picoCTF Reverse Engineering Challenge: FlagHunters Lyric Reader

Challenge Overview

The challenge presents a Python script that simulates a lyric reader for a fictional song. The script embeds a secret flag inside the lyrics and plays through them line by line. The goal is to reverse engineer the control logic and extract the hidden flag.

Initial Analysis

The Python script begins by reading the contents of `flag.txt` and embedding it within a multi-line string variable named `secret_intro`. This content is then prefixed to the full lyrics of the song stored in `song_flag_hunters`. The function `reader(song, startLabel)` controls the printing of lyrics based on positional markers and commands such as `REFRAIN`, `RETURN`, and `END`.

Key Observations

The control flow of the song playback is based on specific labels and conditions:

- The script splits each line on the semicolon character and processes each segment.
- When a line contains `CROWD`, the script prompts the user for input.
- If the user enters a line matching the pattern `RETURN <line number>`, it is interpreted as a jump in the song flow.
- The line following the label `[VERSE1]` is where the playback starts.

Exploitation Strategy

The vulnerability lies in the handling of the `CROWD` input. The script replaces the entire line with the user-supplied string and interprets it as executable lyric content. By supplying the input:

;RETURN 1

at the first `CROWD` prompt, the following occurs:

1. The input is split on the semicolon, yielding `RETURN 1` as a separate command.
2. The interpreter detects the `RETURN 1` command and updates the line pointer `lip` to index 1.
3. Line index 1 corresponds to the beginning of the lyrics, which includes the `secret_intro` containing the flag.
4. The loop repeats from the top and reveals the flag again.

This manipulation allows for a forced reprint of the embedded flag without needing to read the file directly or reach the `END` keyword.

Conclusion

This challenge demonstrates the impact of control flow injection through unsanitized input. Although the script does not use shell commands, the custom language logic allows for user-influenced redirection. By understanding the structure of the interpreter and its command parsing, we were able to reveal the flag early in the program's execution.

Mitigation Recommendations

To prevent similar issues in real-world interpreters or command processors, input handling should validate or restrict control keywords. In particular:

- Avoid direct substitution of user input into interpreted logic
- Maintain a whitelist of allowed input formats for user interaction
- Use separate execution contexts for user inputs and control flow