A Framework for Development of Runtime Monitors

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

***Abstract*- Software Testing is the process used to assure the correctness, completeness, performance, security and reliability of the software. Different software testing techniques are used during pre-deployment phase of the software. But, these do not ensure that all possible behaviors of implementation are analyzed, executed and tested. Because of the incomplete assurance from the testing methodology, software can sometimes behave very differently during the post-deployment phase. This is termed as software anomaly and they are caused mostly by 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 during the post-deployment phase, a technique known as runtime monitoring can be used. This paper presents a framework for the development of runtime monitors to accomplish post-deployment monitoring of software to detect and prevent path traversal attack.**

I. INTRODUCTION

The increased size and complexity of modern software applications have made them susceptible to be easily exploited by attackers. Performing complete software testing is a critical and an important verification activity to be performed during the pre-deployment phase, so that a highly secure and reliable software application is available to users in the post-deployment phase.

Despite the existence of different software testing methodologies and tools [2] for the assurance of correctness, completeness, security and reliability; the software can sometimes behave very differently during the post-deployment phase. The most common reason being failure to test the software thoroughly during pre-deployment phase as the testing activities are time consuming, expensive and difficult [4]. Dynamically testing the software during the pre-deployment phase does not assure complete identification of all software vulnerabilities. This in turn leads to exploitation of vulnerabilities causing attacks such as buffer overflow, SQL injection, path traversal attack, etc. In order to obtain highly secure and reliable software, monitoring the software during post-deployment phase requires major attention thus motivating us to employ techniques, methodologies, and tools to assure the security and reliability of the software after it is deployed.

Continuously monitoring or observing the behavior of executing software and determining whether it complies with the specified properties is termed as runtime monitoring [17] and is performed by software runtime monitors. Our proposed framework uses the information obtained from performing the two pre-deployment software testing techniques, to help us in the development of runtime monitors to accomplish post-deployment runtime monitoring of the software.

The framework described in this paper combines two pre-deployment testing techniques: 1) Data Flow Testing and 2) Basis Path Testing. Data anomalies are caused due to improper declarations and manipulation statements existing in high-level programs. Data Flow Analysis testing [9] is an effective approach to detect improper use of data and can be performed either statically or dynamically. In static Data Flow Analysis, the source code is inspected to track the sequences of uses of data items without its execution. However, in the dynamic data flow analysis, the sequences of actions are tracked during execution of the program. Basis Path testing is a white box testing technique that identifies the minimal set of all legal execution paths [2] from both the control flow graph of the program and by the calculation of cyclomatic complexity, the measure of number of independent paths in the program being considered.

In this paper, we propose a framework for the development of runtime monitors to achieve post-deployment monitoring and any possibility of path traversal attack to the software application is detected and prevented.

The paper is organized as follows. In Section 2, we present the proposed framework. In Section 3, we discuss the development of a software application used by our proposed framework to detect path traversal attack and the results obtained. In Section 4, we present Game Inspired Defense Architecture (GIDA) framework. Section 5 discusses the related work. Conclusion and future work is discussed in Section 6.

II. PROPOSED FRAMEWORK

We propose a framework to perform post-deployment monitoring of the software. Our main goal is to identify the set of critical paths to be monitored in software during runtime such that the detection and prevention of path traversal attack is achieved.

The proposed framework in Figure 1 consists of the following phases: (a) Software Repository, (b) Critical Variable Identification, (c) Data Flow Testing, (d) Basis

978-1-4673-1938-6/12/$31.00 ©2012 IEEE

Path Testing, (e) Path Identification Function, (f) Monitor Development, and (g) Monitor Integration. In our proposed framework, we focus mainly on Java applications, and hence we use Monitoring Oriented Programming (MOP) [18, 19, 20] tool to integrate the monitor with the source code.

The documents related to software requirements and development such as functional requirements, security specifications and source code are obtained from the software repository. 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. We define critical operations as a set of functions which affect the internal data of the software.

By performing data flow testing of the identified critical variables present in the software, we identify all possible sub-paths thus obtaining the life cycle of the critical variables identified. Basis path testing is then performed to identify the minimum number of legal execution paths. Since basis path testing leads in the reduction of monitorable paths, the complexity of our proposed method in terms of integrating monitors across multiple paths also reduces.

The path identification function to identify the set of paths to monitor in an application for the detection and prevention of path traversal attack is defined as follows:

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

```
Let, Critical_Path = { }
for every Basis Path P^j ϵ PATH and
    for every critical variable sub path P_C^i ϵ P_C
        if (P^j ∩ P_C^i == P_C^i )
            CRITICAL_PATH = CRITICAL_PATH U {Pj}
```

In the above ($0<=i<=m$) and ($0<=j<=k$).

