Secure Environment for Unsafe Component Loading

E.Kodhai[#1], Gnanasundari.A[*2],

[#] *Associate Professor, Department of Information and Technology,*

**P.G. Student, Department of Computer Science and Engineering,*
Sri Manakula Vinayagar Engineering College, Puducherry.

[1]kodhaiej@yahoo.co.in

[2]gnanasundari15.5@gmail.com

***Abstract*-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. 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. 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. A technique to detect unsafe dynamic component loadings is proposed. The paper proposes a secure environment to detect unsafe component loading in dual systems such as client and server.**

***Keywords*— Dynamic loading, component resolution, component loading.**

I. INTRODUCTION

Dynamic loading is widely used in designing and implementing software. Its benefits include modularity and generic interfaces for third-party software such as plug-ins. It also helps to isolate software bugs as bug fixes of a shared library can be incorporated easily. Dynamic loading components are utilization requires local file system access on the end host. The following problems are occurred in the local and remote dynamic components loading. In local system, the file does not exist in the specified path or the specified search directories, hijacking the components. Although in the remote system, the browser automatically download arbitrary files to the user's Desktop directory without any prompting, vulnerable program starts up through the shortcut, an archive file containing a document and a malicious component.

A. Remote Attacks

Buffer overflows have been the most common form of security vulnerability for the last so many years. Then came web browsers that are plagued with vulnerabilities, providing hackers with easy access to computer systems via browser-based attacks.

One such project was Mashup OS, that proposed new abstractions to facilitate improved sharing among multiple principles hosted in the same web page. Mixed concrete and symbolic execution important technique for finding and understanding software bugs, including security relevant ones. However, previous to Loop-Extended Symbolic Execution on binary programs, symbolic execution techniques were limited to examining one execution path at a time, in which symbolic variables reflect only direct data dependencies. A key limitation of single-path symbolic execution is that it interacts poorly with loops, a common programming construct. This approach is applied to the problem of detecting and diagnosing buffer overflow vulnerabilities, in a tool that operates on unmodified Windows and Linux binaries. Rather than trying a large number of inputs in an undirected way, this approach often discovers an overflow on the first candidate it tries. Unsafe component resolutions may cause an application to load unintended components.

1) "Carpet Bomb"-Based Attack: The Carpet Bomb attack can lead to remote code execution in conjunction with unsafe DLL loading on Microsoft Windows. In particular, when the browser accesses a malicious webpage, attackers can make the browser automatically download arbitrary files to the user's desktop directory without any prompting [3]. This flaw leads to remote code execution if a vulnerable application checks in the desktop directory first for resolving a DLL. When Internet Explorer 7 runs, it loads this DLL file and executes arbitrary code [4].

2) *"Shortcut with Component" Attack*: Sending a victim an archive file containing a shortcut to a vulnerable program and a malicious component can also cause remote code execution. If the vulnerable program starts up through the shortcut, it loads the component and executes malicious code.

3) *"Document with Component" Attack:* Opening a document can load particular files placed in the same directory as the document. This vulnerability can be exploited to launch remote code execution attacks by sending a victim an archive file containing a document and a malicious component.

B) Serious Security Vulnerabilities

An unsafe component loading can cause serious security vulnerabilities in software. They can be zombie, virus etc., the user may not know that the system software is infected but it can damage the system when the infected code is getting executed once the user clicks on the file injected by the hacker.

II. RELATED WORK

Brumleyet al. [11] , proposed a tool for automatically protecting against integer-based vulnerabilities, efficiently detecting integer-based attacks against C programs at run time. A compiler extension that compiles C programs to object code that monitors its own execution to detect integer-based attacks is analyzed. This compiler is a useful and lightweight software testing tool. The limitation is it has a run-time defense mechanism that may generate false positives when programmers use integer overflow deliberate and it can miss some integer bugs because it does not model certain C features.

Safe component resolution [12], present a mechanism safe-open, to prevent unsafe component resolutions in Unix by detecting modifications to path names by untrusted users on the system. A dynamic analysis is performed to discover unsafe component loading vulnerabilities in the software. Vulnerability analysis and detection testing and analysis techniques for detecting software vulnerabilities have been well explored.

