The Ideal Computing System Framework – A Novel Security Paradigm

Orhio Mark Creado*, Bala Srinivasan**, Phu Dung Le*** & Jeff Tan****

*PhD Student, Caulfield School of Information Technology, Monash University, Caulfield, Victoria, AUSTRALIA.
E-Mail: mark.creado{at}monash{dot}edu
**Professor, Clayton School of Information Technology, Monash University, Clayton, Victoria, AUSTRALIA.
E-Mail: srini{at}monash{dot}edu
***Lecturer, Caulfield School of Information Technology, Monash University, Caulfield, Victoria, AUSTRALIA.
E-Mail: phu.dung.le{at}monash{dot}edu
****High Performance Computing Specialist, IBM Research Collaboratory for Life Sciences, IBM Research - Australia, Melbourne, Victoria, AUSTRALIA. E-Mail: jeffetan{at}au1{dot}ibm{dot}com

***Abstract*—**Computing system security is a growing area of increasing importance; yet existing infrastructure is vulnerable due to its reliance on dated mechanisms to ensure secure operations by trusted entities. There exists a need wherein a computing system must explicitly be secured from itself. This over dependence on the operating system to ensure overall system security can be viewed as a vulnerability in itself. This paper presents a novel framework towards securing a computing system by proposing a structuring concept of a monitor defined at a micro level, with specific characteristics aimed to satisfy specific security properties. With multiple monitors defined for each component, they facilitate for each component to be responsible for securing itself and its operations in a decentralized manner and collectively ensuring the security of the overall system thereby reducing any single point of failure.

***Keywords*—**Computing System Security; Decentralized Security; Explicit Trust; Security Framework.

***Abbreviations*—**Computing System (CS); Ideal Computing System (ICS); Operating System (OS).

I. INTRODUCTION

MODERN day computing systems are complex machines. The integration of various components and varied platforms has not only made securing a Computing System (CS) more important but more challenging. In CSs, many identity verification mechanisms and trust validation techniques currently used often view security and trust as a unitary metric; and are often exploited due to this fact. The reactive approach towards fixing exploited vulnerabilities involves the band-aid solution; which, more often than not, tend to be short term fixes that sometimes lead to newer vulnerabilities due to code reuse from predecessor applications.

The concept presented in this paper is towards proposing a new novel security paradigm which stipulates that the complete security of a system can only be achieved if each of its underlying constituents are also secured. In order to do this, we propose the concept of monitors and reduce a CS to its most fundamental unit - the operating code. Each monitor has defined characteristics, which operate with the goal of satisfying specific security properties. By isolating the various components of a CS, various monitors collaborating at the component level can determine the overall security of a component by its satisfaction of the underlying properties; and collectively with monitors of other components allow for communications between components which are deemed secure only. This proposed approach reduces the reliance on any one specific component, such as the Operating System (OS), to be responsible for securing the execution of all other components; and fosters a decentralized and distributed model which allows for the compartmentalized isolation of components.

As this paper is an extension of a prior published article by Creado et al., (2014A), the rest of this paper is structured as follows. Section 2 proposes a brief overview of relevant literature and outlines the existing open foundational problems. Section 3 cover the original proposed idea but add more details pertaining to the proposed operations by outlining the proposed framework and its approach towards achieving distributed and decentralized security. Section 4 proposes an evaluation methodology for the original idea and adds a significant contribution to the original published work by outlining an objective evaluation of the model in terms of

challenges applicable to feasibility of the model and lists achievable countermeasures which aim to resolve them. Section 5 concludes the paper and outlines further directions in terms of future work planned for this research project.

II. BACKGROUND

Trust and security aren't necessarily synonymous, and their misappropriation within a CS is an undesirable consequence [Abadi, 2004]. Security is traditionally viewed either in terms of data confidentiality or data integrity. With this distinction many models have been proposed which normally accredit one aspect of security efficiently, but not both. With the growth of modern technology, the quest for more user friendly devices with the emphasis on performance has become the norm; but the requirement for securing those devices has remained the same.

Security in CSs has been a long standing dilemma; some of the earliest examples of secure CS proposals are those of MULTICS [Rushby 1986; James et al., 2009], PSOS [Feiertag & Neumann 1979; Neumann & Feiertag, 2003], and KSOS [Berson, 1979]. These systems are nonetheless dated but have formed the foundational framework for most modern day commodity systems. The main drawback with these models was their overreliance on the OS to maintain overall system security. Various mathematical formulations justified the definition of many security properties, but the distinction between system and user level processes was unclear. Since then many proposals which have either developed on or enhanced these models have aimed to increase the security of a CS either through authentication mechanisms or access control procedures administered through software or hardware based solutions. Some examples of these include EROS [Shapiro et al., 1999], and TCG's TPM [Martin, 2008; Trusted Computing Group, 2013]; of which a resulting application includes Microsoft's NGSCB [Microsoft, 2003; 2003A; 2003B; Abadi & Wobber, 2004; Peinado et al., 2004]. Other alternatives have included the development of micro kernels which aimed to reduce the amount of operating code within kernel space, an example is proposed by Klein et al., (2009); and lastly, the use of virtualization technology.

However, between these approaches, the main similarity has been the provision of using the OS as a fulcrum for processing instructions; thereby, making the security of the CS its inherent responsibility. Hansen (1975; 1976) defined the purpose of the OS was to manage the allocation and utilization of system resources and even proposed an OS structuring paradigm to achieve the same. Commodity OSs have been designed within a hierarchical structure as proposed by Lister & Sayer (1977), wherein all system operations occur at ring 0 but securing ring 0 from itself remains the main challenge [Bratus et al., 2009].

