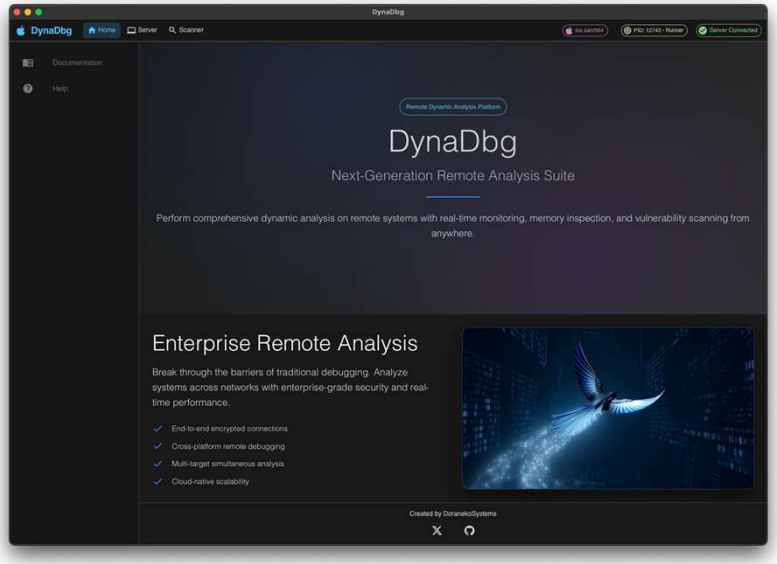
Creating a GUI-based macOS/iOS ARM64 Debugger

September 14, 2025
Kenjiro Ichise
@DoranekoSystems

Security-Nexus-Hub

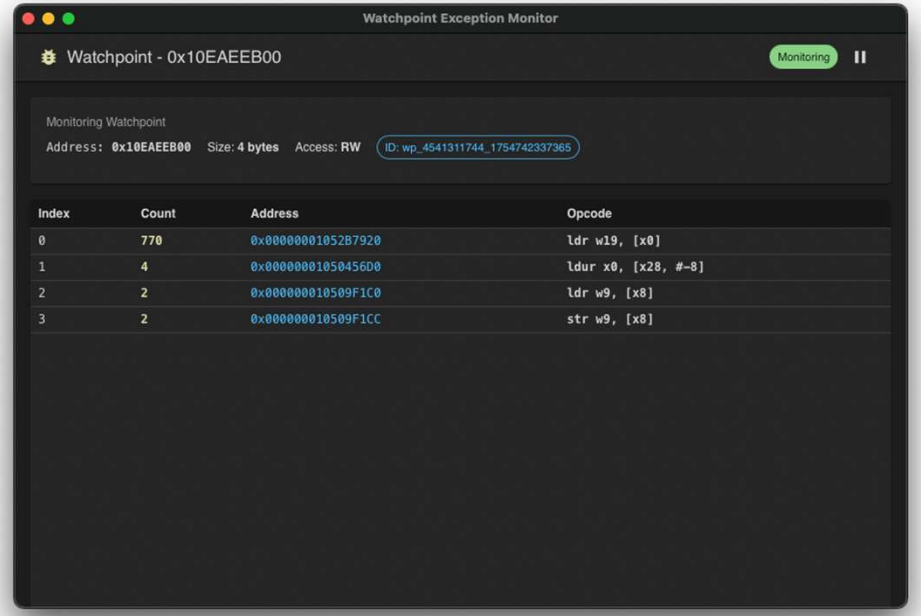
What I’m creating this time

- Aiming to create a general-purpose GUI debugger like x64dbg.
  x64dbg:https://x64dbg.com
- Basic functionality supports Windows/Linux/Mac/Android/iOS.
- Currently developing debuggers for each platform.



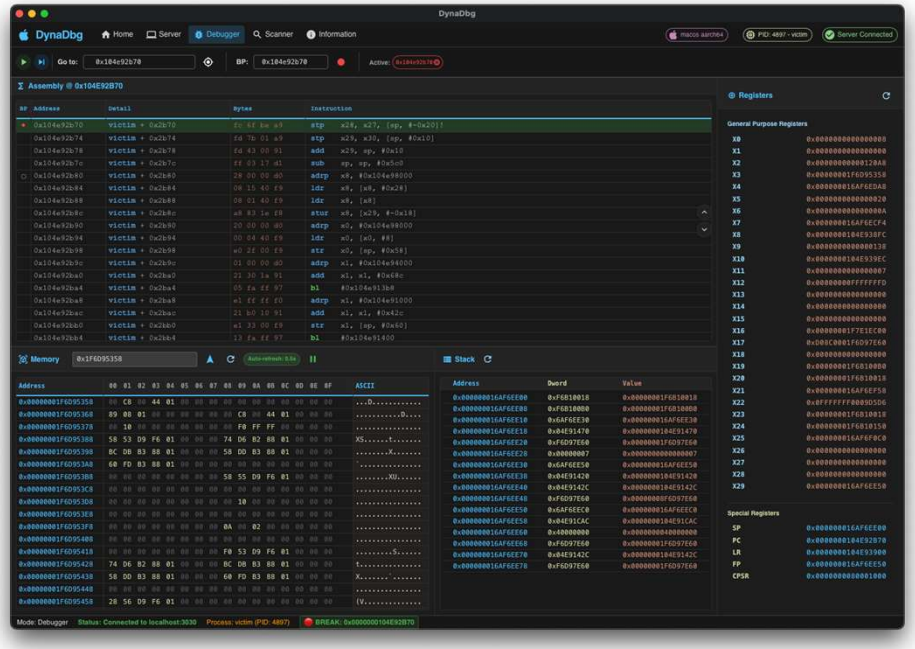
Design

- (Host) Desktop Application: Tauri (Rust + React/Vite)
- (Remote) Backend API: Rust (Warp)
- OS-specific native components: C++



