International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 08 Issue: 03 | Mar 2021 www.irjet.net p-ISSN: 2395-0072

Secure Isolation and Migration of Untrusted Legacy Applications

Dr. Khalid Ahmed Ibrahim

Associate Professor, Faculty of CS & IT, Karary University, Omdurman, Sudan

---

Abstract - To address the system security and availability issues, we have developed peas and pods. Pea provides a least privilege environment that can restrict processes to the minimal subset of system resources needed to run. Pod provides a group of processes and associated users with a consistent, machine-independent virtualized environment. This mechanism allows system administrators the flexibility to patch their operating systems immediately without worrying over potential loss of data or needing to schedule system downtime. We have implemented peas and pods in Linux without requiring any application or operating system kernel changes. Our measurements on real world desktop and server applications demonstrate that peas and pods impose little overhead

Key Words: Security, Peas and Pods, System Resource, Operating System, Schedule, Kernel

1.INTRODUCTION

Security problems can wreak havoc on an organization's computing infrastructure. To prevent this, software vendors frequently release patches that can be applied to address security issues that have been discovered. However, software patches need to be applied to be effective. It is not uncommon for systems to continue running unpatched applications long after a security exploit has become wellknown . This is especially true of the growing number of server appliances intended for very low-maintenance operation by less skilled users. Furthermore, once a patch has been released, exploits of unpatched applications based on reverse engineering the patch now occur as quickly as a month later whereas such exploits took closer to a year just a couple years ago .

Software updates to existing applications may not address security problems that result from users accidentally downloading and executing malicious code. Recently a security hole was discovered in a popular mp3 player that could result in arbitrary code being executed if a user played a maliciously constructed mp3. If the mp3 player were run within a simple sandbox that limited the player to one's collection of mp3s, the damage the malicious code could accomplish would be severely limited. Over the years, complex services like Sendmail have similarly been exploited to allow malicious code to be run within its context. Since Sendmail runs with privilege, the malicious code also runs with privilege. A sandbox can be used to protect an entire machine from a faulty service, such as Sendmail. However, these services don't run by themselves, but also depend on other aspects of the machine, such as programs a user might want to call from a Procmail script to filter their mail. Consequently, one might end up including the entire machine within the sandbox. Since common sandboxes simply provides a single namespace, they don't provide good security solutions for the complex services in use today.

Furthermore, even when software updates are applied to address security issues, they commonly result in system services being unavailable. Patching an operating system can result in the entire system having to be down for some period of time. If a system administrator chooses to fix an operating system security problem immediately, he risks upsetting his users because of loss of data. Therefore, a system administrator must schedule downtime in advance and in cooperation with all the users, leaving the computer vulnerable until repaired. If the operating system is patched successfully, the system downtime may be limited to just a few minutes during the reboot. If the patch is not successful, downtime can extend for many hours while the problem is diagnosed and a solution is found. For systems that need to provide a high degree of availability, downtime due to security-related issues is not only inconvenient but costly as well. While application servers can sometimes mirror application state between servers and allow an application to continue even when one server has to be taken down, they only work in specific situations. For instance, a regular user's desktop can not be mirrored between servers. Even for applications that can mirror their data, the application has to be designed to interface with the mirroring architecture, resulting in application specific solutions that are difficult to generalize. We introduce Pea-Pods to provide a solution to these security problems.

Pea-Pods provide two key abstractions, peas (Protection and Encapsulation Abstraction) and pods (PrOcess Domain). A pod is a lightweight migratable virtual execution environment that looks just like the underlying operating system environment. A pea is a least privilege environment within a pod that allows access to a subset of processes and resources in the pod. In tandem, peas and pods decouple process execution from the underlying operating system to provide transparent, secure isolation and migration of untrusted applications. Pea-Pods can isolate untrusted applications within sandboxes, preventing them from causing harm to the underlying system or other applications if they are compromised.

Pea-Pods achieve these goals through three distinguishing characteristics.

First, a pod provides a consistent private virtual namespace that gives all processes within it the same virtualized view of the system. This virtualized view isolates sandboxed processes from the underlying system by associating virtual identifiers with operating system resources and only allowing access to resources that are made available within the virtualized namespace. This isolation mechanism provides a simple way to control what operating system resources are accessible to a group of processes. Similarly, it allows a pod to define a complete set of users which can be distinct from those supported by the underlying system.

Second, a pea provides a least privilege encapsulation layer within a pod that can limit certain processes from interacting with other processes and accessing file system and network resources. This is effective for preventing compromised applications from attacking other processes and resources of the system. We provide intuitive tools to easily and dynamically create Pea-Pods tailored for individual applications or groups of applications.

Third, Pea-Pod virtualization is integrated with checkpoint-restart mechanism that decouples processes from dependencies on the underlying system and maintains process state semantics to enable processes to be migrated across different machines. The checkpoint-restart mechanism employs an intermediate format for saving the state associated with processes and Pea-Pod virtualization. This format provides a high degree of portability to support process migration across machines that are running operating systems that differ in the security and maintenance patches applied. It also enables application services to be checkpointed on a system and restarted after the underlying operating system is upgraded and the system is restarted. We have implemented Pea-Pods in a prototype system as a loadable Linux kernel module. We have used this prototype to securely isolate and migrate a wide range of unmodified legacy and network applications. We measure the performance and demonstrate the utility of Pea-Pods across multiple systems running different Linux 2.4 kernel versions using three real-world application scenarios, including a full KDE desktop environment with a suite of desktop applications, an Apache/MySQL web server and data base server environment, and a Sendmail/Procmail e-mail processing environment. Our performance results show that Pea-Pods can provide secure isolation and migration functionality on real world applications with low overhead.