The Saturn tool [14] expresses program properties as boolean constraints, which models pointers and heap data down to the bit level. Since dynamic checking runs code, it is limited to just executed paths, but can more effectively check deeper properties. Examples include program executions that loop on bad inputs, or byzantine errors that occur when a formatting command (such as for printf) is not properly obeyed. Many of the errors in this paper would be difficult to discover statically. To overcome this EXE testing is applied which shows how aggressive symbolic execution can be used to find such security holes inreal systems code and other interesting errors [15].

Data Space Randomization [13], in which the most critical updates from software vendors have been based on vulnerabilities such as buffer over flows heap overflows, exploitation of these vulnerabilities with the most promising defenses being based on randomization. Two randomization techniques that have been explored are address space randomization (ASR) that randomizes the location of objects in virtual memory, and instruction set randomization(ISR) that randomizes the representation of code. These method explore a third form of randomization called data space randomization(DSR) that randomizes the representation of data stored in program memory.

Valueguard [16] , is a countermeasure for data-only attacks caused by buffer overflows. Valueguard's detection technique consists of inserting canary values in front of all memory objects and verifying them when the objects are used. These countermeasure operates on the source code level and does not require any modifications to the target platform valueguard can be used either as a testing tool by developers before deployment of an application or as a run-time protection monitor for critical applications. Using valueguard a previously unreported buffer overflow in the olden benchmark suite was found and it showed that valueguard can detect and stop data-only attacks that many other generic countermeasures cannot. It overcomes all the attacks that can be caused by buffer overflow.

Bas Cornelissenet et al.[18], proposed a tool quantification of the usefulness of trace visualization for program comprehension. To gain a deeper understanding of the nature of task its added value, some 8types of tasks is tested in EXTRAVIS which is a tool for execution of trace visualization. To fulfill these goals, design and execution of a controlled experiment in which how the tool affects (1) the time that is needed for typical comprehension tasks, and (2) the correctness of the solutions given during those tasks is measured.

Loop-Extended Symbolic Execution [17] , on Binary Programs is an important work wherein the goal is to extend the symbolic expressions computed from a single execution by incorporating additional information reflecting the effects of loops that were executed. In single-path symbolic execution, the values of variables are either concrete or are represented by a symbolic expression. But some of the values considered concrete by single-path symbolic execution are in fact indirectly dependent on the input because of loops. To make loop-extended symbolic execution more tractable, the task is split into two parts by introducing a new class of symbolic variables, which we call trip counts. Each loop in the program has a trip count variable that represents the number of times the loop has executed at any moment to keep track of the paths being executed.

III.BACKGROUND

A) Dynamic Analysis

Dynamic analysis is to detect component loading. There are three types of information collected. The simplest way to determine the memory address of a variable we want to inject is to obtain it at execution time. The information are system calls are invoked for dynamic loading, image loading, process and thread identifiers. The collected information is stored in a profile that is later used for reference.

System call analysis is a widely used analysis technique to understand program behavior because a sequence of invoked system calls can provide useful information on a program execution. To capture system-level actions for the dynamic component loading, system calls that cover all possible control-flow paths of the dynamic loading procedure are instrumented , which enables the procedure offline to be reconstructed. Also its parameter information for detecting unsafe component resolutions is detected. Specifically, the

target component specification (i.e., specified fullpath or filename) and the directory search order can be obtained from the system call parameters.

Actual loadings of target components through dynamic binary instrumentation is done. The loading information is needed for reconstructing the loading procedure in a combination with the information captured by the system call instrumentation. It also indicates the resolved full path determined by the loading procedure. Unsafe component loading is detected using resolved path by the unsafe checker as shown in figure 2.

If the target program uses multithreads and each thread loads a component dynamically, the instrumented system calls for each loading can be interleaved, which makes it difficult to correctly reconstruct the loading procedure of each thread. To solve this problem, process and thread identifiers along with the other information on instrumented system is detected.

B) DETECTION OF UNSAFE COMPONENTS

Unsafe component resolution is classified into unsafe resolution and resolution failure as sorted out with its necessary conditions in Table 1.

1) Resolution Failure of a Target Component:

To detect failed resolution of a target component, the number of image loads and the number of failed resolutions during the dynamic loading procedure is detected. In particular, if no image is loaded and the resolution of the component failed, the component loading is said to be a resolution failure.

2)Unsafe Resolution of a Target Component:

To check whether the target component is specified by its file name is necessary because a full path specification does not iterate through the search directories for resolution. If a file name is used, then the resolved path of the target component by retrieving the first element of a list of image loads in the dynamic loading procedure is noted.

TABLE I

CONDITIONS FOR COMPONENT LOADING

TYPE	CONDITION
Resolution failure found	Target component is not found
Unsafe resolution	i) Target component is specified by its name ii) Target component is resolved by iterating through multiple directories iii) There exists another searched directory before resolution

IV. SYSTEM OVERVIEW

The proposal is an effective dynamic analysis to detect vulnerable and unsafe dynamic component loadings. The work introduces an automated technique to detect and analyse vulnerabilities and errors related to the dynamic component. The technique is to implement a set of practical tools for detecting unsafe component loadings.The overall architecture is shown in figure 3. The exploitability of unsafe component loadings in terms of local and remote attacks is identified. The analysis is done in two phases in order to reduce the performance overhead incurred during dynamic binary instrumentation. The first phase is online phase and the second phase is offline phase as shown in figure 1. First, a sequence of system-level actions are captured for dynamic loading during a program's execution then dynamic binary instrumentation is used to generate the profile on its runtime execution The detection of vulnerable components is carried out using algorithm in figure 2 with reference to table 1.

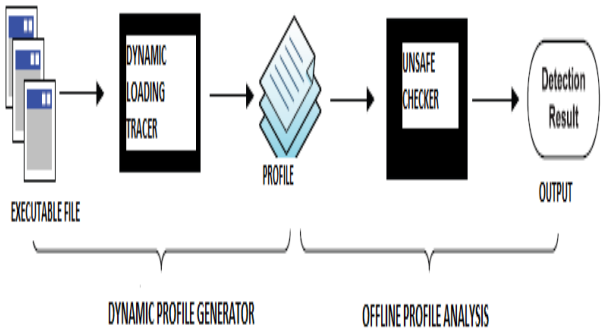

Figure1. Unsafe component detection

Local attacks assume that attackers can access the local file system on a victim host, while remote attacks assume that attackers can only send data to the victim user.

Input: S (a sequence of actions for a dynamic loading)
Auxiliary functions:
TargetSpec(S): return target specification of S
DirSearchOrder(S): return directory search order used inS
ImgLoad(S): return the image loadings in S
ResolutionFailure(S): return the resolution failures in S
ChainedLoading(S): return actions for the chained loadings in S

```
IsUnsafeResolution(filename, resolved_path, search_dirs):
check whether the resolution is unsafe
1: img_loadsImgLoad(S)
2: failed_resolutionsResolutionFailure(S)
3: if jimg_loadsj ¼¼ 0 then
4: if jfailed_resolutionsj ¼¼ 1 then
5: Report this loading as a resolution failure
6: end if
7: else
8: spec TargetSpec(S)
9: dirsDirSearchOrder(S)
10: if spec is the filename specification then
11: resolved_pathimg_loads[0].resolved_path
// retrieve the first load
12: if IsUnsafeResolution(spec,resolved_path,dirs)
then
13: Report this loading as an unsafe resolution
14: end if
15: end if
16: chained_loadsChainedLoading(S)
17: for each_load in chained_loadsdo
18: OfflineProfileAnalysis(each_load)
19: end for
20: end if
```

Figure 2. Offline Profile Analysis

To accomplish remote attacks exploiting unsafe component loadings, attackers need to place malicious files in the DLL-hijacking directories from remote sites. However, accessing the file system of a remote host is generally prohibited. For example, the system directory is not accessible remotely unless the directory is shared to the remote user or the system is exploited by other vulnerabilities to enable it. Due to this difficulty in remote exploitation, unsafe component loadings have not been considered as a serious security threat.

To find remote attacks on unsafe DLL loadings caused by the following three conditions: resolution failure, file name specification and standard or alternate search order the loading can be focused. Dynamic loading is performed by the dlopen system call. The first phase of the technique is platform-dependent, while the second is platform-independent. Unsafe component loading is essentially a type of programming defects[5,6]. Therefore, this problem often arises in operating systems that support dynamic loading.

A) MODULES

The proposed work is carried out as a set of modules. They are as follows

1. Erroneous Program
2. Profile construction
3. Unsafe checker
4. Results and forward

1) Erroneous Program:

A Java program created with error and bugs and erroneous program files. This program is executed by the server. This vulnerability can be exploited to launch remote code execution attacks by sending the victim an archive file containing a document and an erroneous component. The user is given with the erroneous code and it is passed on to the server.

The output of each and every phase namely the construction module, the profile generation are all based on the result acquired from the erroneous code module. Too many codes made of erroneous program may spoil the system, so an effective analysis of the bugs and proper identification of the tools is required. In order to analyze the unsafe components first the erroneous code must be given to the system administrator for the future development. The functional requirement of this is erroneous code.

2) Profile construction:

The profile from the running program contains system calls, image loading, thread process and identifiers. The malicious program is taken and analyzed using the three conditions. A detailed analysis has been widely used to understand software behavior and to adopt this approach for component loading. Dynamically instrument the binary executable under analysis to capture a sequence of system-level actions for dynamic loading of components. During the instrumented program execution, we collect three types of information such as system calls invoked for dynamic loading, image loading and process and thread identifiers. The collected information is stored as a profile for the instrumented application and is analyzed in the offline profile analysis phase.

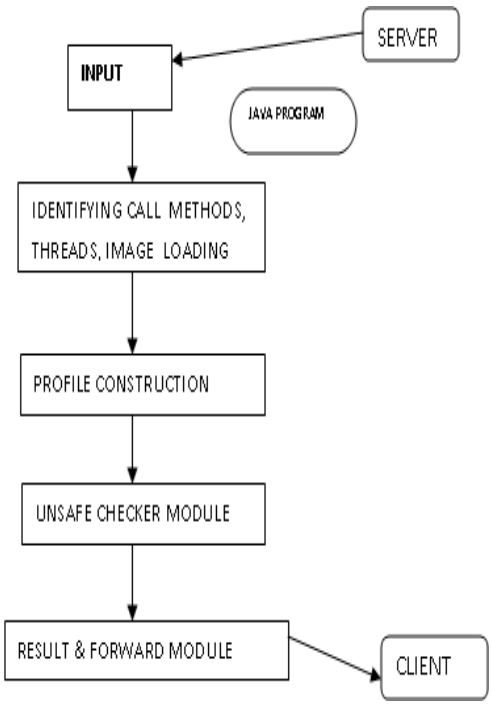

Figure3. Architecture Design

System Calls Invoked for Dynamic Loading

System call analysis is a widely used analysis technique to understand program behavior because a sequence of invoked system calls can provide useful information on a program execution.

To capture system-level actions for the dynamic component loading, we instrument system calls that cover all possible control-flow paths of the dynamic loading procedure, which enables us to reconstruct the procedure offline. Besides the name of an instrumented system call, its parameter information is also collected for detecting unsafe component resolutions. Specifically, the target component specification (i.e., specified fullpath or filename) and the directory search order can be obtained from the system call parameters. Although the directory search order can vary according to the underlying system and program settings, it is computed by operating systems and provided as parameters to the relevant system calls for dynamic loading. Furthermore, results of the instrumented system calls provide both the control flow in the loading procedure and error messages generated by the operating systems. Such information is used for the reconstruction of the dynamic loading procedure and the detection of unsafe loadings.

Image Loadings: We also capture actual loadings of target components through dynamic binary instrumentation. The loading information is needed for reconstructing the loading procedure in combination with the information captured by the system call instrumentation. It also indicates the resolved full path determined by the loading procedure. We use this resolved path to detect unsafe component loading.

Process and Thread Identifiers: Since this approach is based on the system call instrumentation, it is important to consider multithreaded applications. If the target program uses multithreads and each thread loads a component dynamically, the instrumented system calls for each loading can be interleaved, which makes it difficult to correctly reconstruct the loading procedure of each thread. To solve this problem, we capture process and thread identifiers along with the other information on instrumented system

3) Unsafe Checker Module:

The unsafe components are analyzed based on Profile information. All components are loaded from the profile. Group a sequence of action in the profile by process and thread identifiers as the actions performed by different threads may be interleaved due to context switching. Grouping separates the sequences of loadings performed by different threads. Divide the sequence for each thread into sub sequences of actions, one for each distinct dynamic loading. This can be accomplished by using the first invoked call for loading as a delimiter. Obtain a list of groups, each of which contains a sequence of actions for loading a component at runtime. This gives the possible control flows in the loading procedure. In this function, unsafe components are collected from the program. The program components are checked by the conditions. To detect failed resolution of a target component, simply check the number of image loads and the number of failed resolutions during the loading procedure.

In particular, if no image is loaded and the resolution of the component failed, then report the component loading as a resolution failure.

