Availability Enhancement Model for Virtual Machines over Hypervisor Attacks

A. Ravishankar
Juniper Networks, India Pvt. Ltd.
Email: arthir@juniper.net

C. Subramaniam
Department of Computer Science, SRM University, Chennai, India
Email: Chandrasekaran.s@ktr.srmuniv.ac.in

***Abstract*—The objective of the research work is to propose an enhanced availability model for the virtual machines over hypervisor attacks. In recent years, virtualization has turned out to be a most promising area in the field of Information Technology. However, there is always security threats existing in all domains and virtualization is not new to it. Among the various components that lie in virtual environment, hypervisors have always been an ideal target for attack as they provide a single entry point into the virtual environment. Designing a model that could stand these security threats has always been a challenge. The proposed work focuses on the architecture of the virtual environment, the limitations of the various physical resources to be virtualized and addresses various security related issues. It analyses various attack vectors in a quantitative manner individually and integrates all the recovery actions to enhance the overall availability of the virtual machines.**

***Index Terms*—hypervisor, hypervisor attack, security, virtual machines, virtualization**

I. Introduction

With the massive growth in the field of Information Technology, virtualization is playing a major role in the development of new IT infrastructures. Virtualization enables multiple logical abstractions using a single physical hardware running a hypervisor which supports multiple host and guest operating systems [1]. By allowing multiple virtual networks to cohabit on a shared physical substrate virtualization provides flexibility, promotes diversity, and promises increased manageability in the Internet [2]. Virtualization has gained lot of popularity in recent decades and has successfully powered the cloud. It consists of number of virtual machines (VMs) which is a self-contained operating environment—software that works with, but is independent of, a host operating system. In other words, it's a platform-independent software implementation of a CPU that runs compiled code. Intel's "Vanderpool" chip-level virtualization technology was one of the first of these innovations. AMD's "Pacifica" extension provides additional virtualization support. These virtual machines are hosted over a crucial component called the hypervisor that acts as heart of the entire virtual environment. Unfortunately, like any other software or hardware entity, these hypervisors are also prone to security attacks. Any form of security breach ideally shaken all the hosted virtual machines. Each and every interaction between the virtual machine and the hypervisor has become a potential attack vector. Compromised hypervisors not only enable illegitimate access to information but also provide impressive computing power [3]. By altering the metadata it is possible to tie up resources on other hypervisors, force the migration of virtual machines, and confuse load balancing processes in the system manager. NIST's National Vulnerability Database [4] showcases the difficulty of shipping a bug free hypervisor. These software vulnerabilities are easily exploited by attackers thereby breaching confidentiality, availability and integrity of other virtual machine's code [5]. Various techniques are proposed in order to mitigate the security threats that are lying in the hypervisors such as enabling direct access to hardware from VM level, new processor architecture, etc. however, these solutions suffer from few drawbacks as they have either changed the control flow process in such a way that the current cloud environment doesn't support them or provide additional performance overhead. In addition to this, some of the new architectures proposed insist on additional software code that increases the attack surfaces. The proposed model mitigates these shortcomings and provides a comprehensive solution to enhance the availability of virtual entities during hypervisor attacks.

The remaining sections are organized as follows. Section II explores the possibilities of a hypervisor attacks and vulnerabilities at the execution or exploitation mode in a virtual environment. Section III introduces a mathematical approach to the enhanced availability model of the hypervisor in the case of such attacks with suitable modular components the hypervisor site. Section IV discusses about the communicating processes that cover the health status of the hypervisor with its normal states and discloses the formal representation of the secured hypervisor. Section V brings out the possible

Manuscript received July 7, 2014; revised September 15, 2014.

©2015 Engineering and Technology Publishing
doi: 10.12720/ijsps.3.1.88-93

simulations through CPN Tool and Section VI concludes the work with specifications about the future work.

