Volume 3, Issue 6, June 2013 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

# Inclusion of Facts Collateral Issues with Context Switching Semaphore Using Self Healing Tactics

**S.MEENATCHI**
School of Information Technology and Engineering,
*VIT, UNIVERSITY,*
*Vellore, Tamil nadu., India*

**C.NAVANEETHAN**
School of Computing Science and Engineering,
*VIT,UNIVERSITY,*
*Chennai, Tamil nadu, India.*

***Abstract— The key purpose of a self healing scheme is to distribute and make safe the data of any system at the same time. "Self-healing" techniques are reliable computing methodology. Specially self-healing systems have to think for itself without the user input, also it should able to boot up and backup systems. Though, sharing and protection are two opposing goals. Protection programs might be altogether cut off from each other by executing them on separate non-networked computers, however, this precludes sharing.***

***Keywords- Self-healing; Semaphore; Security***

## I. INTRODUCTION

Self-healing mechanisms are complement approaches that stop attacks from succeeding by preventing the insertion of code, transfer of control to injected code, or mistreatment of existing code. Approaches to repeatedly protecting software systems have typically focused on ways to proactively or at run time care for an application from hit. These proactive approaches comprise writing the system in a "safe" language, connecting the system with "safe" libraries, switching the program through artificial diversity, or compiling the program with stack integrity check up. The technique of program shepherding validates branch instructions to prevent transfer of control to injected code and to make sure that calls into local libraries create from applicable sources. Control Flow Integrity (CFI), observing with the intention of high-level programming frequently assumes properties of control flow that is not enforced at the machine level. The use of CFI enables the well-organized completion of a software Outline call stack with muscular protection guarantees. Though, such techniques generally spotlight on integrity security at the expense of availability. Control flow is often degraded because input is finally integrated into part of an instructions opcode, set as a jump target, or forms part of an argument to a sensitive system call.

## II. SELF HEALING SYSTEMS

### *A. Self Healing Approach*

Self-healing is a loom to detect inappropriate operations of software applications, connections and business processes, and then to start corrective action without disturbing users. Healing systems that require human intervention or intervention of an agent external to the system can be categorized as assisted-healing systems. The key focus or complementary idea as compared to dependable systems is that a self-healing system should improve from the abnormal state and return to the healthy or normal state and function as it was prior to trouble. Some scholars treat self-healing systems as an independent one while others view as a subclass of traditional fault tolerant computing systems. The system watches itself for indication of irregular actions. When such actions are found, the system enters a self-diagnosis mode that aims to identify the mistake and take out as much information as possible with respect to its reason, symptoms, and collision on the system. The system tries to settle in itself by generating candidate fixes, which are tested to find the greatest target state. Self-healing systems can carry decision making in a huge method for managerial and organizational situations. Several of the decision support systems (DSS) offer inactive forms of decision sustain, where the decision-making process depends upon the user's initiative. Such active involvement is especially needed in complex decision-making environments.

### *B. Architecture of (S-H)*

The term "self" in self-healing architecture is referred to the action or response initiated automatically within the system. A general architecture of a self-healing system is shown in Fig. 1.

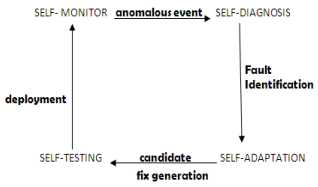


Fig. 1 A general architecture of a self-healing system

 

*C. Self Healing method*

The effective remediation strategies include failure-oblivious computing, error virtualization, rollback of memory updates, and data-structure repair. These approaches may cause a semantically incorrect continuation of execution attempts to address this difficulty by exploring semantically safe alterations of the programs environment. The method is later introduced in a modified form as failure-unaware computing, since the program code is broadly rewritten to comprise the necessary checks for every memory access, the system incurs overheads for a variety of different applications. Data-structure Repair is the most critical concerns with getting better from software faults and weakness exploits is ensuring the reliability and accuracy of program data and state.

*D. Why Self Healing*

The software notoriously buggy and crash-prone is despite considerable work in the fault tolerance and reliability. The current approach to ensuring the security and availability of software consists of a mix of different techniques:

*1) Proactive techniques*: Seek to make the code as dependable as possible, through a combination of safe languages, libraries and compilers, code analysis tools, formal methods and development methodologies.

