

zppp project – un semplice esercizio di asm coding sotto LinuxPubblicato da **Ritz** il 19/11/2000Livello **intermedio****Introduzione**

La programmazione in Assembly sotto Linux non sembrava essere, fino a poco tempo fa, un argomento seguito da molte persone, almeno qui in Italia. Proprio per questo e' nato il RACL, Reversing & asm coding for Linux, che ha lo scopo di diffondere l'"arte" del reverse engineering e asm coding riferiti a questo so. Per le varie info al riguardo rimando al sito <http://racl.oltrelinux.com>. Cosa sia il reverse engineering o la programmazione in Assembly non lo spieghero' certo in questo articolo, qui infatti cerchero' di introdurre brevemente a *come* in Linux si puo' programmare in asm, portando anche un semplice esempio pratico. Scarica i sorgenti.

Iniziamo

Naturalmente, Linux e' un sistema operativo a 32-bit, il quale opera in protected mode [piccola nota: quando dico asm coding intendo sempre in architetture ia32, ovvero da 386 in su =)].

Quando scriviamo un programma in asm per Linux dobbiamo prima di tutto scegliere la sintassi che desideriamo adottare (a chi proviene dalla programmazione in asm sotto win32 o dos questo aspetto risultera' probabilmente strano). Nello scrivere un listato asm da compilare e linkare proprio sotto Linux, infatti, abbiamo a disposizione 2 scelte differenti proprio nella sintassi a nostra disposizione. La prima, la classica sintassi Intel, e' quella normalmente utilizzata in win o dos, mentre la seconda, la sintassi AT&T (quella utilizzata originariamente in sistemi UNIX), presenta alcune caratteristiche che la differenzia da quest'ultima e la rende, secondo molti, meno ambigua, ad esempio:

- in un'operazione, il registro di destinazione deve *sempre* essere il secondo.
- il nome dei registri e' preceduto dal simbolo "%". Ad esempio per copiare eax in ebx scriveremo al posto di mov ebx, eax (sintassi Intel) mov %eax, %ebx (sintassi AT&T).
- la lunghezza dell'operando va posposta all'operazione che opera su di esso. Ad esempio, per copiare bx (word) in ax (word) si scrivera' movw %bx, %ax, per il byte si usa la lettera b, per la dword la lettera l.
- Ogni operando immediato va posposto al simbolo "\$" (quello di Bill Gate\$), ad esempio addl \$5,%eax.
- Non mettere un prefisso a un operando indica che e' un indirizzo di memoria. Es. movl \$ciao,%eax muove l'offset di ciao in eax, mentre movl ciao,%eax muove il contenuto di ciao (la dw da esso puntato) in eax.

A seconda della sintassi con cui viene scritto il listato sorgente, si sceglia' quindi il compilatore. Nel caso sia stata scelta la classica sintassi Intel il compilatore piu' utilizzato e' il NASM, in caso contrario si potra' usare, ad esempio, il GAS (Gnu Assembler), contenuto nel GCC.

Una volta che il sorgente viene compilato, bastera' semplicemente linkarlo con dei linker tipo gcc o ld ottenendo come risultato finale un elf eseguibile.

Per quanto riguarda invece le classiche funzioni di i/o da console, file, etc., in Linux abbiamo a disposizione due diverse possibilita', ovvero possiamo decidere se chiamare la funzione del kernel relativa al nostro scopo o se preferiamo invece utilizzare le libc.

Nel secondo caso bastera' eseguire semplici call alla relativa funzione passano i relativi argomenti nello stack, avendo l'accortezza, una volta che la funzione e' stata eseguita, di pulire lo stack (in Linux e' obbligatorio farlo *sempre* dopo che si torna da una funzione).

Scegliendo la prima ipotesi, invece, dovremo fare delle chiamate al kernel, e cio' in Linux avviene attraverso l'int 0x80. Prima di eseguire questo int, in eax sara' messo il numero di funzione richiesto, quindi rispettivamente in ebx, ecx, edx, esi, edi i relativi argomenti. In valore di ritorno si trovera' in eax e non sara' utilizzato lo stack. E' cmq buona norma non mettere in eax direttamente il numero della funzione, poiche' tali numeri con le successive versioni dei kernel possono variare, ma utilizzare invece nomi che li identifichino, quali sys_write o sys_exit.

