Volume 6, Issue 6, June 2016 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Trends in Measuring and Preventing Software Security Weaknesses– A Survey

Kevin O. Gogo, Jane Kiruki
Department of Computer Science, Chuka University,
Kenya

Abstract- On our daily basis operations, there has been several usage of technology, these technologies have to use custom software, commercial-out-of-shelve software or embedded software; making software a very vital part of today's life and technology. In this regard, then software has to be secured from the several threats that constitute software vulnerability. Hence in this paper we examine the trends that have been put to measure and prevent the software security weaknesses. In examining these measures we briefly examined software security, then reviewed the trends that have / are being put in place in measuring and preventing software security and weaknesses. Our contribution in this research paper is a survey of current trends in measuring and preventing software security weaknesses.

Keywords- Software security weaknesses, software security measurements, software security weakness prevention, and software quality.

I. INTRODUCTION

On our everyday operations there is need to engage computers to perform our duties, in order to perform most of our communication and computing activities. These computers are driven by the software in order to perform our duties, making software a very important part of computers and its operations. It is through these software security measures that we can guarantee that our data in computers are safe, hence pointing at software security and quality. In this paper we are going to examine the trends in measuring and preventing software security weaknesses, which we will first review software security, software security measurements, software security weaknesses, and finally the trends that have / are being put in place in measuring and preventing software security and weaknesses.

II. SOFTWARE SECURITY

Software is a program that computers and computerized devices use to operate and control other devices; they give instructions on how to operate computers and computerized devices [14]. Reference [21] explains that software security is the ability to build software that will function in the presence of unintended and malicious attacks; it involves securing of software to minimize the number of vulnerabilities. Vulnerabilities are weaknesses in the requirements, design and implementation of software, which attackers exploit to compromise the system.

Further clarifications by [7], shows that software security is no exception; nearly every major business-critical application deployed today contains vulnerabilities, these vulnerabilities range from buffer overflow and cross-site scripting and so many other that are less well-known types of vulnerabilities. These problems can be exploited to cause considerable harm by external hackers or malicious insiders. Security teams know that these errors exist, but are mostly ill equipped to quantify the problem and solve them perfectly.

To further understand software security we have to introduce software security measurement and software security weaknesses: -

A. Software security measurement (metrics)

Measurement or metrics is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined rules [10].

Reference [11] elaborates that there are two broad types of measurement, namely direct and indirect measurement. *Direct measurement* of an attribute is measurement which does not depend on the measurement of any other attribute. *Indirect measurement* of an attribute is measurement which involves the measurement of one or more other attributes. It turns out that while some attributes can be measured directly; we normally get more sophisticated measurement if we measure indirectly as compared to directly. Some uses of Measurement include: - Assessment and Prediction

Vulnerability density is one of the major aspects of security measurement. Vulnerability density is a normalized measure, given by the number of vulnerabilities per unit of code size. Vulnerability density can be used to compare software systems within the same category [3]. To measure the code size we have two options. First, we can use the size of the installed system in bytes; the advantage of this measure is that information is readily available. However, this measure will vary from one installation to another. The second measure is the number of lines of source code, which is simple and correspondence to defect density metric in the software engineering domain.

Vulnerability density is the number of vulnerabilities in the unit size of a code. The vulnerability density is given by: $V_D = {}^{V}/_{S}$, where $_S$ is the size of the software and $_V$ is the number of vulnerabilities in the system. Vulnerability density allows us to compare the quality of programming in terms of how secure the code is. If the instruction execution rates and other system attributes are the same, a system with a higher defect or vulnerability density is likely to be compromised more often. Estimating the exact vulnerability density would require us to know the number of all the vulnerabilities of the system [16].

References [20] and [3], further clarifies that Vulnerability density can be used as a metric for estimating the number of residual vulnerabilities in a newly released software system given the size of the software. The vulnerability density of a newly released software system should be comparable to its comparable predecessor assuming that it is designed using the same process. Hence, vulnerability density can be used by users to assess the risk when considering the purchase of a new software system. Developers can use the vulnerability density metric for maintenance planning, and planning for their testing process, and deciding when to stop testing.

B. Software security weakness

Software security weaknesses are vulnerabilities in the software that can be exploited by a determined intruder to gain access to the system, or a defect which enables an attacker to bypass security measures in order to alter the normal behavior of the system [6].