Current Development Status

- macOS/iOS version
  - Implemented Hardware Breakpoint/Hardware Watchpoint.
  - Implemented functionality to advance by one instruction from the breakpoint address (Single Step).



Prev: How to Create a Memory Search Tool

- Reference:
  DeNA/mempatch
  https://github.com/DeNA/mempatch

  In the source code, the following files under jni handle memory read/write operations for each OS.
  - Memory_Linux.cpp
  - Memory_Dwarwin.mm
  - Memory_Windows.cpp

- Minimum requirements for a memory search tool
  - Function to read memory from other processes.
  - Function to obtain memory maps of other processes

Example: To read memory from other process

- Windows:
  - Obtain handle with OpenProcess(PROCESS_VM_READ permission).
  - Read memory with ReadProcessMemory.
- Linux/Android:
  - Method 1: ptrace system call (PTRACE_PEEKDATA, etc.).
  - Method 2: Direct reading from /proc/{pid}/mem file.
  - Method 3: process_vm_readv (efficient, Linux 3.2 and later).
- macOS/iOS:
  - Obtain task port with task_for_pid (requires privileges).
  - Read memory with mach_vm_read_overwrite.

※Admin privileges or security setting adjustments are required for each OS.

How to Create an macOS/iOS Debugger

- Many debuggers utilize LLDB's debugserver.
- debugserver uses a communication method based on GDB's remote serial protocol, enabling memory reading and breakpoint.
- Packet format: $<data>#<checksum>
- Examples of data section:
  - Attach to target PID: vAttach;{pid:02x}
  - Read specified size from target address: x{address:02x},{size}
  - Read register value at target index: p{index:02x}

For technical exploration purposes, I am challenging myself to create a custom implementation without using debugserver!

Basic Principles of macOS/iOS Debugger

- Use debugger privileges to properly handle exceptions that occur in the target process.
- Debugging flow for macOS/iOS (overview)
  1. Connect to target process (task_for_pid)
     Obtain administrative privileges for the process you want to debug, enabling memory read/write and thread operations.
  2. Start monitoring exceptions with specified conditions (task_set_exception_ports)
     Register with the system to receive exceptions (breakpoints, memory violations, etc.) that occur within the process in the debugger.
  3. Wait for exceptions (mach_msg_server)
     Wait for exception notification messages from the system in an infinite loop, automatically calling handler functions when received.
  4. Handle exceptions (catch_exception_raise)
     Analyze received exception information (which thread had what abnormality) and pass it to the main debugger processing.

Single Step Method on macOS/iOS ARM64

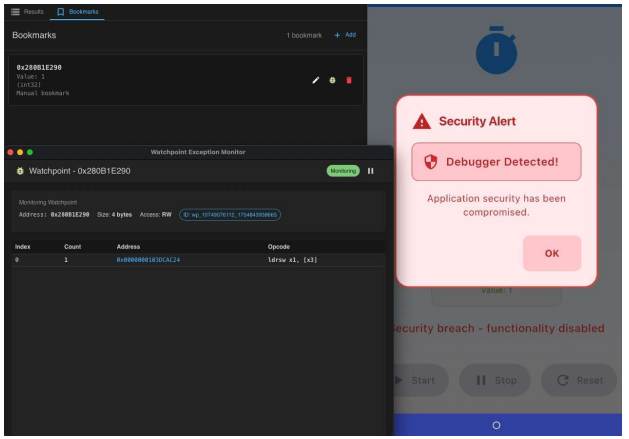
- Single Step (advance one instruction) flow for macOS/iOS.
- Operations are performed on the target thread

1. Get current debug state (thread_get_state)
2. Enable single step bit (debug system control register: mdscr_el1)
3. Apply changes (thread_set_state)

- Characteristics of macOS/iOS
  Unlike Linux/Android, macOS/iOS allows direct access to debug system control registers.
- On Linux/Android, when advancing one instruction, ptrace(PTRACE_SINGLESTEP…) is used, but the kernel controls mdscr_el1 on the kernel side.

What I Learned Through Development

- As a side benefit, I deepened my understanding of macOS/iOS anti-debug techniques.
- Method 1: Countermeasure by obtaining exception monitoring status for own process task_get_exception_ports.
- Method 2: Monitor debug control registers thread_get_state Monitor debug-related flags that are set.



References

- DynaDbg
  https://github.com/DoranekoSystems/DynaDbg