CSs serve an end user, and therein is introduced the fallacy for a distinction between system processes and user processes. The importance of this research is to alleviate this ambiguity by fostering a trust based operating environment mimicking those which exist in human interactions. Existing technology is still based on a black and white analogy when it comes to trust within a CS; a common example is that of a username and password. There is no facilitation for anything in between. Existing technology attempts to resolve some of these modern threats by abstracting the responsibility to the end user. This results in two very undesirable situations. First, compromising user friendliness of the system; and second, the assumption of computer security literacy of the end user to be able to secure themselves. The plethora of hacking attacks against big corporations, governments, and other organizations is proof that our existing infrastructure can be prone to human error in its implementation. There is a need for the foundational model of secure computing to implement security as a vital part of its operation rather than as an optional add on available at the discretion of the end user.

This paper further elaborates on the Ideal Computing System (ICS) framework concept of Creado et al., (2014A); in which they identify five main components of a CS, namely: Hardware, OS, Services, Applications, and Users. They proposed a paradigm wherein a CS is representable as its most fundamental unit - its underlying operating code; so as to be able to render system security as a combination of individual characteristics. Through the linear progression of each of these individual characteristics, the overall security of the system could be achieved. The security characteristics they identified applicable to operating code were: Invulnerable, Has Integrity, Is Verified, Is Trustworthy, and Is Secure. The proposed concept model provided a hybrid approach of using the OS structuring concepts of 'Monitors' [Hansen, 1972] supported by a model which required the definition of explicit trust [Creado et al., 2014] for all operating code. This explicit trust model facilitated for an architecture which allowed for the implementation of a trust engine, wherein trust and security are considered two parts to a whole. By ensuring that all operating code satisfies these characteristics and their underlying properties, the proposed trust engine made it possible to represent trust as a deterministic value, based on the degree of satisfaction of each of the outlined characteristics and past operational history. This introduced the concept of evolving trust wherein this calculated trust metric associated with any operating code, is capable of evolving over time to increase or decrease based on correct executions in its operational history. In order to secure the CS from itself, the notion of monitors was introduced to enforce these characteristics and their underlying properties. The monitors also facilitated for the compartmentalized isolation of components to allow for decentralized security; whilst simultaneously, reducing any single points of failure within the system by eliminating the dependence of any one component on another for its own secure operations.

III. ICS SECURITY WITH MONITORS

This section outlines the role of the underlying monitors proposed within the ICS framework and how they are

integrated into the CS to ensure overall system security. Figure 1 graphically illustrates a high level concept of the integration between the proposed monitors to each identified component, and that of each component in relation to each operating stage of a CS. In the following paragraphs, we formally define the proposed concept of monitors; and further define the specific operations pertaining to each identified monitor; and lastly, define the monitor operations in relation to the operating stages of a CS.

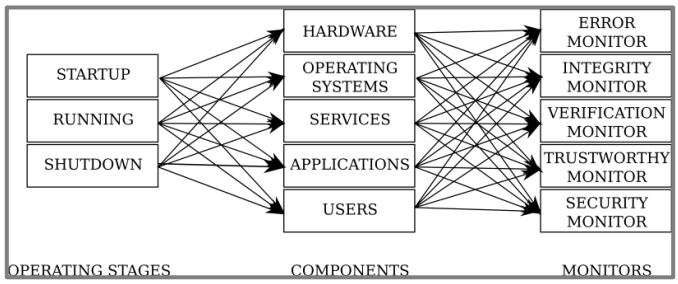

Figure 1: ICS Framework Monitor Integration

3.1. Notations

For conciseness, wherein words have been abbreviated, Table 1 outlines a partial list of common notations used through the rest of this paper, with the remainder outlined in text.

Table 1: List of Notations

Notation	Description
Ψ	Computing System
$Comp$	Component
EM	Error Monitor
IM	Integrity Monitor
$Instr$	Instruction
M	Monitor
$OpHist$	Operation History
Sec	Secure
Sig	Signature
SM	Security Monitor
$State$	A system state of operation
TL	Trust Level
TM	Trustworthy Monitor
$TrustAlg$	Trust Algorithm
VM	Verification Monitor

3.2. Monitor Description

Definition 1: The ICS framework defines a Monitor (M) tobe a self-contained programmed unit, which operates independent of any other entity within a strictly defined operation boundary specified by a set of characteristics which govern its underlying attributes. Each monitor defined within the ICS framework is a quintuple of the following characteristics:

$$M = \{ C, O, A, T, S \}$$

where,

- Class (C) - Defines the level of operation and the level of information sharing supported. Includes:
  - System-Only (C_o) - Operates over the entire system but allow for system only access.
  - System-Wide (C_w) - Operates over the entire system but allow for information sharing to secure entities.
- Mode (O) - Defines the operational state and mode of information procurement and storage. Includes:
  - Analysis (O_a) - Statistical analysis of historic data using an active observe-react-respond approach.
  - Tracking (O_t) - Information storage using a passive observe-track-record approach for later analysis.
  - Hybrid (O_h) - Combination of analysis and tracking modes, allowing for run-time switching.
  - User (O_u) - Analysis, Tracking, or Hybrid modes for operation within user space with restricted access.
- Access Level (A) - Defines a calculated trust threshold to enable resource access for an entity. Includes:
  - System Access (A_s) – Highest level of trusted system-only operations isolated from user intervention.
  - Normal Access (A_n) - Routine non system, user operations which require a high level of trust.
  - Restricted Access (A_r) - Sandboxed operations with restricted access to resources.
- Trust Level (T) - Defines a deterministic metric calculated from successful operational history. Includes:
  - Functional Trust - Basic trust requirements for operating code to execute. Levels include:
    - Operational (T_o) - Highest level reserved for system-only operations.
    - Verifiable (T_v) - Basic level required for all entities.
    - Denied (T_d) - Reserved for non-trusted entities, not permitted to operate within the system.
  - Transactional Trust - Allows trust levels to evolve over time. Levels include:
    - Transitional (T_t) - Serves as an intermediary level between operational and verifiable trust levels.
    - Untrustable (T_u) - Serves as an intermediary level between verifiable and denied trust levels.
- Scope (S) - Defines the operational visibility within other component layers. Includes:
  - Global (S_g) - Global operation over the entire system supporting information sharing at most levels.
  - Local (S_l) - Local operation over specific components sharing information via a global scope monitor.

