ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Issue 8, August 2016

Using Sandbox to Prevent the Malicious Code

P. G. V. Suresh Kumar[1] Seelam Sowjanya[2]

Associate Professor, Department of IT & SC, AAiT, Addis Ababa University, Addis Ababa, Ethiopia[1]

Assistant Professor, Department of CSE, St. Mary's Engineering Collage, Chebrolu, Guntur, India [2]

ABSTRACT: Sandboxing refers to mechanism of having a confined environment for untrusted programs which can cause potential damage if their boundaries are not defined properly. Example of such kind of programs are client browser based environments such as java applets, flash player, Microsoft silver light etc. Since these application run on clients computer/browser, can cause potential damage if their boundary of allowed resources are not defined or they can be used by hackers to hack the client systems. Apart from this sandboxing provide good security to prevent the malicious codes from trusted code. In this paper, we explore the security of code by the use of sandboxing method and we also proposed some concept for enhancing the security by access controls rules for the codes. Our idea is to distribute the access related roles to each type of user.

KEYWORDS: Sandboxing, Access List.

I. INTRODUCTION

Sandboxing is a generic security term that applies to any environment that reduces the privileges under which an application runs**.** Sandbox technique, which is a proactive approach to stop the malicious actions of internet code before any malicious activity can occur. The Sandbox application sits between the malicious code and the operating system and intercepts any system calls that code made. It checks its policy database to determine just what the kind of code is allowed to perform certain kind of task and immediately block any unauthorized activity [6]. This section gives an overview of Windows Presentation Foundation sandbox created for Web Browser Applications.

This Paper gives an overview of the technologies used to create the WBA security sandbox, summarizes the sandbox feature set, and provides a rationale for the tradeoffs made during development [9]. It is intended to:

- Explain the types of operations possible within Web Browser Applications.
- Enable experienced developers to understand the design of the security sandbox.
- Help developers understand what is disallowed in the sandbox "by design" and what might be a bug for which they can provide feedback to Microsoft.

The truth is that when you type a web site's URL in your browser's address bar and push Go button to surf that site, you do not go to that web site, but the site actually come to you. Fancy graphics, flashy animations and all kinds of auto executable codes that includes ActiveX controls and Java Applets gets download on your computer which are capable to perform practically any destructive any task in your computer with or without your knowledge. This is a study of creating restricted environments for programs that are considered unsafe. UNIX itself offers many restrictions to what process can and cannot do. Where additional security is required (e.g. ftp daemon) the CHROOT system call is used to restrict access to a specific area of the file system. In this study, we examine how these access restrictions can be used to create a safe execution environment.

II. RELATED WORK

A lot of work has been done on software sandboxing including Java virtual machines, and FreeBSD. Most software methods of application sandboxing view the entire application being sandboxed as a monolithic module and do not allow access control of system resources between application subcomponents. Over the years there have been numerous proposals for systems that impose discretionary access controls on programs. These systems can be roughly placed in three categories, in increasing order of complexity for the execution environment.

ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Issue 8, August 2016

Systems that trust programs: Programs that have been vetted are considered trusted, while the rest are given only limited access. Examples include Microsoft Active X controls and the system presented by Lai et. al. [3] these systems assume that all bugs or vulnerabilities can be detected before deployment. This assumption has been demonstrated time to time again as utopian.

Regulate file access: File access is considered a key capability in most systems since it involves the long term memory of the system. Unauthorized modifications to files can threaten the integrity of the system itself or the data that is stored in it.

Full Access Control: All requests made by the application are passed through a discretionary access control mechanism that enforces policy. The checks may be at the operating system call level as in Janus [1] and SubOS [4], or at the library call level [5].Virtual machines such as the Java VM and VMware also provide a restricted environment in which programs may operate. We focus on related work that has been targeted towards any of these two aspects of browser security:

- Protecting the host system state from browser (or browser plug-in) originated attacks.
- Protecting the browser state from malicious browser plug-ins that may attack the host browser directly or via the OS kernel [9].

The Tahoma web browsing system uses the Xen virtual machine monitor for isolating browser instances and its associated policies. In the Tahoma system, a manifest defines the browser policies such as URLs the browser can access and other web application characteristics. A trusted computing base called “Browser Operating System (BOS)” executes the management functions for Xen. The Tahoma BOS has a kernel which manages browser instances and storage for the system. A network proxy enforces network access policies for web applications based on the policies in the manifest. Some of the key differences with Tahoma are that our approach does not require graphics or device virtualization additionally our goal is to ensure sandboxing of plug-ins within the browser to allow for differentiated and controlled access to the platform from such plug-ins. Feather-weight Virtual Machine (FVM) is an OS-level software virtualization architecture. FVM uses namespace virtualization to rename system resources at the OS system call interface. The OS based virtualization approaches are themselves susceptible to privileged malicious code that can tamper or unhook the virtualization hooks within the operating system. Such tampering affects all the sandboxes running on the host. Wang et. al. [10, 11] presents a system utilizing a honeypot within a virtual machine (VM). Within the VM, the tool creates an instance of the Microsoft Internet Explorer, navigates to a specific URL, and waits for a few minutes. Any changes to the VM’s file system and registry are flagged. If a change to the file system outside of the temporary directory of the browser is detected, the site is classified as malicious. The system then shuts down the potentially infected VM and starts a clean one to analyze the next URL. A general problem of this approach is that it does not scale well. In our approach, the hypervisor can also sandbox any unauthorized code detected on the host machine such that malware will not be able to tamper with sandboxing services. Ozone HIPS is a self protection tool designed to protect the operating system by disallowing tampering with system resources and/or disallow loading of unauthorized code. Ozone uses two protection layers called memory protection layer to guard against attacks that hijack execution by corrupting memory and a process protection layer that sandboxes process to continuously monitor their access to the important system processes. It uses address space randomization to prevent attacks that depend on static address exploits from being successful. In the process protection layer, Ozone enforces access control by sandboxing kernel service calls. Sandboxes in Ozone are described by a policy text file stored on the host. Our approach is similar to the Ozone approach but differs in the architecture for memory protection– we use a higher privilege hardware mode for memory protection. Our approach also does not require address space randomization for the protected processes since memory-based attacks are mitigated by strict separation of pages referenced from the process page tables.