2) *Debugging techniques*: Aim to make post-fault analysis and recovery as easy as possible for the programmer that is responsible for producing a fix.

3) *Runtime protection techniques*: Try to detect the fault using some type of fault isolation, which address specific types of faults or security vulnerabilities.

4) *Containment techniques*: Seek to minimize the scope of a successful exploit by separating the process from the rest of the system, *e.g.*, all the way through the use of virtual machine.

5) *Byzantine fault-tolerance and quorum techniques*: Rely on redundancy and diversity to create reliable systems out of unreliable components.

*E. Elements*

In the Self-healing process model, there are different categories of aspects to the self-healing system,

1) *Fault model*: Self-healing systems include the view of dependable computing is called a fault model must be specified for any fault tolerant system. The fault model answers the question of what faults the system is to stand for. Self-healing systems contain a fault model in terms of what injuries (faults), which are predictable and able to self-heal.

2) *Fault duration*: Faults can be stable, irregular or temporary due to an environmental circumstance. It is important to state the fault duration assumption of a self-healing approach to understand what situations it addresses.

3) *Fault manifestation*: The severity of the fault expression, it affects the system in the lack of a self-healing response. The faults cause instant system crashes, but, many faults cause less horrible penalty, such as system slow-down due to extreme CPU loads, whipping due to memory hierarchy overloads, resource leakage, file system excess, and so on.

4) *Fault source*: Thee source of faults can affect self-healing strategies due to accomplishment of defects, necessities defects, operational mistakes, and etc. Self-healing software is designed only to withstand hardware failure such as loss of memory or hardware failure and not software failures.

5) *Effect*: The effect of a failure is the range of the component that is compromised by that fault. Various self-healing mechanisms are probably appropriate depending on the effect of the failures and hence, the granularity of healing events.

6) *Fault profile expectations*: The origin of the fault is the profile of fault occurrences that is predictable. It considered for self-healing might be only expected faults that is based on design analysis or faults that are unexpected. Additionally, faults might be random and independent, might be correlated in space or time, or might even be intentional due to malicious intent.

*F. Traditional Methods of Security*

The traditional methods can overcome only the effects of passive threats and not the active threats for the authenticate users. They reduce user-friendliness and also, the amount of OS resources required to provide high protection. Different protocol architectures are used for providing safety for each and every layer of the OSI model and it may not be broad. Alternately, in this work, the information is allowed to flow freely through the fetch and decode cycles while an access or authentication is made only between the decode and execute cycle before the data is permanently written into the memory by the user (if authenticated). This is shown in Fig. 2.

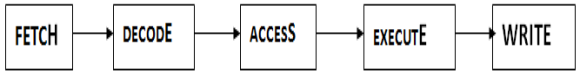


Fig. 2 Information Flow

*G. Proposed features of Security issues*

The proposed hardware is shown in Figure 3. The features of the proposed hardware are that it is (i) PCI compliant and (ii) Mounted in a single chip

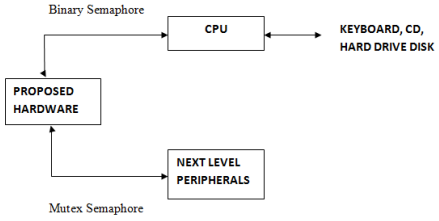


Fig. 3 Proposed hardware

 

1) *Robustness*: It provides defence against vulnerabilities with few false positives or false negatives.
2) *Flexible*: It adapts easily to cover the continuously evolving threats.
3) *End-to-End*: The security policy flows throughout all the seven layers of the OSI model.
4) *Scalable*: It can co-exist with the existing circuitry without any modification.

## III. METHODOLOGY

*A. Semaphores and Resource Sharing*

A semaphore is a protected variable or an abstract data type which restricts the access to shared resources such as shared memory in a multiprogramming environment. It is a primitive synchronization mechanism for sharing CPU time and resources. It is a traditional solution to avoid race conditions.

*B. Operation of Semaphore*

The "value" of a semaphore is the number of units of resources which are available. To keep away from busy-waiting, a semaphore has an connected queue of processes (usually First in First Out). If a process performs a "A" operation on a semaphore which has the value zero, the process is added to the semaphores queue. When another process increments the semaphore by performing a "B" procedure, and there are processes on the queue, one of them is detached from the queue and continues the operation.

