ISSN: 2278 – 1323
International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)
Volume 2 Issue 12, December 2013

DETECTION OF UNSAFE COMPONENT LOADINGS USING DYNAMIC ANALYSIS TECHNIQUE

N. Geethanjali[1], S.Priyadarshini[1], Dr.S.Karthik[2]

Abstract— Dynamic loading plays a major role in software development process. The flexibility in it allows linking a component dynamically and using its exported functionalities. Dynamic loading allows the system to perform, a library can be loaded or unloaded in the memory, then the loaded library functions and variables can be retrieved and executed or accessed. The dynamic analysis technique deals with detection of vulnerable and unsafe dynamic component loadings in the system. This work introduces the first automated technique to detect and analyze vulnerabilities and errors related to the dynamic component loading, and also detects for safe components loaded in the memory. The analysis comprises of two phases namely, Online Phase to apply dynamic binary instrumentation to collect runtime information on component loading, and Offline Phase to analyze the collected information to detect vulnerable component loadings. The technique uses a set of practical tools for detecting and removing unsafe component loadings on Microsoft Windows and Linux. An extensive analysis of unsafe component loadings on various types of popular software has been conducted.

***Index Terms*— Binary Instrumentation, Dynamic loading, Unsafe Component Loading, Vulnerability.**

I. INTRODUCTION

Dynamic loading is defined to either load or unload a library into memory at run time. It helps us to retrieve the address of functions and variables present in the library. With the help the retrieved address of those functions and variables it can be executed or accessed from the library. It is possible to discover available libraries, and potentially gain extra functionalities for a program to startup even in the absence of libraries, which is not so in static and loadtime linking. In software development dynamic loading plays a vital role.

Manuscript received Dec, 2013.

N. Geethanjali, *Department of Computer Science and Engineering, SNS College of Technology, Coimbatore, India.*

S.Priyadarshini*, Department of Computer Science and Engineering, SNS College of Technology, Coimbatore, India.*

Dr.S.Karthik*, Department of Computer Science and Engineering, SNS College of Technology, Coimbatore, India.*

For implementing software plug-ins dynamic loading is most frequently used technique. Consider an example, with dynamic loading the Apache Web Server's "dynamic shared object" (*.dso) plug-in files are libraries which are loaded at runtime. In implementing computer programs where multiple different libraries may supply the requisite functionality and where the user has the option to select which library or libraries to provide.

Dynamic loading is not supported by all systems. Dynamic loading is provided by UNIX-like operating systems such as Mac OS X, Linux, and Solaris with the C programming language "dl" library.

Dynamic loading is provided in the Windows operating system through the Windows API. To protect system resources operating systems may provide mechanisms. Consider an example, Microsoft Windows supports Windows Resource Protection (WRP) to prevent system files from being replaced. Loading of a malicious component located in a directory searched before the directory where the intended component resides, is not prevented in Microsoft Windows.

Only because of the remote code executions attacks the problem of an unsafe dynamic loading has received attention to solve resolution failure and vulnerabilities. Its exploitation requires local file system access on the victim host, thus they were not considered seriously. Consider an example, that an attacker sends a vulnerable program (e.g., a Word document) and a malicious DLL, then the victim opens the document after extracting the archive file, the vulnerable program will load the malicious DLL. This leads to remote code execution in the system. Thus this technique will check for those files and stop execution for such files temporarily.

LoadLibrary or LoadLibraryEx is used to accomplish loading on Windows and with dlopen on UNIX-like operating systems. GetProcAddress on Windows and dlsym on UNIX-like operating systems is used for extracting the content of a dynamically loaded library is achieved with. Some software components utilize functionalities at runtime exported by other components such as shared libraries. There are three phases in this operation: resolution, loading, and usage. Even more, an application resolves the needed target components.

II. RELATED WORK

Dinakar Dhurjati and Vikram Adve proposes the problem of enforcing correct usage of array and pointer references in C and C++ programs which remains unsolved. The approach proposed by Jones and Kelly (extended by Ruwase and Lam) is the only one known of that does not require significant manual changes to programs, but it has extremely high overheads in the two versions. The author describes a collection of techniques that dramatically reduce the overhead of this approach, by exploiting a fine-grain partitioning of memory called Automatic Pool Allocation. Together, these techniques bring the average overhead checks down to only 12% for a set of bench- marks (but 69% for one case). This shows that the memory partitioning is a key to bringing down this overhead. It also shows that this technique successfully detects all buffer over- run violations in a test suite modeling reported violations in some important real-world programs.

David Molnar , Xue Cong Li and David A. Wagner proposes for security vulnerabilities, common root cause is considered as integer bugs, including integer overflow, width conversion, and signed/unsigned conversion errors. The author introduces a new method for discovering integer bugs using dynamic test generation on x86 binaries, and describes key design choices in efficient symbolic execution of such programs. The author implemented his methods in a prototype tool SmartFuzz, which is used to analyze Linux x86 binary executables. To aid in triaging and reporting bugs found by SmartFuzz and the black-box fuzz testing tool zzuf, the author also created a reporting service, metafuzz.com.

The report on experiments is gathered by applying these tools to a range of software applications, including the mplayer media player, the exiv2 image metadata library, and ImageMagick convert. Also another report on using SmartFuzz, zzuf, and metafuzz.com to perform testing at scale with the Amazon Elastic Compute Cloud (EC2) is gathered. For reference, till date, the metafuzz.com site has recorded more than 2; 614 test runs, comprising 2; 361; 595 test cases. The experiment found approximately 77 total distinct bugs in 864 compute hours, costing us an average of $2:24 per bug at current EC2 rates. The author quantifies the overlap in bugs found by the two tools, and we show that SmartFuzz finds bugs missed by zzuf, including one program where Smart- Fuzz finds bugs but zzuf does not.

Chris Grier, Samuel T. King and Dan S. Wallach argues that browsers should be responsible for specifying and enforcing security policies for browser plug-ins. Browsers can significantly reduce the impact of plug-in vulnerabilities and eliminate much of the risk posed by today's plug-in exploits, by enabling the browser to make security decisions on behalf of the plug-in. The author proposes policies for document access, persistent state, network connections and other devices that browser-based security policy.

For videos, music, and documents, web browser plug-ins have become the present tool on the Internet. The web applications have faced radical change due to new, feature-rich plug-ins. Consider YouTube works mostly behind the plug-in Flash Player YouTube work – without the streaming video support added in Adobe Flash 7, YouTube would not have taken off. Unfortunately, plug-ins are riddled with security vulnerabilities and expose users to significant risk. In today's browsers, Plug-ins are considered to be the single largest source of vulnerabilities, accounting for 476 reported vulnerabilities in 2007 compared to 163 for browsers, including IE, Firefox, and Safari combined. To implement its own security policy and enforcement mechanisms that fail when an attacker can exploit a plug-in, each plug-in is responsible itself. A plug-in API is used by the plug-ins to interact with the browser, such as the NPAPI, supported by the browser. The common functionality of Plug-in APIs in browsers is that, the browser provides plug-ins with document (i.e., DOM) access, network connectivity and interaction with other browser components. Except for some document accesses, each plug-in is responsible for restricting the use of this API by the plug-in content as well as controlling access to the underlying system.

Shuo Chen†, Jun Xu‡, Emre C. Sezer‡, Prachi Gauriar‡, and Ravishankar K. Iyer proposes a technique for non-control data attacks. Most memory corruption attacks and Internet worms follow a familiar pattern known as the control-data attack. Hence, many defensive techniques are designed to protect program control flow integrity. Although earlier work did suggest the existence of attacks that do not alter control flow, such attacks are generally believed to be rare against real-world software. The key contribution of this paper is to show that non-control-data attacks are realistic. The author demonstrates that many real-world applications, including FTP, SSH, Telnet, and HTTP servers, are vulnerable to such attacks. For each case, the generated attack results in a security compromise equivalent to that due to the control data attack exploiting the same security bug.

A variety of application data including user identity data, configuration data, user input data, and decision-making data are corrupted due to Non-control-data attacks. The success of these attacks and the variety of applications and target data suggest that potential attack patterns are diverse. Though attackers are currently focused on control-data attacks, but it is clear that when control flow protection techniques shut them down, they have incentives to study and employ non-control-data attacks. This emphasizes the importance of future research efforts to address this realistic threat.

David Brumley, Tzi-cker Chiueh, and Robert Johnson present the design and implementation of RICH (Run-time Integer CHecking), a tool for efficiently detecting integer-based attacks against C programs at run time. There will be a frequent programming error and attack in C integer bugs, when a variable value goes out of the range of the machine word used to materialize it, e.g. when assigning a large 32-bit int to a 16-bit short. We show that safe and unsafe integer operations in C can be captured by well-known sub-typing theory. The RICH compiler extension compiles C programs to object code that monitors its own execution to detect integer-based attacks. Here the author implemented RICH as an extension to the GCC compiler and tested it on several network servers and UNIX utilities. Inspite of the radical change in integer operations, the performance overhead of RICH is very low, averaging about 5%. As per results given by the author, RICH found two new integer bugs and caught all but one of the previously known bugs we tested. These results show that RICH is a useful and lightweight software testing tool and run-time defense mechanism. RICH may generate false positives when programmers use integer overflows deliberately and it can miss some integer bugs because it does not model certain C features.

III. PROPOSED WORK

This project deals with the analysis of software components. The technique is used to identify the unsafe components present in the system. The major role of project is to first collect the executable files that are present in the system. A profile is generated for storing all DLL files contained in the system. When a DLL is found to be unsafe it can identified, stop its execution, or delete the file permanently. Fig.1 demonstrates the procedure in detecting the unsafe components.

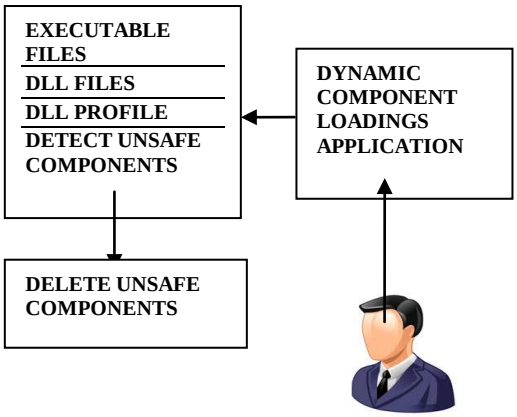

Fig. 1. Detection of Unsafe Components

Dynamic binary instrumentation is used to identify the performance of the component before its execution itself. Binary instrumentation code will detect the components behavior of the components contained in the system. It helps the system to avoid resolution failure or unsafe resolution. Thus the unsafe components are detected easily and removed. The removal of unsafe components may affect the execution of safe components in the system, hence this approach detects the safe components in the system and their performance is checked.

IV. MODULE DESCRIPTION

EXECUTABLE FILES MODULE:

In this module consider, in our operating system there are so many application software installed, that software are in executable format. What are the application is being installed in our computer, that executable file are displayed in this module. This module processes all running executable files. If we want to add new application then we select the specific application like notepad from list box in run option. After that the notepad exe file added to the list. You can view newly added file in that list. If you don't want to show that file from that list, you can stop by using stop option.

DLL FILES MODULE:

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.

This module shows list of DLL files, if one application is running, then number of DLL files are loading during the application loading, so we can show the number of DLL file names from this module. Microsoft Windows supports both types of target DLL specifications: full path and filename. For filename specifications, there exist Windows-specific mechanisms to resolve target DLLs. In particular, Microsoft Windows supports Side-by-Side Assembly and maintains Known DLLs to determine the target DLL full path directories without performing iterative directory searching

DLL PROFILE MODULE:

This module shows what DLL files are running for which application and profile of DLL. Dynamic analysis has been widely used to understand software behavior. We also adopt this approach for detailed analysis of component loading. Specifically, we 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: 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. It will show the following details like file version, file size, file modification date, and file description, product name, product version

DETECTION MODULE:

An unsafe component loading can cause serious security vulnerabilities in software. This module presents a dynamic analysis technique for detecting unsafe component loadings. To detect unsafe component resolutions, the technique first captures a sequence of system-level actions for dynamic loading during a program's execution. The technique uses dynamic binary instrumentation to generate the profile on its runtime execution. It then reconstructs the dynamic loading information from the profile offline and check safety conditions for each resolution. Because the technique only requires binary executables, it is robust and can be applied to analyze not only open source applications but also commercial off-the-shelf products. Alternatively, we could also detect an unsafe component loading during the program execution. However, we divide our analysis into two phases (i.e., the dynamic profile generation and the offline profile analysis) to reduce the performance overhead incurred during dynamic binary instrumentation. From this module we can detect unsafe files and delete that files.