3.3. Monitors and their Operations

In this section, we further elaborate on the defined monitors and their operations for each of the five defined security characteristics - Invulnerable, Has Integrity, Is Verified, Is Trustworthy, and Is Secure. The ideology behind the concept is the definition of a monitor for each characteristic to facilitate the enforcement of that characteristic for each component operating within the CS. Figure 2 outlines the interdependency between each of the monitors in order to ensure system security.

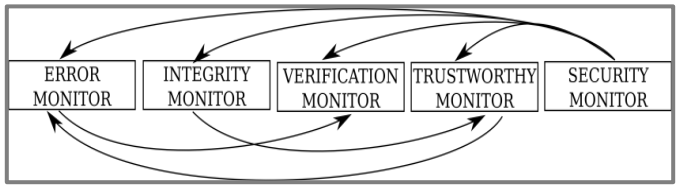

Figure 2: Monitor Interdependency

The following list itemizes each monitor's higher order purpose and its required operational parameters.

3.3.1. Error Monitor (EM)

The ICS framework defines an error as the probability of ensuring that all executed instructions are processed correctly. This is specifically applicable for all input and output parameters which must comply with the appropriate data types as required by the calling operating code. The required parameters can be represented as the following set:

$$EM = \{Code, Instr_i, O_i\} | i \in N$$

As the error monitor would be required to passively monitor instruction processing post execution; for any $Code$ with n instructions and O being the result of executing an instruction, its operations can be simplified as follows:

$for\ i = 1\ to\ n$
$O_i^* = Execute[EM(i)]$
$\quad if\ O_i^* = Error(State)\ then\ Oi = O_i^*$
$next$

Post execution, the error monitor must update the operational history for the specific $Code$, and produces the following output:

$$EM(Code) = \{O, OpHist(Code)\}$$

3.3.2. Integrity Monitor (IM)

The ICS framework determines integrity by ensuring that all operating code is verified and validated for an ownership signature prior to execution so as to ensure accountability. The required parameters can be represented as the following set:

$$IM = \{Code, Hash(Code), Sig[Hash(Code), User], O\}$$

Prior to execution, the integrity monitor would be required to verify the attached signature against a calculated signature of the $Code$ being executed.With O beingthe result of the validation process, its operations can besimplified as follows:

$S^* = Sig^*[Hash(Code), User]$
$If\ S^* = Sig[Hash(Code), User]\ then$
$\quad O = true\ else\ O = false$
$endIf$
$return\ O$

Prior to execution, the integrity monitor must provide a Boolean metric to ascertain the validity of the provided integrity signature and produces the following output:

$$IM(Code) = \{O\}$$

3.3.3. Verification Monitor (VM)

The ICS framework determines verification by ensuring that proper state management is maintained for each executed instruction; a hybrid concept adopted from the theory of finite state machines. The ICS framework incorporates state management at the instruction level to ensure no unauthorized changes in system states occur during execution of each instruction. The required parameters for the operation of the verification monitor can be represented as the following set:

$$VM = \{Code, Instr_i, O_i, O_i^*\} | i \in N$$

The verification monitor would be required to actively monitor the execution of each instruction so as to verify the state prior to execution remains unchanged post execution. With O being the result of the verification process, its operations can be simplified as follows:

$for\ i = 1\ to\ n$
$\quad if\ O_i \neq O_i^*\ then\ exit$
$\quad else$
$O_i^* = Exec[VM_i]$
$O_i = O_i^* i$
$\quad endIf$
$next$

Post execution, the verification monitor must update the error monitor with any state exceptions caught so as to have the operational history updated accordingly by producing the following outputs:

$$VM(Code) = \{O, O^*, Error(State)\}$$

3.3.4. Trustworthy Monitor (TM)

The ICS framework determines trustworthiness by ensuring the proper definition and validation of a trust level for all operating code based on its past historical performance (captured and recorded from previous executions obtained via the Error Monitor). This is facilitated by the implementation of a trust engine which incorporates a trust algorithm capable of producing a deterministic trust metric to be used as a trust level based on static inputs of past operational history and execution outcomes (at this time, the trust algorithm, trust category trust metrics, and trust level trust metrics are out of scope of this paper). The required parameters for the operation of the trustworthy monitor can be represented as the following set:

$$TM = \{Code, TrustAlg, OpHist, TL, TL^*\}$$

The trustworthy monitor would be required to calculate the trust level (deterministically calculable via a trust algorithm based on the Error Monitor and Integrity Monitor) for all operating code prior to execution to ensure the proper allocation of permissions and access levels to resources. With O being the result of the calculation process, its operations can be simplified as follows:

$TL^* = TrustAlg(Code, OpHist)$
$if\ TL \neq TL^* then\ TL = TL^*$
$return\ TL$

Prior and post execution, the trustworthy monitor must provide a trust level validation and trust level update based on the operational performance and execution outcome by producing the following output:

$$TM(Code) = \{TL\}$$

3.3.5. Security Monitor (SM)

The ICS framework determines security through the collective enforcement of all defined monitors via a dedicated security monitor to ensure the overall security of the CS is maintained by facilitating interaction between components and monitors prior to executing instructions. The security monitor must determine the overall level of security of the component by assessing the satisfaction of requirements from the other monitors. Furthermore, inter-component communication is facilitated by the security monitor of the calling component communicating with the security monitor of the called component to ensure both components are deemed secure prior to instruction execution. The required parameters for the operation of the security monitor can be represented as the following set:

$$SM = \{Code, EM, IM, VM, TM\}$$

The security monitor would be required to ensure that the execution of the operating code has appropriate access levels defined for resource and system access. The operation of the security monitor ideally has two independent phases which determine these levels. With O being the result of the each process, its operations can be simplified as follows:

Phase 1: Prior Execution

$$EstablishEM(Code, Instr, O)$$
$$Establish\ IM\{Code, Hash(Code), Sig[Hash(Code), User], O\}$$
$$if\ IM(Code)\{O\} = true\ then$$
$$Establish\ VM(Code, Instr, O, O^*)$$
$$Establish\ TM(Code, TrustAlg, OpHist, TL, TL^*)$$
$$Set\ Access\ Levels\ (TL)$$

