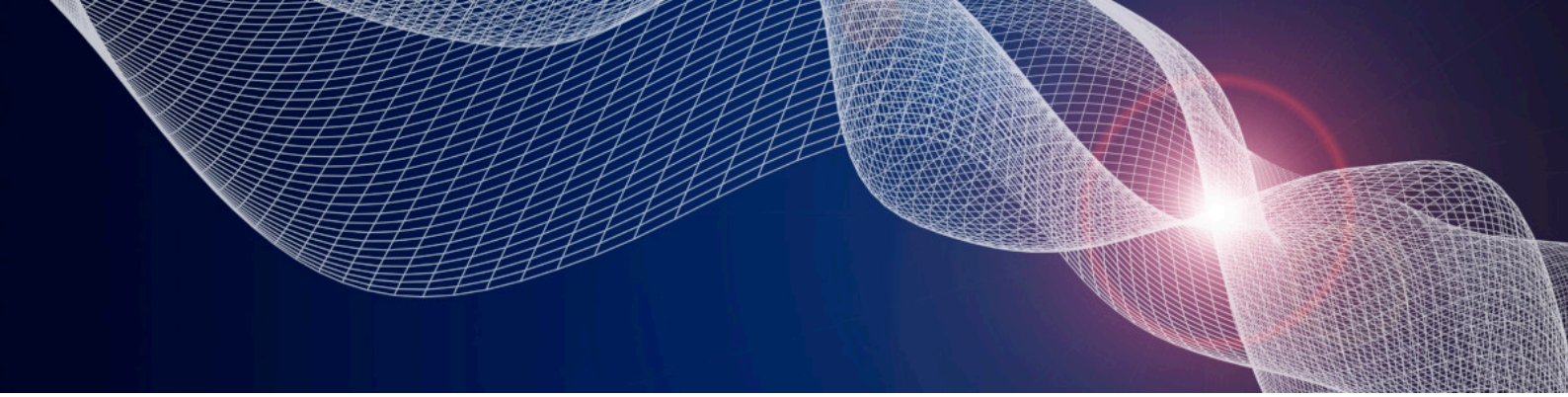

4. Dynamic Analysis

4.1 Program Analysis

Download `ex3-win.exe` from the chapter repository and import it into *Ghidra*. It's a very simple binary which was developed explicitly to be relatively simple to reverse engineer. Reality unfortunately often is less kind to us ;-)

4.2 Running a Program (30')

Open `ex3-win.exe` in the *Debugger* tool, connect the appropriate debugger and load the program with an argument (you'll figure out later what argument is expected, just put something for now).

1. Inspect the "Dynamic Listing" component and compare it to the static "Listing" component. Why are they showing completely different memory locations?
2. Have a look at the "Registers" component. Where are the *instruction pointer* and *stack pointer* pointing? Where can you find these memory locations?
3. What is displayed in the "Stack" component?

4.2.1 Suspending at a Known Location (45')

Set a breakpoint right at the beginning of the `main` method. Do what's necessary to get the program suspended there.

1. How do the "Dynamic Listing" and the static "Listing" components present now? Why is it different from before?
2. How is the "Stack" component changed? What happens if you click on the different lines presented there?
3. Where can you find the values for `argc` / `argv`? What are these? Can you find the argument you passed based on these?
4. How do you go about finding the actual text values passed to the program?

4.2.2 Stepping (60')

Advance the program to the method comparing the key entered to the value expected - probably best by setting another breakpoint.

1. How would you go about finding the expected value (without modifying program state - we only learn about that later ;-))?
2. Get the first three correct characters.

4.2.3 Modifying Register Values (60')

1. Using this new way of "tricking" the program, how can you go about finding the entire expected value?
2. Do it! What's the key?

4.2.4 Modifying Memory (60')

Instead of modifying registers to get the correct value, we can also modify memory to get the program to accept the current input and move on. How do you do that?

4.3 Patching Programs (30')

As a last exercise, get the program to accept any input. Document which instruction(s) you patched with which new one(s).

If you'd like to do something a little more interesting to finish up, get the program to accept your first name as a key - and your first name only!