🚀 Bash: Input Validation & Structured Output Script

📌 Overview

This project demonstrates a clean Bash scripting pattern: **accepting user input, validating arguments, and producing structured output** using arrays and conditional logic.

Although the “cargo bay” theme is fictional, the skills are directly transferable to **security automation**, where scripts must safely handle user input, restrict valid options, and fail predictably.

🎯 Objective

- Store structured values using Bash arrays
- Accept a single command-line argument (`forward`, `midship`, `aft`)
- Validate inputs and handle error cases reliably
- Print formatted results in a consistent, readable layout

🧩 Scenario

In cybersecurity and system administration, small CLI utilities are frequently used to:

- Display system status (services, users, ports, configs)
- Validate parameters before running sensitive actions
- Prevent unexpected script behavior caused by invalid input

This script mirrors that pattern by requiring one valid argument and returning helpful messages when input is missing or invalid.

🛠️ Tools & Technologies

- Bash shell scripting
- Arrays
- Conditional logic (`if / elif / else`)
- Command-line arguments (`$1`)
- Exit codes (`exit 1`)

🔍 Methodology

1️⃣ Structured Data Storage

- Defined three arrays representing separate datasets:
    - `forward_bay`
    - `midship_bay`
    - `aft_bay`
- Each array contains exactly **three items**, keeping the output predictable and testable

Security relevance:
This is the same structure used to store:

- allowlisted values
- known-good configuration options

- controlled menu-based scripts to reduce operator mistakes

2️⃣ Input Validation & Control Flow

- Checked whether an argument was provided (`$# -eq 0`)
- Allowed only three valid values (`forward`, `midship`, `aft`)
- Rejected everything else with a clear message and non-zero exit code

Security relevance:
Input validation reduces risk from:

- unexpected runtime behavior
- ambiguous results
- incorrect operator usage during incident response or automation tasks

---

3️⃣ Consistent, Human-Readable Output

- Printed inventory content in a structured numbered format
- Ensured outputs are predictable and easy to read

Security relevance:
Clear output formatting improves:

- SOC analyst workflow
- handoffs between responders
- auditability of script outcomes

✅ Usage

```
./cargo_manifest.sh forward
./cargo_manifest.sh midship
./cargo_manifest.sh aft
```

🧠 Key Takeaways

- Validating arguments is a core scripting habit that prevents errors and strengthens reliability
- Arrays make scripts easier to scale compared to repeated string variables
- Predictable output formatting improves usability and documentation quality