II. Hypervisor Attack

Hypervisor is a software or firmware that creates and manages multiple virtual machines. It decouples the operating system and applications from their physical resources. A type 1 hypervisor has its own kernel and it is installed directly on the hardware, or "bare metal". It hosts a software component called a Virtual Machine Monitor (VMM) that is responsible for allocation of system's processes and other resources such as memory, etc. for the guest operating systems. Realizing this functionality requires frequent interactions between the hypervisor and the virtual machines. An ideal hypervisor attack usually happens during such interactions, exploiting the bugs present in the hypervisor. A malicious request tries to gain kernel level access in the hypervisor during such interactions and comfortably manages to attack other VMs that are hosted over the same hypervisor. Thus, a compromised hypervisor not only cause harm to a single VM but brings down the entire virtual environment. In recent times variety of attacks is encountered and according to recent statistics, more than one-fourth of security threats that lie in virtual environment are cornered towards the hypervisor. There is wide range of hypervisor attacks registered starting from "escape-to-hypervisor" attack to "hyperjacking". Security attacks such as "hyperjacking" has adverse effect as it crafts a thin hypervisor which takes complete control over the virtual and physical entities. In addition to these, numerous Trojan programs are designed which are capable of running without any sign of its very existence in the existing system. Further, the hypervisor weakness or vulnerability gets multiplied wherever more number of access, delete or inject operations are performed with a threat of marrying malicious metadata into the maiden hypervisor.

Creating a virtual machine has become as simple as copying a file and pasting it in a different location. Such type of endless scaling also creates significant security issues as they increase the number of interactions between virtual machines and the hypervisor. Users frequently use several or even dozens of special purpose VMs for testing, demonstration e.g., "sandbox" VMs to try out new applications, or for particular applications which are not provided by their regular OS (e.g. a Windows VM running Microsoft Office). Thus, the total number of VMs in an organization grows at an explosive rate, proportional to available storage [6]. This exponential increase in the number of VMs creates room for dormant VMs that are highly inactive. Many a times, these dormant VMs are left out while implementing security procedures such as updating access policies thereby providing room for unchecked "back doors" in the VMs [7]. These "back doors" are best exploited by the attackers as they act as an ideal entry point into the secure virtual environment.

Technically, a security attack can be described as an event which is mostly uncertain, i.e. the time of attack and the impact of attack are highly unpredictable. It is a deducible deviation from the Control Flow Process (CFP) to be executed by the hypervisor for all sorts of input, states and transitions. Let attack A_i, shown in equation (1), be an event which is parameterized by the factors as time of attack t_a, duration of attack d_a and impact of attack i_a

$$A_i = <t_a, d_a, i_a> \quad (1)$$

An attack A_i, ideally exploits the various vulnerabilities that are present inside the hypervisor. The vulnerability of a hypervisor can be described as a state of security flaw whose probability of existence depend upon the design of the hypervisor h_d, interaction between *VMs* residing on different/same hypervisors VM_i and the code visibility C_v to other domains, as per equation (2).

$$P(v) = <h_d, VM_i, C_v> \quad (2)$$

Thus, the event of attack A_i exploiting the vulnerability factors V_j is represented in the equation (3)

$$E(A_i, V_j) = \sum A_i \, x \, [P(v)] \quad (3)$$

III. Enhanced Availability Model for Virtual Environment

The availability in the context of virtual machines is the portion of the time during which the hypervisor is functioning as per the functional and performance specifications. The percentage can be called as the ratio of uptime to total time where the total time is the sum of uptime and the mean time to repair. The repairing is to be carried out once the hypervisor has been attacked either from outside or due to internal design flaws. The challenges are cross-platform systems management for both the virtual and physical machines when the hypervisor is attacked. Especially, when the data migration or live VM migration take place, it is simpler to enhance the performance of legacy applications but ensuring security involves additional efforts.

The real issues and challenge to enhance the availability of the virtual machines are determined by the availability of fixed type of virtualization either server or application or network when the hypervisor is attacked, and the durability over which the deployed virtual machine should have the expected behavior. The availability of the virtual machine A_{vm} at time t, is determined by the availability of the hypervisor $A_{hypervisor}$ which is determined by the reliability $R_{hypervisor}$ and security factors $S_{hypervisor}$ that exist with respect to the hypervisor.

$$A_{vm}(t) \, \alpha \, A_{hypervisor}(t)$$

$$A_{hypervisor}(t) \, \alpha \, R_{hypervisor}(t) \, x \, S_{hypervisor}(t)$$

To model and design a formal and efficient virtualization technique with correct type, duration and scalable virtual entities for both computing and networking solutions, it is possible to incorporate a duty based strategy on disposable and universal virtual machine configurations and networking interfaces.