*C. Binary Semaphore*

In binary semaphore, if there is only one source, the semaphore takes value "1" or "0".This is shown in Fig. 4.

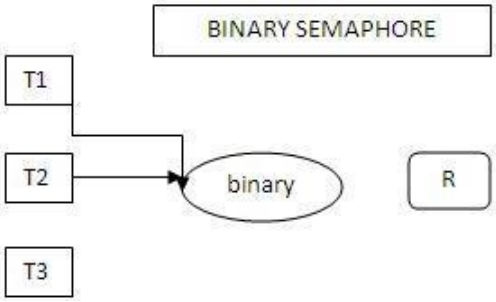


Fig. 4 Binary Semaphore

Suppose a "A" operation busy-waits (uses its turn to do nothing) or maybe sleeps (tells the system not to give it a turn) until a resource is available, where upon it immediately claims one. Now, let "B" be the operation that simply makes a resource available again after the process has completed using it. The "B" and "A" operations have to be unique, i.e., no process could be preempted in the heart of one of those operations to run a different operation on the identical semaphore. When a semaphore is being used, it takes value "0" and when it takes the value "1", the process directly starts execution without waiting.

*D. Counting Semaphore*

The counting semaphore concept can be extended with the ability of claiming or returning more than one unit from the semaphore. When several resources are to be distributed by several operations, such a semaphore is used. All the resources must be of the same type. This is shown in Fig. 5.

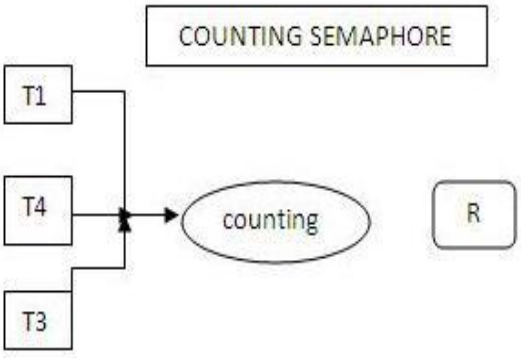


Fig. 5 Counting Semaphore

In this, the initial value of semaphore is set equal to number of resources available. As the semaphores are being used, the value keeps decrementing. As the semaphores are being released, after use, its value keeps incrementing. "Zero" value refers to empty semaphore.

*E. Mutex Semaphore*

A mutex is a binary semaphore with extra features like ownership or priority inversion protection. Mutexess are meant to be used for mutual exclusion only. This is shown in Figure 6.

 

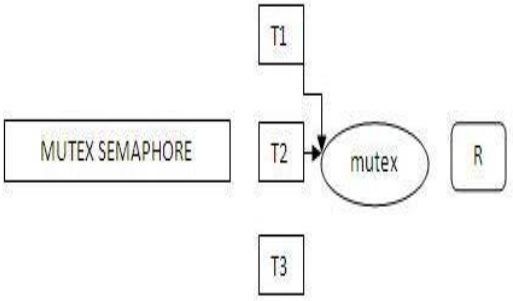


Fig. 6 Mutex semaphore

Initially the semaphore value is assigned to zero. Once a process attains ownership, it can access the resources as many times as it wants and each time, it accesses, the semaphore value is incremented.

*F. Characteristics of Semaphore*

The individuality of the three semaphores are shown in TABLE I.

TABLE I
THE INDIVIDUALITY OF THE THREE SEMAPHORES

| BINARY SEMAPHORE | COUNTING SEMAPHORE | MUTEX SEMAPHORE |
|---|---|---|
| Anyone can release the semaphore. Used for mutual exclusion and event announcement. | Only after a task has attained access, it can release the semaphore. | Only the owner can release the semaphore. Used only for mutual exclusion. |

## IV. RESULTS AND DISCUSSION

*A. Implementation*

In the existing architecture of computers, the restricted access of connecting a high speed low memory device to a low speed high memory device is solved by using a transitional memory unit called the cache. The cache exists between the high speed CPU and the lower capacity memories. Though, there is no protection between the data link layer of the CPU and the cache. The planned card is placed on the PCI bus at the highest probable speed and a shortest connection is recognized to the CPU via snooping. To recognize how the security layer is to be implemented, the knowledge of the three basic terms is required: subject, object and capability. Subject refers to the user or entity which acts on behalf of the user on the system. Objects may be referred to as resources within the system. The main term however is capability which is basically a "token". The ownership of a capability by a subject confers admission right for an object. They cannot be easily customized, but they can be reproduced.