Thus, the image loading is equal to zero. This is the necessary condition for resolution failure because a program may attempt to load a component that is already loaded. To avoid reporting any false resolution failures, explicitly check whether a resolution failure has occurred. Check whether the target component is specified by its file name because a full path specification does not iterate through the search directories for resolution. If a file name is used, then retrieve the resolved path of the target component by retrieving the first element of a list of image loads in the dynamic loading procedure. The first element of the list corresponds to the target component because, if the target component is already loaded or its resolution is failed there exists no image loading in the loading procedure. The target component is always loaded for the first time during its runtime loading. Based on the resolved full path, the target component specification, and the applied directory search order, determines whether to classify this as an unsafe resolution by checking the directories searched before the resolution. To detect unsafe component resolutions in the chained loading procedure by performing the above mentioned conditions recursively.

4) Result and forward module:

The detected unsafe components results are forwarded to the user. The user is able to view the result based on which the unsafe components are identified. Based on which we can resolve system call at runtime and confirm file existence of resolution.

V. CONCLUSION

The technique to detect unsafe dynamic component loadings is described. The technique works in two phases. It first generates profiles to record a sequence of component loading behaviors at runtime using dynamic binary instrumentation. It then analyzes the profiles to detect two types of unsafe component loadings: resolution failures and unsafe resolutions. Thus a technique is proposed for the unsafe components detection and analyzed in dual system that is the client and server.

ACKNOWLEDGMENT

The authors thank the anonymous reviewers for their detailed and constructive review of this work.

REFERENCES

[1] Taeho Kwon, Studen and Zhendong Su, "Automatic Detection of Unsafe Dynamic Component Loadings", IEEE transactions on software engineering, 2012

[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_attack, 2011

[3] "About the Security Content of Safari 3.1.2 for Windows," http:// support.apple.com/kb/HT2092, 2011.

[4] "IE's Unsafe DLL Loading," http://www.milw0rm.com/ exploits/2929, 2011.

[5] Tielei Wang, TaoWei, Zhiqiang Lin, Wei Zou,"IntScope: Automatically Detecting Integer Overflow Vulnerability in X86 Binary Using Symbolic Execution"

[6] Mark Mitchell, Jeffrey Oldham, and Alex Samuel, "Secure Linux Programming"- Book

[7] "Researcher Told Microsoft of Windows Apps Zero-Day Bugs 6 Months Ago," http://www.computerworld.com/s/article/print/9181358/Researcher_told_Microsoft_of_Windows_apps_zero_ day_bugs_6_months_ago, 2011.

[8] "Zero-Day Windows Bug Problem Worse than First Thought, Says Expert," http://www.computerworld.com/s/article/9180978/Zero_day_Windows_bug_problem_worse_than_first_thought_ saysexpert, 2011

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

[10] T. Kwon and Z. Su, "Automatic Detection of Unsafe Component Loadings", Software Testing and Analysis, 2010.

[11] D. Brumley, D.X. Song, T. Chiueh, R. Johnson, and H. Lin, "RICH: Automatically Protecting against Integer-Based Vulnerabilities", Network and Distributed System Security , Mar. 2007.

[12] S. Chari, S. Halevi and W. Venema, "Where Do You Want to Go Today? Escalating Privileges by Pathname Manipulation", Mar. 2010.

[13] Bhatkar, S., Sekar, R.," Data space randomization", Detectionof Intrusions and Malware & Vulnerability Assessmentconference , 2008

[14] Y. Xie and A. Aiken, " Scalable error detection using booleansatisfiability", IGPLAN-SIGACT symposium on Principles of programming languages, 2005.

[15] C. Cadar, V. Ganesh, P.M. Pawlowski, D.L. Dill, and D.R. Engler, "Exe: Automatically Generating Inputs of Death", 2006.

[16] Steven Van Acker, Nick Nikiforakis, Pieter Philippaerts, Yves Younan, and Frank Piessens,"ValueGuard: Protection of native applications against data-only buer over ows" ACM Conference on Computer and Communications Security

[17] PrateekSaxenaPongsinPoosankam, Stephen McCamant Dawn Song,"Loop-Extended Symbolic Execution on Binary Programs"ACM,2009

[18] Bas Cornelissen, Andy Zaidman, Arie van Deursen,"Trace Visualization for Program Comprehension:A Controlled Experiment", Technical Report, Delft University of Technology, 2009

