ISSN: 2278 – 1323
International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)
Volume 1, Issue 10, December 2012

SURVEY ON DYNAMIC ANALYSIS TO DETECT VULNERABILITIES AND UNSAFE COMPONENT LOADINGS

Geethanjali.N[1], Maragatham.T[2], Dr. Karthik. S[3]

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. Dynamic loading is a mechanism by which a computer program can, at run time, load a library into memory, retrieve the addresses of functions and variables contained in the library, execute those functions or access those variables, and unload the library from memory. An effective dynamic analysis to detect vulnerable and unsafe dynamic component loadings is proposed. This work introduces the first automated technique to detect and analyze vulnerabilities and errors related to the dynamic component loading. This analysis has two phases: 1) Online Phase to apply dynamic binary instrumentation to collect runtime information on component loading , and 2) Offline Phase to analyze the collected information to detect vulnerable component loadings . The technique uses a set of practical tools for detecting 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 a mechanism by which a computer program can, at run time, load a library into memory, retrieve the addresses of functions and variables contained in the library, execute those functions or access those variables, and unload the library from memory. Unlike static linking and loadtime linking, this mechanism allows a computer program to startup in the absence of these libraries, to discover available libraries, and to potentially gain additional functionality.

Dynamic loading is most frequently used in implementing software plugins.For example, the Apache Web Server's `*.dso` "dynamic shared object" plugin files are libraries which are loaded at runtime with dynamic loading. Dynamic loading is also used 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.

Manuscript received Dec 21, 2012.

Geethanjali.N, *Department Of Computer Science and Engineering, SNS College Of Technology, Coimbatore, India, 9344332080*

Maragatham. T*, Department Of Computer Science and Engineering, SNS College Of Technology, Coimbatore, India,*

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

Not all systems support dynamic loading. UNIX-like operating systems such as Mac OS X, Linux, and Solaris provide dynamic loading with the C programming language "dl" library. The Windows operating system provides dynamic loading through the Windows API. Operating systems may provide mechanisms to protect system resources. For example, Microsoft Windows supports Windows Resource Protection (WRP) 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.

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 thery 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.

Loading the library is accomplished with `LoadLibrary` or `LoadLibraryEx` on Windows and with `dlopen` on UNIX-like operating systems. Extracting the contents of a dynamically loaded library is achieved with `GetProcAddress` on Windows and with `dlsym` on UNIX-like operating systems. Some software components utilize functionalities at runtime exported by other components such as shared libraries. 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.

II. RELATED WORK

a. Backwards compatible array bounds checking for c with very low overhead

The problem of enforcing correct usage of array and pointer references in C and C++ programs 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 of 5x-6x and 11x–12x in the two versions. The author describes a collection of techniques that dramatically reduce the overhead of this approach, by exploit- ing 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 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.

b. Dynamic Test Generation To Find Integer Bugs in x86 Binary Linux Programs

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 methods 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). 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.

c. How I Learned to Stop Worrying and Love Plugins.

The paper argues that browsers should be responsible for specifying and enforcing security policies for browser plugins. Browsers can significantly reduce the impact of plugin vulnerabilities and eliminate much of the risk posed by today’s plugin exploits, by enabling the browser to make security decisions on behalf of the plugin. 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 plugins have become the present tool on the Internet. The web applications have faced radical change due to new, feature-rich plugins. Consider YouTube works mostly behind the plugin Flash Player YouTube work – without the streaming video support added in Adobe Flash 7, YouTube would not have taken off. Unfortunately, plugins are riddled with security vulnerabilities and expose users to significant risk. In today’s browsers, Plugins 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 plugin, each plugin is responsible itself. An plugin API is used by the plugins to interact with the browser, such as the NPAPI, supported by the browser. The common functionality of Plugin APIs in browsers is that, the browser provides plugins with document (i.e., DOM) access, network connectivity and interaction with other browser components. Except for some document accesses, each plugin is responsible for restricting the use of this API by the plugin content as well as controlling access to the underlying system.

d. Non-Control-Data Attacks Are Realistic Threats

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.

e. RICH: Automatically Protecting Against Integer-Based Vulnerabilities

The authors 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. 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. This made us present a survey on different tools and techniques that is used to detect vulnerabilities and attacks in the system. To detect vulnerable and unsafe component loadings we have discussed some of the available papers to detect attacks bugs in the system. These references could also help to detect an unsafe component loading during the program execution. However, popular softwares 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. In future, the analysis may also include code coverage problem along with the dynamic analysis technique .

Acknowledgment

The authors would like to thank the Editor-in-Chief, the Associate Editor and anonymous Referees for their comments.

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 Plugins," 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 is currently pursuing her M. E. degree in Software Engineering at SNS College of Technology, Coimbatore. Her area of interests are Networks and Software Engineering.

Maragatham. T received B.E degree in Computer Science and Engineering at M. Kumarasamy College of Engineering Karur in 2004. She has received Master Degree in Engineering from Anna University Coimbatore in 2011. She published papers in two international journals. She has presented papers in national and international conferences.

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.