Attack Surface and Hardening Report for `time_daemon.c`

Scott Nguyen

1 Introduction

This report presents an analysis of the attack surface of a simple Linux daemon program written in C, `time_daemon.c`, which logs the current time to syslog every second. The daemon is launched at boot, detaches from the terminal, and gracefully terminates on signal. This report outlines the potential attack surfaces, their associated risks, and proposes steps to harden the code.

2 Attack Surface Model

An attack surface is the total set of points where an unauthorized user could try to enter data to or extract data from a program. For the `time_daemon`, the primary attack surfaces include:

- **Signal Handling:** The program listens for `SIGTERM` and `SIGINT`. An attacker could attempt to kill the process if it runs with elevated privileges.
- **Environment Variables:** If the daemon inherits untrusted environment variables (e.g., from init system), it could behave unpredictably.
- **Logging to Syslog:** If log messages are not properly formatted, they might cause confusion or log injection.
- **Process Privileges:** If the daemon is run as root, it increases the risk in case of compromise.
- **Resource Usage:** Unbounded logging or poor time retrieval could affect system performance.

3 Security Risks Identified

1. **Signals sent by any user:** If the daemon runs without privilege separation, any user could terminate it.
2. **Lack of input sanitization:** While the program only logs internal time, the formatting functions could still fail with unexpected inputs.
3. **No privilege dropping:** The daemon runs with inherited user privileges; if started as root, it remains root.

4. **Missing PID file:** This could allow multiple instances to run simultaneously, leading to unpredictable behavior.
5. **Basic signal handling:** Using `signal()` instead of `sigaction()` is less safe and portable.

4 Hardened Implementation Summary

The following changes are recommended or already implemented to reduce the attack surface:

- Replace `signal()` with `sigaction()` to improve robustness and prevent handler override vulnerabilities.
- Add privilege dropping after daemonization if initially started as root (e.g., via `setuid()`).
- Validate and sanitize all output to syslog to avoid malformed entries or log injection.
- Create and lock a PID file in `/var/run` to prevent multiple instances.
- Set resource limits (e.g., with `setrlimit()`) to prevent log spam or fork bombs.
- Clear sensitive or unnecessary environment variables after startup.

5 Conclusion

By identifying and reducing the attack surface of `time_daemon.c`, we make the daemon safer and more reliable for real-world embedded use. This process helps ensure that even small programs follow best practices in software security and maintainability.

Appendix: Source Code

```c
// cc -o time_daemon time_daemon.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <syslog.h>
#include <time.h>
#include <signal.h>

static volatile int running = 1;

void signal_handler(int sig) {
    if (sig == SIGTERM || sig == SIGINT) {
        running = 0;
    }
}

int main() {
    pid_t pid, sid;

    // Fork the parent process
    pid = fork();
    if (pid < 0) {
        exit(EXIT_FAILURE);
    }
    if (pid > 0) {
        // Exit the parent process
        exit(EXIT_SUCCESS);
    }

    // Create a new session ID for the child process
    sid = setsid();
    if (sid < 0) {
        exit(EXIT_FAILURE);
    }

    // Change working directory to root
    if ((chdir("/")) < 0) {
        exit(EXIT_FAILURE);
    }

    // Close standard file descriptors
    close(STDIN_FILENO);
    close(STDOUT_FILENO);
    close(STDERR_FILENO);

    // Setup signal handling to allow clean shutdown
    signal(SIGTERM, signal_handler);
    signal(SIGINT, signal_handler);

```

```c
    // Open connection to syslog
    openlog("time_daemon", LOG_PID, LOG_DAEMON);

    while (running) {
        time_t now = time(NULL);
        if (now == ((time_t) -1)) {
            syslog(LOG_ERR, "Failed to get current time");
        } else {
            struct tm *ptm = localtime(&now);
            if (ptm == NULL) {
                syslog(LOG_ERR, "Failed to convert time to local time");
            } else {
                char time_str[64];
                if (strftime(time_str, sizeof(time_str), "%Y-%m-%d %H:%M:%
                    S", ptm) == 0) {
                    syslog(LOG_ERR, "strftime returned 0");
                } else {
                    syslog(LOG_INFO, "Current time: %s", time_str);
                }
            }
        }
        sleep(1);
    }

    syslog(LOG_INFO, "Daemon terminated");
    closelog();

    return EXIT_SUCCESS;
}
```

Listing 1: `time_daemon.c`