MQP Pre-Draft 1

Zachary Porter

2022-02-14

1 Problem Statement

Many software applications are build on the assumption that code, once compiled, is static and non-modifiable. I think it is possible to build binary wrappers that modify executables (at run time) to allow for easy program modification of non-exposed program functionality.

2 Software that already exists to deal with this topic

- frida (Any): Allows for arbitrary injection of compiled javascript with QuickJS into a currently running process. This mainly works on the function level and allows for injection of code on arbitrary function entry and exit (of dynamically linked libraries). This is a super cool tool and something I might work heavily with.
- mach inject (macOS) : Allows copying over code into another processes address space and spawning threads to run that code.
- DynamoRIO (any) : Similar to Frida. "Allows arbitrary modifications to application instructions via a powerful IA-32/AMD64/ARM/AArch64 instruction manipulation library. DynamoRIO provides efficient, transparent, and comprehensive manipulation of unmodified applications running on stock operating systems."
- The backdoor factory (any) : Allows patching of executables to run shell code and then revert the binary to its original state.
- Cheat Engine (Windows + macOS) : This tool is a memory scanner / debugger. Its primary functionality comes through repeatedly pausing an application and then performing memory scans to identify addresses that satisfy certain criteria. This allows the user to isolate and identify addresses in memory that contain values they are interested in (such as in-game health, etc). It also allows for code injection and game-focused / DirectX hacks.
- scanmem (Linux) : This tool is very similar to Cheat Engine however it lacks other game-focused features and primarily focuses on memory address isolation.
- radare2 : Advanced decompiler with many features built in like disassembly and code-flow graph generation.
- ghidra : Advanced code analysis tool to compete with IDA Pro.

- RR : Record and replay software which allows for the recording and then replaying of a programs execution. This works via capturing all non-deterministic factors of a program's execution and then using those for the replay. "rr records a group of Linux user-space processes and captures all inputs to those processes from the kernel, plus any nondeterministic CPU effects performed by those processes (of which there are very few). rr replay guarantees that execution preserves instruction-level control flow and memory and register contents. The memory layout is always the same, the addresses of objects don't change, register values are identical, syscalls return the same data, etc. " This is a super cool tool and something that I might work heavily with.

Etc. There are more tools out there but these are the main ones. I will add to this list as I find more tools that I find interesting.

3 What these tools don't solve

These tools are great for information collection / reverse engineering and even basic editing of binaries however they fail to make binary editing feasible to anyone but those who have a deep understanding of what is most likely happening on these processes. Non-simple program modifications with tools like Cheat Engine are incredibly difficult and are often done in tandem with tools like gdb and ghidra (I have watched a few advanced tutorials on the software).

4 Minimum Definition of Success

Proof of concept that accomplishes:

1. Ability to record execution of programs
2. Ability to determine relevant addresses and instructions for a certain time period
3. 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.

This does not include code insertion or modification however that is just because I want to set the bar low before I dive into the subject over the summer.

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 Timeline

I plan on doing a great deal of research and solving some of the inspirational problems during Spring and Summer 2022. This includes activities like reading papers, studying tools like rr and frida, creating proof of concepts that can record xlib, etc. I plan on truly starting in the beginning of A term 2022 and finishing either at the end of A or B term 2022. I have finished my classes for my major but haven't done my MQP so at this point I'm leaning towards just calling it B term but I need to plan things out more.

9 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)

10 Target Platform

Linux with kernel 5.15+ running on x86 via a user who *could* run as root if needed. This isn't really something I am interested in changing at the moment. This is the hardware I run and want to develop on for the time being.

11 MQP Advisor / Co-Advisor

Right now, I really don't know how MQP advisors work. Right now I am envisioning 30 minutes every week or every odd week of meeting + a few occasional emails. Mostly I am looking for your brilliance and wisdom that you have accumulated in order to help me know what paths I should explore.

12 Team

I am currently in talks with a friend of mine, Ted Clifford to join this MQP. He has worked for Mitre for two summers and is well versed in security and is passionate about this layer of the stack. However he is employed by Mitre at the moment and isn't sure if he would rather just do an MQP with Mitre.

I am still reaching out to people who might be interested. If you meet anyone, please send them my way.