Some of the known software weaknesses according to [17] are: -

1) *Buffer overflow*: this mostly occurs with fixed length buffers when some data is going to be written beyond the boundaries of the current defined capacity. This could lead to malfunctioning of the system since the new data can corrupt the data of other buffers or processes, and inject malicious code, and then the execution sequence of the program could be altered and the injected code can take control of the system.
2) *XSS or cross site scripting*: consists in the injection of code in the pages accessed by other users. If exploited an attacker can bypass access controls, perform phishing or identity theft.
3) *SQL injection:* it consists in the injection of code with the intension of exploiting the content of a database. Usually happens because the inputs are not handled correctly and the attacker can get sensitive information from the database.

Further elaborations from [18] indicate that software security weaknesses could also be due to: -

1) *Overconfidence*: and use of non standard and non tested algorithms by the programmers
2) *Malice of the software developers*: hence they can inject say Trojan horse, bugs, viruses, worms, malware etc onto the software.
3) *Complacency*: experiences software developer assumes certain aspects of security i.e. testing hoping that they have been doing it right always
4) *Rush to finish*: this can be due to pressure from management or set deadlines in software development
5) *Memory lapses*: can cause someone to forget testing or adding some line of code in a program resulting in vulnerabilities

Reference [24] stresses that some of the security measures have been build to overcome the software security weaknesses include: -

1) *Secure channel and envelops*: a mechanism that provides communication in a secure way, message is transmitted through secure channels and encapsulated envelops
2) *Authentication*: this software security measure ensures data that is accessed by users are authenticated
3) *Protection and authorization*: process that protects data from unauthorized access and only enable users perform what they are only authorized to do
4) *Auditing and intrusion detection*: this method allows posteriori access of data and information allowing the detection of unauthorized access

Most software security vulnerabilities fall into one of a small set of the following categories:
Buffer overflows, un-validated inputs, access-control problems, weaknesses in authentication, authorization, or cryptographic practices. A list of some common types of software weaknesses includes [8]: -

1) Buffer overflows, format strings, etc
2) Structure and validity problems
3) Common special element manipulations
4) Channel and path errors
5) Handler errors
6) User interface errors
7) Pathname traversal and equivalence errors
8) Authentication errors
9) Insufficient verification of data
10) Code evaluation and injection
11) Randomness and predictability

III. TRENDS IN MEASURING SOFTWARE SECURITY WEAKNESSES

Software security being an area still under research, There are a number of trends that have been put across in measuring the software security weaknesses, some of the trends in measuring the software security weaknesses include: -

A. Porting software to different browsers

Porting browsers to mobile platforms may lead to new vulnerabilities whose solutions require careful balancing between usability and security, and might not always be equivalent to those in desktop browsers. Comparison between mobile and desktop browsers was compared in terms of the software security, a number of software vulnerabilities were identified on the mobile browsers which were not found on the desktop browsers; Two new classes of vulnerabilities specific to mobile browsers and demonstrate their risk by launching real-world attacks including display ballooning, login CSRF and click jacking. It was conclude that usability considerations are crucial while designing mobile solutions and display security in mobile browsers is not comparable to that in desktop browsers [4].

B. Build the software then Break: Penetration Testing

Reference [19] elaborates that developers create applications with only a minimum of attention paid to security, and the applications are deployed. The operations team then attempts to compensate for the problematic software with perimeter security. At some point the operations team may bring in penetration testers to find the problems before hackers or malicious insiders do. The penetration testers generally have a fixed schedule for performing their work, and their goal is to find a small number of serious problems on the software. The problem with most penetration tests is that they often find the same types of problems over and over again, though it is an advance in measuring software security weakness.

C. Measuring and ranking software weaknesses attacks based on vulnerability analysis

Reference [22] and [25] explains that it is very difficult to design network software where nobody could exploit the vulnerability. As the number of software vulnerabilities increases, the research on software vulnerabilities becomes a focusing point in information security. However, multiple attacks may target one software product at the same time, and it is necessary to rank and prioritize those attacks in order to establish a better defence. One way of measuring software security weakness is measurement to compare and categorize vulnerabilities, and a set of security metrics to rank attacks based on vulnerability analysis.

