Runtime Monitoring – A Post-deployment Security Testing Technique

Ramya Dharam and Sajjan. G. Shiva
Department of Computer Science
University of Memphis
Memphis, TN-38152, USA
{rdharam, sshiva}@memphis.edu

***Abstract*-Increased usage of software systems in recent years has in turn led to high need for ensuring the (i) confidentiality, (ii) integrity, and (iii) availability of these software systems. Pre-deployment security testing techniques do not ensure that all possible behaviors of implementation are analyzed, executed and tested. This in turn causes the software to often behave differently than what it was designed for during its post-deployment, termed as Software Anomaly and is caused mostly due to external attacks such as SQL injection, cross-site scripting, path-traversal attack, etc. To detect such anomalies and to ensure the security and reliability of software systems, a post-deployment security testing technique known as runtime monitoring can be used. This paper reviews various security testing techniques and details runtime monitoring security testing technique. The paper also introduces a framework for the development of runtime monitors to accomplish security testing.**

Keywords-Software Testing; Security Testing; Pre-deployment; Post-deployment; Runtime Monitoring; Software Behavior; Data Flow Testing; Basis Path Testing.

I. INTRODUCTION

Software systems are used by companies and organizations to provide services like online banking, online shopping and social networking also; over the recent years our dependence on them has increased drastically in everyday routine activities. So we expect these systems to be secure and reliable when we are paying bills, shopping online, making transactions etc. Software testing is the most common and popular software development life cycle (SDLC) activity used by the organizations to assure the correctness, quality, security and reliability of the software systems to its users.

Software testing is a series of activities conducted with the intent of finding errors in the software and is not just used for finding and fixing bugs but also to ensure that the system is working according to the specifications [1]. Software testing can be classified into the following four categories based on the purpose: 1) Correctness Testing 2) Performance Testing 3) Reliability Testing and 4) Security Testing [1].

Security testing is primarily performed to validate a system's conformance to specified security requirements and to identify the potential security vulnerabilities i.e. errors that an attacker can exploit within the system. From a pure software development perspective, security vulnerabilities identified through security testing can be viewed in the same manner as "traditional" software bugs that are discovered through software testing processes. "Traditional" software bugs are those deficiencies identified through non-security-related test functions such as unit and subsystem level tests, or stress, performance and load tests. Security Vulnerabilities can have a much greater financial impact on an organization than traditional software bugs. Financial impact can affect both the software development organizations and the end customer organization that uses the software [2].

The security vulnerabilities can be addressed by implementing different types of pre-deployment security testing techniques [1], tools and scanners [3] within the software development life cycle to identify and resolve security issues. But, software systems are still susceptible to attacks, regardless of the strength of design and implementation, so monitoring software behavior is an excellent defensive technique [4].

In order to obtain highly secure and reliable software, testing the software during its post-deployment requires major attention thus motivating us to employ techniques, methodologies, and tools to assure the security and reliability of the software once deployed.

In this paper, we describe a post-deployment security testing technique called runtime monitoring and also introduce a framework to develop runtime monitors to accomplish runtime monitoring of the software system. Continuously monitoring or observing the behavior of executing software and determining whether it complies with the specified properties is termed as runtime monitoring [5] and is performed by software runtime monitors.

The paper is organized as follows. In Section 2, we present security testing techniques and introduce runtime monitoring security testing technique. In Section 3, we introduce a framework for development of runtime monitors to achieve runtime monitoring of deployed software. Section 4 discusses the related work. Conclusion is discussed in Section 5.

II. SECURITY TESTING TECHNIQUES

Security testing ensures that the systems used by organizations and users are secured from any unauthorized attack [1]. The overall goal of security testing is to reduce vulnerabilities within software systems. A variety of testing techniques are specifically used to probe security. There are two major aspects of security testing: testing security functionality to ensure that it works and testing the subsystem in light of malicious attack. Security testing is motivated by probing undocumented assumptions and areas of particular complexity to determine how a program can be broken [2].

Security testing for software has recently migrated beyond the domain of network port scanning to include checking the software's intrinsic behavior. This test goes far deeper than a simple black box test that probes on the presentation layer and it goes even beyond the functional testing of security apparatuses [6]. Security Testing will help us to achieve the Confidentiality, Integrity, Authorization and Availability of the software systems.

Different security testing techniques exist as shown below in Figure 1.