The enhanced availability model is proposed for Type 1 Hypervisors where the software or firmware runs

©2015 Engineering and Technology Publishing

directly on the hardware. The proposed model focuses on establishing various modules (shown in Fig. 1), namely, Hypervisor Health Monitor (HHM), Ambiguous Request Analyzer (ARA), Resource Switching Verifier (RSV), Timing Call Checker (TCC) and Injected Interface Manager (IIM) in order to enhance various quality factors thereby bringing in availability when the hypervisor as a whole is attacked. The Hypervisor Health Monitoring (HHM) module verifies the incoming and outgoing data of the hypervisor by taking health samples periodically. The expected transitions and flows are defined by the CFP (Control Flow Process). Any deducible deviation from CFP defines the unstable or attacked state of the hypervisor.

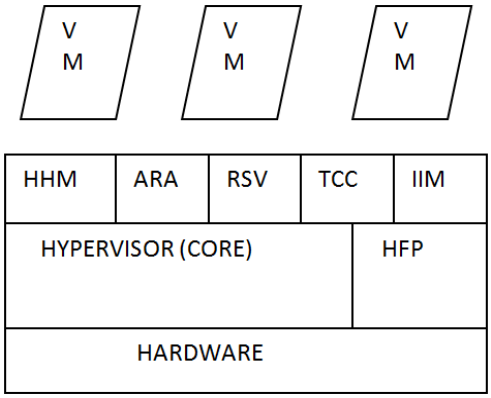

Figure 1. Highly available secured hypervisor

Any event of attack triggers the TCC (Timing Call Checker) module. The timing call checker provides an absolute remedy in the case of hypervisor attacks based on the intrinsic delays that are observed during hypervisor attacks. The TCC scrutinizes the average response time of the hypervisor in all condition and interact with other running VMs or active VMs once the delay exceeds the normal value.

The Ambiguous request analyzer (ARA) plays as a security guard for the hypervisor. It checks the integrity of the request before submitting any task to the hypervisor. It makes use of Embedded Intelligence to check the data the request is trying to access and the code it is trying modify in order to gain access into the guest or host. This module reduces the load over the hypervisor by enabling policy definition at an abstraction layer that lies above the hypervisor. Various policies are defined within the module to limit the functionality of the request coming to the hypervisor from the overlying entities. This reduces the need for the implementation of various security mechanisms such as Mandatory Access Control (MAC) within the hypervisor. The Resource Switching Verifier (RSV) component proposed in the model supports the hypervisor by verifying the switching time of the virtualized resources for the next application. The timing call checker (TCC) component disallows any delayed call from any one of the domains or VMs. In the case of suspected requests based on the reports from ARA, RSV and TCC top-ups, the hypervisor is available only to the legal and authenticated requests through injected interface management module (IIM).

IV. Process Model of Hypervisor

The process model depicts the control transfer between various modules stated in Fig. 1 under versatile conditions. The various processes with accepted and limited colored tokens (designed as per CPN) are declared in the communicating sequential process model of the hypervisor with limited concurrency. The hypervisor states are defined as “Active”, “Risk”, “Attack”, “Wait” and “Footprint”. The “Active” state represents the fully functional state of the hypervisor where all the VMs come up and operate as expected. Under active state, multiple state transitions are possible depending on the event that follows it. For example, the hypervisor may be attacked by some external forces or the hypervisor might get overloaded due to the creation of numerous VMs that affects the performance of the entire virtual environment and create room for attacks due to lacking updates in dormant VMs thereby creating “back doors”. When the hypervisor gets loaded with multiple dormant VM, the hypervisor state transitions to “Wait” state where in, the auxiliary code gets executed that removes the dormant VMs that are highly inactive, hosting insensitive data. However, auxiliary code doesn’t remove dormant VMs that host sensitive data such as critical configurations, encryption keys, etc. It will rather ensure that such VMs are kept up-to-date with required access policies and other security procedure. The hypervisor remains in “Wait” state during this process and moves back to “Active” state upon completion.

The ARA (Ambiguous Request Analyzer) module introduced in section III acts as a guard process that intimates the hypervisor about malicious trials for entry into the core hardware. Under such suspicious conditions, the hypervisor moves from “Active” state to “Risk” state. Moving the hypervisor to “Risk” only indicates that some ambiguous request is being analyzed by ARA and this transition doesn’t affect the normal functioning of the hypervisor. If these ambiguities are resolved in the ARA module (shown in Fig. 1), the hypervisor moves back to “Active” state.