The vulnerability information is retrieved from a vulnerability management ontology integrating commonly used standards like CVE (common vulnerable and exposure), CWE (common weaknesses and exposure), CVSS (common vulnerability scoring system) etc. This approach can be used in many areas of vulnerability management to secure information systems, such as vulnerability classification, mitigation and patching, threat detection and attack prevention.

D. Search-Based Refactoring

Security metric have been proposed to assess the security of software applications based on the principles of either reduce attack surface and grant least privilege. These metrics alone cannot help inform the developer in choosing designs that provide better security; they cannot on their own show exactly how to make an application more secure. Even though they could, the onerous task of updating the software to improve its security is left to the developer. They present an approach to automated improvement of software security based on search-based refactoring, which uses the search-based refactoring platform, code implementer to re-factor the code in a fully-automated fashion, which could then reduce vulnerabilities [12].

E. Vulnerability-centric requirements engineering framework

According to [9] security breaches occur because of exploitation of vulnerabilities within the system. They propose a methodological framework for security requirements elicitation and analysis centred on vulnerabilities. The framework offers modelling and analysis facilities to assist system designers in analysing vulnerabilities and their effects on the system; identifying potential attackers and analysing their behavior for compromising the system; and identifying and analysing the countermeasures to protect the system. The framework proposes a qualitative goal model evaluation analysis for assessing the risks of vulnerabilities exploitation and analysing the impact of countermeasures on such risks.

IV. TRENDS IN PREVENTING SOFTWARE SECURITY WEAKNESSES

A number of trends to prevent software security weaknesses have been devised and brought forward. In this chapter we will review some of them. The reviewed advances include: -

A. Security common criteria bench marking

The main problem faced by system administrators nowadays as suggested by [24] is the protection of data against unauthorized access or corruption due to malicious actions. They represent key concepts on security, also providing the basis for understanding existing challenges on developing and deploying secure software systems. They point out that there are several security common criteria that have been put in place to secure software, such as orange, common criteria for IT security evaluation to define generic rules that allow developers to specify security attributes that meet their requirements. These benchmarks are created by centre for internet security which is a non-profit organization, and focus on practical aspects of configuration of systems and the value each system should have to ensure overall security of real installations.

B. Decomposing the elements of complex systems

Reference [23] advices that the way a software system is decomposed into a set of decomposition elements greatly affects the amount of effort spent on the development and maintenance of that system. It is therefore advised that software architects should decompose the software system such that the resulting decomposition elements can evolve as independent as possible. If the software is in a state of the decomposition, find the decomposition weaknesses and improve the decomposition if necessary. Big and complex software usually tend to have some defects, hence in ensuring the safety of complex systems, the decomposition of large system to decomposed systems is required. Some of the ways of system decomposition, in files classes and methods is by: -

1. Making them belong to the same subsystem
2. Developed by the same group of developers

C. Measure Software Security as Part of Software Quality

Software security needs to be treated just as another aspect of software quality. Contrary to the traditional quality assurance aimed at verifying a set of features against a specification, software security requires much more than well-implemented security features. Most cases traditional quality issues do not guarantee good results with respect to security issues. Hence, in respect to this, you have to focus specifically on security in order to improve it. Good security is not necessarily a by-product of good quality [19].

D. Source code analysers

In order to identify certain security defects, a source code analyser could be used. A source code analyser searches the code for patterns that represent potentially known vulnerabilities and presents the code that matches these patterns to a human auditor for review. The three key attributes for good source code analysis are accuracy, precision and robustness [21]. A source code analyser should accurately identify vulnerabilities that are of concern to the type of program being analysed. For example, in web applications typically at risk such as SQL injection, cross-site scripting, and access control problems, among others should be searched. Further, the analysis results should indicate the likely importance of each result. The source code analyser must also be precise, pointing to a manageable number of issues without generating a large number of false positives.

E. Diversity-Based Approaches to Software Systems Security

Modern information systems run significant similar software. This is referred to IT monoculture. As a consequence, software systems share common vulnerabilities, which enable the spread of malware. The principle of diversity can help in mitigating the negative effects of IT monoculture on security. One important category of the diversity-based software approaches for security purposes focuses on enabling efficient and effective dynamic monitoring of software system behavior in operation [13].