The capabilities of the objects are to be stored in the non-readable part of the Hard Disk Drive. For a subject to access a exact object, it must own the capability for doing so. Hence, before a subject accesses a resource via the CPU, it will first go through a screening test from the hardware on whether or not its capabilities let it to access such resources. Hence, a security layer is now added to the data link layer between the CPU and the cache. In addition, user also must be prevented from creating random capabilities. This can be accomplished by placing the capabilities in individual "Capability Segments" which users cannot access. One more loom is to add a tag bit to each primary storage location. This bit, unreachable to the user is "ON" if the site contains a capability. It ought to be noted that the hardware restricts the manipulation of the location contents to appropriate system routines. If the last remaining capability is destroyed, then that object cannot be used in any manner. In this work, special provisions are made for controlling the copying and movement of capabilities (as well as interpretation) depending on the hardware concerned.

## V. CONCLUSION

In this work, self-healing systems prove that more and more significant in countering the software based attacks, which recover and protect the data from interrupted services. Self-healing systems suggest an active form of decision support, without human involvement that can detect the failure and it can overcome it. Also, By means of clever architectural models, a self-healing system can select the suitable patch up plan to deploy the broken component, if there is more than one component that needs to be healed, can prioritize a fault component over the others, etc.

**References**

[1]. Bouricius, W.G., Carter, W.C. & Schneider, P.R, "Reliability modeling techniques for self-repairing computer systems", in proceedings of 24th National Conference, ACM, 1969, pp. 395-309.

 

[2]. Shelton, C., Koopman, P.&Nace, W., "A framework for scalable analysis and design of system-wide graceful degradation in distributed embedded systems", WORDS03, January 2003.

[3]. G. Edward Suh, J. W. Lee, D. Zhang, and S. Devadas, "Secure Program Execution via Dynamic Information Flow Tracking", in proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-XI), October 2004.

[4]. Martin Abadi, Mihai Budiu, Ulfar Erlingsson, and Jay Ligatti, "Control-Flow Integrity: Principles, Implementations, and Applications", in proceedings of the ACM Conference on Computer and Communications Security (CCS), 2005.

[5]. B. Demsky and M. C. Rinard, "Automatic Detection and Repair of Errors in Data Structures", in proceedings of ACM OOPSLA, October 2003.

[6]. M. Locasto, S. Sidiroglou, and A.D Keromytis, "Software Self-Healing Using Collaborative Application Communities", in proceedings of the Internet Society (ISOC) Symposium on Network and Distributed Systems Security (SNDSS), February 2006.

[7]. J. Kong, X. Hong, J.-S. Park, Y. Yi, and M. Gerla, "L"Hospital: Self-healing Secure Routing for Mobile Ad-hoc Networks", in Technical Report CSD-TR040055, Dept. of Computer Science, UCLA, January 2005.

[8]. Michael E. Shin and Jung Hoon An, "Self-Reconfiguration in Self-Healing Systems", in proceedings of the Third IEEE International Workshop on EASE'06, pp 89-98 (2006).

[9]. E. M. Dashofy, A. V. D. Hoek, and R. N. Taylor, "Towards architecture-based self-healing systems", in p*roceedings of the first workshop on Self-healing systems,* Charleston, South Carolina, pp. 21-26, 2002.

[10]. H.You, V.Vittal, Z.Yang, "Self-healing in power systems: an approach using islanding and rate of frequency decline based load shedding", IEEE Transaction on Power System, Vol.18, No.1, 2003, pp.174 -181.

[11]. T.A. Ramesh Kumaar and Dr.I.A.Chidambaram, "Self-Healing Strategy for Dynamic Security Assessment and Power System Restoration", in International Journal of Computer Science & Emerging Technologies, (E-ISSN: 2044-6004) Vol. 2, Issue 2, April 2011.

 