Phase 2: Post Execution

$$Validate\ VM(Code) = \{O, O^*, Error(State)\}$$
$$Validate\ EM(Code) = \{O, OpHist\}$$
$$Update\ Trust\ Level\ TM\{Code, TrustAlg, OpHist, TL, TL^*\}$$

3.4. Monitor - Component Integration and Operations

The proposed work allows for the definition of independent monitors for each of the various components with defined characteristics governing that monitor's operations and access levels within the system. This is achieved by the definition of a monitor engine, the purpose of which is to ensure the execution of secure and verified code for defining monitors assigned to each calling component. Thereby facilitating for a distributed model of control and process execution, a compartmentalized model is fostered which allows for isolated execution of operations so as to ensure that any compromise within any one specific component does not indirectly affect another component's operations. We address each of the components identified in previous sections and how the proposed concept of monitors facilitates for isolated operations so as to ensure overall system security. Figure 3 graphically depicts a conceptual model which defines the interaction between various components and the monitor engine to facilitate monitor creation for each component within the system.

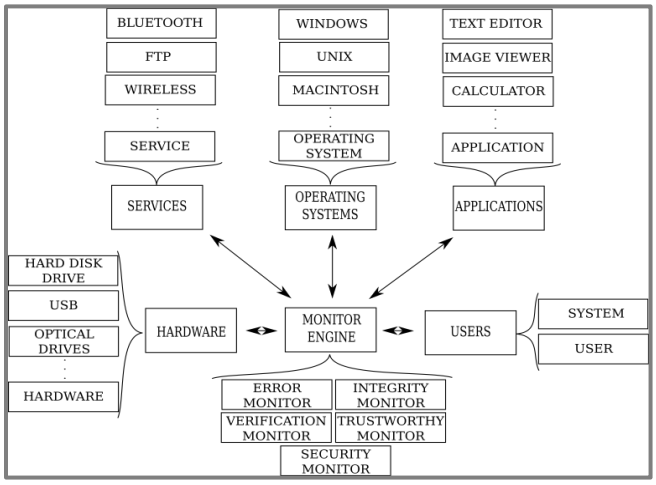

Figure 3: Conceptual Layout of ICS Monitors

We identify the three main stages of operation of a CS; namely: start-up, running, and shutdown; and apply them to the operations of the five identified components. Where applicable the steps outlined can be reversed for the shutdown stage of operation; and components not involved during any specific stage do not initiate any steps during that stage of operation.

The proposed work illustrates six main steps involved in the process of securing a component, which are repeated for each additional component within any specific component category. The key variation between the various component categories is defined by the monitor characteristics assigned to each set of monitors. These steps include:

1. Validate the existence of programmable component interfaces which are part of the CS.
2. Verification of each component
3. Assign each calling component a set of 5 independent monitors with appropriate metrics and characteristics.
4. Returning allocated monitors to communicate with the associated component.
5. Associating all operations of the component with the allocated monitors.
6. Establishing monitor-component communication guidelines to facilitate interaction with the rest of the CS.

The following lists each specific component category identified within a CS and their monitor definition via the monitor engine along with the applicable monitor classifications and also outlines the six stage process discussed above.

3.4.1. Secure Hardware (SHW)

Hardware often comprises both physical and digital security. Hardware is often the first to be instantiated and the last to be accessed during the lifecycle of operation within a CS. Figure 4 graphically illustrates the monitor operations during this stage. Important to note is that each hardware component is represented as an individual component within the CS and is assigned its own set of monitors which ascertain to its own secure operation.

The following describes the monitor characteristics:

- Class – Can be system only or system wide class to allow for system only use or user enabled system use.
- Mode – Can be in analysis or tracking modes to allow for system intervention or passive observation.
- Access Level – Can be system, normal, or restricted access to resources and other component monitors.
- Trust Level – At this component level, must operate with any of the functional trust category levels.
- Scope – At this component level, must always have a global scope.

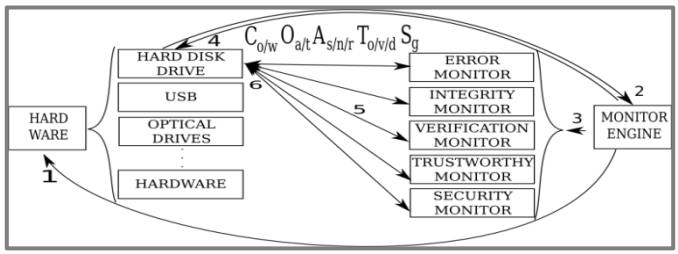

Figure 4: Secure Hardware

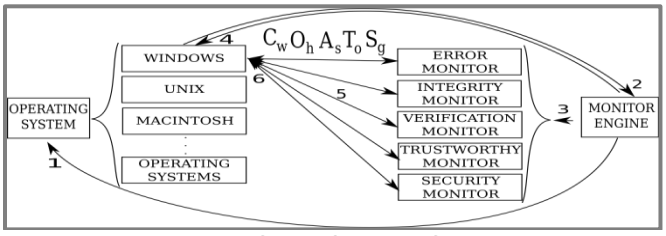

Figure 5: Secure Operating System

3.4.2. Secure Operating System (SOS)

The OS often serves a wide array of functional uses within a CS. The proposed work views an OS as just another component. There are two main reasons for this abstraction; namely, the OS is viewed as providing some functionality to the end user and is required to not only secure itself, but is also responsible for securing all other components. As an OS is just code, its reliability is underpinned by its underlying operating code. The proposed framework ensures the adherence of the OS's underling code to the defined security model requirements. Second, as an added outcome, the proposed approach could be designed to potentially allow multiple OSs to execute side by side simultaneously, as applications providing functionality without resorting to virtualization. Figure5 graphically illustrates the monitor operations during each stage.