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

In the monitor development phase, we map the identified critical paths to regular expressions and use the MOP tool to generate monitor. The generated monitor is then integrated in the specific locations of the applications source code, such that post-deployment runtime monitoring of all the identified critical paths is achieved.

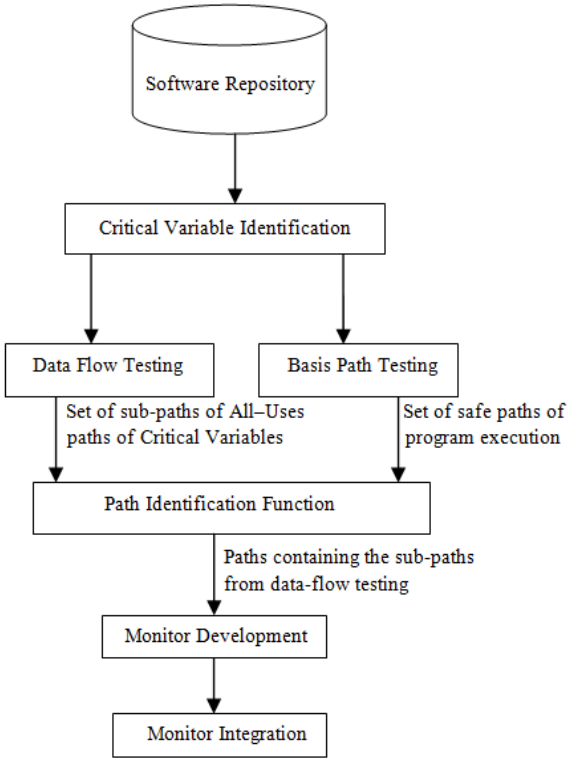

Figure 1: Runtime monitoring framework for Path Traversal Attacks.

III. EXAMPLE: AN INSTANCE OF PATH/DIRECTORY TRAVERSAL ATTACK

Directory traversal attack [24, 25] occurs when an attacker attempts to access files under a directory that are not intended to be accessed. This attack works on applications that accept input from users and uses it in a path that is used by the application to access a filesystem. In this section we provide an example of software application that is vulnerable to directory traversal attack. We further briefly describe about the construction of runtime monitors for the software application using our proposed framework to detect and prevent directory traversal attacks.

If any special characters like ".." (dot-dot) are included in the input given by an attacker that modify the meaning of the path, then the application will misbehave and allow the attacker to access confidential information or unauthorized resources [25]. By using the runtime monitor, we identify this abnormal behavior of the program as an attack and immediately halt the execution of the program notifying the administrator.

The application developed has the following environment set up: each user's information is stored in a separate .txt file and all the files are stored in a single directory under "\home\users". Two files user1.txt and user2.txt which contains user's information are stored under "\home\users" directory. Any user has unrestricted access to all the files present under "users" directory but, has a restricted access to all the other files present in other directories such as "\etc" (i.e. etc directory under the root directory).

User requests the application to provide access to the files present under “users” directory and gives the filename as input to the application. The application receives input from the user and further resolves it into path such as “\home\users\user1” and the access to the file user1 is provided to the user.

The attacker now tries to gain access to password file present in the “passwd” directory by performing the directory traversal attack and provides an input like “..\..\etc\passwd”. The application receives the input from the user and resolves it into path such as “\home\users\..\..\etc\passwd” and further resolve the special character "..\" in the user input to the parent of current working directory and thus gives the attacker access to “\etc\passwd” directory.

To prevent the attacker from gaining access to “passwd” directory, the runtime monitor developed by using the proposed framework observes the behavior of the executing program. When the program resolves the special character “..\” given in the user input and tries to give the access of root directory, this abnormal behavior of the executing program is immediately identified by the runtime monitor. The monitor then halts the execution of the program notifying the administrator. Thus, the runtime monitor developed prevents the attacker from gaining access to the root directory.

IV. GIDA FRAMEWORK

The framework proposed in the current paper is used to enhance the working of the Target Self-monitoring Application component present in the GIDA architecture as shown in figure 2.

Shiva et al. in [21, 22] proposed Game Inspired Defense Architecture (GIDA) as a holistic approach designed to secure target applications/network against probable attacks.

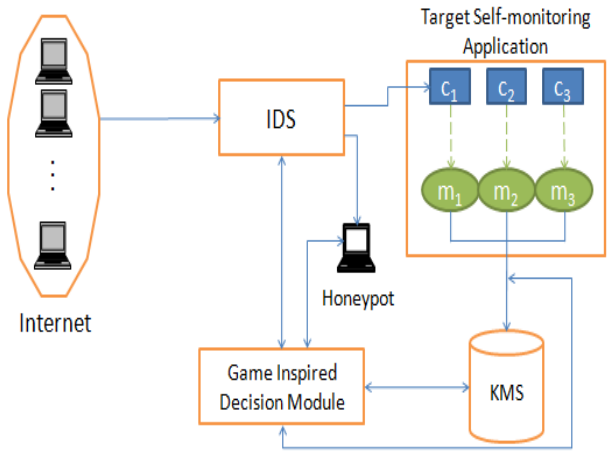

Figure 2: Game Inspired Defense Architecture (GIDA) [21]

GIDA architecture shown in figure 2 consists of the following components, namely: Intrusion Detection System (IDS), Knowledge Management System (KMS), Game Inspired Decision Module (GIDM), Honeypot and Target Self Monitoring Application (TSMA). In the rest of paper, we refer TSMA as Monitoring System (MS).