F. Developing and reusing of software security patterns

Security as one essential quality requirement has to be addressed during the software development process. In order to obtain sound architectures and correct requirements, knowledge which is gained in the solution space, for example from security patterns, should be reflected in the requirements engineering of software development [2]. They propose an iterative method that takes into account the concurrent development of requirements and architecture descriptions systematically, and reuses security patterns for refining and restructuring the requirement models, by applying problem-oriented security patterns. These problem-oriented security patterns adapt existing security patterns in a way that they can be used in the problem-oriented requirements engineering, which bridges the gap between security problems and security architectural solutions.

G. Neural network based security tool for analysing software design for security flaws

Software can be developed more securely, in order to achieve this, it has been suggested that security needs to be integrated into every phase of software development lifecycle (SDLC). In line with this view, security tools are now used during SDLC to integrate security into software applications [1]. They propose a neural network based security tool for analysing software design for security flaws. The trained neural network was able to match possible attack patterns to design scenarios presented to it, and with the information on the attack pattern identified, developers can make informed decision in mitigating risks in their designs

H. Security alert through accessing the end user usability

Commercial security product can suffer from a usability perspective, thus lacking the necessary attention to design in relation to their alert interfaces [15]. They argue that based upon a related set of usability criteria, usability alerts should be included in internet security packages. Their findings reveal that the interface design combined with the user's relative lack of security knowledge are two major challenges that influence their decision making process. The analysis of the alert designs showed that it would be desirable to consider the user's previous decisions on similar alerts, and modify alerts according to the user's previous behavior in securing software weaknesses.

I. Pattern Matching using Law finder

Pattern matching with Law finder, consist searching a pattern string inside the source code and give us results of the number of occurrences of it. Suppose we consider C language, the pattern could be any call to possible dangerous

functions like "getc", or even UNIX command "grep". This method is limited since it returns false positives because there is no analysis of the results; additionally its effectiveness is limited since it depends on the exact writing of the strings [26]. Law finder also uses a more elaborated pattern matching process to find possible vulnerabilities and sort them by their risk levels. This risk level depends on the function and on the values of the parameters assigned to the function.

V. CONCLUSION AND FUTURE WORK

A. Conclusions

Among other several software security authors, software security is a very vital aspect with the current growth in communication industry, where big data and lots of information have to be passed, and securely stored in the systems [5]. For the user to trust his information with the systems then a lot has and is being done to ensure the security of these systems. In this paper we have looked at some security issues, then the various measures of preventing software security weaknesses such as porting software in different browsers, build and breaking, among a number of frameworks.

Finally we looked on the various measures of preventing the software security measures such Security common criteria bench marking, decomposing the elements of complex systems, Measure Software Security as Part of Software Quality, source code analysers, Diversity-Based Approaches to Software Systems Security, developing and reusing of software security patterns, Neural network based security tool for analysing software design for security flaws, and pattern matching.

B. Future work

Software security and quality, being a revolving area, we expect that in future all the software being developed should be secure despite the numerous new security threats that are up-coming. As future research work, there is need to develop software security tools that are able to analyse and advice on the current security threats, as well as include software defences in the software development and software updates.

ACKNOWLEDGMENT

We acknowledge the support of Dr Kimwele of Jomo Kenyatta University of Agriculture and Technology for guiding us into software development.

REFERENCES

[1] Adebiyi, A., Arreymbi, J., and Imafidon, C., *A Neural Network Based Security Tool for Analyzing Software Technological Innovation for the Internet of Things*, IFIP Advances in Information and Communication Technology Volume 394, 2013, pp 80-87.

[2] Alebrahim, A., and Heisel, M., *Towards Developing Secure Software Using Problem-Oriented Security Patterns Availability*, In Reliability and Security in Information Systems, Lecture Notes in Computer Science Volume 8708, 2014, pp 45-62

[3] Alhazmi, O.H., Malaiya, Y.K., and Ray, I., *Measuring, analyzing and predicting security vulnerabilities in software systems*, Elsevier – Science direct, 2007.