Ma veniamo ora ad un esempio pratico: il programma che spieghero' di seguito e' un semplice tool che serve a configurare una connessione a Internet. Esso pone all'utente varie domande, terminate le quali va a scrivere alcuni file e script di configurazione. Per scriverlo utilizzeremo la classica sintassi Intel. I src verranno compilati con NASM e linkati con gcc.

Dato che ' il primo esempio ho deciso di utilizzare sia il kernel che le libc per le chiamate varie.

```
global main

extern printf
extern scanf
extern strlen

NULL equ 0
```

Prima di tutto dichiariamo l'entrypoint "main" (la funzione main() necessaria per il linker gcc) che indica in punto di inizio esecuzione del nostro codice. Fatto cio' dichiariamo le funzioni esterne delle libc che ci serviranno, nella fattispecie printf, scanf e strlen (non chiedetemi a cosa servono per favore:)).

Infine gli equates, con classico NULL = 0.

```

section .data

graphic1      db      0Ah,0Dh,'\
               +-----+',0Dh,0Ah,NULL
head          db      '\
               | zppp v1.0 by Ritz for RACL |',0Dh,0Ah,NULL
graphic2      db      '\
               +-----+',0Ah,0Dh,0Ah,0Dh,NULL
copyright     db      'This program is totally FREE and comes with NO WARRANTY. Run from root
account.      ',0xA,0xD,NULL
explain       db      'By using this simple program you will now configure a new Internet
connection.',0Dh,0Ah,'Please answer to the following questions [^C to
terminate]...',0Dh,0Ah,0Dh,0Ah,NULL
file1         db      0xA,'Writing /etc/ppp/options... ',NULL
file2         db      'Writing /etc/ppp/pppscript... ',NULL
file3         db      'Writing /etc/ppp/pap-secrets... ',NULL
file4         db      'Writing /etc/resolv.conf... ',NULL
script        db      'Writing connection script... ',NULL
done          db      'done.',0xD,0xA,NULL
errormex      db      'error!! Leaving... ',0xD,0xA,NULL

phone         db      '. Write here your ISP phone number: ',NULL
phonebuffer   TIMES 0x15 db NULL
lenphone      dd      NULL

device        db      '. Write here your modem device path or its symbolic link: ',NULL
devicebuffer  TIMES 0x15 db NULL
lendev        dd      NULL

user          db      '. Write here the username for your account: ',NULL
userbuffer    TIMES 0x15 db NULL
lenuser       dd      NULL

pw            db      '. Write here the password for your account: ',NULL
pwbuffer      TIMES 0x15 db NULL
lenpw         dd      NULL

domain        db      '. Write here the domain name of your ISP: ',NULL
domainbuffer  TIMES 0x15 db NULL
lendomain     dd      NULL

dns1          db      '. Write here the first DNS IP for your account (needed): ',NULL
dns1buffer    TIMES 0x15 db NULL
lendns1buffer dd      NULL

dns2          db      '. Write here the second DNS IP for your account (n = none): ',NULL
dns2buffer    TIMES 0x15 db NULL
lendns2buffer dd      NULL

perfect       db      0xD,0xA,'\
The configuration has been completed successfully!',0xD,0xA,NULL
advice        db      '\
To connect to the Internet, simply execute the script /usr/sbin/zppp.',0xD,0xA,NULL
advice2       db      '\
To let all users be able to connect, change the mode of /usr/sbin/pppd to
4755.',0xD,0xA,0xD,0xA,NULL
enjoy         db      'Enjoy!',0xD,0xA,'@2000 by Ritz for RACL',0xD,0xA,0xD,0xA,NULL

format        db      '%s',NULL
handle        dd      NULL