The proposed model holds a Hypervisor Foot Print (HFP) module (shown in Fig. 1) which is considered as the heart of the system that stays isolated from the core. The footprint module keeps track of various tasks that happen at core level without intervention. It takes control over the overlying VMs under critical scenarios that are caused because of external attacks on the hypervisor.

Ideally, a hypervisor gets attacked adversely, when an attacker manages to run some arbitrary code in the kernel space of the hypervisor through illegitimate entry. On detection of such malicious entry, the proposed model transitions the control to the HFP module and moves the hypervisor to “footprint” state indicating that the normal functioning of the hypervisor is taken over by the HFP module. This enhances the availability of various entities in the virtual environment as they are made available even when the hypervisor as a whole gets attacked. The control transfer to HFP module is done to enable the hypervisor to recover in this duration. However, the virtual machines can be kept functional by the HFP

©2015 Engineering and Technology Publishing

module only for a specific time duration within which the core module of the hypervisor is expected to recover. Recovery before the HTP timeout moves the hypervisor to "Active" state and the control is taken care by the hypervisor (core).

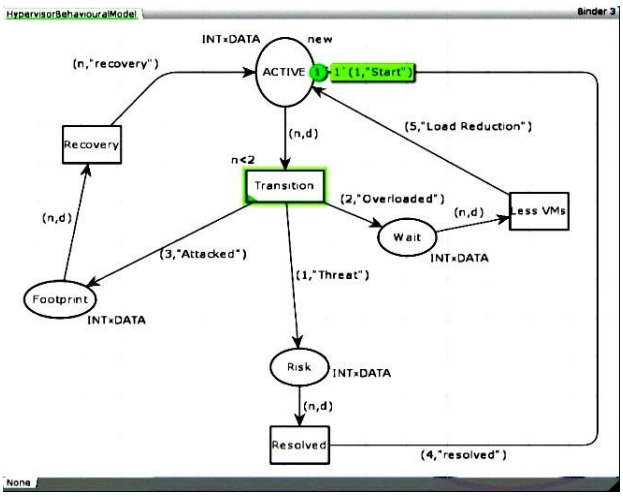

Figure 2. Communicating process model of hypervisor

The formal model of secured hypervisor has to be checked for the different functional modules proposed towards its security. Model checking is a method to verify whether a model obtained satisfies the formal specification. The availability of the virtual machines can be enhanced if the hypervisor is healthy and attack free. So the study is focused towards the hypervisor attacks and its prevention to safe guard the number of VMs running on it. The vulnerability of the hypervisor has to be understood as a scenario where the event of attack will be realized by running suitable modules each calling their respective functions. The phase of the attack prevention is to describe the procedure that are to be followed one after the other with all its conditions checked as per the status of the respective conditions of the hypervisor as shown below:

Let S represent the set of states represented in the communicating process model (shown in Fig. 2), C represents set of possible configuration and S ´represent the set of state transition triggers.

```
set of states S = { ACTIVE, RISK, WAIT, FOOTPRINT};
set of state_triggers S ´= {ATTACKED, THREAT, OVERLOADED};
set of configuration C = {VM_config, process_config, interface_type};

procedure hypervisor_health_monitor (state S)
        trusted_domain_check();
        if ¬S: = ACTIVE then
                ambiguous_request_analyzer();
        else return 1;
        fi
endproc

procedure ambiguous_request_analyzer()
        if S ´: = THREAT then
                S ´: = ATTACKED;
                hypervisor_footprint (S ´);
    else if S ´= OVERLOADED then
                until ((S: = WAIT) V remove dormant VMs);
                S: = ACTIVE;
        fi
endproc
```

```
procedure hypervisor_footprint (state S ´)
        transfer control_flags;
        transfer register_sets;
        while (¬S ´: = Recovered) do
                sleep (10);
        od
        if (S ´= Recovered) then
                post_attack_health_status();
                transfer control_to_master;
                S: = ACTIVE;
        fi
endproc
```

```
procedure boolean post_attack_health _status()
        check configurarion C;
        if S: = ACTIVE && RSV: = TRUE
                return 1;
endproc
```

V. Simulation Using CPN Tools

The communicating process model of the hypervisor is explored by simulating it using Coloured Petri Net (CPN). A CPN model of a system describes the states of the system and the events (transitions) that can cause the system to change state [8]. It enables step-by-step execution thereby providing a perfect "walk through" for the proposed model.

