Mission Name
The Restrains

Historical Context
Ethan, armed with the Claire Recon datacard, infiltrates the Skytech Recon facility. Their objective: to dismantle the auto-restraints on their vehicle, a critical step to ensure their journey to Euphea can proceed.

Technical Overview
Inside the confines of Skytech, Ethan must navigate the Recon system. This mission's success hinges on exploiting a vulnerability found in the webpage's Samples formulary. The flaw in question relates to command injection, offering a pathway to bypass the vehicle's auto-restraints.

Mission Brief
Ethan, leveraging the Claire Recon datacard, you are tasked with penetrating the security of the Skytech Recon area. Your mission is to exploit the system's vulnerabilities to disengage the auto-restraints on your vehicle, enabling travel to Euphea. Return with evidence of your successful system exploitation as proof of mission completion. Best of luck in your endeavor!

Detailed Assignment
With the Claire Recon datacard as his key, Ethan embarks on a crucial mission within the Skytech Recon facility. The task at hand involves neutralizing the automated restraints imposed on their mode of transport, thereby facilitating their impending voyage to Euphea.

Operational Venue
PARIS, FRANCE | PREPARING TO DEPART

Tools

- User: clay
- Password: R3c0n-Claire-999

Questions

What is the name of the vulnerability found?

- Command Injection

Main wildcard used to prompt the flag?

- ?

What is the filter used in the regular expression (//)?

- /[^A-Za-z0-9\-\?\/\.]/

Hints

1. Check out sample's formulary.
2. Try to inject Linux code in samples formulary.
3. Wildcard ?? will help

Categories

- Web
- Command Injection

Write Up

Given the scenario within the Recon Skytech system, where you've identified a command injection vulnerability in the Samples section, the task at hand involves exploiting this flaw to retrieve the flag, despite the lack of direct code visibility. Here's how to approach this blind exploitation based on the provided code snippet and goal:

Understanding the Filter

The application applies a filter to the input, allowing only alphanumeric characters, hyphens (-), question marks (?), slashes (/), and periods (.):

- $filter = preg_replace('/[^A-Za-z0-9\-\?\/\.]/', '', $request->get('name'));

Additionally, it blocks any input containing the strings '.txt', 'home', 'ubuntu', or 'Index', which likely are measures to prevent easy access to critical files or directories:

- if(strstr($filter, '.txt') || strstr($filter, 'home') || strstr($filter, 'ubuntu') || strstr($filter, 'Index'))

Crafting the Payload

Given these restrictions, the goal is to craft a payload that bypasses the filter and does not contain the explicitly forbidden strings, yet still navigates the file system to the flag's location.

The hint suggests using a path traversal exploit chain with obscured directory names to bypass the filter, aiming to reach `/home/ubuntu/flag.txt` without directly using the blocked keywords.

Exploit Chain Explanation

The suggested exploit chain:

- ../../../../../../????/??????/flag????

relies on directory traversal (`../../`) to move up from the current directory to the root (`/`) and then guess the path to the flag file without using the blocked words. The question marks (`?`) indicate uncertainty about the exact path, suggesting the need for guessing or fuzzing the directory names.

Executing the Exploit

1. Blind Guessing/Fuzzing: Since direct references to 'home', 'ubuntu', and specific file extensions are filtered, you'll need to employ a combination of guesswork and systematic fuzzing. Tools like Burp Suite Intruder or custom scripts can automate attempts to discover the correct path by substituting the question marks (`?`) with likely directory names or by incrementally guessing directory structures.

2. Exploit Execution: Once you've formulated a plausible exploit path that adheres to the filter's constraints, submit it through the vulnerable parameter. In practice, this might involve manipulating the URL or HTTP request sent to the Samples section, like so:

- http://recon.skytech.com/samples?name=../../../../../../bin/bash

This example URL is purely illustrative; the actual payload would attempt to traverse to the flag's directory without hitting the filters.

3. Retrieving the Flag: Successful exploitation leads to the execution of a command or script that reads and returns the contents of `/home/ubuntu/flag.txt`, assuming the correct path is guessed and navigated.

Flag Information

flag{r3g3x_filt3r_not_always_works}