# **HTB — Code (Experience & Takeaways)**



I recently worked through an HTB lab called **Code**, which focused on a Python-based web app. It taught me a lot about restricted execution environments, sandbox bypass techniques, and privilege escalation through system misconfiguration.

***Tools that helped:***

* nmap — for initial service discovery
* Python Code editor — built into the target’s web app
* [CrackStation](https://crackstation.net/) — for cracking password hashes
* tar, bunzip2 — for transferring and extracting backups
* **“tar -xjf [filename]”** automatically decompresses and extracts the archive in one move.
* -x → extract
* -j → handle bzip2 compression
* -f → specify the filename

This saves time by skipping the manual decompression step with bunzip2.

***Key Concepts I Learned:***

* Python Sandbox Bypass

The built-in code editor filtered out dangerous functions like import, os, and open. I’m not super experienced with Python, but I learned that you could still use exception handling to interact with the app’s memory. Using:

raise Exception(globals())

I was able to dump all global variables available to the app. This revealed database models like *User*. Which meant that from here I could query the app’s memory to pull sensitive data. Even without needing direct access or imports.

* SQLAlchemy In-Memory Dumping

Once I saw the User model was available, I learned you could use SQLAlchemy's built-in query functionality:

User.query.all()

This allowed me to extract usernames and password hashes directly from memory. No shell, no file access. Just objects already loaded in the running Python app.

* Weak Hashes

The hashes I pulled were in MD5 format, which are easy to crack. I used CrackStation to recover the original passwords quickly. This gave me credentials for an actual user account, which I then used to gain SSH access to the machine.

* Privilege Escalation via Backups Scripts

After logging in, I checked my sudo privileges and found that I could run a backup script (backy.sh) as root without needing a password. This script used a task.json file to define what directories to archive. I realized I could manipulate that file to read directories that would otherwise be off-limits. Even as root.

* Path Traversal Bypass

The script was supposed to restrict backups to /home/ and /var/ only. But it relied on basic string matching, not actual path validation. By setting the backup target to:

/var/....//root/

I was able to trick it into archiving the /root/ directory — bypassing the filter and giving me access to the final flag.

## ***Final Thoughts:***

Even though I’m not super confident writing Python, this box taught me a lot about how applications behave in memory and how you can still interact with them in creative ways. Even when things are restricted. I followed a lot of trial-and-error, but over time I started to understand why certain things worked. It also made me realize how powerful misconfigured scripts can be, especially when they run with root privileges and rely on weak logic like basic string matching. Overall, this was a solid learning experience and definitely helped me level up my problem-solving mindset.