Abstract

Modern websites evolved to interactive applications which process confidential user data, such as credit card numbers, passwords, and private messages, on a daily basis. This sensitive data requires reliable protection from cyber criminals who exploit vulnerabilities in the applications' source code. Particularly web applications developed in PHP, the most popular server-side scripting language on the Web, are prone to security vulnerabilities. Although the developers' awareness is rising for the traditional types of vulnerabilities, such as *cross-site scripting* and *SQL injection*, they still persist due to faulty security mechanisms or intricate language features. Besides, more complex vulnerability types, such as *second-order vulnerabilities* or *PHP object injections*, are comparatively unknown and actively exploited by attackers.

Manual detection of complex vulnerabilities in modern PHP applications with hundreds of thousands lines of code is expensive, time-consuming, and requires deep security knowledge. With the help of static code analysis, security vulnerabilities can be detected in an automated fashion and subsequently remediated. However, previous work in this area focused only on the detection of a few traditional vulnerability types and dismissed more complex occurrences or types of vulnerabilities. Additionally, these approaches do not scale to large code bases or do not support major language features.

In this thesis, we present novel techniques designed for the efficient and precise static analysis of PHP code in order to automatically detect traditional and complex security vulnerabilities. A comprehensive configuration and simulation of over 1 200 PHP built-in features allows us to precisely model the highly dynamic PHP language. By creating block and function summaries, we are able to efficiently perform a backwards-directed taint analysis for 36 different types of vulnerabilities. More specifically, our string analysis is the first to evaluate the interaction between different types of security mechanisms, encodings, sources, sinks, markup contexts, and PHP settings. Furthermore, we are the first to detect *second-order* vulnerabilities and related *multi-step* exploits. Based on our novel forwards-directed object analysis, we are the first to automatically generate attack sequences used against *PHP object injection* vulnerabilities.

We implemented a prototype based on our approach. Our evaluation shows that it is capable of finding severe and complex vulnerabilities in modern real-world applications, previously missed by other approaches: in total, we detected 321 previously unknown vulnerabilities in 23 popular PHP applications, for example in *Joomla*, *phpBB*, and *os-Commerce*. Finally, we used our prototype to study prevalent practices of developers and attackers. We first studied how developers utilize security mechanisms in practice regarding different markup contexts, and which common pitfalls exist. Then, we analyzed features and backdoors in popular PHP shells used by attackers.