The IDS is used to monitor the network traffic and capture as much information possible about the network behavior and its usage statistics. The information thus collected will be used to identify the attack that has occurred. In our proposed framework IDS is considered as a network level sensor.

We consider software applications developed in modular fashion consisting of different modules/components represented using C_1, C_2, and C3 as shown in Figure 2. Different modules/components execute either in different target systems or on a single target system connected to internet. Each module/component is instrumented with their respective runtime monitor, represented using m_1, m_2 and m_3 as shown in Figure 2. The runtime monitor will detect the abnormal behavior of the module/component during its execution which may be caused due to existing internal errors or due to external attacks performed on the modules/components. The process of identifying the abnormal behavior by the runtime monitor will be performed by utilizing the framework proposed in the current paper. The runtime monitor is considered to be an application level sensor.

Once the abnormal behavior is identified, the runtime monitor can perform any of the following functions:
i) If the monitor is able to identify the attack that has occurred, it will immediately execute defense action by itself to secure the module/component. The simplest and direct defense action to implement by the runtime monitor is to immediately halt the execution of the entire module/component depicting the abnormal behavior.
ii) If the monitor is not able to identify the attack that has occurred, then it forwards the information gathered related to the abnormal behavior to both KMS and GIDM, to identify the attack occurred and wait for the response.

The output from either the network or the application level sensors is forwarded as input to GIDM which is the brain and an important component of the proposed Game Inspired Defense Architecture (GIDA). GIDM processes the input information received from sensors (IDS and MS) and could take either of the following mentioned actions based on the input received and its knowledge about the attacks occurred:
i) If GIDM is able to identify the attack occurred, then it suggests a defense.
ii) If GIDM is not able to identify the attack occurred, it forwards the input received either from IDS or MS about the occurred abnormal behavior to KMS, for identification of the attack that has occurred and a suitable defense action to be executed.

iii) If neither KMS nor GIDM is able to identify the attack occurred, then GIDM invokes the honeypot which is primarily used for analyzing traffic and gathering additional information from the attacker.

In all the above mentioned cases the defense actions suggested either by GIDM or by KMS will be executed either by GIDM itself to protect the target network/ application or will be forwarded to either the IDS or MS for them to execute the suggested defense action.

The KMS focuses on determining the type of attack and it consists of game models mapped to the kinds of attack they can address. The KMS is based on a cyber-attack taxonomy called AVOIDT [23]. Based on the parameters provided as inputs from GIDM, KMS uses AVOIDIT to identify the characteristics of an attack. Once an attack is identified, a candidate game model which can defend against such an attack is selected and notified to GIDM. GIDM then either executes the suggested game model as the defense action to protect the target network/application or GIDM will forward the defense action information to either IDS or MS for them to execute the suggested defense action.

GIDA functions in the following sequential manner to provide security to either the target application/network:
1. Receive inputs from either the network or application sensors.
2. Identify the attack occurred either by its prior knowledge or with the help of KMS or honeypot.
3. Select a relevant game model for the occurred attack by either utilizing its prior knowledge or with the help of KMS.
4. Execute the selected game model or allow the sensors to perform defense actions to secure target application/network against the attacker.

V. RELATED WORK

Khan in [1, 2] describe about the different 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 [3] 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 [5, 6, 7] 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 above described papers focus on achieving quality and security of software during the pre-deployment phase. In the current paper we focus on proposing a technique for achieving quality and security not only during the pre-deployment phase of the software but, also during the post-deployment.

Boujarwah et al. in [8] propose a dynamic data flow analysis technique to test Java program to detect data flow anomalies in them. Similarly Saleh et al. in [9] introduce an approach to use dynamic data flow analysis technique to test concurrent Java programs against synchronization anomalies. Hong et al. in [10] present a vulnerability analysis framework to detect security holes in Java web applications. The objective of vulnerability analysis is to find out the unknown security holes in a system. Zhao et al. in [12] propose a framework for analyzing the vulnerability for a Java bytecode program using data flow analysis technique.

