Liquid Glass Challenge(incomplete solve)

Goal:
Understand and exploit the custom WebAssembly “liquid glass” image filter used in the challenge.

What I did

1. **Recon:**
   - Inspected the provided JavaScript bundle and found the `apply_liquidglass` function.
   - Discovered that the raw WAT (WebAssembly Text Format) was embedded directly in the JavaScript.
     Extracted this WAT to study how the filter works without running `wasm2wat` myself.
2. **Disassembly:**
   - Decompiled the WAT to C code using `wasmdec` to get a higher-level view of the algorithm.
     Understood the nested loop structure, pixel offset calculations, and buffer writes.
3. **Tried Dynamic Analysis:**
   - Tried to replicate the WASM module’s behavior in Python using `wasmtime`.
     Managed to load the module and access its memory.
     But struggled with properly writing image bytes into the WASM `Memory` because `wasmtime` in Python does not allow direct assignment like JS’s `Uint8Array`.
     Realized I need to learn more about how to handle WASM memory buffers properly in Python.
4. **Status:**
   - Did not yet get the Python version working.
   - Next step is to study `wasmtime` or other Python WASM runtimes to correctly read/write the WASM `memory` buffer.

Key Takeaways

- Located and extracted embedded WAT from JavaScript.
- Decompiled to C for easier reading.
- Understood the core image distortion algorithm.
- Need to improve skills with WASM runtimes in Python to complete dynamic testing.

Next:

- Learn WASM memory API in Python (`wasmtime.Memory`, `memoryview` or `ctypes` wrappers).
- Try simple WASM test modules first to practice reading and writing bytes.
- Then redo the image exploit with full control over input/output.