A Framework for Development of Runtime Monitors

Ramya Dharam

University of Memphis, Department of Computer Science: GTCS Research Lab

Advisor: Dr. Sajjan .G. Shiva

Abstract

Software Testing is a process used in Software Development Life Cycle to help identify the correctness, completeness, security, reliability and quality of the developed software. Different software testing techniques (Static and Dynamic) are used during pre-deployment phase of the software. But, this does not ensure that all possible behavior of its implementation are analyzed, executed and tested.

Because of this incomplete assurance by the testing methodology, the software can sometime behave unexpectedly during its execution in the post-deployment phase which is termed as "Software Anomaly". Such software anomalies are caused mostly due to external attacks such as SQL Injection, Cross Site Scripting, etc.

To detect such anomalies and to ensure the security and reliability of software during its post-deployment phase, dynamic software testing techniques such as "Runtime Monitoring" can be used.

We propose a framework for development of runtime monitors to achieve post-deployment software testing for detecting and preventing Path Traversal Attack.

Introduction

Different software testing methodologies and tools exists to assure the quality of the software during its pre-deployment phase but still, the software can sometimes behave unexpectedly during its post-deployment phase and the most common reason being, failure to test the software thoroughly during its pre-deployment phase due to time constraints. This in turn leads to exploitation of software vulnerability to perform attacks such as Path Traversal, SQL Injection, etc.

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

Runtime monitoring is one such post-deployment dynamic software testing technique which could be used to achieve our objective.

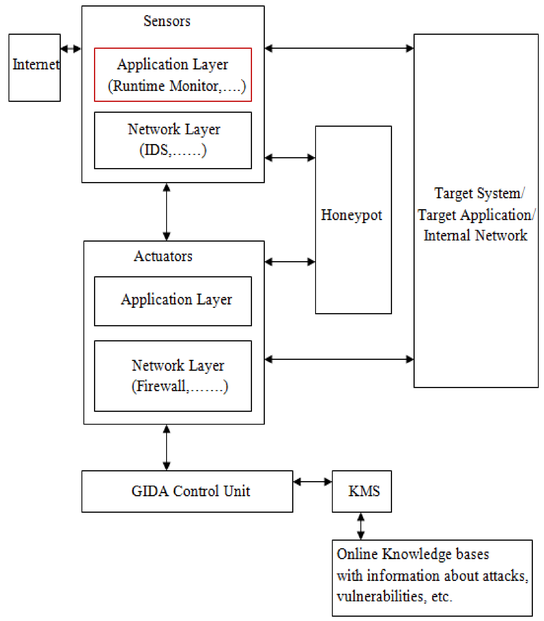
Continuously monitoring or observing the behavior of executing software and determining whether it complies with the specified properties is termed as "Runtime Monitoring" and is performed by "Runtime Monitor" which is a software program.

Our proposed framework uses the above idea along with combining the two pre-testing software techniques 1) Data Flow Testing Technique and 2) Basis Path Testing Technique to help us in the development of runtime monitor to perform post-deployment testing.

We use the technique of program instrumentation where-in the monitor developed is integrated with the software and then executed to achieve runtime monitoring of the software.

Monitoring the software during its execution using our proposed framework, any possibility of Path Traversal Attack can be detected and prevented.

GIDA Framework



Internal Network/Target System/Target Application: is being protected by the GIDA defense system.

Internet: is the external world with which the target communicates during its operation.

Sensors & Actuators: is the channel through which all the interactions happen between the Internet and the Target System/Application.

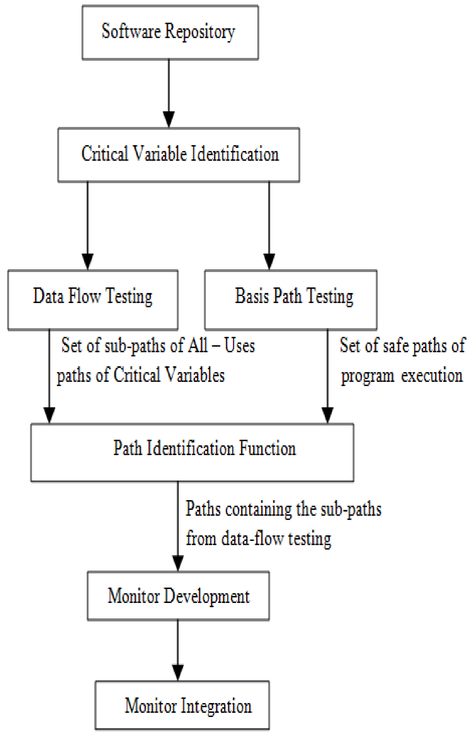
Knowledge Management System (KMS): is the knowledge center where the information related to the observed anomalies is sent in and a defense action plan is got as the output to the GIDA Control Unit block.

Honeypot: is the trap to which a suspected user is sent to understand his intentions.

Online Knowledge Base: consists of the online knowledge bases related to attacks, vulnerabilities, risks etc.

GIDA Control Unit: is the central control unit for the whole defense operation of GIDA. It is connected to Sensors & Actuators which intimates when there is any anomaly observed. It sends control signals to Sensors & Actuators to abort a traffic or operation, drive it to honeypot or communicate with KMS to decide what to do.

Proposed Framework



Let C = {C^1, C^2……., C^m} be a set of *m* critical variables identified during Critical Variable Identification phase and P_C = {{ P_C^1 } U { P_C^2 } U …..,{ 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 ……, 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 ($P^j \cap P_C^i == P_C^i$)
    CRITICAL_PATH = CRITICAL_PATH U {P^j}

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

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

In the monitor development phase, we map the identified critical paths to regular expressions and since we focus mainly on Java applications, MOP tool is used to generate the monitor and then instrumented with the source code of the application for monitoring all its identified critical paths.

Experimental Setup

Path Traversal Attack also known as Directory Traversal Attack occurs when an attacker attempts to access files under a directory that are not intended to be accessed. This attack works on applications that receive input from users and uses it in its path to provide access to files.

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 is 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 by giving 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.txt is provided to the user.

Results

The attacker now tries to gain access to the password.txt file present in "passwd" directory by performing the directory traversal attack and providing an input like "..\..\etc\passwd".

The application after receiving the input from the user will resolve 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. Thus the attacker could gain access to "\etc\passwd" directory and access the files present.

To prevent the attacker from gaining access to "passwd" directory, the runtime monitor developed by utilizing the proposed framework observes the behavior of the executing program and when the program resolves the special character "..\" given in the user input and tries to give access of root directory to the attacker, this abnormal behavior of the executing program will be immediately identified by the runtime monitor.

It then prevents the attacker from gaining access to the directory by halting the execution of the program and notifies the administrator about the possible attack.

Conclusion and Future Work

The proposed framework can be used for development of runtime monitors to detect and prevent Path Traversal Attack in an application during its post-deployment phase.

We plan to extend the framework for detecting more complex attacks such as SQL Injection, Cross Site Scripting, etc and further enhance the functionality of monitors such that it can send signals to GIDA Control Unit and implement defense action plan suggested by KMS.