Cain et al. in [13] propose an approach for performing dynamic data flow analysis of Object Oriented programs, by constructing a meta model of the runtime structure of an Object Oriented program. Guangmei et al. in [14] describe about path testing and the procedure for automatically generating basis set of path for path testing. Most of the software errors can be detected during software testing using a kind of white box testing technique called path testing. The paper stresses the importance of automatically generating correct basis set of paths required for performing path testing. In the current paper we focus on using data flow analysis and basis path testing technique in our proposed framework to accomplish the development of software runtime monitors to detect and prevent path traversal attack.

F. Chen et al. in [18, 19, 20] describe about the Monitoring Oriented Programming (MOP) framework and a Java based tool called Java-MOP that implements the MOP framework. Monitors are used to check the dynamic behavior of the executing software with respect to specified properties. User defined recovery code which includes raising exceptions, outputting messages could be executed if any violations of the specifications are found. For our experimentation purpose we use Java-MOP tool.

VI. CONCLUSION

In this paper, we proposed a framework for development of runtime monitors used to perform post-deployment monitoring of the software to detect and prevent path traversal attack. Thus using our proposed framework we ensure that the quality and security of software is achieved not only during its pre-deployment phase but, also during its post-deployment phase and any possible exploitation of vulnerability in the software by an external attacker is detected and prevented.

We further intend to automate the entire process of using the proposed framework to develop the runtime monitors and also extend the framework for detecting more complex attacks like SQL injection, XSS, etc.

VII. REFRENCES

[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] 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.
[3] M. Curphey and R. Arawo, "Web Application Security Assessment Tools", IEEE Security & Privacy, Volume: 4, Issue: 4, Pages: 32 - 41, Jul – Aug 2006.
[4] A. Orso, "Monitoring, Analysis and Testing of Deployed Software", Proceedings of the FSE/SDP workshop on Future of Software Engineering Research, 2010.
[5] 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.
[6] 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.
[7] Y. W. Huang and D. Lee, "Web Application Security-Past, Present, and Future", Computer Security in the 21st Century, Pages: 183 – 227, 2005.
[8] A. S. Boujarwah, K. Saleh, J. Al-Dallal, "Dynamic Data Flow Analysis for Java Programs", Information Software Technology, Volume: 42, Issue: 11, April 2000.
[9] K. Saleh, A. S. Boujarwah, J. Al-Dallal, "Anomaly Detection in Concurrent Java Programs Using Dynamic Data Flow Analysis", Information and Software Technology, Volume: 43, Issue: 15, December 2001.
[10] T. Hong, C. Hua, Z. Gang, L. Qiang and Z. Jinjin, "The Vulnerability Analysis Framework for Java Bytecode", 15th International Conference on Parallel and Distributed Systems (ICPADS), December 2009.
[11] H. Chen, T. Zou and d. Wang, "Data Flow Based Vulnerability Analysis and Java Bytecode", Proceedings of the 7th Conference on 7th WSEAS International Conference on Applied Computer Science, November 2007.
[12] G. Zhao, H. Chen and D. Wang, "Data Flow Based Analysis of Java Bytecode Vulnerability", The Ninth International Conference on Web Age Information management (WAIM), July 2008.
[13] A. Cain, T. Y. Chen, D. D. Grant, F. C. Kuo and J. G. Schneider, "An Object Oriented Approach Towards Dynamic Data Flow Analysis", The Eighth International Conference on Quality Software, 2008.
[14] Z. Guangmei, C. Rui, L. Xiaowei and H. Congying, "The Automatic Generation of Basis Set of Path for Path Testing", Proceedings of 14th Asian Test Symposium (ATS), 2005.
[15] Dr. H. Schligloff and Dr. M. Roggenbach, "Path Testing".
[16] J. Poole, "A method to determine a basis set of paths to perform program testing", Report 5737, U.S. Department of Commerce/National Institute of Standards and Technology, Gaithersburg, Md, USA, November 1995.
[17] 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.
[18] 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.
[19] 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.
[20] F. Chen, M. d'Amorium and G. Rosu, "Checking and Correcting Behaviors of Java Programs at Runtime with Java MOP", Electronic Notes in Theoretical Computer Science, Volume 144, Issue 4, May 2006.
[21] S. Shiva, H. Bedi, C. Simmons, M. Fisher, R. Dharam, "A Holistic Game Inspired Defense Architecture", International Conference on Data Engineering and Internet Technology (DEIT), March 2011.
[22] S. Shiva, S. Roy and D. Dasgupta, "Game Theory for Cyber Security", 6th Cyber Security and Information Intelligence Research Workshop, April 2010.
[23] C. Simmons, S. Shiva, D. Dasgupta, and Q. Wu, "AVOIDT: A Cyber Attack Taxonomy", Technical Report: CS-09-003, University of Memphis, August 2009.
[24] OWASP: The Open Web Application Security project. Category: Path Traversal Attack
[25] CWE – Common Weakness Enumeration. CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal').