Attack Surface Hardening

I. Overview

This program is a Linux daemon written in C that runs in the background after double-forking and logs the system time every second using syslog. It is designed to run as a minimal IoT kind of background task.

II. Attack Surface

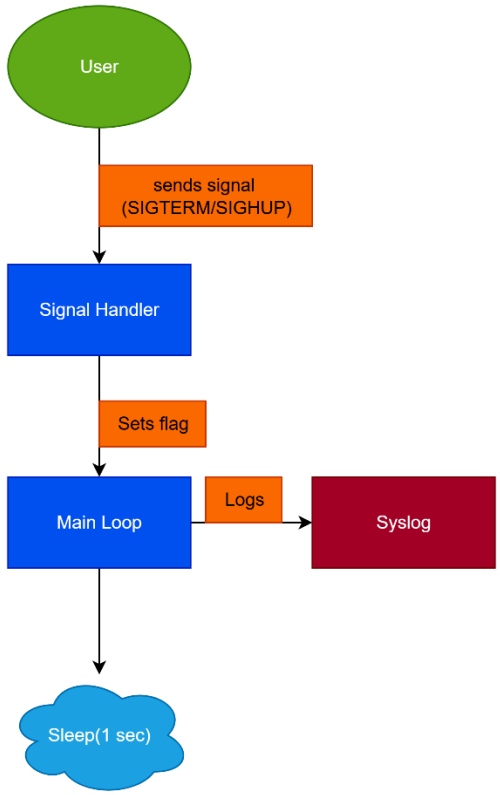
The attack surface includes all the ways that I assume an attacker could influence or interact with the daemon program.

Signals	The daemon accepts SIGTERM and SIGUP. If signal handling is not implemented correctly, it could case problems like crashes or premature exits.
Env.	The daemon would normally inherit parent env variables, but this program does not use any. The env might still affect behavior by locale settings though.
Syslog	If this is not rate limited or properly formatted, then the daemon could be vulnerable to log flooding or log injection.
File Descriptors	The daemon calls localtime() and strftime() to format timestamps. These expect a structured time and date and a properly sized buffer. If the buffer is too small or the time is corrupted, formatting could fail or the memory could be overwritten.
Runtime Behavior	The main loop runs forever. If it's not bounded properly or monitored, it could keep running forever even if there is abnormal conditions. An infinite resource consumption scenario is possible.

III. Hardening Measures

Risk	Hardening
Unsafe signal handling	Used sig_atomic_t flags and then handled the signals in the main loop.
Unchecked return values	Validated return values
Time format overflow	Checked strftime() result and added error logging for it.
File descriptor leakage	Closed and redirected stdin, stdout, stderr to /dev/null
Logging reliability	Added LOG_CONS to openlog() to make sure message goes to the console if syslog fails.
Unexpected input	Avoided getenv() or command line args to limit the input vectors.

IV. Flow of Program



V. Conclusion

There isn't a lot to say about the changes to the daemon. It was researching different attack paths that malicious individuals might use, and the solutions others have thought of for them. This exercise helped identify ways that were unknown to me of how a background system-level C program could be influenced by external actors. Hardening this was mostly focused on eliminating any unsafe operations and sanitizing the inputs (which also means if there is no input!). The final version of this program resists most of the basic manipulation or misuse that I have found.