A Survey on Vulnerabilities Related to Security Tactics

Anjumol T Many[1] Ushus Maria Joseph[2]
[1,2]Department of Computer Science & Engineering
[1,2]Mar Baselios Christian College Idukki, Peermade-685531

Abstract— Architectural security tactics are used for resisting, detecting and to recover from many kinds of attacks. Software architects often adopt some security mechanisms for this purpose. Consequently, flaws in the implementation of security tactics or their deterioration during software evolution and maintenance can introduce severe vulnerabilities that could be exploited by attackers. This shows that, a programmer does not have an idea about the types and also the impacts of vulnerabilities. In this paper, we conduct a deep study about the types and effects of vulnerabilities.

Key words: SAM, RBAC, UML

I. INTRODUCTION

Security tactics are a useful tool that can help people immediately start reasoning about secure software design. A security tactic is a design concept that addresses a security problem at the architectural design level. In particular, incorrect implementation of security tactics or the deterioration of security tactics during coding and maintenance activities can result in vulnerabilities in the security architecture of the system, we refer to these vulnerabilities as tactical vulnerabilities. The correct implementation of "Manage User Sessions" tactic in a web application would allow the system to keep track of users that are currently authenticated. Once the user authenticates him/herself with this forged session identifier, the attacker would be able to hijack or steal his/her authenticated session. Although architects have used the "Manage User Sessions" tactic in the architecture design of the web application, the developers have failed to implement it correctly. So, we conduct a deep study about the vulnerabilities related to security tactics.

Software architectural specifications using SAM is a concept, in which SAM is a framework based on two complementary formalisms—Petri nets and temporal logic. Formally analyse SAM software architecture specifications using well-known techniques—symbolic model checking with tool Symbolic Model Verifier. SAM supports just formal analysis in many techniques. Security Code Review done with some process, is simple: Make sure you know what you're doing, prioritize and Review the code. UML metamodel level allows tool developers to build support for creating patterns and for checking conformance to pattern specifications. The pattern specification technique to support practical and rigorous pattern-based model transformation techniques. In dynamic rule space, software architectures should be viewed and analysed as multi-layered overlapping DRSpaces, Able to identify a large number of structural and evolutionary problems. Some formal models for secure software architectures contains some techniques to construct secure architecture designs by assembling already verified security pattern models. This approach produces reusable results, and is able to uncover relevant architectural security flaws. An empirical analysis of the relationship between peer code review and security vulnerabilities shows peer code review to prevent security vulnerabilities. It can identify attempts to insert malicious code into the codebase.

Architecture design methods, such as ADD, describe an idealization of how architects perform their duties. Frameworks might generate new requirements .This shows a principled way to use frameworks in the architecture design. Two novel hotspot patterns, Unstable Interface and Implicit Cross-module Dependency based on Baldwin and Clark's design rule theory, in the automatic detection of architectural smells. These patterns identify the most error-prone and change-prone files, they also pinpoint specific architecture Problems. DRSpace-based analysis approach to identify architectural design flaws. In an architecture centric approach, focuses entirely on the design decisions, doing this analysis does not require access to the source code, just a knowledge of the structural and historical relations between files. A machine learning approach for discovering and visualizing architectural tactics in code. Mapping these code segments to Tactic Traceability Patterns, and monitoring sensitive areas of the code for modification events.

II. TECHNIQUES TO IDENTIFY SOME VULNERABILITIES

Archie detects architectural tactics such as heartbeat, resource pooling, and role-based access control (RBAC) in the source code of a project; constructs traceability links between the tactics. , Archie's primary contribution is in the area of architectural preservation through detecting and tracing architectural concerns. The method of systematic review is applied with the purpose of identifying, extracting and analyzing the main proposals for security ontologies. The main identified proposals are compared using a formal framework.

Next is, various recent developments in the area of design level vulnerabilities identification are reviewed. Future research directions are also proposed which can be used by the researchers for further extension of their research in the area of secure software design. A semantic template for each type of vulnerability is created from information in the Common Weakness Enumeration dictionary. Next, known vulnerabilities and related concepts in the repository are tagged with concepts from the template.

