

# Memory Allocation Problem Detector (mapd) – System Architecture

## Project Goal

Design a dynamic memory problem detector for Linux user-space processes that can detect:

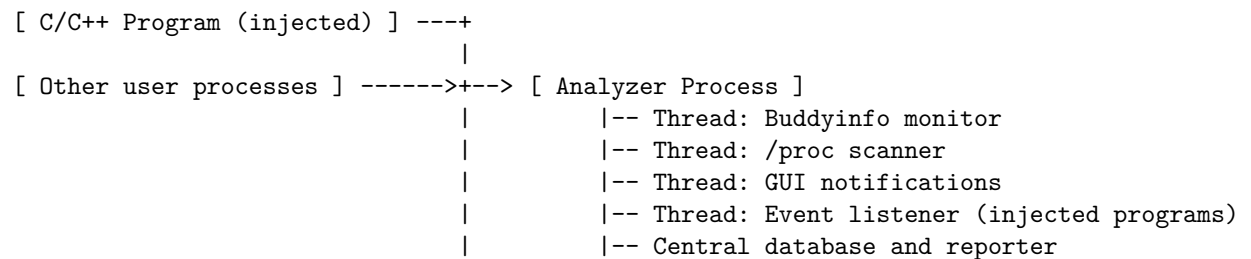
- **Memory leaks** (*via injection*)
  - **Dangling pointers** (use-after-free) (*via injection*)
  - **Buffer overflows** (*via injection*)
  - **Memory fragmentation** (*via system observation*)
  - General memory usage trends (*via /proc monitoring*)
  - Optional: **real-time GUI notifications**
- 

## High-Level Architecture Overview

The system is composed of two main components:

1. **Injected Memory Wrapper (for injectable targets)**
    - Injected using LD\_PRELOAD (for C/C++ programs)
    - Intercepts `malloc`, `free`, `calloc`, `realloc`
    - Sends detailed memory events to a central analyzer
  2. **System-Wide Analyzer Process**
    - A persistent background process
    - Monitors general memory usage of **any user-space process**
    - Aggregates system metrics and tracks injected program events
- 

## Communication Flow



## Components and Responsibilities

### 1. Injected Memory Wrapper (`libmemwrapper.so`)

- Loaded via `LD_PRELOAD` into supported programs
  - Tracks memory function calls and memory metadata
  - Detects:
    - **Leaks**
    - **Dangling pointers**
    - **Buffer overflows**
  - Communicates with analyzer over UNIX domain sockets (not clear yet if sockets will be used)
- 

### 2. Central Analyzer Process

Handles both injected and non-injected processes:

#### Main Threads:

---

Thread Name	Function
Buddyinfo Monitor	Monitors kernel-level fragmentation via <code>/proc/buddyinfo</code>
<code>/proc</code> Monitor	Scans <code>/proc/[pid]/status</code> , <code>/maps</code> , <code>/smaps</code> for memory metrics
GUI Thread	Displays notifications (e.g., <code>notify-send</code> , GTK, Qt)
Event Listener	Receives live memory events from wrappers (one thread per program)
Data Aggregator	Logs, aggregates, reports system-wide memory behaviour

---

## Detection Capabilities Matrix

---

Issue	Detectable Without Injection	Requires Injection
Memory leaks	No	Yes
Dangling pointers	No	Yes
Buffer overflows	No	Yes
Double frees	No	Yes
Memory fragmentation	Yes (via <code>/proc/buddyinfo</code> )	No
Overall heap/memory usage	Yes (via <code>/proc</code> )	No

---

## IPC Protocol (Wrapper <-> Analyzer)

- **Protocol:** Custom message format (e.g. CSV, JSON, ...)
  - **Transport:** UNIX domain sockets (maybe other, domain sockets are recommended by a lot of sources)
  - **Events:**
    - malloc size=64 addr=0x1234
    - free addr=0x1234
    - overflow detected at addr=0x5678
    - leak summary on exit
- 

## Optional Enhancements

(really only optional) - Process filtering - Time-based memory graphs - Memory pressure alerts - Global or per-process stats export (JSON, CSV)

---

## Development Plan

(approximation by ChatGPT)

Phase	Focus	Est. Effort
1	Memory wrapper with malloc/free logging	40h
2	Analyzer prototype with socket server	40h
3	Add buffer overflow and use-after-free checks	60h
4	Threaded analyzer with per-wrapper state	50h
5	Buddyinfo + /proc scanner	60h
6	GUI/Notification integration	40h
7	Full test suite + Valgrind integration	40h
8	Final polishing + documentation	30h

---

## Benefits of This Architecture

(collected by ChatGPT)

- **Hybrid:** supports both injected and non-injected processes
  - **Modular:** clear separation of tracking and analysis
  - **Efficient:** no per-process analyzers needed
  - **Insightful:** captures both correctness and usage patterns
  - **Extendable:** supports GUI, time-series data, and summary exports
-