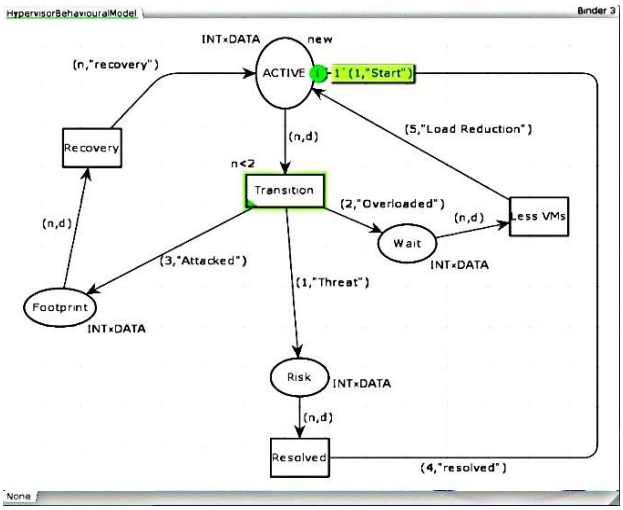

Figure 3. Hypervisor state - active

A CPN model consists of tokens that have data values attached to them. The various hypervisor states are represented as places (drawn as ellipses) and the

transitions are represented in rectangular boxes. Fig. 3 shows the initial state of the system where the hypervisor is in "active".

The possible transitions from "active" state are depicted in the form of tokens. Each token carry an integer and a string value that is taken as input by the next place or transition (within the CPN model). The string value sent in the token hold the current state of the hypervisor.

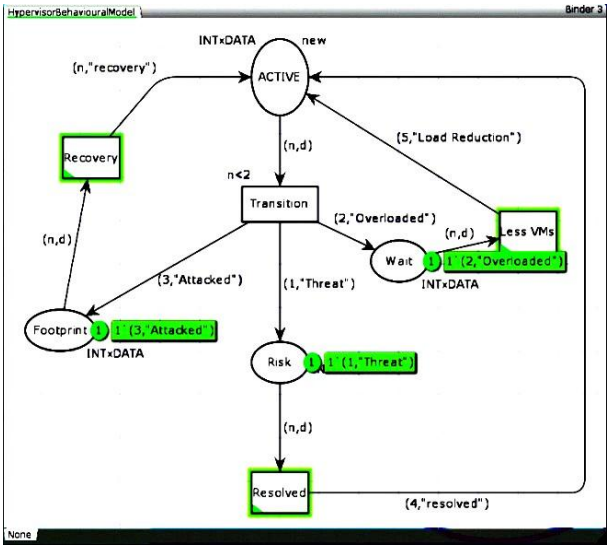

Figure 4. Hypervisor state - vulnerable/attacked/overloaded

The token values sent between the processes states hold the token number (int) and the state information (string). These token values are shown in the places (shown in ellipse). A hypervisor in "active" state might either get attacked or might identify a threat of or might get overloaded with too many VMs. These triggers transition the hypervisor to appropriate states (shown in Fig. 4) based on the token values (n, d) sent.

Fig. 5 depicts the recovered state of the hypervisor after the security measures taken by appropriate modules like RSV, TCC and IIM as and when the suspicious event is detected.

