Volume 5, Issue 12, December 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Advances in Measuring and Preventing Software Security Weaknesses

Adrian Bosire, Michael Kimwele
Institute of Computer Science and Information Technology,
Jomo Kenyatta University of Agriculture and Technology, Kenya

Abstract –There are several software security standards in place but they are not comprehensive, as such, one needs an array of tools so as to achieve a single security functionality. The survey guides us through the classical implementation to the modern as well as proposed future endeavors so as to achieve an integrated and robust platform for developing a reliable model. Some of the techniques and models have been tested and have proven to be effective and efficient in achieving the main goal of software security. The conclusion gives us the direction to take with regard to attaining ideal software weakness prevention.

Key Words – Software Security, Software metrics and measures, software Security vulnerability, Software Weakness Prevention

I. INTRODUCTION

This research seeks to distinguish the existent software security metrics and explore the particular techniques applied to curb security breaches. The number of security breaches has increased, the scale and cost has nearly doubled. 11% of respondents in a study carried out by Price Waterhouse Cooper in 2015 said that they changed the nature of their business as a result of their worst security breach. Nevertheless, we can assert our control over determinate entities hence the need for metrics. In software security we need such parameters so as to ascertain the nature of threats and vulnerabilities both quantitatively and qualitatively. This will effectively help us in identifying the problems, quantifying and ultimately prescribing an ideal solution [20]. Therefore, a security metric as a quantitative measure indicating to which extend the considered entity possesses the attribute of being secure [17].

Past ventures into the software security problem focused mostly on post software production but in recent times the focus has shifted to the software development life cycle. The attackers basically exploit implementation bugs and architectural flaws existent in the system. Both bugs and flaws lead to vulnerabilities that can be exploited [3].

Professionals in the Information Technology agree that insecure software is one of the root causes of security breaches and often leads to a number of business problems [7]. Security breaches of all kinds are growing in complexity, sophistication, and impact. The root cause of software security can arguably be the fact that software developers and security managers have different goals. Simply put, the developers are under pressure to deliver solid software containing the functionality requested within a specific timeframe whereas the security managers assess risks and find solutions for the critical issues as soon as possible. Ultimately resulting to implementation of a layer of defense via security technologies which should not be the case [21]. This is supported by a WhiteHat's Website Security Statistics Report 2015 which states that we need more secure software, not more security software [22]. Nevertheless, this is easier said than actually done despite an abundance in standards, regulations, frameworks and models.

II. SOFTWARE METRICS VS MEASURES

There is a difference between the two terms Measurements and Metrics. A measurement is an indication of the size, quantity, amount or dimension of a particular attribute of a product or process. A Metric is a measurement of the degree that any attribute belongs to a system, product or process. Simply put, software measurement gives rise to software metrics [1], [18]. Metric Classification Software metrics can be divided into two categories; product metrics and process metrics. Product metrics are used to assess the state of the product, tracking risks and discovering potential problem areas. The team's ability to control quality is assessed. Process metrics focus on improving the long term process of the team or organization. Measures can also be categorized effectively based on Steven's Taxonomy into ordinal scale, nominal scale, interval scale and ratio scale. Apparently, consistent measurement units will yield reliable metrics.

This is echoed by National Institute of Standards and Technology - NIST in an article on metrics and measures [14]. Further on the classifications of software measures with regard to the general aspects is quite befitting. These classes are:

1. Size: number of elements, lines of code or classes
2. Complexity: computational, algorithmic, logical or functional
3. Quality: composite, emergent, or high-level aspects: reliability, efficiency, resilience, usability

III. TRINITY OF SOFTWARE SECURITY ISSUES

According to the Open Web Application Security Project - OWASP insecure software undermines most of our critical infrastructure [16]. Moreover, the difficulty of achieving software security increases exponentially with the increase in

complexity of our digital infrastructure. This brings about the concept of the triad of trouble. This trinity of trouble is classified as connectivity, complexity and extensibility [3].

Connectivity refers to the ubiquitous nature of the internet and so is the software which at the very least is used on a given network. Complexity refers to the distributed nature of the ever-growing network as well as the increasing lines of codes in the software's we use. Lastly, Extensibility refers to the unforeseen ways in which software evolves for adaptability purposes.