Attack-based approaches are based on knowing your enemy and assessing the possibility of similar attacks. Although the taxonomy proposed here is incomplete and imperfect, it provides an important first step. It focuses on collecting common errors and explaining them in a way that makes sense to programmers. This new taxonomy is made up of two distinct kinds of sets, which we're stealing from biology: a phylum and a kingdom.

Three approaches to software security including penetrate & patch, secure operational environment, and secure software engineering were mentioned. It is clear that early detection of security problems and countering them in the software development cycle will save time and energy

spent on removing flaws after software release. A variation of one approach to misuse detection, state transition analysis, by using pattern matching to detect system attacks. Knowledge about attacks is represented as specialized graphs. These graphs are an adaptation of Colored Petri Nets with guards representing signature context and vertices representing system states.

A methodology to develop countermeasures against code injection attacks, and validates the methodology by working out a specific countermeasure. This methodology is based on modeling the execution environment of a program. Such a model is then used to build countermeasures.

Design and implementation methods exist for reducing the security vulnerabilities of software. In addition, a variety of source code security checkers are available that use automatic scanning to indicate potential security pitfalls in a software application. Nevertheless, the imperative nature of the majority of programming languages used now may lie at the heart of the security vulnerabilities.

A new technique for finding potential buffer overrun vulnerabilities in security-critical C code. The key to success is to use static analysis: Then formulate detection of buffer overruns as an integer range analysis problem. One major advantage of static analysis is that security bugs can be eliminated before code is deployed.

REFERENCES

[1] Formally analysing software architectural specifications using SAM, Tianjun Shi, Junhua Ding, Yi Deng,2002
[2] A Process for Performing Security Code Reviews, Richard Ford, rford@se.fit.edu Michael A. Howard, mikehow@microsoft.com
[3] A UML-Based Pattern Specification Technique Robert B. France, Member, IEEE Computer Society, Dae-Kyoo Kim, Student Member, IEEE, Sudipto Ghosh, Member, IEEE Computer Society, and Eunjee Song, Student Member, IEEE.
[4] Design Rule Spaces: A New Form of Architecture Insight Lu Xiao, Yuanfang Cai Drexel University Philadelphia, PA, USA {lx52, yfcai}@cs.drexel.edu Rick Kazman University of Hawaii & SEI/CMU Honolulu, HI, USA kazman@hawaii.edu.
[5] Reusable formal models for secure software architecture Thomas Heyman, Riccardo Scandariato and Wouter Joosen IBBT-DistriNet, KU Leuven first.last@cs.kuleuven.be
[6] Peer Code Review to Prevent Security Vulnerabilities: An Empirical Evaluation Amiangshu Bosu and Jeffrey C. Carver University of Alabama, Tuscaloosa, AL, USA asbosu@ua.edu, carver@cs.ua.edu
[7] A Principled Way to Use Frameworks in Architecture Design Humberto Cervantes, Autonomous Metropolitan University, Mexico City Perla Velasco-Elizondo, Autonomous University of Zacatecas Rick Kazman, University of Hawaii
[8] Hotspot Patterns: The Formal Definition and Automatic Detection of Architecture Smells Ran Mo∗, Yuanfang Cai∗, Rick Kazman†, Lu Xiao∗ ∗ Drexel University †University of Hawaii & SEI/CMU Philadelphia, PA, USA Honolulu, HI, USA {rm859,yc349,lx52}@drexel.edu kazman@hawaii.edu
[9] Towards an Architecture-centric Approach to Security Analysis Qiong Feng∗, Rick Kazman†, Yuanfang Cai∗, Ran Mo∗, Lu Xiao∗ ∗ Drexel University †University of Hawaii & SEI/CMU Philadelphia, PA, USA Honolulu, HI, USA {qf28, yc349, rm859, lx52}@drexel.edu kazman@hawaii.edu
[10] Detecting, Tracing, and Monitoring Architectural Tactics in Code Mehdi Mirakhorli∗ Member, IEEE Jane Cleland-Huang† Member, IEEE ∗Department of Software Engineering Rochester Institute of Technology, Rochester, NY mehdi@se.rit.edu †School of Computing DePaul University, Chicago, IL 60604 jhuang@cs.depaul.edu

