MQP A Term Plan of Work

Zachary Porter

2022-08-22

1 Problem Statement

It is currently difficult to identify the flow of ”program-level”/macro logic from static or even dynamic analysis.

2 Goal Statement

I hope to use record and replay technology to allow researchers to move easily peer into a binary to find important addresses and to identify the most important instructions for ”program-level” logic.

3 Minimum A-Term Deliverables

- Code capable of tracing data through a program's execution
- A plan for comparing the traces and identifying shared points between traces
- A platform for my UI that I am comfortable with
- Recording infastructure to support scrobbling along the timeline of a program
- Tests for all code I have written (within reason)
- Documentation for all code that reasonably requires documentation

4 Minimum Definition of Success

Proof of concept that accomplishes:

- Ability to record execution of programs
- Ability to determine relevant addresses and instructions for a certain time period
- Ability to compare multiple code flow graphs and create a combined one that shows where the different graphs differ. This would be used to determine addresses where a change of state can result in a change of program flow.

5 Maximum Reasonable Definition of Success

Proof of concept that accomplishes:

- Seeing all instructions that modify an address throughout the execution of a program (or in a smaller section of time)
- Ability to insert compiled javascript via frida during execution / ability to modify instructions if frida doesn't pan out.
- Ability to see all code that was run during a certain time interval
- Ability to pop open a GDB shell or open Ghidra to the current location at any time during the debugging process
- Ability to either create a layer over an executable to insert new instructions or modify memory addresses at runtime (when no longer debugging).
- Ability to show a version of a code-flow graph (Not exactly a code flow graph as that isn't terribly interesting for most people, but rather a graph that shows all of the variables used in a code segment and their relationships to a segment of code)
- Ability to compare two separate executions of a section of code and see where they differ (i.e. see where jump instructions are executed and show different code flow graphs).
- Ability to select hone in on certain processes
- Ability to do this with programs that use X11 (this might be very hard. Not something I am going to tackle for a while).

6 Use Cases

- Changing non-configurable default settings of a binary (like default font in MS-Word)
- Adding/removing key bindings to a non-configurable binary (Macros have proven to be useful. This would be super-macros.)
- Instrumentation of binaries in complex manners (write to a file on: phone-home messages, GUI draw events, etc. (I obsessively publish almost all of my data to a MongoDB cluster. This would allow me to do that with closed source apps.))
- Understanding what causes a program to crash / comparing execution states (reverse engineering).
- Removing outdated hardware-based limitations (For retro-video game modding)
- Changing constants that affect program flow in a way a user doesn't desire (Error popus, music playing, etc.)
- ... this will change *significantly* as the project develops

7 Biggest concerns / possible challenges

Perspirative challenge: one that can be overcome with the application of time and energy (perspiration).
Inspirational challenge: one that can be overcome with the application of careful thought and research

- JIT / Dynamic linking (Inspirational): Anything that modifies executable code at runtime worries me.
- Xorg / anything that uses Unix domain sockets (Inspirational): I don't know how to playback these events or understand them in a temporal sense. This might not be a big worry but it is something I've spent a lot of time thinking about.
- Code insertion (Perspirative): I really don't want to underestimate this part of the project.
- UI (Mix): UI's are not my area of expertise. I can write simple things but this is something that may take more time than I expect. (Depending on what path is chosen for the UI)

8 Required Research

(in order of priority)

1. Deep dive on recording software and inexpensive interrupts
2. Fairly deep dive on function inlining and functions that are the same across compilation unit boundaries (weak references / ODR violations). I am also curious about LTO (link time optimzation) and how that changes the produced binary
3. Deep dive on concolic programming / SAT solving. I understand the basics but now I need to understand more than that. I also not very familiar with the tools in the space.
4. Deep dive on dynamic linking (Good lectures 1 2) (This will play a large role in program understanding and I need to really understand how it works at a low level.)
5. Shared memory and unix domain sockets with regards to Xorg (Ability to record xorg will have a large on final product so I need to start that early.)
6. Additional research on Frida and possibly LiveRecorder
7. Code flow graph generation techniques (If a tool that is suitable for this exists I would rather use that).
8. JIT compilation (for now I will stay far clear of anything with a non-static set of instructions)

9 Target Platform

Linux with kernel 5.15+ running on x86 via a user who *could* run as root if needed.