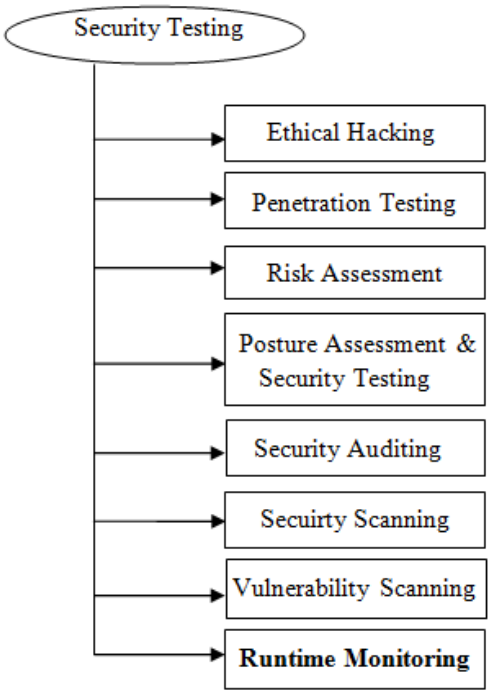

Figure 1. Different Security Testing Techniques [1]

1. Ethical Hacking: Ethical Hacking is also known as intrusion testing. It is performed by an ethical hacker who is a network and computer expert and hacks software systems on behalf of its developers and testers to discover the possible security vulnerabilities existing in the software system and report the problems instead of taking advantage of them which a malicious hacker could exploit.

2. Penetration Testing: The term "penetration testing" refers to testing the security of a computer system and/or software application by attempting to compromise its security, and in particular the security of the underlying operating system and network component configurations [7]. It is the process of attempting to gain access to the resources without any knowledge of usernames, passwords and other normal means of access. If the focus is on computer resources, then examples of a successful penetration would be obtaining or subverting confidential documents, pricelists, and databases and other protected information. The main thing that separates a penetration tester from an attacker is permission. The penetration tester will have permission from the owner of the computing resources that are being tested and will be responsible to provide a report. The goal of a penetration test is to increase the security of the computing resources being tested. In any cases, a penetration tester will be given user-level access and in those cases, the goal would be to elevate the status of the account or user other means to gain access to additional information that a user of that level should not have access to [8].

3. Risk Assessment: Risk assessment determines the relevance of vulnerabilities occurred and varies over time. It is a method of assessing the risk depending on the type of loss and the probability of the occurrence of the loss. It helps in preparing a possible backup-plan for any type of potential risk and hence, contributes towards the security conformance. Conducting interviews, discussions and performing analysis of the same are different ways in which risk assessment is carried out.

4. Posture Assessment and Security Testing: This is a combination of Security Scanning, Risk Assessment and Ethical Hacking and helps the organizations to know where it stands in the context with Security conformance [1].

5. Security Auditing: Security Auditing includes the usage of direct inspection like code walk-through and interviewing staff to perform systematic evaluation of security of the application developed, Operating Systems and any system on which it is being developed.

6. Security Scanning: Security Scanning is all about scanning the network, systems, or applications to discover possible vulnerabilities in them. Security scanning consists of utilization of a wide variety of automated software tools to perform routine tests and checks. Security scanning finds the areas of vulnerability in network, systems or applications without breaking into them illegally.

7. Vulnerability Scanning: Vulnerability Scanning is also known as vulnerability assessment. It identifies and classifies the vulnerabilities in an application, network and computer system. A variety of vulnerability scanners are available for use which consists of a database containing all the information required to identify the vulnerabilities detected during the scanning process.

In spite of the usage of all the above mentioned pre-deployment security testing techniques, software systems are still vulnerable to attacks during its post deployment. To

achieve security of software systems during its post-deployment a security testing technique known as Runtime Monitoring can be used.

Runtime Monitoring involves monitoring the program behavior which is an important part of any testing process. Usually it means examining the behavior of the program under test and asking whether this observed behavior is symptomatic of vulnerability in the software. This examination can be harder in security testing than it is in traditional testing, because the tester is not necessarily comparing actual program behavior to expectations derived from specifications. Rather, the tester is often looking for unspecified symptoms that indicate the presence of unsuspected vulnerabilities [3]. To perform runtime monitoring we introduce a framework which is used to develop runtime monitors to monitor the behavior of the software system during its post-deployment.

III. PROPOSED FRAMEWORK

We introduce a framework described in [9] to perform post-deployment security testing of the software. Our main goal is to identify the set of critical paths to be monitored in software during runtime such that security vulnerabilities in the software system are identified.