2. MATERIALS AND METHODOLOGY

Mainly this project is the combination of Operating System Concept and Network Security Here we need xml parser and little j2ee knowledge to know how to deploy a component .Ant is the tool to built the xml tree for target location.Linux environment we prefer to use .With this we can upgrade the system without restart .Process migration can done using file transfer Protocol and sand box creation is done using java and checkpointing can be done using c.Pea-pod layer is mirror image of underlying operating system in this layer kernel versions are deployed in it.Any upgradation can be done by shifting the currently running Processes to the near by system and activate the newer version there by no change in the Underlying Kernel version.Hence upgradation without restart .

How it works :

It work like this,the applications that are running in out-dated versions are called legacy untrusted applications we are going to securely migrate this to latest version that is running in the nearby computer, by using digital signature to each processes that are currently running in the older versions.After updating our system,the processes is once again remigrated to the same system .This mechanism provides high uptime without needing to schedule system downtime for upgradation.Main module is the Checkpoint restart mechanism , which means that after migration we restart the process from the point we have left,not

from the beginning .Main aim of this project is to upgrade the system without restarting thus providing high uptime rather the allocating system downtime for the upgradation .

3. COMPARISON WITH EXISTING SYSTEMS

Existing system consisting of migration without using digital signature, it is disadvantage Since it is not secure .Existing system does not support any larger sandbox migration hence this software plays a vital role in larger organization where system downtime results in unavailability of services and loss of data. Existing system contains more number of lines of coding hence it results in large overhead. Peapod layer construction is only 50 lines of coding hence it results in lower overhead and performance of the system up gradation technique is improved.Isolation of processes and allocating of processes ID is difficult and confusing.This software uses private virtual name space hence it is easier to isolate processes and avoid confusion of allocating processes ID after migration

4. CONCLUSION

The Pea-Pod system provides an operating system virtualization layer that decouples process execution from the underlying operating system. The virtualization layer supports two key abstractions for encapsulating processes, peas and pods. Pods provide lightweight sandboxes that mirror the underlying operating system environment, and peas provide fine-grain least privilege environments within pods. Together, peas and pods can isolate untrusted applications within sandboxes, preventing them from being used to attack the underlying host system or other applications even if they are compromised. The Pea-Pod sandboxes can be transparently migrated across machines running different operating system kernel versions. This enables security patches to be applied to operating systems in a timely manner with minimal impact on the availability of sandboxed application services. Pea-Pod secure isolation and migration functionality is achieved without any changes to applications or operating system kernels. We have implemented Pea-Pods in a Linux prototype and demonstrated how peas and pods can be used to improve computer security and application availability for a range of applications, including e-mail delivery, web servers and databases, and desktop computing. Our results show that Pea-Pods can provide easily configurable, secure migratable sandboxes that can run a wide range of desktop and server Linux applications in least privilege environments with low overhead.

REFERENCES

[1].WWW.PeaPod.org

[2]. WWW.Checkpointing .org

[3]. WWW.TrustedComputing.Org

[4]. Y. Artsy, Y. Chang, and R. Finkel. Interprocess communication in charlotte. I*EEE Software*, pages 22–28, Jan 1987.

[5] A. Baratloo, N. Singh, and T. Tsai. Transparent Run-Time Defense Against Stack smashing Attacks. In *Proceedings of the USENIX Annual Technical Conference*, 2000.Honolulu, Hawaii, Apr. 1996.

[6] R. Rashid and G. Robertson. Accent: A communication oriented network operating system kernel. In *Proceedings of the 8th Symposium on Operating System Principles*, pages 64–75, Dec 1984.

[7] E. Rescorla. Security holes... Who cares? In *Proceedings of the 12th USENIX Security Conference*, Washington, D.C., Aug. 2003.

[8] M. Rozier, V. Abrossimov, F. Armand, I. Boule, M. Gien,M. Guillemont, F. Herrman, C. Kaiser, S. Langlois, P. L´eonard, andW. Neuhauser. Overview of the Chorus distributed operating system. In *Workshop on Micro-Kernels and Other Kernel Architectures*, pages 39–70, Seattle WA(USA), 1992.

[9] P. Smith and N. C. Hutchinson. Heterogeneous process migration:The Tui system. *Software – Practice and Experience*,28(6):611–639, 1998.

[10] R. Spencer, S. Smalley, P. Loscocco, M. Hibler, D. Andersen,and J. Lepreau. The Flask Security Architecture: System Support for Diverse Security Policies. In *Proc. of theEighth USENIX Security Symposium*, Aug. 1999

[11] A. Whitaker, M. Shaw, and S. D. Gribble. Scale and Performance in the Denali Isolation Kernel. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI 2002)*, Boston, MA, Dec. 2002.

[12] E. Zadok and J. Nieh. FiST: A Language for Stackable File Systems. In *Proceedings of the AnnualUSENIX technicalConference*, pages 55–70, June 2000.

FLOWCHART