With new security incidents surfacing almost every single day, software vendors must meet ever-increasing standards of quality in systems security. As such, major vendors now spend massive amounts of resources to test their products and minimize the probability of exploitable software bugs—the kind of bugs that allow attackers to completely compromise a target system. In particular, much effort is devoted to very dangerous bugs that corrupt memory. Despite all these efforts, memory corruption bugs are still pervasive in systems software, with well-known examples such as the Heartbleed and StageFright vulnerabilities putting millions of users at risk. Since it is impossible to completely eradicate these bugs, the only option is to mitigate their impact in practice.

This dissertation considers all the stages in which one may protect software against attacks based on memory corruption bugs: testing before deployment, detection of corruption attempts after deployment for the bugs that we cannot find by testing, and exploit containment for the bugs that we cannot easily detect. Such a symbiosis between software testing and systems security can get us closer to a solution. Lessons learned in systems security help us concentrate the testing efforts on the areas of the program which are expected to be most vulnerable. Lessons learned in software testing help us incorporate validation techniques into the workflow of security solutions. Drawing from such lessons, this dissertation presents a concert of techniques at different layers of systems security, including bug finding, memory corruption detection, and control-flow enforcement.

In bug finding, this dissertation introduces guided fuzzing based on security prioritization, which aims to prioritize the testing effort around likely vulnerable code artefacts. Such prioritization typically occurs as part of white-box testing with manual input from developers, but we show that we can automate this process using...
static source analysis. This strategy lays the ground for effective modular testing, by automatically inferring the need to test certain code artefacts and avoid others.

In addition, this dissertation explores reverse engineering techniques to extend the guided fuzzing approach to scenarios where no source information is available. There is a significant gap in the literature regarding the classification of pointer-based data structures in binary programs. I bridge this gap via a novel approach based on behavioral tracking and a classifier based on refinement.

The dissertation also identifies issues in existing state-of-the-art control-flow enforcement schemes, including the one implemented in the GCC compiler today. I counter this threat by developing a novel white-box testing framework for a certain class of control-flow enforcement schemes, which allows developers to check if their mitigation fully respects C++ semantics, while at the same time offering the strongest possible protection for the program. I then leverage the framework to build a new mitigation, which fully achieves these goals, while also being faster than the variant used in GCC.

Finally, this dissertation shows that a major reason that enforcement of memory safety rules in C/C++ is inefficient lies in the overhead imposed by metadata tracking. I developed a new scheme, called variable metadata shadowing, which alleviates these concerns. My scheme is flexible enough to implement a wide range of memory safety checks, while imposing minimal performance and memory overhead during the execution. I then validate the metadata tracker in a novel type enforcement system, which detects type confusion bugs in real-world software (such as browser with over a million lines of code). The metadata tracker forms the backbone of this mitigation mechanism and I show that it can protect real applications with much greater efficiency than any other solution.

In conclusion, this dissertation presents evidence that we need a strong coupling between the areas of software testing and systems security. The best solution is to layer different techniques on top of one another and I show how each layer can be greatly enhanced via the proposed symbiosis.