```

```

; -----
optionspath      db          '/etc/ppp/options',NULL
optionshandle    dd          NULL

optionsbuffer    db          'lock',0xA
                 db          'defaultroute',0xA
                 db          'noipdefault',0xA
                 db          'modem',0xA
optionsins       TIMES      0x15 db NULL
optionsbuffer2   db          '115200',0xA
                 db          'crtsets',0xA
                 db          'passive',0xA
                 db          'asyncmap 0',0xA
                 db          'name "'
optionsins2      TIMES      0x15 db NULL

; -----

pppscripthandle  dd          NULL
pppscriptpath   db          '/etc/ppp/pppscript',NULL

pppscriptbuffer db          'TIMEOUT 60',0xA
                 db          'ABORT ERROR',0xA
                 db          'ABORT BUSY',0xA
                 db          'ABORT "NO CARRIER"',0xA
                 db          'ABORT "NO DIALTONE"',0xA
                 db          '" " "AT&FH0"',0xA
                 db          'OK "atdt'
pppscriptins    TIMES      0x15 db NULL
pppscript2      db          'TIMEOUT 75',0xA
                 db          'CONNECT',NULL

; -----

papsecretshandle dd          NULL
papsecretspath  db          '/etc/ppp/pap-secrets',NULL

papsecretshandle dd          NULL
papsecretsl     db          '" "'
papsecrets      TIMES      0x40 db NULL

; -----

scriptbuffer     db          '/usr/sbin/pppd -detach connect "/usr/sbin/chat -v -f
/etc/ppp/pppscript"',NULL
scriptpath       db          '/usr/sbin/zppp',NULL

; -----

resolvhandle     dd          NULL
resolvpath       db          '/etc/resolv.conf',NULL

resolv           db          'search '
resolvins        TIMES      0x15 db NULL

resolv2          db          'nameserver '
resolvins2       TIMES      0x15 db NULL

resolv3          db          'nameserver '
resolvins3       TIMES      0x15 db NULL

-----

```

Come vedete dalla sezione .data il prg dovrà creare 5 file: /etc/ppp/options, /etc/ppp/pppscript, /etc/ppp/pap-secrets, /etc/resolv.conf e lo script di connessione, che potremo benissimo mettere in /usr/sbin/zppp. I "TIMES 0xN db NULL" equivalgono semplicemente alle dichiarazioni "db N dup(NULL)" del TASM, ovvero riempiono di NULL una quantità N di byte. Il resto dovrebbe essere tutto chiaro.

Ovviamente i contenuti dei file dovranno avere degli spazi vuoti proprio per permettere l'inserimento dei dati personali per la connessione.

NOTA: i src sono stati scritti come semplice esempio di coding, come potete vedere da voi stessi gli overflow sono infiniti, ma questo (visto che non abbiamo bisogno che il prg sia necessariamente sicuro) a noi non interessa.

Una volta dichiarata la sezione .data possiamo mettere sotto tutti i dati inizializzati che ci serviranno. Piccola nota: i dati non

inizializzati andrebbero messi ad esser precisi in .bss, ma in questo semplice esempio ho usato ugualmente questa sezione.
Ora inizia la sezione .text

section .text

main:

```
    push    dword graphic1
    call    printf
    add     esp, 4

    push    dword head
    call    printf
    add     esp, 4

    push    dword graphic2
    call    printf
    add     esp, 4

    push    dword copyright
    call    printf
    add     esp, 4

    push    dword explain
    call    printf
    add     esp, 4

    push    dword phone
    call    printf
    add     esp, 4

    push    dword phonebuffer
    push    dword format
    call    scanf
    add     esp, 8

    push    dword device
    call    printf
    add     esp, 4

    push    dword devicebuffer
    push    dword format
    call    scanf
    add     esp, 8

    push    dword user
    call    printf
    add     esp, 4

    push    dword userbuffer
    push    dword format
    call    scanf
    add     esp, 8

    push    dword pw
    call    printf
    add     esp, 4

    push    dword pwbuffer
    push    dword format
    call    scanf
    add     esp, 8

    push    dword domain
    call    printf
    add     esp, 4

    push    dword domainbuffer
    push    dword format
    call    scanf
    add     esp, 8
```

```

push    dword dns1
call    printf
add     esp, 4

push    dword dns1buffer
push    dword format
call    scanf
add     esp, 8

push    dword dns2
call    printf
add     esp, 4

push    dword dns2buffer
push    dword format
call    scanf
add     esp, 8

```

Anche qui tutto e' molto semplice: il prg infatti fa tutte le domande necessarie e aspetta la risposta da parte dell'utente. Notate **SEMPRE** che lo stack viene pulito con l'istro add esp. Potete farlo anche pushando lo stack in dei registri che non vi servono, ma qui oh preferito il primo metodo.