[4] Amrutkar, C., Singh, K., Verma, A., Traynor, P., *Vulnerable Me: Measuring Systemic Weaknesses in Mobile Browser Security Information Systems Security*, Lecture Notes in Computer Science Volume 7671, 2012, pp 16-34

[5] Bandhakavi, S., King, S.T., Madhusudan, P., and Winslett, M., *Vetting Browser Extensions for Security Vulnerabilities.* In: Proceedings of the USENIX Security Symposium, SECURITY, 2010.

[6] Barth, A., Felt, A.P., Saxena, P., Boodman, A., *Protecting Browsers from Extension Vulnerabilities.* In: Proceedings of the 17th Network and Distributed System Security Symposium, NDSS, 2010.

[7] Chess, B., *Metrics That Matter: Quantifying Software Security Risk.* Springer, 2010.

[8] Common Weakness Enumeration website — CWE™, accessed at https://cwe.mitre.org on 20th April 2015

[9] Elahi, G., Yu, E., Zannone, N., *A vulnerability-centric requirements engineering framework: analyzing security attacks, countermeasures, and requirements based on vulnerabilities* In Requirements Engineering , Volume 15, Issue 1, 2010 pp 41-62

[10] Fenton, N., *Software Measurement: a necessary Scientific Basis*, IEEE transactions on software engineering. Vol. 20, no. 3, 1994.

[11] Finkelstein, L., *A review of the fundamental concepts of measurement,* Measurement**,** vol. 2, no.1, 1984, pp. 25-34.

[12] Ghaith, S., and Cinnéide, M., *Improving Software Security Using Search-Based Refactoring Search Based Software Engineering* Lecture Notes in Computer Science Volume 7515, 2012, pp 121-135

[13] Gherbi, A., and Charpentier, R., *Diversity-Based Approaches to Software Systems Security Technology* In Communications in Computer and Information Science Volume 259, 2011, pp 228-237

[14] Haley, C., Laney, R., Moffett, J., Nuseibeh, B., *Security requirements engineering: a framework for representation and analysis*. IEEE Trans Software Eng 34: 2008, pp. 133-153

[15] Ibrahim, T., Furnell, M. S., Papadaki, M., and Clarke, L. N., *Assessing the Usability of End-User Security Software*, Trust, Privacy and Security in Digital Business, Lecture Notes in Computer Science Volume 6264, 2010, pp 177-189

[16] Jajodia, S., and Wijesekera, D., (Eds.), Data and Applications Security, LNCS 3654, pp. 281–294, 2005. © IFIP International Federation for Information Processing.

[17] Jimenez, W., Mammar, A., and Cavalli, A., *Software Vulnerabilities, Prevention and Detection Methods: A Review* digital version, 2010.

[18] Kizza, M., J., *Introduction to Computer Network Vulnerabilities* in Guide to Computer Network Security, Computer Communications and Networks 2015, pp 87-103, Springer Verlag London

[19] Lipner, S., and Howard, M., *the Trustworthy Computing Security Development Lifecycle* In Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC'04), 2004, pp. 2-13

[20] Mayer, N., Dubois, E., Matulevicius, R., and Heymans, P., *towards a measurement framework for security risk management.* In: Proceedings of modeling security workshop, 2008.

[21] McGraw, G., et al. , *Building Security* in, IEEE Security and Privacy Magazine, 2005.

[22] Radke, K., Boyd, C., Nieto, J. G., Brereton, M., *Ceremony Analysis: Strengths and Weaknesses Future Challenges in Security and Privacy for Academia and Industry*, IFIP Advances in Information and Communication Technology Volume 354, 2011, pp 104-115

[23] Vanya, A., Klusener, S., Premraj, R., Rooijen, N., Vliet, H., *Identifying and Investigating Evolution Type Decomposition Weaknesses*, In Views on Evolvability of Embedded Systems, Embedded Systems 2011, pp 53-68

[24] Vieira, M., and Antunes, N., *Introduction to Software Security Concepts* in Innovative Technologies for Dependable OTS-Based Critical Systems 2013, pp 29-38, Springer

[25] Wang, J., Guo, M., Wang, H., and Zhou, L., *Measuring and ranking attacks based on vulnerability analysis* in Information Systems and e-Business Management, Volume 10, Issue 4, 2011, pp 455-490

[26] Wheeler, D., 2007, *Flaw finder*, available online at http://www.dwheeler.com/flawfinder/ accessed on 12th April 2015

