🧬 ARM ELF Malware Analysis Report

🔧 Environment Setup Summary

- **Rootfs Used:** OpenWRT ARM uClibc rootfs (manually patched with `libc.so.0` and `ld-uClibc.so.0` from `arm_tendaac15`)
- **Emulator:** Qiling Framework
- **Hooked Elements:**
  - Fake system calls (`open`, `socket`, `connect`, etc.)
  - Deceptive file system mappings (`/proc`, `/dev`, `/etc`)
  - Patched missing symbol `__uClibc_main` using `ql.set_api()`

🔬 Behavioral Breakdown

1. ✅ Execution Started Cleanly

- Entry point mapped and reached.
- Interpreter `/lib/ld-uClibc.so.0` loaded correctly.
- Dynamic linking succeeded; over 50 symbols resolved (e.g., `system`, `socket`, `execv`, etc.).

2. ✅ Library Dependencies Resolved

- Opened `/lib/libc.so.0` successfully (fake).
- Memory segments allocated with `mmap2` and marked executable.
- Initial `write()` syscall output shows malware banner and error propagation.

3. 🔐 System Calls Attempted

Category	Syscalls Observed	Simulated Behavior
File I/O	`open`, `fstat`, `close`, `access`, `unlink`	All returned fake success
Networking	`socket`, `connect`, `bind`, `setsockopt`	Pretended to succeed (fd=3)
Memory	`mmap2`, `munmap`, `brk`	Normal ARM memory management
Process	`fork`, `execv`, `getpid`, `getppid`	Allowed but monitored
Signal	`sigprocmask`, `signal`, `kill`	Setup observed, allowed
Sleep/Wait	`nanosleep`, `sleep`, `usleep`	Ignored to skip delays
IOCTL & misc	`ioctl`, `fcntl`, `readdir`, `opendir`	Returned 0 or stubbed safely

4. ⚠ Missing Symbol Crash Avoided

- Symbol `__uClibc_main` was not found in libc → initially caused crash
- Bypassed using `ql.set_api()` to inject a fake `__uClibc_main()`

5. 🧠 Deception Mode Behavior

- All API calls returned fake-success
- Malware continued past initialization logic
- `write()` messages confirmed it **believed it had failed to load libc**, then **believed it failed to resolve **`__uClibc_main`**
- Eventually exited cleanly with code `0x1`

🎯 Recommendations for Next Steps

- Enable full memory tracing for dropped payloads (hooks on `write`, `execv`)
- Patch additional libc stubs (like `system`, `readdir`, etc.) to monitor intent
- Simulate C2 replies to `recv` to monitor command flow
- Compare behavior against other ELF samples to identify reused logic

---

🗃️ Artifacts

- `mimic_run_2.py` → Fully hooked deception script
- `arm_uclibc rootfs` → Contains patched `libc.so.0`, `ld-uClibc.so.0`
- Qiling logs → All syscall interactions

✅ Outcome

Malware reached execution logic beyond interpreter and linking. It was deceived into believing libc resolved but couldn't initialize. It showed behavior typical of:

- IoT persistence techniques
- Signal/child handling
- Network backdoor setup (ports, C2 IP)

Environment now fully capable of dynamically observing similar ELF samples.
