International Journal of Information and Computation Technology.
ISSN 0974-2239 Volume 3, Number 10 (2013), pp. 1009-1014

http://www. irphouse.com /ijict.htm

Detecting Unsafe Component Loadings using Static Techniques

K.B. Hemanth, G. Ramesh and K. Prabhakar

Department of CSE, JNTUA College of Engineering
Ananthapuram, India.

Abstract

Dynamic loading of software components (e.g., libraries or modules) is a widely used mechanism for an improved system modularity and flexibility. Correct component resolution is critical for reliable and secure software execution. However, programming mistakes may lead to unintended or even malicious components being resolved and loaded. In particular, dynamic loading can be hijacked by placing an arbitrary file with the specified name in a directory searched before resolving the target component. Although this issue has been known for quite some time, it was not considered serious because exploiting it requires access to the local file system on the vulnerable host. Recently, such vulnerabilities have started to receive considerable attention as their remote exploitation became realistic. It is now important to detect and fix these vulnerabilities. In this paper, we present the static analysis based automated technique to detect vulnerable and unsafe dynamic component loadings.

1. Introduction

DYNAMIC loading is an important mechanism for software development. It allows an application the flexibility to dynamically link a component and use its exported functionalities. Its benefits include modularity and generic interfaces for third-party software such as plug-ins. Because of these advantages, dynamic loading is widely used in designing and implementing software.

A key step in dynamic loading is component resolution, i.e., locating the correct component for use at runtime. Operating systems generally provide two resolution methods, either specifying the fullpath or the filename of the target component. With

fullpath, operating systems simply locate the target from the given full path. With filename, operating systems resolve the target by searching a sequence of directories, determined by the runtime directory search order, to find the first occurrence of the component.

Although flexible, this common component resolution strategy has an inherent security problem. Since only a file name is given, unintended or even malicious files with the same file name can be resolved instead. Thus far this issue has not been adequately addressed. In particular, we show that unsafe component loading represents a common class of security vulnerabilities on the Windows and Linux platforms (cf., Section 5). Operating systems may provide mechanisms to protect system resources. For example, Microsoft Windows supports Windows Resource Protection (WRP) [1] to prevent system files from being replaced. However, these do not prevent loading of a malicious component located in a directory searched before the directory where the intended component resides.

The problem of an unsafe dynamic loading had been known for a while, but it had not been considered a serious threat because its exploitation requires local file system access on the victim host. The problem has started to receive more attention due to recently discovered remote code execution attacks [2], [3], [4], [5], [6], [7], [8], [9]. Here is an example attack scenario. Suppose that an attacker sends a victim an archive file containing a document for a vulnerable program (e.g., a Word document) and a malicious DLL. In this case, if the victim opens the document after extracting the archive file, the vulnerable program will load the malicious DLL, which leads to remote code execution.

With current breakthrough with availability of open source code directories like source forge people download the sources and use them as dynamic components integrated with their code. But the trust of that code is not for sure and it may have severe vulnerabilities. This motivates us for the current paper work.

In this paper work, we propose a static code analysis technique to detect a component is safe to load or unload. Library user who intends to use a dynamic component must provide the source code of the component and his source code which uses the dynamic component to the tool. This analyses the use and point if any suspicious vulnerability is present. Also we will check for security violation in the dynamic component code using the CVC rules.

2. Related Work

Software components often utilize functionalities exported by other components such as shared libraries at runtime. This operation is generally composed of three phases: resolution, loading, and usage. Specifically, an application resolves the needed target components, loads them, and utilizes the desired functions provided by them. Component interoperation can be achieved through dynamic loading provided by operating systems or runtime environments.

For example, the LoadLibrary and dlopen system calls are used for dynamic loading on Microsoft Windows and Unix-like operating systems, respectively. Dynamic loading is generally done in two steps: component resolution and chained component loading.

In order to resolve a target component, it is necessary to specify it correctly. To this end, operating systems provide two types of target component specifications: fullpath and filename. For fullpath specification, operating systems resolve a target component based on the provided fullpath. For example, a fullpath specification /lib/libc-2.7.so for the libc library in Linux determines the target component using the specified full path. For filename specification, operating systems obtain the full path of the target component from the provided file name and a dynamically determined sequence of search directories. In particular, an operating system iterates through the directories until it finds a file with the specified file name, which is the resolved component. For example, suppose that a target component is specified as midimap.dll and the directory search order is given as C:\Program Files\iTunes;C:\Windows\System32;. . . ;$PATH on Microsoft Windows. If the first directory containing a file with the name midimap.dll is C:\Windows\System32, the resolved full path is determined by this directory.