Our framework shown in Figure 2 consists of the following phases: (a) Software Repository, (b) Critical Variable Identification, (c) Data Flow Testing, (d) Basis Path Testing, (e) Path Identification Function, (f) Monitor Development, and (g) Monitor Integration.

The framework first uses a software repository which consists of a collection of documents related to requirements, security specifications, source code, etc. to find the critical variables. Identification of critical variables, which accept data from the external world and also part of critical operations, is accomplished in the critical variable identification phase of the framework. Then, a combination of basis path and data flow testing techniques is used to find all the legal/valid execution paths the critical variables can take during their lifetime in the application. Runtime monitors are then developed to observe the path taken by the critical variables and check them for compliance with the obtained legal paths. During runtime, if the path taken by the identified critical variable violates the legal paths obtained, implies that the critical variable consists of the malicious input from the external user and is trying to obtain unauthorized access to the information. This abnormal behavior of the application due to the critical variables is identified by the runtime monitor and immediately notifies to the administrator. The framework described is shown in Figure 2 and consists of three main steps which are discussed below in detail.

Identification of critical variables: Scan the software repository to identify all the critical variables present in the source code. Critical variables are those which interact with the external world by accepting user input, and also which are part of critical operations that involve query executions.

Build legal execution paths: By combining data flow and basis path testing, legal execution paths of the application are obtained. Data flow testing of the critical variables help in identification of all the legal sub paths that can be taken by critical variables during the execution. Basis path testing is performed to identify the minimum number of legal execution paths of the software. Since basis path testing leads to reduced number of monitorable paths, the complexity of our proposed technique in terms of integrating monitors across multiple paths also reduces.

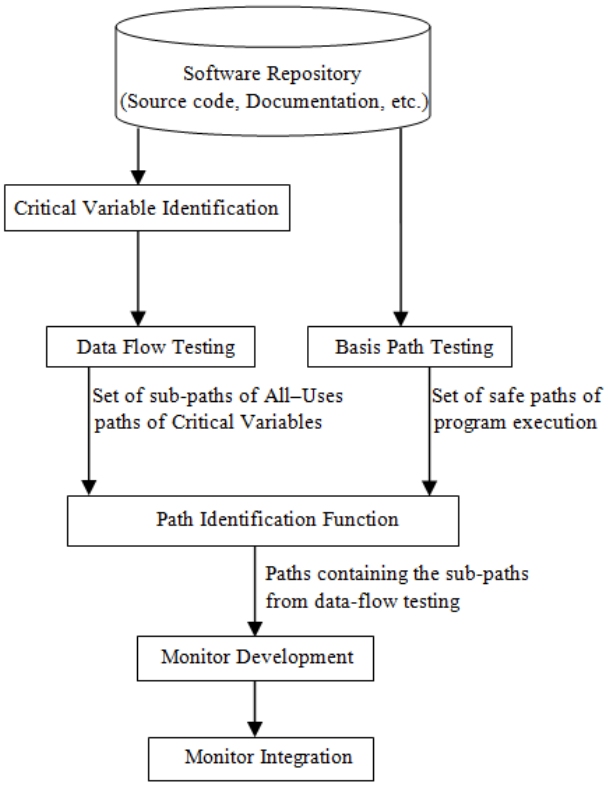

Figure 2. Runtime monitoring framework

Let $C = \{C^1, C^2 \ldots\ldots, C^m\}$ be a set of *m* critical variables identified during critical variable identification phase. Let $P_C = \{\{ P_C^1 \} \cup \{ P_C^2 \} \cup \ldots..,\{ P_C^m\}\}$ be a set of critical variable sub paths such that, P_C^i is a set of all valid sub paths a critical variable C^i can take during its lifetime in the software, $i \in [0, m]$ and is identified by performing data flow testing on C^i. Let $P = \{P^1, P^2 \ldots\ldots, P^k\}$ be a set of k legal paths identified using basis path testing and CP is a set of paths we intend to monitor. CP is identified using the pseudo code shown below:

```
CP = { }
for every P^j ϵ P and
    for every P_C^i ϵ P_C
        if (P^j ∩ P_C^i == P_C^i)
            CP = CP U { P^j }
```

where, $i \in [0, m]$ and $j \in [0, k]$.

We thus identify all the critical paths of the software to be monitored.

Runtime Monitoring: In this phase, we map the identified critical paths to regular expressions and use the monitoring oriented programming (MOP) [10, 11] tool to generate monitors. The generated monitor is then integrated with the respective module of the application for monitoring the critical paths.

IV. RELATED WORK

Khan in [1, 12] describe about the different pre-deployment software testing techniques that can be used for finding errors in the software applications and classifies them based on their purpose. The paper also describes about different white box testing techniques such as control flow testing, data flow testing, branch testing, basis path testing and loop testing respectively. Curphey et al. in [13] describe about different tools and techniques that can be used for analyzing security vulnerabilities in Web application's during its pre-deployment phase.

Huang et al. in [14, 15] describe the use of software testing techniques like dynamic analysis, black box testing, fault injection and behavior monitoring in achieving Web application security. The tools and techniques mentioned in the paper focus on achieving quality and security of software during its pre-deployment phase. In the current paper we describe about a security testing technique for a software system during its post-deployment.

V. CONCLUSION

In this paper, we described runtime monitoring as a post-deployment security testing technique. We also introduced a framework for development of runtime monitors used to perform post-deployment security testing of the software system. Thus, using our proposed framework we ensure that the quality and security of software is achieved not only using pre-deployment security testing techniques but also can be achieved using post-deployment security testing technique.

VI. REFERENCES

[1] Mohd. Ehmer Khan, "Different Forms of Software Testing Techniques for Finding Errors", International Journal of Computer Science Issues, Vol. 7, Issue 3, No 1, May 2010.

[2] https://buildsecurityin.us-cert.gov/bsi/articles/best-practices/testing/255-BSI.html

[3] https://buildsecurityin.us-cert.gov/bsi/articles/tools/black-box/261-BSI.html

[4] G. McGraw, "Software Security", IEEE Security & Privacy, Volume 2, Issue 2, Pages: 80-83, Mar-Apr 2004.

[5] N. Delgado, A. Gates, and S. Roach, "A Taxonomy and Catalog of Runtime Software Fault monitoring Tools", *IEEE Transactions on Software Engineering*, Volume: 30, Issue: 12, Pages: 859 - 872, Dec 2004.

[6] G. McGraw and B. Potter, "Software Security Testing", IEEE Security & Privacy, Volume 2, Issue 5, Pages 81-85, Sept-Oct 2004.

[7] https://buildsecurityin.us-cert.gov/bsi/articles/best-practices/penetration/655-BSI.html

[8] S. Northcutt, J. Shenk, D. Shackleford, T. Rosenberg, R. Siles, and Steve Mancini, "Penetration Testing: Assessing Your Overall Security Before Attackers Do", SANS Analyst Program, June 2006.

[9] Ramya Dharam, Sajjan. G. Shiva, "A Framework for Development of Runtime Monitors", International Conference on Computer and Information Sciences (ICCIS), Kuala Lumpur, Malaysia, June 2012.

[10] F. Chen, M. d' Amorium and G. Rosu, 'Monitoring-Oriented Programming: A Tool-Supported Methodology for Higher Quality Object-Oriented Software", Technical Report UIUCDCS-R-2004-2420, 2004.

[11] F. Chen and G. Rosu, "Java MOP: A Monitoring Oriented Programming Environment for Java", in Proceedings of Eleventh International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), 2005.

[12] Mohd. Ehmer Khan, "Different Approaches to White Box Testing Technique for finding Errors" , International Journal of Software Engineering and Its Applications, Vol. 5, No. 3, July 2011.

[13] M. Curphey and R. Arawo, "Web Application Security Assessment Tools", IEEE Security & Privacy, Volume: 4, Issue: 4, Pages: 32 - 41, Jul – Aug 2006.

[14] Y.W. Huang, S. K. Huang, T. P. Lin & C. H. Tsai, "Web Application Security Assessment by Fault Injection and Behavior Monitoring", Proceedings of the 12th International Conference on World Wide Web, 2003.

[15] Y. W. Huang, F. Yu, C. Hang, C. H. Tsai, D. T. Lee and S. Y. Kuo, "Securing Web Application Code by Static Analysis and Runtime Protection", Proceedings of the 13th International Conference on World Wide Web, 2004.

[16] J. D. Meier, "Web Application Security Engineering", IEEE Security & Privacy, Volume 4, Issue 4, Pages: 16-24, Jul-Aug 2006.

[17] M. R. Stytz and S. B. Banks, "Dynamic Software Security Testing", IEEE Security & Privacy, Volume 4, Issue 3, Pages: 77-79, May-June 2006.