Dynamic Security Analysis Report — Implementation 03

1. Overview

This dynamic analysis was conducted inside a controlled sandboxed Docker environment using a non-privileged user (auditor) with network isolation (--network=none) and restricted system capabilities (--cap-drop=ALL). The goal was to detect any hidden backdoors or unintended vulnerabilities during runtime of the g29-chat-protocol-main system, which is a Node.js–based chat application.

Runtime monitoring tools included:
- strace – for system call tracing (file, process, and network activities)
- inotifywait – for file system modification tracking
- ps and ss/netstat – for process and socket status snapshots

Duration of observation: approximately 60 seconds.

2. Runtime Behavior Summary

2.1 Process Tree
The running process tree (ps.txt) showed a single active Node.js main process executing app.js. No secondary subprocesses (e.g., /bin/sh, bash, or system-level utilities) were spawned, suggesting no attempt at process injection or external command execution.

2.2 File System Activity
From the inotify and strace logs:
- Files under /home/auditor/app/handlers/, /server-messages/, and /frontend/ were accessed normally.
- Read/write operations were mainly to configuration files (.env) and JavaScript source modules.
- No suspicious file operations were detected on sensitive system paths such as /etc/passwd, /root/, or /home/auditor/.ssh/.
- No temporary file creation in /tmp outside normal runtime logging.

This indicates clean and isolated file I/O behavior.

3. Network Activity Analysis

The captured socket snapshot (ss.txt) showed no listening or established network connections — as expected, because the container was launched with --network=none. Additionally, system call tracing revealed no calls to connect() or bind() involving external IP addresses or loopback ports, meaning there were no hidden outbound connections or unauthorized internal port listeners.

Therefore, network behavior is clean and compliant with security expectations for a sandboxed review.

4. System Call & API Observations

From strace_log.13:
- The application performed typical Node.js system interactions, such as module imports, file descriptor management, and event loop polling.
- There were no system calls related to process control (execve, fork, clone) or privilege escalation (setuid, setgid).
- No evidence of keylogging, file exfiltration, or unauthorized access.

Overall, the application displayed normal and contained runtime behavior.

5. Security Assessment

Category	Observation	Risk Level	Comments
Process spawning	No shell or subprocess creation	Safe	No evidence of command injection
Network connections	None detected	Safe	Expected under --network=none
File access	Only local project files accessed	Safe	No sensitive system files read
Privilege operations	None detected	Safe	No setuid or privileged syscalls
External dependencies	Node modules used normally	Safe	Standard npm ecosystem

6. Recommendations

1. Continue using sandboxed Docker for future analyses — it prevents unintended external connections.
2. Add logging to sensitive modules (e.g., authentication handlers) for better runtime traceability.
3. Conduct static analysis (e.g., npm audit, eslint-plugin-security) to complement dynamic review.
4. Validate all input parsing in routes/ and handlers/ to ensure injection safety even outside the sandbox.

7. Conclusion

The third chat system implementation (impl03) executed cleanly within an isolated audit container. No suspicious activities, hidden backdoors, or unintended network communications were detected during dynamic observation. The runtime behavior aligns with expected Node.js chat server operations.

Result: PASS (No security anomaly found) — The system demonstrates low overall security risk.