The following describes the monitor characteristics:

- Class – At this component level, must only operate in a system wide class.
- Mode – Are always hybrid to allow for runtime changes in operation.
- Access Level – Must have system level access to allow inter-monitor communications between components.
- Trust Level – Must operate with operational trust to ensure overall system security is maintained.
- Scope – At this component level, must always have a global scope.

3.4.3. Secure Services (SSV)

Services are defined as system functionality which are normally offered by a CS directly at start-up, with the possibility of user initiation or termination during run time. Some services are directly dependent on the availability of associated hardware components and due to this nature can be dependent on defining common monitors where associated components do not have any embedded operational code, or disparate monitors which operate on each independent set of operating code at each component level respectively. Figure6 graphically illustrates the monitor operations during each stage.

The following describes the monitor characteristics:

- Class – System services operate with a system only class. User services operate with a system wide class.
- Mode – Always operate in tracking mode to ensure tracking and future analysis for all service activity.
- Access Level – Operate as normal or restricted, based on the associated trust level.
- Trust Level – Can operate under any of the defined trust levels under either category except denied trust.
- Scope – At this component level, must always have a local scope.

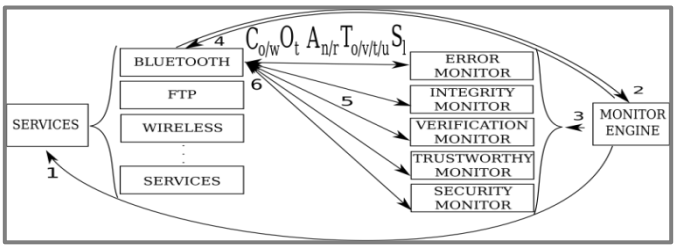

Figure 6: Secure Services

3.4.4. Secure Applications (SAP)

Applications are similar to services with the exception of the requirement to be initiated and terminated by a user. Figure 7 graphically illustrates the monitor operations during each stage.

The following describes the monitor characteristics:

- Class – Operate with system only or system wide class.
- Mode – Can operate under analysis or tracking modes.
- Access Level –Operate as normal or restricted access, based on the associated trust level.
- Trust Level – Can operate under any of the defined trust levels under either category.
- Scope – At this component level, must always have a local scope.

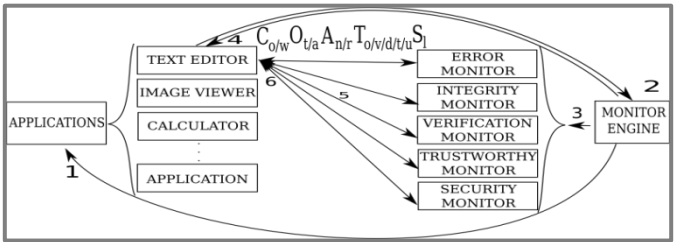

Figure 7: Secure Applications

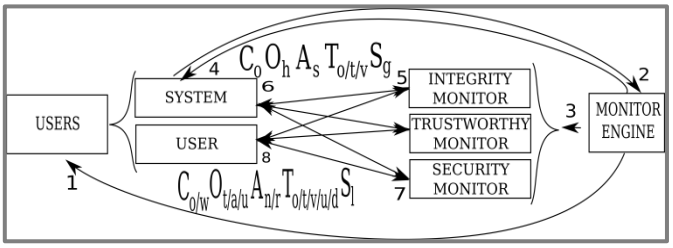

Figure 8: Secure User

3.4.5. Secure Users (SUR)

Users, in this context, can be viewed as either human users, other networked devices, or the system itself. Due to this blurred distinction this component operates on a different scale. The system user and the defined human users within the CS operate with disparate set of monitors. Also, due to the nature of the user component, only three monitors are defined for each user; namely: Integrity Monitor, Trustworthy Monitor, and Security Monitor. Figure 8 graphically illustrates the monitor operations during each stage.

The following describes the monitor characteristics:

- Class – System user operates as system only; user monitors operate as system only or system wide.
- Mode – System user as hybrid; user monitors operate as analysis, tracking, or user.
- Access Level – System user has system level access; user monitors have normal or restricted access.
- Trust Level – System user has a functional trust level; user monitors can have any of the defined trust levels.
- Scope – System user has a global scope; user monitors have a local scope.

3.4.6. Achieving Overall System Security

The overall system security can be defined as the summation of the security of each individual component involved. For simplicity, we can formally represent the security associated with each component independently so as to ensure it adheres to the security properties outlined for a secure CS represented as a set of disparate entities which provided for some functionality; defined as:

$$Sec(\Psi) = \{X_1, X_2, \dots, X_n\}$$

where $X \in \{Comp(x_i), Code(u_i), User\ (t_i)\} | i \in N$.

Furthermore, we can abstract the security of each individual component to the enforcement of each monitor associated with it; defined as:

$$Sec(Y_i) = \{EM(Y_i) \cup IM(Y_i) \cup VM\ (Y_i) \cup TM(Y_i) \cup SM(Y_i)\}$$

where $Y \in \{x_i, u_i, t_i\}, x \subset Comp(x_i), u \subset Code\ (u_i), t \subset User(t_i) |\ i \in N$.

Securing the CS can hence be reduced as the collective securing of each individual component, defined as:

$$Sec(\Psi) = \sum_{i=0}^{n} \{EM[W_i] \cup IM[Z_i] \cup VM[W_i] \cup TM[Z_i] \cup SM[Z_i]\} \quad (1)$$

where $W \in \{x_i, u_i\}, Z \in \{x_i, u_i, t_i\}, W = Z, i \in N$.

IV. EVALUATION

Whilst the proposed system does offer merits in terms of increased levels of security within the CS; both at the component level as well as the system level, it can be prone to a few lapses in terms of practicality and feasibility. In this section we address some of the issues pertaining to the framework and propose some alternatives to resolve those circumstances and further enhance usability.

4.1. Performance Degradation

One of the biggest challenges in securing CSs has been the trade-off in terms of performance. In computing, any increase in the number of instructions executed results in a gradual reduction in performance over time. Accepting this premise, the security provided by the ICS framework is the collective securing of each component in terms of the additional overhead as was proposed in Equation (1),

$$Sec(\Psi) = \sum_{i=0}^{n} \{EM[W_i] \cup IM[Z_i] \cup VM[W_i] \cup TM[Z_i] \cup SM[Z_i]\}$$

where $W \in \{x_i, u_i\}, Z \in \{x_i, u_i, t_i\}, W = Z, i \in N$, we can further represent each underlying monitor as a set of instructions which must be executed to enforce the applicability of the monitor; with O representing the total number of instructions,we can represent this as follows:

$$Sec(\Psi) = \sum_{i=0}^{n} \{O_{EM[W_i]} \cup O_{IM[Z_i]} \cup O_{VM[W_i]} \cup O_{TM[Z_i]} \cup O_{SM[Z_i]}\} \quad (2)$$

Due to the complexity of the associated parameters, statistical analysis is thereby limited, but quantification can be done in terms of executing code. For commodity systems executing n lines of code, the ICS framework requires the execution ofa proportionally larger number of instructions respective to $n$which can be represented Equation (2) as follows:

$$Sec(\Psi) = \sum_{i=0}^{n} \{\Omega_1 + n_i\} \quad (3)$$

where

$$\Omega_1 = \frac{O_{EM[W_i]}}{n_i} \cup \frac{O_{IM[Z_i]}}{n_i} \cup \frac{O_{VM[W_i]}}{n_i} \cup \frac{O_{TM[Z_i]}}{n_i} \cup \frac{O_{SM[Z_i]}}{n_i}$$

This can be further reduced by facilitating for abstraction of monitor enforcement at higher levels in the execution stack rather than lower levels. By enforcing the requirements of the monitors at the application level, the number of executions required to be executed at lower levels would be reduced and directly proportional to the increase in the size of n. If we let l be the number of added instructions we can represent Equation (3) as follows:

$$Sec(\Psi) = \sum_{i=0}^{n} \{\Omega_2 + (n + l)_i\} \quad (4)$$

where

$$\Omega_2 = \frac{O_{EM[W_i]}}{(n+l)_i} \cup \frac{O_{IM[Z_i]}}{(n+l)_i} \cup \frac{O_{VM[W_i]}}{(n+l)_i} \cup \frac{O_{TM[Z_i]}}{(n+l)_i} \cup \frac{O_{SM[Z_i]}}{(n+l)_i}$$

As we can observe, irrespective of the size of n, the ICS framework's additional overhead in terms of enforcing securityis directly proportional to the number of instructions whichit executes, thereby allowing for a scalable and deterministicmodel in terms of operation execution.

4.2. Security Analysis

Determining security as a quantitative metric has always been a challenge in theoretical computer science. To facilitate for this calculation, the ICS model incorporates a probabilistic model. For a skilled adversary, with sufficient time, knowledge, and resources; the probability of finding a security vulnerability can be represented as 1. However, through secure coding practices and property enforcement, the probability of a security vulnerability existing is directly dependent on the underlying operating code. Let us assume λ to be the defaultprobability of an error existing within a component's operating code. With this notion, if we accept the premise that the security provided by the ICS framework as the collective securing of each component proposed in Eq. (1), we can define the overall security of the system in terms of the operation of each monitor by transforming Eq. (2) as follows:

$$Sec(\Psi) = \sum_{i=0}^{n} 1 - \{\lambda_{EM[W_i]} \cup \lambda_{IM[Z_i]} \cup \lambda_{VM[W_i]} \cup \lambda_{TM[Z_i]} \cup \lambda_{SM[Z_i]}\} \quad (5)$$

Keeping in mind the traditional calculation of probabilities, with O representing the total number of instructions, we can define the calculation of λ as follows:

$$\lambda = \frac{1}{O} \equiv \frac{n}{O} \equiv \frac{n+l}{O} \quad (6)$$

By substitution of Eq. (6), we can transform Eq. (5) as follows:

$$Sec(\Psi) = \sum_{i=0}^{n} \{1 - [(n+l)_i . \Omega_3]\} \quad (7)$$

where

$$\Omega_3 = \frac{1}{O_{EM[W_i]}} \cup \frac{1}{O_{IM[Z_i]}} \cup \frac{1}{O_{VM[W_i]}} \cup \frac{1}{O_{TM[Z_i]}} \cup \frac{1}{O_{SM[Z_i]}}$$

From this we observe that, since all operational code must be executed in order to be functional, accounting for the additional overhead in terms of adherence to each of the outlined security properties, the decrease in performance is negligible in comparison to the increase in levels of security attained.

4.3. System Complexity

Previous models of monitor implementations, although effective, suffered from nested monitor calls [Andrew, 1977]. The ICS framework alleviates this problem; however, the nesting of monitors at various component levels and the intricacies of the inter monitor communications doesn't allow for a large margin of error in monitor operations. The requirement for a streamlined and structured process flow can be observed from the example process between components outlining a system start-up event within the ICS framework shown in figure 9.

Keeping in mind the three outlined stages of system operation, the process is started and terminated by a system user being initiated first, followed by the hardware components, the OS, the services, the applications, and lastly the human users. The creation and assignment of monitors to each component and the tracking of their operations can be overwhelming for single processor machines; however, with the advancement of technology and the development of parallel processing it remains a vital enhancement to bolster secure execution of operating instructions.

