ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization) Vol.2, Special Issue 1, March 2014

Proceedings of International Conference On Global Innovations In Computing Technology (ICGICT'14)

Organized by

Department of CSE, JayShriram Group of Institutions, Tirupur, Tamilnadu, India on 6th & 7th March 2014

A Technique to Prevent Dynamic Unsafe Component Loadings

M.Malathi[1], Mrs.P.Sumathi[2], M.E., Dr.S.Chitra M.E., Ph.D[3]

2nd year M.E, Department of Computer Science and Engineering, Er.Perumal Manimekalai College of Engineering, Hosur, Tamilnadu, India[1]

AP, Department of Computer Science and Engineering, Er.Perumal Manimekalai College of Engineering, Hosur, Tamilnadu, India[2]

Prof., Department of Computer Science and Engineering, Er.Perumal Manimekalai College of Engineering, Hosur, Tamilnadu, India[3]

ABSTRACT: Dynamic loading of software components (e.g., libraries or modules) is a widely used mechanism for improved system modularity and flexibility. In general, an operating system or a runtime environment resolves the loading of a specifically named component by searching for its first occurrence in a sequence of directories determined at runtime. Correct component resolution is critical for reliable and secure software execution, however, programming mistakes may lead to unintended or even malicious components to be 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 started to receive considerable attention as their remote exploitation became realistic; it is now important to detect and fix these vulnerabilities.

I. 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. It also helps to isolate software bugs as bug fixes of a shared library can be easily incorporated. For these advantages, dynamic loading is widely used in designing and implementing software.A key step in dynamic loading is component resolution, i.e., how to locate 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. Operating systems may provide mechanisms to protect system resources, such as Windows Resource Protection (WRP) in Microsoft Windows Vista. However, these do not prevent loading of a malicious component located in a directory searched before the directory where the intended component resides.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization) Vol.2, Special Issue 1, March 2014

Proceedings of International Conference On Global Innovations In Computing Technology (ICGICT'14)

Organized by

Department of CSE, JayShriram Group of Institutions, Tirupur, Tamilnadu, India on 6th & 7th March 2014

The problem of 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. Here is an example attack scenario. An attacker sends an archive file containing a document for a vulnerable program (e.g., a Word document) and a malicious DLL to a victim. If the victim opens the document, the vulnerable program will load the malicious DLL and the host machine can be subverted in more detail potential remote code execution attack vectors exploiting vulnerable dynamic component loading.

II. EXISTING APPROACH

Existing system provides the first static binary analysis aiming at detecting all possible loading-related errors. The key challenge is how to scalably and precisely compute what components may be loaded at relevant program locations. Our main insight is that this information is often determined locally from the component loading call sites.

This motivates us to design a demand-driven analysis, working backward starting from the relevant call sites. In particular, for a given call site c, we first compute its context-sensitive executable slices, one for each execution context. Then we emulate the slices to obtain the set of components possibly loaded at c. This novel combination of slicing and emulation achieves good scalability and precision by avoiding expensive symbolic analysis. We implemented our technique and evaluated its effectiveness against the existing dynamic technique on nine popular Windows applications. Results show that our tool has better coverage and is precise—it is able to detect many more unsafe loadings. It is also scalable and finishes analyzing all nine applications in not possible.

III. PROPOSED FRAMEWORK

Dynamic loading of software components is a commonly used mechanism to achieve better flexibility and modularity in software. For an application's runtime safety, it is important for the application to load only its intended components. However, programming mistakes may lead to failures to load a component, or even worse, to load a malicious component.

Recent work has shown that these errors are both prevalent and severe, sometimes leading to remote code execution attacks. The work is based on dynamic analysis by monitoring and analyzing runtime component loadings. Although simple and effective in detecting real errors, it suffers from limited code coverage and may miss importantvulnerabilities. Thus, it is desirable to develop effective techniques to detect all possible unsafe component loadings automatically. Dynamic profile generation

In this phase,instrument runtime executions of the binary executable under analysis to capture a sequence of system-level actions for dynamic loading of components. In particular, collect three types of information during the instrumented program execution: 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 will be analyzed in the offline profile analysis phase.

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 (with names of the invoked functions and their arguments) can provide useful information on program execution. To capture system-level actions for 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, also collect its parameter information for detecting unsafe component resolutions. Specifically, the target component specification 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 setting, it is computed by operating systems at the higher-level and provided as parameters to the relevant system calls for dynamic

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization) Vol.2, Special Issue 1, March 2014

Proceedings of International Conference On Global Innovations In Computing Technology (ICGICT'14)

Organized by

Department of CSE, JayShriram Group of Institutions, Tirupur, Tamilnadu, India on 6th & 7th March 2014

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 component resolutions.

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

Process and thread identifiers, because the approach is based on system callinstrumentation, it is important to consider multi-threaded applications. If the target program uses multi-threads 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, capture process and thread identifiers along with the other information on instrumented system calls.

IV. OFFLINE PROFILE ANALYSIS

In this phase, extract each component loading from the profile and detect defects in the resolution of a target component and its dependent components .In the first step of this offline phase, extract each component loading from the profile. To this end, first group a sequence of actions in the profile by process and thread identifiers as the actions performed by different threads may be interleaved due to context switching. This grouping separates the sequences of dynamic loadings performed by different threads. Next, we divide the sequence for each thread into sub-sequences of actions, one for each distinct dynamic loading. This can be achieved by using the first invoked system call for dynamic loading (e.g., dlopen) as a delimiter. After this step, 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 dynamic loading procedure. Note that each group contains loading actions for both the target component and the load-time dependent components.
The analysis detects the two types of unsafe component resolution .Resolution failure and resolution hijackingwhich are directly derived from the definition of each unsafe component resolution, for each component loading.