Fatto cio' va a sistemare uno a uno i file che poi saranno scritti:

```

-----

push    dword devicebuffer
call    strlen
add     esp, 4
mov     dword [lende], eax
mov     ecx, eax
inc     ecx
mov     byte [devicebuffer+eax], 0xA

mov     esi, dword devicebuffer
mov     edi, dword optionsins
repz   movsb

mov     ecx, 0x28
mov     esi, dword optionsbuffer2
mov     edi, dword optionsins
add     edi, dword [lende]
inc     edi
repz   movsb

push    dword userbuffer
call    strlen
mov     dword [lenuser], eax
mov     byte [userbuffer+eax], ''

mov     ecx, eax
inc     ecx
mov     esi, dword userbuffer
mov     edi, dword optionsins
add     edi, 0x29
add     edi, dword [lende]
repz   movsb

mov     eax, dword optionsbuffer
add     eax, 0x4E
add     eax, dword [lende]
add     eax, dword [lenuser]
mov     dword [eax], NULL

```

Per fare tutte queste operazioni come vedete bisogna fare un po' di taglio e cucito;) nel senso che i byte dei buffer non si trovano tutti al loro posto, anzi, quindi repz movsb rulez. Come avrete gia' notato, nel NASM per muovere un offset in un registro si scrive "mov reg, dword buffer", mentre per muovere il contenuto del buffer in questione nello stesso registro si scrive "mov reg, dword [buffer]". Un po' diverso anche qui da TASM o MASM. Inoltre, viene accettato il prefisso 0x per indicare i valori hex. Dopo che i byte sono stati tutti allineati correttamente e il buffer e' pronto per essere scritto nel file, vengono eseguite le ultime 5

istruzioni del blocco sopra presentato, che hanno lo scopo di mettere byte NULL alla fine del buffer per questioni di sicurezza nel caso in cui essi non fossero già presenti perché sono stati sovrascritti nelle operazioni di "allineamento". Lo so avrei potuto usare delle strutture per fare le stesse cose ma non sarebbe cambiato molto e per i nostri scopi questi buffer vanno benissimo.

Le stesse operazioni vanno fatte anche per tutti gli altri file:

```
-----  
  
    push    dword phonebuffer  
    call   strlen  
    mov    dword [lenphone], eax  
    add    esp, 4  
  
    mov    byte [phonebuffer+eax], ''  
    mov    byte [phonebuffer+eax+1], 0xA  
    mov    ecx, eax  
    add    ecx, 2  
  
    mov    esi, dword phonebuffer  
    mov    edi, dword pppscriptins  
    repz  movsb  
  
    mov    ecx, 0x13  
    mov    esi, dword pppscript2  
    mov    edi, dword pppscriptins  
    add    edi, eax  
    add    edi, 2  
    repz  movsb  
  
    mov    eax, dword pppscriptbuffer  
    add    eax, 0x71  
    add    eax, dword [lenphone]  
    mov    dword [eax], NULL  
  
; -----  
  
    push    dword userbuffer  
    call   strlen  
    mov    dword [lenuser], eax  
  
    mov    ecx, eax  
    inc    ecx  
    mov    esi, dword userbuffer  
    mov    edi, dword papsecrets  
    repz  movsb  
  
    mov    dword [papsecrets+eax], 0x22092A09  
  
    push    dword pwbuffer  
    call   strlen  
    add    esp, 4  
    mov    dword [lenpw], eax  
  
    mov    byte [pwbuffer+eax], ''  
    mov    ecx, eax  
    inc    ecx  
  
    mov    esi, dword pwbuffer  
    mov    edi, dword papsecrets  
    add    edi, dword [lenuser]  
    add    edi, 4  
    repz  movsb  
  
    mov    eax, dword papsecrets  
    add    eax, 0x6  
    add    eax, dword [lenuser]  
    add    eax, dword [lenpw]  
    mov    dword [eax], NULL  
  
; -----  
  
    push    dword domainbuffer  
    call   strlen  
    add    esp, 4
```

```

mov     dword [lendomain], eax
mov     byte [domainbuffer+eax], 0xA
mov     ecx, eax
inc     ecx
mov     esi, dword domainbuffer
mov     edi, dword resolvins
repz   movsb

push   dword dns1buffer
call   strlen
add    esp, 4
mov    dword [lendns1buffer], eax
mov    byte [dns1buffer+eax], 0xA
mov    ecx, eax
inc    ecx
mov    esi, dword dns1buffer
mov    edi, dword resolvins2
repz   movsb

cmp    dword [dns2buffer], 0x0000006E
jz     otherdns

