Each 6.1800 lecture will come with an outline. You can fill this in during lecture, after lecture, or not at all — it's entirely up to you how you use it. The goal of these outlines is to help you understand the main points that you should be taking away from each lecture. In some cases we will also include examples of things you should be able to do after each lecture.

*In the past, these outlines have proved to be an effective tool for studying for the exams. Note that the outlines are **not exhaustive**; there will be topics and nuances in lecture that aren't captured by the outline.*

Lecture 22: Low-level attacks
*Note: You do **not** need to be able to use gdb or write code in C after this lecture.*

Stack-smashing

- What is our threat model for stack smashing?
- How does the program stack change when a function is called?
    - *Example: what gets pushed onto the stack at the start of a new function?*
- How does the program stack facilitate returning to the previous function after a function call ends?
- What assumption do the basic stack-smashing attempts in this lecture rely on?

*Given each of the examples in lecture, you should be able to explain the **basic steps** that an adversary would need to take in order to achieve their goal. Specifically, you should understand:*

- What the program stack looks like
- What the adversary's intent is (what variable(s) are they trying to overwrite, and in doing so, what are they trying to accomplish)
- What the "payload" should look like (i.e., the string that is passed as an argument to main). E.g., given a specific architecture: how many bytes should the payload be, what content should be included in the payload and where, etc.

Compilers

- What does a compiler do?
- How could a user detect Thompson's hack in the first version of our hacked compiler (the one that only inserts a backdoor into UNIX)?
- Why does this method of detection not work with our "hacked v2.0 C compiler"?