A Method and Case Study for Using Malware Analysis to Improve Security Requirements

Nancy R. Mead, Carnegie Mellon University, Software Engineering Institute, Pittsburgh, PA, USA

Jose Andre Morales, Carnegie Mellon University, Software Engineering Institute, Pittsburgh, PA, USA

Gregory Paul Alice, Carnegie Mellon University, Seattle, WA, USA

ABSTRACT

In this paper, the authors propose to enhance current software development lifecycle models by implementing a process for including use cases that are based on previous cyberattacks and their associated malware. Following the proposed process, the authors believe that developers can create future systems that are more secure, from inception, by including use cases that address previous attacks. In support of this, the authors present a case study of a malware sample that is used to generate new requirements for a mobile application.

Keywords: Cyberattacks, Malware, Requirements, SDLC, Software Security

1. INTRODUCTION

Hundreds of vulnerabilities are publicly disclosed each month (NIST, 2014). There are many definitions of what is meant by a vulnerability. The definition that is a good fit for our research is provided by Information Technology Security Evaluation Criteria (ITSEC) “The existence of a weakness, design, or implementation error that can lead to an unexpected, undesirable event compromising the security of the computer system, network, application, or protocol involved.” Exploitable vulnerabilities typically emerge from one of two types of core flaws: code flaws and design flaws. Both types of flaws have facilitated several cyberattacks, a subset of which have had global repercussions.

In the context of vulnerabilities, we view a code flaw as a vulnerability...
in the code base that requires a highly technical, crafted exploit to compromise a system; examples are buffer overflows and command injections. *Design flaws* are weaknesses in the system that may or may not require a direct code exploit in order to compromise the system. Examples include failure to validate certificates, non-authenticated access, automatic granting of root privileges to non-root accounts, lack of encryption, and weak single-factor authentication. A *malware exploit* is an attack on a system that takes advantage of a particular vulnerability. *Use cases* (Jacobson, 1992) describe a scenario conducted by a legitimate user of the system. Use cases have corresponding requirements, including security requirements. A *mis-use case* (Alexander, 2003) describes use by an attacker and highlights a security risk to be mitigated. Misuse cases describe a sequence of actions that can be performed by any person or entity in order to harm the system. Exploitation scenarios are often documented more formally as misuse cases. In terms of documentation, misuse cases can be documented in diagrams alongside use cases and/or in a text format, similar to that of a use case.

Several approaches for incorporating security into the software development lifecycle (SDLC) have been documented. Most of these enhancements have focused on defining enforceable security policies in the requirements gathering phase and defining secure coding practices in the design phase. Although these practices are helpful, cyberattacks based on core flaws have persisted.

Major corporations such as Microsoft, Adobe, Oracle, and Google have made their security lifecycle practices public (Lipner & Howard, 2005; Adobe, 2014; Oracle, 2014; Google, 2012). Collaborative efforts such as the Software Assurance Forum for Excellence in Code (SAFECode) (Simpson, 2008) have also documented recommended practices. These practices have become de facto standards for incorporating security into the SDLC. These security approaches are limited by their reliance on security policies such as access control, read/write permissions, and memory protection, and their reliance on standard secure code writing practices such as bounded memory allocations and buffer overflow avoidance. These processes are helpful in developing secure software products, but—given the number of successful exploits that occur—they fall short. For example, techniques such as design reviews, risk analysis, and threat modeling typically do not incorporate lessons learned from the vast landscape of known successful cyberattacks and their associated malware.

In this paper, we propose that current SDLC models can be further enhanced by including misuse cases derived from malware analysis. Our focus is on the vulnerabilities resulting from design flaws. We also propose an open research question: Are specific types of systems prone to specific classes of malware exploits? If this is the case, developers can create future systems that are more secure, from inception, by including use cases that address previous attacks.
Threat Representation Methods for Composite Service Process Models
www.igi-global.com/article/threat-representation-methods-composite-service/77914?camid=4v1a

SOAda: A New Architecture to Enrich SOA with a Decisional Aspect
www.igi-global.com/article/soada/114604?camid=4v1a

Developing Secure, Unified, Multi-Device, and Multi-Domain Platforms: A Case Study from the Webinos Project
www.igi-global.com/chapter/developing-secure-unified-multi-device-and-multi-domain-platforms/188223?camid=4v1a