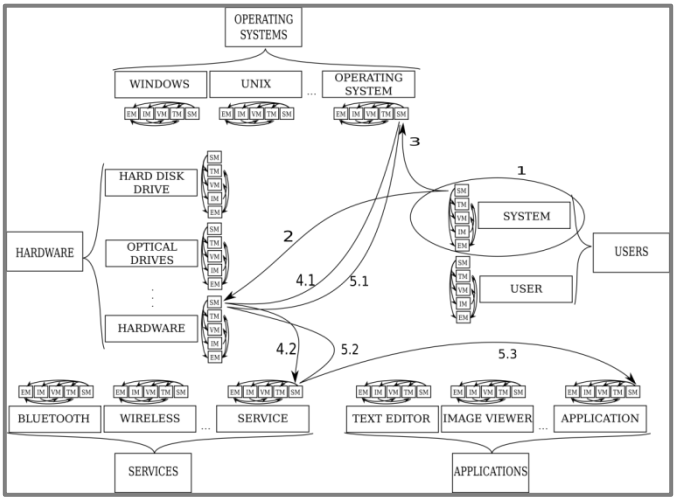

Figure 9: System Security using Monitors*
[*Dependent monitors for each component are detailed more elaborately in figure 2]

The ICS framework is designed so as to allow for compartmentalized isolation of operations and components and the designed monitors are defined with specific classes and scopes which collectively determine the monitor hierarchy within a CS. Inter-monitor complications can be avoided with the utilization of good programming techniques to code applications which do not rely extensively on code reuse for multiple sensitive applications, specifically for system operations. Furthermore, resolving this design consideration at a much lower level within the execution stack, such as the development of an underlying instruction set architecture and processing scheduler, could completely alleviate the challenges posed by a complex design structure.

4.4. Discussion

The proposed evaluation methodology presented can still be subject to criticism in regards to its approach of assessing security and performance associated with a CS in terms of its operating code being executed. It remains important to emphasize that proof methodology is also one of the most challenging aspect in theoretical computer science. The proposed model, at this point in time, aims at addressing the gap in the literature wherein a CS should be able to secure

itself from itself. With this premise, the goal of the evaluation methodology remains to ascertain a metric wherein only the additional overhead resulting from the operation of the various monitors impacts the overall performance and security of the system. Figure 10 depicts an ideal scenario which the ICS framework aspires to achieve.

Due to the overwhelming combinations possible from the various parameters accounted for within the set definitions for each monitor; the presented methodology has aimed to demonstrate the most optimum scenario. As we can observe from the graph, which is not to scale, that it remains possible to balance the additional overhead resulting from additional instructions being executed through monitor enforcement with minimal degradation in performance through higher level abstraction of monitor operations which can be enforced a adopting a more secure system architecture.

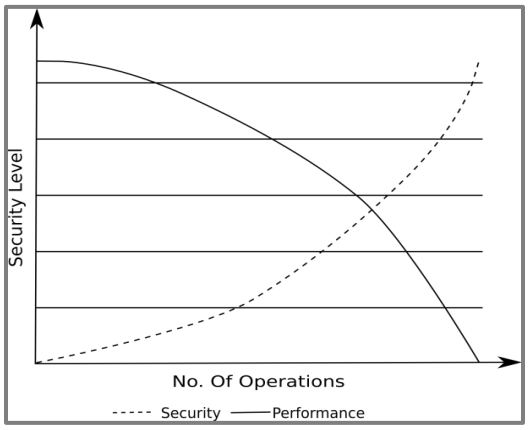

Figure 10: Security vs. Performance Tradeoff

V. CONCLUSION

CS security is a growing area of increasing importance. The existing infrastructure is still vulnerable as it relies on dated mechanisms to ensure secure operations by trusted entities. Despite many advances in protection mechanisms the main challenge has still been securing the system from itself. The quest for performance oriented computing has compromised the actual security offered.

This paper presented a novel concept of decentralized system security wherein the OS or any other component is not held responsible for securing the entire system. The ideology presented is that of a hybrid approach which uses structuring concepts at a more micro level within the CS so as to define a framework which allows each component to secure its own operations whilst adhering to a set of security properties. Furthermore, the generalization of these properties makes them more universally applicable rather than limiting to any specific platform or application. The evaluation of the framework presents a more realistic approach which is achievable but requires the redesign of the fundamentals which underpin our computing hardware and software. By defining all aspects of computing as instructions to be executed, the proposed work provides some symbolic relationship between performance and security in terms of number of operations wherein a reasonable quantifiable metric is deterministically achievable.

Revisiting Section II, we reinforce our main argument for the requirement of a CS to secure itself from itself. Although interaction with CSs has become more interactive, e.g. touch screens and voice inputs, the underlying foundations are still the same. The rate of progress of enhancing security is still not as proactive as it should be. The proposed model has addressed three vital areas of computing fundamentals. Firstly, it addresses the need to enhance security at lower levels within a CS by addressing the most basic fundamental unit of operation. Secondly, it addresses the missing link in computing wherein security is currently viewed either in terms of data confidentiality or data integrity but not necessarily both. Thereby facilitating for a bridged model between purely mathematical models and the real world systems, which are require to be more user friendly. And lastly; it addresses the ambiguity of the notion of security and trust in CSs by providing a benchmark for evaluating the requirements for secure operations, supported by a logically sound model which separates and justifies the inclusion of security enforcing characteristics.

The author's outcome for this work is the hope for defining a new path towards envisioning secure systems with this paper being a stepping stone towards a more secure modern day computing architecture. Planned future work includes the formal definition of the security properties and their relationship to a CS wherein all operations and components are reduced to their underlying operating code. The evolving trust paradigm is in progress to determine a deterministically viable method for calculating the level of trust for an entity within a CS. The last aspect includes the development of an underling instruction set architecture capable of supporting the proposed monitors and incorporating the concept of evolving trust via the means of a trust engine.

REFERENCES

[1] P.B, Hansen (1972), "Structured Multiprogramming", *Communications of the ACM*, Vol. 15, No. 7, Pp. 574–578.

[2] P.B. Hansen (1975), "The Programming Language Concurrent Pascal", *IEEE Transactions on Software Engineering*, Vol. 1, No. 2, Pp. 199–207.

[3] P.B. Hansen (1976), "The Solo Operating System: Processes, Monitors, and Classes", *Software: Practice and Experience*, Vol. 6, No. 2, Pp. 165–200.

[4] A.M. Lister & P.J. Sayer (1977), "Hierarchical Monitors", *Software: Practice and Experience*, Vol. 7, No. 5, Pp. 613–623.