```

; inizio sezione inutile se non c'e' il secondo dns -----

```

push   dword dns2buffer
call   strlen
add    esp, 4
mov    dword [lendns2buffer], eax
mov    byte [dns2buffer+eax], 0xA
mov    ecx, eax
inc    ecx
mov    esi, dword dns2buffer
mov    edi, dword resolvins3
repz   movsb

mov    ecx, dword [lendns2buffer]
add    ecx, 0xB
mov    esi, dword resolv3
mov    edi, dword resolvins2
add    edi, dword [lendns1buffer]
inc    edi
repz   movsb

```

; fine sezione inutile se non c'e' il secondo dns -----

otherdns:

```

mov    ecx, dword [lendns1buffer]
add    ecx, dword [lendns2buffer]
add    ecx, 0x16
mov    esi, dword resolv2
mov    edi, dword resolvins
add    edi, dword [lendomain]
inc    edi
repz   movsb

mov    eax, dword resolv
add    eax, 0x1F
add    eax, dword [lendomain]
add    eax, dword [lendns1buffer]
add    eax, dword [lendns2buffer]
mov    dword [eax], NULL

```

Soliti lavori di allineamento, con l'accortezza di aver inserito l'opzione di poter utilizzare anche un solo server dns.

Solo dopo che tutti i buffer sono a posto possono essere scritti su file. E qui usero' le chiamate al kernel al posto di chiamate quali fopen()).

```

-----
mov    eax, 0x8
mov    ebx, dword optionspath

```

```

mov     ecx, 0x1A4
int     0x80
cmp     eax, -1
jz      near error
mov     dword [handle], eax

```

Il numero della chiamata e' 8, cioe' sys_create. in ebx va l'offset di optionspath, in ecx i permessi.

Come si impostano i giusti permessi? Semplice: considerato che il valore numerico di un permesso (che so, 755) e' sempre espresso in sistema ottale, bastera' convertire tale valore in hex e quindi metterlo in ecx, nient'altro.

```

-----
push   dword optionsbuffer
call   strlen
add    esp, 4

mov    ebx, dword [handle]
mov    ecx, dword optionsbuffer
mov    edx, eax
mov    eax, 0x4
int    0x80

```

Con il pezzo sopra di codice andiamo a scrivere nel file che abbiam oappena creato tramite sys_write, numero 0x4. Il numero di byte da scrivere lo ricaviamo con un strlen del buffer che ci interessa.

Ora non ci resta che chiudere l'handle.

```

-----
mov    eax, 0x6
mov    ebx, dword [handle]
int    0x80

```

Fatte queste operazioni per i vari file, non dovremo fare altro che compilare il tutto con

```
$ nasm -f elf zppp.asm
```

e quindi andare a linkare il file .o con

```
$ gcc zppp.o
```

per avere come risultato l'eseguibile a.out.

Conclusioni

Il nostro piccolo tool e' terminato e funziona a dovere.

Per i soliti suggerimenti, critiche, appunti, bug e cosi' via mandatemi pure una mail.

Byz,

Ritz

----- ritz@freemail.it -----

----- ritz@oltrelinux.com ----

Attenzione

Le informazioni contenute in questo documento sono a puro scopo didattico; l'autore non incoraggia chi volesse servirsene per scopi illegali.