IV. SOFTWARE SECURITY VULNERABILITIES

The Joint Technical Committee ISO/IEC defines security as "the degree to which a product or system protects information and data so that persons or other products or systems have degree of data access appropriate to their types of authorization" [5]. A vulnerability as a weakness in an asset, process, or piece of software. Therefore a security vulnerability leaves a company or individual user exposed to the risk of exploitation with possible negative outcome [8].

The following is a list of some of the vulnerabilities presented by OWASP and are particular to web applications [16].

1. Injection flaws: Occurs when the attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.
2. Broken Authentication and Session Management: Related Application functions not implemented correctly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities.
3. Cross-Site Scripting (XSS): Occur whenever an application takes untrusted data and sends it to a web browser without proper validation or escaping. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.
4. Insecure Direct Object References: A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data.
5. Security Misconfiguration: Secure settings should be defined, implemented, and maintained, as defaults are often insecure. Additionally, software should be kept up to date.
6. Sensitive Data Exposure: Sensitive data deserves extra protection such as encryption at rest or in transit, as well as special precautions when exchanged with the browser.
7. Missing Function Level Access Control: Applications need to verify access control checks on the server when each function is accessed.
8. Cross-Site Request Forgery (CSRF): A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other automatically included authentication information, to a vulnerable web application. This allows the attacker to force the victim's browser to generate requests the vulnerable application thinks are legitimate requests from the victim.
9. Using Components with Known Vulnerabilities: Components, such as libraries, frameworks, and other software modules, almost always run with full privileges. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications using components with known vulnerabilities may undermine application defenses and enable a range of possible attacks and impacts.
10. Unvalidated Redirects and Forwards: Web applications frequently redirect and forward users to other pages and websites, and use untrusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages.

MITRE Corporation maintains a general list of types of software vulnerabilities known as the Common Weaknesses Enumeration – CWE. This is not to be confused with the Common Vulnerabilities and Exposures – CVE list which aims to give a common name for each specific instance of vulnerabilities and security exposures hence standardize. CWE seeks a common name and definition for each kind of weakness hence generalize. CWE thus yields the following classification [9]:

1. SQL injection (CWE-89)
2. Buffer overflow (CWE-120)
3. Missing encryption of sensitive data (CWE-311)
4. Cross-site request forgery (CWE-352)
5. Use of a broken or weak crypto algorithm (CWE-327)
6. Integer overflow (CWE-190)

There is also another taxonomy that has gained recognition from organizations such as MITRE Corporation, NIST as well as National Vulnerability Database – NVD and has proven to be acceptable. Vulnerabilities can be classified using schemes based on cause [15]. The eight classes are as follows:

1. Input Validation Error (IVE) (Boundary condition error (BCE), Buffer overflow (BOF)): Such types of vulnerabilities include failure to verify the incorrect input and read/write involving an invalid memory address.
2. Access Validation Error (AVE): These vulnerabilities cause failure in enforcing the correct privilege for a user.
3. Exceptional Condition Error Handling (ECHE):
4. These vulnerabilities arise due to failures in responding to unexpected data or conditions.
5. Environmental Error (EE): These vulnerabilities are triggered by specific conditions of the computational environment.

6. Configuration Error (CE): These vulnerabilities result from improper system settings.
7. Race Condition Error (RC): These are caused by the improper serialization of the sequences of processes.
8. Design Error (DE): These are caused by improper design of the software structure.
9. Others: Includes vulnerabilities that do not belong to the types listed above, sometimes referred to as nonstandard.

Common Vulnerability Scoring System - CVSS is the universal open and standardized method for rating vulnerabilities and determining the urgency of response. CVSS is currently maintained by the Forum of Incident Response and Security Teams – FIRST and is arguably the preferred classification model due its extensive nature as a benchmark for both metric and measure [2]. CVSS rates each vulnerability in three scopes on a scale 0 – 10:

1. Base metrics: Objective characteristics of the vulnerability. These include the Exploitability metrics comprising of Attack Vector (AV), Attack Complexity (AC), Privileges Required (PR) and User Interaction (UI), Scope (S) and Impact Metrics made up of Confidentiality Impact (C), Integrity Impact (I) and Availability Impact (A)
2. Temporal metrics: The risk change over time. These metrics include Exploit Code Maturity (E), Remediation Level (RL) and Report Confidence (RC)
3. Environmental metrics: The vulnerability's applicability to your organization. The metrics are Security Requirements (CR, IR, and AR) and Modified Base Metrics

STRIDE is a mnemonic for a threat model designed by Microsoft. It can also be used to classify vulnerabilities and does not attempt to rank or prioritize them [12]:

1. Spoofing: Impersonating something or someone else
2. Tampering: Modifying data or code
3. Repudiation: Claiming to have not performed an action
4. Information Disclosure: Exposing information to someone not authorized to see it
5. Denial of Service: Deny or degrade service to users
6. Elevation of Privilege: Gain capabilities without proper authorization

DREAD is another system from Microsoft which unlike STRIDE that only classifies, it also ranks them by allocation of sub-scores for each vulnerability [10]. It worth to note that due to it subjective nature in ranking it has been outdated:

1. Damage potential
2. Reproducibility (or Reliability)
3. Exploitability
4. Affected users
5. Discoverability

This goes to show that there are various criteria that may be considered whilst classifying the software security vulnerabilities. Regardless of the same they most more than often revolve around the following two major classes.

1. Coding/Programming Errors
2. Software Design Flaws

Theycan also be considered as implementation bugs and architectural flaws respectively [3].

V. SOFTWARE WEAKNESSES PREVENTION

A multi-layered approach is necessary in order to deploy effective controls against software security weaknesses. There are basically three crucial factors to consider when resolving both current and future software issues [3], [21]:

1. Automated Source Code Review. The number of bugs is relatively proportional to the number of flaws hence it is wise to use an automated tool to review the errors.
2. Architectural Risk Analysis. Bugs are easier to find compared to design flaws hence its importance.
3. Penetration Testing. This is an endeavor to verify and establish the level of threats and vulnerabilities a software system is actually exposed to.

A. Automated Source Code Review

Some of the source code review strategies which have proven to be worthwhile and incorporate an agile approach [6].

1. Review fewer than 200–400 lines of code at a time.
2. Aim for an inspection rate of fewer than 300–500 LOC per hour.
3. Take enough time for a proper, slow review, but not more than 60–90 minutes. Never review code for more than 90 minutes at a stretch.
4. Ensure that authors annotate source code before the review begins.
5. Establish quantifiable goals for code review, and capture metrics so you can improve your processes.
6. Use checklists, because they substantially improve results for both authors and reviewers. Checklists are especially important for reviewers because, if the author forgets a task, the reviewer is likely to miss it also.
7. Verify that the defects are actually fixed.
8. Foster a good code review culture in which finding defects is viewed positively.

9. Beware of the Big Brother effect. Metrics should never be used to single out developers, particularly in front of their peers.
10. Review at least part of the code, even if you can't do all of it.
11. Adopt lightweight, tool-assisted code reviews.

OWASP actually lists some of the techniques that can be used with consideration to the guidelines aforementioned [16]. These techniques include:

1. Data Flow Analysis
2. Control Flow Graph (CFG)
3. Taint Analysis
4. Lexical Analysis

Finally, one can also consider the state of the art review made on the current automated software review tools [23]. These are software's that that automate the whole process.

B. Architectural Risk Analysis

We are also presented with an overview of the whole process of risk analysis with a focus on software threats and vulnerabilities. This process involves software risk management that is further subcategorized into asset identification, the scrutiny of the particular threats or vulnerabilities and finally the steps taken to mitigate the same [4]. Ultimately the discussion gives us a dependable framework that has been tested and proven to work effectively. Nonetheless, these approaches in achieving the same can either adopt a classical approach of scenario-based analysis or modern approach of metric-based analysis [13].

C. Penetration testing

Regardless of the type of software that's being used there's a significant chance that an existing weakness could compromise customer and internal information [3]. Nevertheless, the underlying procedure on how to implement the same should adhere to the methodology [19]. Penetration Test Guidance Special Interest Group and PCI Security Standards Council compiled the information supplement in which the methodology is comprised of three steps:

1. Pre-Engagement: This is the precursor step which recommends that all parties involved (the organization, the tester, and where applicable, the assessor) be informed of the types of testing to be performed, how testing will be performed, and what the testing will target.
2. Engagement: Penetration Testing: Each environment has unique aspects/technology that requires the tester select the most appropriate approach and the tools necessary to perform the penetration test.
3. Post-Engagement: On completion, the involved parties should take the required measures to eliminate the weaknesses discovered and clean the environment.

VI. CONCLUSION

Metrics will easily be invalidated by poor measurements, hence the need to carefully select the appropriate scales of measures. This will form a basis from which one can actualize the framework or model from a concept. In order to achieve optimal software security we need to take due consideration to the development life cycle [11] and also factor in some issues such as the environment, users and technology.

