Do This and Nothing More: Teaching Adversarial Thinking Without Security

Jon Beaulieu
Mazin Jindeel
Aleksandar Straumann
Brandon Paulsen
Peter Peterson
University of Minnesota Duluth
Duluth, MN 55812
pahp@d.umn.edu

Abstract

Software Engineering has a computer security problem; programmers often think only of how to make a program work -- they often don't think about how it could fail. This contrast is at the core of the concept of "Adversarial Thinking" -- that security practitioners (and security conscious developers) need to think critically about about how a sufficiently motivated and intelligent attacker could make their systems fail.

Unfortunately, students are often only taught the concept of Adversarial Thinking in the process of teaching them common vulnerabilities and exploits. This is a problem. First, students who do not take a security course do not learn the value of the adversarial perspective. Second, students learn Adversarial Thinking in the context of specific flaws rather than in the context of programming in general. At the same time, exercises that focus on adversarial aspects of computing often require a significant amount of security expertise from participants and a large time investment from the facilitators.

One way to naturally teach adversarial thinking is to talk about how program specifications are not just a list of features that a program should have, but a contract that states "do this and nothing more." From that perspective, any program behavior not in the specification is a flaw. This concept is valuable for many reasons, not the least of which is that it does not presuppose any particular knowledge of security issues.

We created a framework, DTANM (Do This and Nothing More) that hosts competitions between teams of students. In these short (1-2 hour) competitions, students are given a small piece of code and a specification of how it should behave in terms of inputs,

outputs, and displays. For example, our first program was a simple command line calculator. Students look for flaws in the code that allow it to misbehave. When they find these flaws (e.g., the calculator allows an attempt to divide by zero), they fix them in their code and then force other teams' programs to process those "buggy vectors" by adding them to a list of tests that all teams' programs must periodically process. Teams are scored by comparing their program's behavior to a "gold standard" program (provided by the instructor) that is robust to all known attacks. In this way, teams work to improve their own code, while simultaneously thinking adversarially to discover flaws that enable unexpected behavior -- All without any special "security knowledge" -- only basic programming skills.

Students in UMD's Computer Security course have found this exercise very enjoyable, and the authors hope to expand its use to students in Intro to CS and Programming classes. The framework itself relies primarily on basic input / output and scripting. For each competition, the instructor needs to provide only the source code, a build script, and the "gold standard" version of the program, which means that the DTANM framework can support a broad range of different programs, making it flexible and useful for teaching security principles for years to come.