V. CONCLUSION

Software component loading is done in two ways statically or dynamically. 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. It also helps to isolate software bugs as bug fixes of a shared library can be incorporated easily. Because of these advantages, dynamic loading is widely used in designing and implementing software.

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. However, popular software's such as Microsoft Windows and Linux can be tested with this dynamic analysis. The analysis can be in two phases (i.e., the dynamic profile generation and the offline profile analysis) to reduce the performance overhead incurred during dynamic binary instrumentation.

REFERENCES

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

[2] C. Cadar, V. Ganesh, P. M. Pawlowski, D. L. Dill, and D. R. Engler, "Exe: Automatically Generating Inputs of Death," Proc. 13th ACM Conf. Computer and Comm. Security, pp. 322-335, 2006.

[3] S. Chari, S. Halevi, and W. Venema, "Where Do You Want to Go Today? Escalating Privileges by Pathname Manipulation," Proc. Network and Distributed System Security Symp., Mar. 2010.

[4] S. Chen, J. Xu, E. C.Sezer, P. Gauriar, and R. K. Iyer, "Non-Control-Data Attacks Are Realistic Threats," Proc. 14th Conf. USENIX Security Symp., 2005.

[5] D. Dhurjati and V. Adve, "Backwards-Compatible Array Bounds Checking for C with Very Low Overhead," Proc. 28th Int'l Conf. Software Eng., pp. 162-171, 2006.

[6] C. Grier, S. T. King, and D. S. Wallach, "How I Learned to Stop Worrying and Love Plug-ins," Proc. Workshop Web 2.0 Security and Privacy, May 2009.

[7] C. Grier, S. Tang, and S. T. King, "Secure Web Browsing with the OP Web Browser," Proc. IEEE Symp. Security and Privacy, pp. 402-416, 2008.

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

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

[10]. D. Larochelle and D. Evans, "Statically Detecting Likely Buffer Overflow Vulnerabilities," Proc. 10th Conf. USENIX Security Symp., 2001.

[11]. D. Molnar, X. C. Li, and D. A. Wagner, "Dynamic Test Generation to Find Integer Bugs in x86 Binary Linux Programs," Proc. 18th Conf. USENIX Security Symp., pp. 67-82, 2009.

[12] P. Saxena, P. Poosankam, S. McCamant, and D. Song, "Loop-Extended Symbolic Execution on Binary Programs," Proc. 18th Int'l Symp. Software Testing and Analysis, pp. 225-236, 2009.

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

Geethanjalin. N received B. E. degree in Computer Science and Engineering at SNS College of Technology, Coimbatore in 2011. She received her M. E. degree in Software Engineering at SNS College of Technology, Coimbatore. She is currently working as assistant professor in SNS College of Technology. Her area of interests is Networks and Software Engineering.

Priyadarshini.S received B. E. degree in Computer Science and Engineering at Avinashilingam University, Coimbatore in 2011. She received her M. E. degree in Software Engineering at SNS College of Technology, Coimbatore. She is currently working as assistant professor in SNS College of Technology. Her area of interests is Networks and Software Engineering.

Professor Dr.S.Karthik is presently Professor & Dean in the Department of Computer Science & Engineering, SNS College of Technology, affiliated to Anna University- Coimbatore, Tamilnadu, India. He received the M.E degree from the Anna University Chennai and Ph.D degree from Anna University of Technology, Coimbatore. His research interests include network security, web services and wireless systems. In particular, he is currently working in a research group developing new Internet security architectures and active defense systems against DDoS attacks. Dr.S.Karthik published more than 35 papers in refereed international journals and 25 papers in conferences and has been involved many international conferences as Technical Chair and tutorial presenter. He is an active member of IEEE, ISTE, IAENG, IACSIT and Indian Computer Society.