REFERENCES

[1] Cliff Moser, *Architecture 3.0: The Disruptive Design Practice Handbook*, Routledge publishers, ISBN 1317916859, 9781317916857

[2] Forum of Incident Response and Security Teams – FIRST and CVSS Special Interest Groups – SIG. *Common Vulnerability Scoring System v3.0: Specification Document.*Available: http://www.first.org/cvss/specification-document

[3] Gary McGraw,*Software Security Problems and How to Fix Them: Bug Parades, Zombies and the BSIMM*, Available: https://www.cigital.com/resources/videos/software-security-problems/

[4] Gunnar Peterson, Paco Hope, and Steven Lavenhar, *Architectural Risk Analysis*, Cigital Inc., Available: https://buildsecurityin.us-cert.gov/articles/best-practices/architectural-risk-analysis/architectural-risk-analysis

[5] ISO.org, ISO/IEC 25010:2011 *Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE)-- System and software quality models*, 2011.

[6] Jason Cohen. *11 proven practices for more effective, efficient peer code review*, IBM Developer Works. Available: http://www.ibm.com/developerworks/rational/library/11-proven-practices-for-peer-review/

[7] Kevin Beaver,*Software Security Assurance: Managing for Software Security*, Realtimepublishers 2014, website - http://www.realtimepublishers.com/

[8] Michael Boelen,*Linux Vulnerabilities Explained: From detection to Treatment* Posted on November 3, 2015 Available: http://linux-audit.com/category/vulnerabilities-2/

[9] MITRE Corp, *Common Weakness Enumeration*, Available: http://www.cwe.mitre.org/

[10] Meier, J.D., Alex Mackman, Michael Dunner, Srinath Vasireddy, Ray Escamilla and Anandha Murukan,*Improving Web Application Security: Threats and Countermeasures*, Chapter 3: Threat Model. Available: https://msdn.microsoft.com/en-us/library/ff648644

[11] Michael Howard and Steve Lipner. *The Security Development Lifecycle: SDL: A Process for Developing Demonstrably More Secure Software* (2006)

[12] Microsoft Corporation,*the STRIDE Threat Model*. Available: https://msdn.microsoft.com/en-us/library/ee823878

[13] Mohamed Almorsy, John Grundy, and Amani S. Ibrahim,*Automated Software Architecture Security Risk Analysis using Formalized Signatures*, 978-1-4673-3076-3/13, IEEE, ICSE 2013, San Francisco, CA, USA

[14] National Institute of Standards and Technology NIST. *Metrics and Measures*. Available: https://samate.nist.gov/index.php/Metrics_and_Measures.html

[15] Omar H. Alhazmi, Sung-Whan Woo, Yashwant K. Malaiya,*Security Vulnerability Categories in Major Software Systems*, Proc. IASTED Int. Conference on Communication, Network and Information Security, Oct. 2006.

[16] Open Web Application Security Project - OWASP. *OWASP Top 10 – 2013: The Ten Most Critical Web Application Security Risks*. Available: https://www.owasp.org/

[17] Ouchani, S. and M. Debbabi,*Specification, Verification, and Quantification of Security in Model-based Systems.* Computing, pp.1–21. Available: http://link.springer.com/10.1007/s00607-015-0445-x

[18] Paul E. Black, Karen A. Scarfone, and Murugiah P. Souppaya,*Cyber Security Metrics and Measures*, Wiley Handbook of Science and Technology for Homeland Security, 2008.

[19] Penetration Test Guidance Special Interest Group and PCI Security Standards Council (2015),*Information Supplement: Penetration Testing Guidance*, PCI Data Security Standard (PCI DSS), March 2015.

[20] Price Waterhouse Cooper - PwC,*The UK 2015 information security breaches survey*. Department for Business, Innovation and Skills 1 Victoria Street London SW1H 0ET Available: www.gov.uk/bis

[21] Sumner Blount,*Security Breaches - Challenges and Solutions*. CA Technologies Security Management White Paper November 2012

[22] WhiteHat Security,*Website Security Statistics Report 2015*. Available: https://www.whitehatsec.com/categories/statistics-report/

[23] Yuriy Tymchuk, Andrea Mocci and Michele Lanza,*Code Review: Veni, ViDI, Vici*, PhD Dissertation, Università della Svizzera Italiana USI.