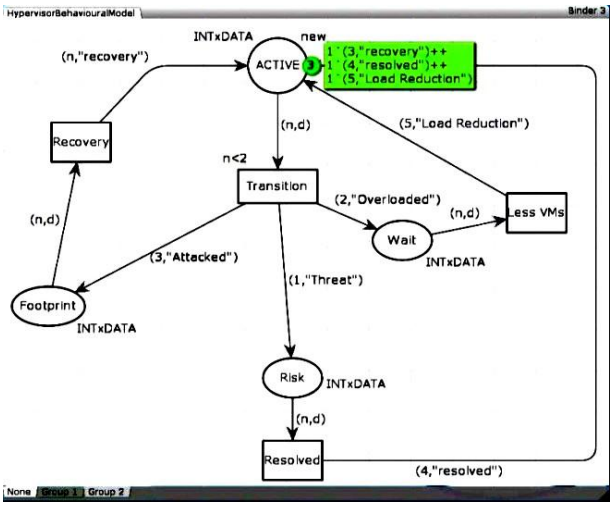

Figure 5. Hypervisor state - recovered

VI. Conclusion

The proposed work has focused on hypervisor security, introduced modules to prevent malicious entry into the hypervisor. The model has also introduced ways to handle the virtual environment once the hypervisor as a whole gets attacked due to disastrous threat that bypassed the firewall and other security enhancements enabled in the system. The model provides a way to make the virtual environment available by transferring control to the footprint module when the bare metal hypervisor goes down completely. The serious limitation of the current work is the uncertain features of all attacks which may affect the resource utilization of the memory of the hypervisor. The focus of the future work will be towards heterogeneous hypervisors integration so as to realize and get ready for the remedial actions to minimize the loss. The other significant area of further work is to focus on the self-assembling virtual machines and having VMs for IO devices and computing cores separately to isolate the exploits of attacks.

Acknowledgment

The authors wish to thank Ravi Jagannathan, Senior Staff Engineer at Juniper Networks, Sunnyvale for his valuable comments which helped in successful completion of the current research work. They also thank Aquin Mathai and Stanzin Takpa from Juniper Networks India Pvt. Ltd., who stood as a strong pillar rendering their continuous support and encouragement that helped in successful delivery of this work.

References

[1] L. Turnbull and J. Shropshire, "Breakpoints: An analysis of potential hypervisor attack vectors," in *Proc. IEEE Southeastcon*, 2013, pp. 1-6.

[2] Q. Duan, "Modeling and Performance analysis on network virtualization for composite network–cloudservice provisioning," in *Proc. 2011 IEEE World Congress on Digital Object Identifier*, 2011, pp. 548-555.

[3] S. Jin, J. Ahn, S. Cha, and J. Huh, "Architectural support for secure virtualization under a vulnerable hypervisor," in *Proc. 44th Annual IEEE/ACM International Symposium on Microarchitecture*, Port Alegre, Brazil, 2011, pp. 272-283.

[4] CVE and CCE statistics. *National Vulnerability Database*. [Online]. Available: http://web.nvd.nist.gov/view/vuln/statistics

[5] J. Szefer, E. Keller, R. B. Lee, and J. Rexford, "Eliminating the hypervisor attack surface for a more secure cloud," in *Proc. Conference on Computer and Communications Security (CCS)*, 2011.

[6] T. Garfinkel and M. Rosenblum, "When virtual is harder than real: Security challenges in virtual machine based computing environments," in *Proc. 10th Conference on Hot Topics in Operating Systems*, 2005.

[7] Requirements and Security Assessment Procedures, PCI Data Security Standards, ver. 2.0, Oct. 2010.

[8] K. Jensen, L. M. Kristensen, and L. Wells, "Coloured petri nets and CPN tools for modelling and validation of concurrent systems," *ACM International Journal on Software Tools for Technology Transfer (STTT)*, vol. 9, pp. 213-254, May 2007.

©2015 Engineering and Technology Publishing

Ms. Arthi Ravishankar, was born in Chennai, India in the year 1992. She completed her B.Tech in Information Technology in REC affiliated to Anna University, India in the year 2013. She has worked on numerous research works and has proposed solutions for various real time issues. She is currently part of Adecco India Pvt. Ltd. and is deputed officially at Juniper Networks, India Pvt. Ltd as System Test Engineer. One of her previous work entitled "Application Safety Enhancement Model using Self Checking with Software Enzymes" was hashed into SOA\NASA Astrophysics database. Her current research domains are cornered towards Virtualization and other areas related to networking.

Dr. Chandrasekaran Subramaniam, has his specialization in Computer Science and Engineering. He completed his B.E (ECE) from C.I.T/UM, Coimbatore, India in the year 1980, followed by M.E (CSE) from N.I.T/B.U, India in the year 1993 and Ph.D in ICE, from C.E.G affiliated to Anna University, India in the year 2004. He has 31 years of rich experience in academic field and is currently working as DEAN & PROFESSOR of CSE department, SRM university, India. He is the Senior Member of IACSIT (Member No. 80336573). He has published his works in more than 90 International Conferences. He has also been a part of various review committees during various International Conferences. He also holds membership in IEEE, ACM, WSEAS, WASET, SEI, IET, I.A. Engg., ASQ, ISTE, CSI. His research works has touched almost all domains and has gained International Recognition.

©2015 Engineering and Technology Publishing