In dynamic loading, the full path of the target component is determined by its specification through the resolution process, and the component is incorporated into the host software if it is not already loaded. During the process of incorporating the target component, the component's loadtime dependent components are also loaded. Fig. 1 illustrates the general procedure of dynamic loading. Suppose component B is loaded by component A. B's dependent components (e.g., component C) are also loaded. We can usually obtain information on B's dependent components from B's file description. This process of chained component loading is repeated until all dependent components have been loaded.

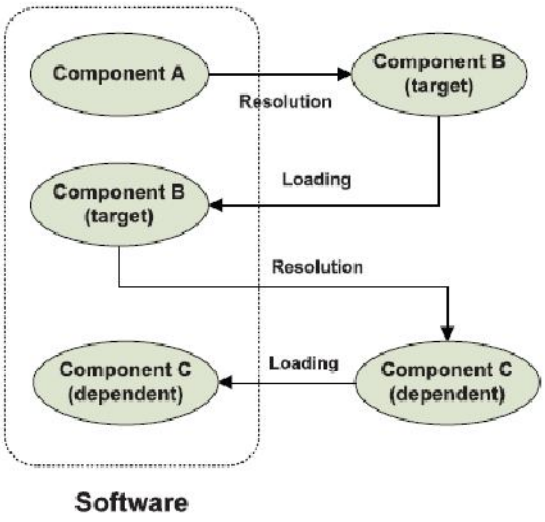

Fig. 1: Dynamic component loading procedure.

3. Overview of Proposed Solution

In this section we present the overview of the solution. The proposed software tool will analyze the source code and find the functionalities expected from the dynamic component. This process is called as contract construction. It will check the dynamic component code for full compliance with the contract and contract violations. This process is called as contract violation check. It also check if there is any security vulnerability in the usage of dynamic component. We use the CVC security violation rules to check for any security violation in the code. This process is called security violation check.

4. Details of Proposed Security Mechanism

4.1 Contract Construction

Contract is nothing but the agreement between user of dynamic component and the dynamic component. Contract is in terms of classes and function. All the classes and functions required by the Contract must be implemented in the dynamic component.

Contract construction is fully automated process. When the source code of our application is provided, it will check for the dynamic component usage in the code and extract the functions and classes usage from the dynamic component and builds the contract. Contract construction is very much language specific and in this paper we will do contract construction for java code. JSR 291 specification for dynamic component will be used for constructing the parser for the java code.

4.2 Contract violation Checking

Based on the contract extracted from the user code, the dynamic component code is parsed and checked for violation. The violation is detected if the matching class or matching function on the class is not found in the dynamic component.

4.3 Security Vulnerability Detection

CWE™ International in scope and free for public use, CWE provides a unified, measurable set of software weaknesses that is enabling more effective discussion, description, selection, and use of software security tools and services that can find these weaknesses in source code and operational systems as well as better understanding and management of software weaknesses related to architecture and design.

To assist in enhancing security throughout the software development lifecycle, and to support the needs of developers, testers and educators, the Common Attack Pattern Enumeration and Classification (CAPEC) is another of these efforts sponsored by DHS CS&C as part of the Software Assurance strategic initiative. The objective of this effort is to provide a publicly available catalog of attack patterns along with a comprehensive schema and classification taxonomy. Linked with CWE, this CAPEC site contains the initial set of content and will continue to evolve with public

participation and contributions to form a standard mechanism for identifying, collecting, refining, and sharing attack patterns among the software community.

We use the following Security violation checking on the code

CWE ID	Description
CVE-2002-0466	Server allows remote attackers to browse arbitrary directories via a full pathname in the arguments to certain dynamic pages
CVE-2002-1483	Remote attackers can read arbitrary files
CWE-400	Resource exhaust
CVE-2001-0255	Suspicious read write
CVE-1999-0674	System call usage

5. Results

We implemented proposed solution in JAVA. We tested out solution for using the 5 libraries on image processing downloaded from source-forge and all these used as dynamic component in imagej application and we measured the number of contract violations. We also found the number of security violations. We found our solution is able to find out more than 75% of compliance faults and security vulnerabilities.