Resolution failure of target component To detect failed resolution of a target component, simply check the number of image loads and failed resolutions during the dynamic loading procedure. In particular, if no image is loaded and its resolution is failed, report it as a resolution failure.
Resolution hijacking of target component Lines 10–15 describe how to detect resolution hijacking of a target component. First 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 the file name is used, we retrieve the resolved path of the target component by retrieving the first element of a list of image loads in the dynamic loading procedure.

Detect unsafe components

1. To detect unsafe components,we first capture a sequence of system level actions during a program execution.

2. We use binary instrumentation to generate its runtime profile,then we check safety conditions for each resolution.

3. Binaryexecutables is robust,not only used for open source but also used for commercial off-the-shelf products

Background on DLL loading procedure

1) Target DLL resolution: Microsoft Windows supports the two aforementioned types of target DLL specifications: fullpath and filename. For the filename specification, 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 fullpath directory without iterating through the directories .

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization) Vol.2, Special Issue 1, March 2014

Proceedings of International Conference On Global Innovations In Computing Technology (ICGICT'14)

Organized by

Department of CSE, JayShriram Group of Institutions, Tirupur, Tamilnadu, India on 6th & 7th March 2014

V. CONCLUSION

In this paper, it described a dynamic analysis technique to detect unsafe dynamic component loadings. 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 resolution hijackings. To evaluate the technique, here implemented a tool to detect unsafe DLL loadings on Microsoft Windows.Evaluation shows that unsafe DLL loading is prevalent and can lead to serious threats. In particular, the tool detected more than 1,700 unsafe DLL loadings in popular software developed by major vendors. It also discovered potential remote code execution attacks exploiting the detected unsafe DLL loadings

VI. FUTURE WORK

We plan to analyze unsafe component loadings in Unix-like operating systems. As we mentioned in Section VI, unsafe component loading is a general security concern, and our approach is general and can also be applied to analyze applications on these systems. We plan to evaluate the prevalence and severity of unsafe component loading for these other important operating systems. Second, we plan to develop static binary analysis techniques to detect unsafe component loadings. Although our dynamic analysis is effective, it may suffer from the standard limitation of dynamic analysis, namely the code coverage problem. We plan to develop sound, practical static analysis techniques to complement the dynamic analysis introduced here.

REFERENCES

[1] About the security content of Safari 3.1.2 for Windows. http: //support.apple.com/kb/HT2092.

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

[3] About Windows Resource Protection.http://www.dependencywalker.com/help/html/dependency types.htm.

[4] PeriklisAkritidis, CristianCadar, CostinRaiciu, Manuel Costa, and Miguel Castro. Preventing memory error exploits with WIT. In Proceedings of the 29th IEEE Symposium on Security and Privacy, 2008.

[5] David Brumley, Dawn Xiaodong Song, Tzi-ckerChiueh, Rob Johnson, and Huijia Lin. RICH: Automatically protecting against integer-based vulnerabilities.In Proceedings of the 14th Annual Network and Distributed System Security Symposium, 2007.

[6] CristianCadar, Daniel Dunbar, and Dawson R. Engler. KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs. In Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation, pages 209–224, 2008.

[7]CristianCadar,VijayGanesh,PeterM.Pawlowski,DavidL.Dill,andDawsonR.Engler.EXE:Automaticallygeneratinginputsofdeath.InProceedingsofthe13thACMconferenceonComputerandcommunicationssecurity,pages322–335,NewYork,NY,USA,2006.ACM.

[8]ShuoChen,JunXu,EmreC.Sezer,PrachiGauriar,andRavishankarK.Iyer.Non-control-dataattacksarerealisticthreats.InProceedingsofthe14thconferenceonUSENIXSecuritySymposium,2005.

[9]ManuelCosta,MiguelCastro,LidongZhou,LintaoZhang,andMarcusPeinado.Bouncer:Securingsoftwarebyblockingbadinput.InProceedingsofthe21thACMSIGOPSSymposiumonOperatingsystemsprinciples,2007.

[10]ManuelCosta,JonCrowcroft,MiguelCastro,AntonyRow-stron,LidongZhou,LintaoZhang,andPaulBarham.Vigilante:End-to-endcontainmentofinternetworms.InProceedingsofthe20thACMsymposiumonOperatingsystemsprinciples,2005.

[11]CVE-2009-0415.http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0415.

[12]DinakarDhurjatiandVikramAdve.Backwards-compatiblearrayboundscheckingforCwithverylowoverhead.InProceedingsofthe28thinternationalconferenceonSoftwareengineering,2006.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization) Vol.2, Special Issue 1, March 2014

Proceedings of International Conference On Global Innovations In Computing Technology (ICGICT'14)

Organized by

Department of CSE, JayShriram Group of Institutions, Tirupur, Tamilnadu, India on 6th & 7th March 2014

[13]dlopenmanpttp://linux.die.net/man/3/dlopen.

[14]Dynamic-LinkLibrarySearchOrder.http://msdn.microsoft.com/en-us/library/ms682586(VS.85).aspx.