są w celu inicjacji usługi ładowania plików różnych formatów oraz podczas samej operacji (tabele *Funkcje binfmt_misc operujące na liście entries* oraz *Funkcje binfmt_misc operujące na systemie plików /proc*).

Tablica 1: Funkcje binfmt_misc operujące na liście entries

Nazwa funkcji	Działanie funkcji
clear_entry(int id)	usuwa format o ident. id
clear_entries()	usuwa wszystkie formaty
get_entry(int fd)	zwraca strukturę binfmt_entry o ident. id, oraz blokuje listę do odczytu
put_entry(binfmt_entry e)	odblokowuje listę do odczytu
check_file(linux_binprm *p)	sprawdza, czy jest na liście format obsługujący plik o parametrach p (sprawdzane jest rozszerzenie pliku oraz charakterystyczny ciąg bitów)

- **usuwanie formatu**: usunięcie z listy i wyczyszczenie wpisu w systemie plików */proc*

4.3.4 Wczytanie pliku

Funkcję load_misc_binary prezentuje *Algorytm load_misc_binary*

Tablica 2: Funkcje binfmt_misc operujące na systemie plików /proc

Nazwa funkcji	Działanie funkcji
entry_proc_setup(binfmt_entry *e)	tworzy wpis do systemu plików /proc, aby można było odczytywać z tamtąd konfigurację
proc_write_register(struct file *f, char *buffer, long count, void *data)	rejestruje nowy format wprowadzając wpis do odpowiedniego pliku w systemie plików /proc
proc_read_status(..., void *data)	odczytuje status formatu w systemie /proc status może być **enabled** lub **disabled**
proc_write_status(..., void *data)	zapisuje status formatu w systemie /proc

4.3.5 Uwagi

Jądro pozwala na wywołanie prawie każdego programu po prostu przez wpisanie jego nazwę w shell'u. Mogą to być na przykład skompilowane programy Javy, Python'a czy Emacs'a.

By to osiągnąć trzeba wskazać modułowi binfmt_misc, który interpreter powinien być wywołany z którym plikiem wykonywalnym. Binfmt_misc rozpoznaje typy binarne przez dopasowanie niektórych bajtów z początku pliku z charakterystycznymi ciągami bajtów (maskując określone bity), jakie są wyspecyfikowane. Binfmt_misc może również rozpoznawać rozszerzenia plików np. '.com' or '.exe'.

Aby faktycznie zarejestrować nowy plik binarnych trzeba ustalić ciąg znaków wyglądający jak: **:nazwa:typ:offset:ciąg bajtów:maska:interpreter:** (przy czym można wybrać znak ':' według własnych potrzeb) i dopisać go do **/proc/sys/fs/binfmt_misc/register**.
Oto co ozaczają poszczególne pola:

- 'nazwa' jest nazwą. W katalogu /proc/sys/fs/binfmt_misc zostanie utworzony nowy plik o tej nazwie
- 'typ' jest jedną z wartości '**M**': rozpoznawanie przez *ciąg bajtów*, lub '**E**': rozpoznawanie przez rozszerzenie
- 'offset' określa przeunięcie ciągu bajtów/maski w pliku liczone w bajtach. Domyślna wartość to 0 w przypadku kiedy to pole jest pominięte np. ':nazwa:typ::ciąg...')
- 'ciąg bajtów' jest ciągiem charakterystycznym dla danego formatu. Może on zawierać ciąg szesnastkowo zakodowanych znaków np. \x0a lub \xA4. Jeśli wybrano rozpoznawanie przez rozszerzenie jest to właśnie to rozszerzenie bez wiodącej '.'. Rozpoznawanie rozszerzenia jest wrażliwe na wielkość liter.
- 'maska' jest opcjonalna (domyślnie dla każdego bitu wynosi 0xff). Maska jest takiej samej długości jak **ciąg bajtów**.
- 'interpreter' to nazwa (wraz z pełną ścieżką) programu uruchamianego z naszym plikiem 'wykonywalnym' jako pierwszym argumentem

Są pewne ograniczenia:

Algorytm 7 load_misc_binary

Dane wejściowe: struct linux_binprm *bprm, struct pt_regs *regs
Wynik: int

weź zmienną fmt typu binfmt_entry
wykonaj fmt = check_file(bprm)
jeśli operacja się powiodła **to**
  weź nazwę interpretera z fmt->interpreter {operacja odbywa się przy zablokowanej liście entries}
w przeciwnym przypadku
  błąd
koniec jeśli
odłuż bprm->dentry {teraz będziemy uruchamiali inny program}
przygotuj nowe parametry wywołania pliku {zwiększana jest liczba argumentów: dodaje się nazwę pliku jako parametr wywołania interpretera}
przypisz bprm->dentry pliku interpretera
przygotuj strukturę bprm {wywołanie funkcji prepare_bprm: patrz sekcja o funkcjach niezależnych}
wywołaj funkcję find_binary_handler {patrz: sekcja o funkcjach niezależnych}

- cały ciąg znaków rejestrujący format nie może przekraczać 255 znaków
- **ciąg bajtów** musi się zawierać (w całości) w pierwszych 128 bajtach pliku
- długość ciągu określającego **interpreter** nie może przekraczać 127 znaków

4.3.6 Zadanie Misc

Zarejestrój pliki nazwa.exe jako pliki wykonywalne.
UWAGA: Do uruchomienia plików exe można użyć programu *dosecec*.

5 Rozwiązania zadań

5.1 Zadanie Script

Przykład może być następujący:

- pierwsza linia pliku /**testdir**/**a.sh** to *#!/testdir/b*,
- pierwsza linia pliku /**testdir**/**b.sh** to *#!/testdir/a*

Warto zauważyć, że błąd, powstały przy wywołaniu pliku, przechwytywany jest przez shell, który zazwyczaj sam próbuje zinterpretować ten plik. W konsekwencji aby sprawdzić działanie przykładu należałoby napisać krótki program, np. w C, wywołujący któryś z tych plików funkcją **execve**, np:

```
#include <unistd.h>
#include <stdio.h>
```

```
main() {
    int ret;
    if ((ret = execve("/testdir/a", 0, 0)) != 0)
        printf("błąd: %d\n", ret);
};
```

Można się przekonać, że program działa poprawnie na przykład modyfikując pierwszą linię **a.sh** na następującą:
#!/bin/sh
(Dobrze jest dodać jakąś treść do pliku, żeby można było zobaczyć, że rzeczywiście się wykonał.

5.2 Zadanie Misc

Zakładam, że obsługa plików wykonywalnych MISC została wkompilowana w jądro (lub odpowiedni moduł jest załadowany), oraz że program *dosexec* znajduje się w katalogu */usr/bin*.
Zadanie można wykonać wydając następującą komendę w shell'u:
echo :DEXE:E::exe::/usr/bin/dosexec: » /proc/sys/fs/binfmt_misc/register