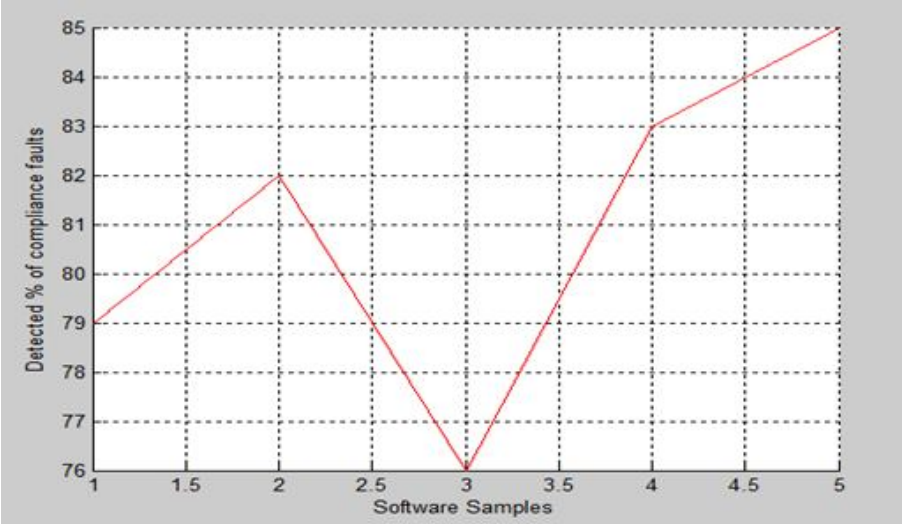

Fig. 2: Number of Faults occurred.

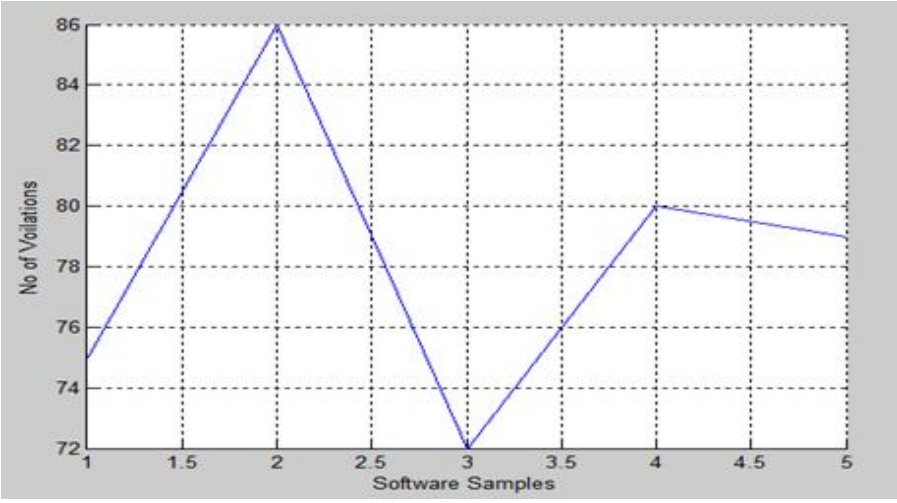

Fig. 3: Number of violations in software.

6. Conclusion

We have proposed and evaluated static analysis solution to detect unsafe component loadings and proved that our solution is able to identify more than 75% vulnerabilities. we propose a static code analysis technique to detect a component is safe to load or unload. This technique involves analyzing the source code and point out if any vulnerability is present.

References

[1] T. Kwon and Z. Su, "Automatic Detection of Unsafe Component Loadings," Proc. 19th Int'l Symp. Software Testing and Analysis, pp. 107-118, 2010.

[2] "Windows DLL Exploits Boom; Hackers Post Attacks for 40-Plus Apps," http://www.computerworld.com/s/article/9181918/Windows_DLL_exploits_boom_hackers_post_attacks_for_40_plus_apps, 2011.

[3] "Hacking Toolkit Publishes DLL Hijacking Exploit," http://www.computerworld.com/s/article/9181513/Hacking_toolkit_publishes_DLL_hijacking_exploit, 2011.

[4] "About Windows Resource Protection," http://msdn.microsoft.com/en-us/library/aa382503(VS.85).aspx, 2011.