III. MOTIVATION

Creating sandboxing application is a very difficult process:-one has to determine what files or services to place within the sandbox to facilitate the execution of application. When a typical user receives a Microsoft Word file and would like to see what is inside but is afraid of what will happen to his workstation if the file contains a virus. One solution would be to place the program (Word) along with the suspect file in a controlled environment (which from now on we

ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Issue 8, August 2016

will refer to as sandbox) and open it there. If the file is infected, the effects of the virus will be localized, but not entirely eliminated [7].

The sandbox must contain all the files needed for executing the application. Gathering a list of these files is a non trivial task. Applications depend in utilities such as lpr or mail, and on shared libraries, loadable modules, configuration files, and the operating system files required by various C library functions (IP service names, localized messages, time zone specifications, etc.). A key consideration is to make sure that the program does not escape from the sandbox, and that the sandbox system never assigns the program greater privileges than it would have if it ran outside the sandbox. Thus, the sandbox must disable access to setuid programs, or not allow them to be executed with permissions other than the ones given to the user running the application.

More sophisticated systems like Janus [1] support centralized policies with fine grained control over the resources that the process may use (e.g. file descriptors, memory, file system space etc.) Our approach dynamically evaluates the requirements of a given program, creating a sandbox specification that will not affect its operation. In doing so we strive to balance what we would like to control against the effort in specifying these restrictions.

IV. PROBLEM IN EXISTING SYSTEM

1. In sandboxing, files may not be added to the access list after the sandbox is running [7].
2. Sandbox software cannot tell you which virus type you have in your computer neither it can remove the virus that had sneaked in. It can only tell you have some suspected, malicious piece of code that can do unwanted things in your computer & protect you from it if you have properly set policies [6].
3. Sometimes task does not give appropriate result due to sandboxing security. For e.g.-sometime browser cannot open files due to sandbox security [9].
4. Do not prevent malicious application from[8]:
   - Accessing network and messing with other machines.
   - Trying to crash host OS.

V. PROPOSED WORK

Our objective for sandboxing applications is based on the following steps:

1. To create a novel method to addressing malicious software by controlling its ability to execute.
2. To specify the resources that an application can access and the type of access that the application is granted. If the application tries to access restricted resources, the sandbox interferes and rejects the access attempt.
3. Run the application with known benign input to create access logs specifying its file access behavior.
4. Use the logs to create an access list that can be morphed into a typical CHROOT environment (like the one used by ftpd). The user can augment this list based on the particular requirements of the application.
5. Create the sandbox as a CHROOT environment based on the access list.

In future we use sandboxing technique to block un-trusted code that change the memory address which is changed by the Root kit, Polymorphic virus and Buffer over flow attack, Moreover, in the current system, files may not be added to the access list after the sandbox is running. We plan to investigate the possibility of asking the user or permission when trying to access a file that is not in the access list.

VI. CONCLUSION

Creating an Internet-facing platform that provides sufficient utility and is secure by default is a difficult task. The sandbox mechanism provides security or defense against the introduction of new and potentially malicious code. This greatly reduces the threat posed by virus, Trojan horses, and even malicious insiders.

ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Issue 8, August 2016

REFERENCES

[1] A. D. Alexandrov, M. Ibel, K. E. Schauser, and Chris J. Scheiman. Extending the Operating System at the User Level: the Ufo Global File System, *USENIX Annual Technical Conference*, Anaheim, California, January 1997.

[2] Goldberg, Ian, David Wagner, Randi Thomas and Eric A. Brewer. A Secure Environment for Untrusted Helper Applications, *USENIX Security Symposium*, 1996.

[3] Ioannidis, Sotiris, A. D. Keromytis, S. Bellovin and J. M. Smith, Implementing a Distributed Firewall, *7th ACM Conference on Computer Communications Security*, November 2000.

[4] Ko, Calvin, G. Fink and K. Levitt, Automated detection of vulnerabilities in privileged programs by execution monitoring, *Proceedings of the 10th Annual Computer Security Applications Conference*, Orlando, FL1994.

[5] N. Lai and T.E. Gray, Strengthening discretionary access controls to inhibit Trojan horses and computer viruses, *Proceedings of the 1988 USENIX Summer Symposium*, pp. 275-286, June 1988.