Dynamic Security Analysis Report — Implementation 07 (Updated Run)

1. Overview

This document provides an updated dynamic analysis report for Implementation 07 (secure-programming-week9). A new run was conducted using an extended execution trace (strace_log.19) to capture additional runtime behavior. The environment remained a fully isolated Docker container configured with no network access, minimal privileges, and restricted resources. The goal was to identify any changes in runtime characteristics, system calls, or potential security anomalies.

Observation duration: ~60 seconds

Tools used:
- strace — system call tracing for network, file, and process behavior
- inotifywait — file monitoring
- ss — socket inspection
- ps — process snapshot
- program.log — runtime console output

2. Process and Runtime Behavior

During this updated analysis run, Node.js successfully launched and attempted to initialize the server module structure. While the previous run terminated early due to a missing module, this trace (strace_log.19) confirms a more complete execution cycle involving internal module imports, file reads, and environment configuration. The Node process started correctly, executed its initialization phase, and then exited cleanly without creating child processes or errors.

Key observations:
- Node runtime initialized standard libraries and core modules.
- No evidence of subprocess spawning or shell execution.
- No signs of privilege escalation or modification of user/group identifiers.
- Process exited normally after completing initialization.

3. File Access Activity

The updated strace logs show multiple successful file open/read operations related to internal project and Node.js library files. Observed paths include application files within `/home/auditor/app/`, JavaScript dependencies, and standard Node runtime modules. No unauthorized access attempts were recorded. The application did not modify or write to sensitive areas such as `/etc/`, `/root/`, or `/home/auditor/.ssh/`.

Summary of file behavior:
- Frequent read operations from project source and library files.
- No write operations to restricted or external directories.
- Temporary files, if created, were confined to `/tmp` and removed automatically.
- No signs of hidden or persistent data creation.

4. System Call Summary (Based on strace_log.19)

Analysis of system calls confirms normal interpreter behavior. The primary syscalls observed include:
- openat(), read(), fstat(), mmap(), mprotect(), close(), brk(), futex()

No evidence of network-related system calls such as connect(), sendto(), recvfrom(), or bind() was found. This verifies that the application made no network attempts during execution. Additionally, no suspicious process management calls (fork(), clone(), execve()) were detected.

Overall, the syscall patterns reflect standard Node.js initialization and I/O operations with no anomalies or system-level modifications.

5. Network and Process Observations

Network inspection (`ss.txt`) again showed zero open or listening sockets. No outbound or inbound connections were attempted by the process, confirming compliance with the `--network=none` configuration.

Process inspection (`ps.txt`) showed a single primary Node.js process with no children or background threads. Memory and CPU usage remained within the expected sandbox limits (PID < 200, <1 GB RAM, 1 CPU core).

6. Security Evaluation

Category	Observation	Status	Remarks
Process Management	No subprocesses or privilege changes	✅ Safe	Single main process executed normally
Network Activity	No socket creation or connection attempts	✅ Safe	Full network isolation verified
File Access	Only local project and Node system files read	✅ Safe	No unauthorized writes or hidden file creation
Privilege Operations	None observed	✅ Safe	User auditor remained

			unprivileged
Code Stability	Clean exit with no errors	✅ Safe	Runtime completed successfully
Backdoor Indicators	None detected	✅ Safe	No suspicious syscalls or persistent processes

7. Recommendations

1. Maintain strict Docker sandbox isolation for all dynamic tests.
2. Verify the completeness of module dependencies before runtime to prevent false negatives.
3. Conduct a brief static audit (e.g., npm audit, eslint-plugin-security) to supplement dynamic findings.
4. Keep runtime logging active for post-analysis verification.

8. Conclusion

This updated analysis confirms that Implementation 07 behaves safely and predictably under sandboxed conditions. The application executed normally, performed standard file reads, and terminated without errors or network communication attempts. No malicious, persistent, or unauthorized activities were detected.

✅ Final Assessment: PASS — No security anomalies or risks observed.
The project demonstrates compliant, stable, and secure runtime characteristics.