[5] L. Andrew (1977), "The Problem of Nested Monitor Calls", *SIGOPS Operating Systems Review*, Vol. 11, No. 3, Pp. 5–7.

[6] R.J. Feiertag & P.G. Neumann (1979), "The Foundations of a Provably Secure Operating System (PSOS)", *Proceedings of the National Computer Conference, AFIPS Press*, Pp. 329–334.

[7] T.A. Berson (1979), "KSOS - Development Methodology for a Secure Operating System", *International Workshop on Managing Requirements Knowledge*, Pp. 365, URL: http://doi.ieeecomputersociety.org/10.1109/AFIPS.1979.68

[8] J. Rushby (1986), "The Bell and La Padula Security Model", URL: www.sdl.sri.com/~rushby/papers/blp86.pdf.

[9] J. Shapiro, et al., (1999), "EROS: A Fast Capability System", *SIGOPS Operating System Review*, Vol. 33, Pp. 170–185.
[10] P.G. Neumann & R.J. Feiertag (2003), "PSOS Revisited", *Proceedings of the 19th Annual Computer Security Applications Conference, IEEE Computer Society*, Pp. 208.
[11] Microsoft (2003), "NGSCB - Hardware Platform for the Next-Generation Secure Computing Base", URL: http://www.microsoft.com/resources/ngscb/documents/FNGSCBhardware.doc.
[12] Microsoft (2003A), "NGSCB - Trusted Computing Base and Software Authentication", URL: www.microsoft.com/resources/ngscb/documents/ngscb_tcb.doc
[13] Microsoft (2003B), "Security Model for the Next-Generation Secure Computing Base", URL: http://www.microsoft.com/resources/ngscb/documents/ngscb_security_model.doc.
[14] M. Abadi (2004), "Trusted Computing, Trusted Third Parties, and Verified Communications", *IFIP International Federation for Information Processing*, Vol. 147, Pp. 290–308.
[15] M. Abadi & T. Wobber (2004), "A Logical Account of NGSCB", *Lecture Notes in Computer Science*, Vol. 3235, Pp. 1–12, URL: http://dx.doi.org/10.1007/978-3-540-30232-2_1.
[16] M. Peinado, Y. Chen, et al., (2004), "NGSCB: A Trusted Open System", *Lecture Notes in Computer Science*, Vol. 3108, Pp. 86–97, URL: http://dx.doi.org/10.1007/978-3-540-27800-9_8.
[17] A. Martin (2008), "The Ten Page Introduction to Trusted Computing", *OUCL*, URL: http://www.cs.ox.ac.uk/files/1873/RR-08-11.PDF.
[18] J.R. James, F. Mabry, et al., (2009), "Secure Computer Systems: Extensions to the Bell-La Padula Model".
[19] G. Klein, K. Elphinstone, et al., (2009), "Sel4: Formal Verification of an OS Kernel", *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*, Pp. 207–220.
[20] S. Bratus, P.C. Johnson, et al., (2009), "The Cake is a Lie: Privilege Rings as a Policy Resource", *Proceedings of the 1st ACM workshop on Virtual machine security*, Pp. 33–38.
[21] Trusted Computing Group (2013), "Trusted Platform Module Specifications in Public Review", URL: http://www.trustedcomputinggroup.org/files/static_page_files/A8B4F2F6-1A4B-B294-D0F8BF38665BFD02/Public%20Review_TPM%20Rev%202.0%20Part%201%20-%20Architecture%2000.99.pdf.
[22] O.M. Creado, B. Srinivasan, et al., (2014), "An Explicit Trust Model Towards Better System Security", *The Fourth International Conference on Computer Science and Information Technology*, Pp. 139–151.
[23] O.M. Creado, B. Srinivasan, et al., (2014A), "A Concept Framework for Decentralizing Computing System Security", *The Second International Symposium on Computer Science and Electrical Engineering*, Pp. 294–299.

Orhio Mark Creado received a B.S in Computer Information Systems from Missouri State University and a Masters of Information Technology Professional (Minor Thesis) from Monash University specializing in cryptographic algorithms. He is currently working towards a PhD degree at Monash University in the Caulfield School of Information Technology. His research interests include: System Security, Cryptography and Cryptology, and Network Security. He has had diverse industry experience in many domains within the IT industry which has supported his venture into computer system security; and currently tutors at Monash University in the disciplines of Networking, Security, Database Management, and Data Mining.

Bala Srinivasan is a Professor of Information Technology in the Faculty of Information Technology, Monash University, Melbourne, Australia. He has authored and jointly edited 6 technical books and authored and co-authored more than 200 international refereed publications in journals and conferences in the areas of Databases, Multimedia Retrieval Systems, Distributed and Mobile Computing and Data Mining. He has successfully supervised 28 research students of which 15 of them are PhDs and, his contribution to research supervision has been recognized by Monash University by awarding him the Vice-Chancellors medal for excellence. He is a founding chairman of the Australasian database conference which is now being held annually. He holds a Bachelor of Engineering Honours degree in Electronics and Communication Engineering from University of Madras, a Masters and a PhD; both in Computer Science, from Indian Institute of Technology, Kanpur. Currently he is in the editorial board of two international journals and program committee member of nearly dozen international conferences.

Phu Dung Le is a Lecturer in the Faculty of Information Technology, Monash University, Melbourne, Australia. His main research interests are: Image and Video Quality Measure and Compression, Intelligent Mobile Agents, Security in Quantum Computing Age. He used to teach Data Communication, Operating System, Computer Architecture, Information Retrieval and Unix Programming. He has also researched in Mobile Computing, Distributed Migration. Currently he is lecturing network security and advanced network security in addition to supervising PhD students.

Jeff Tan was a lecturer and systems administrator at De La Salle University in Manila for six years, where he also obtained his Master's degree, prior to moving to Melbourne in 1999 to study at Monash University. After obtaining his PhD and a year as a systems administrator, he joined the Faculty of Information Technology in 2004 and pursued research with the Monash e-Science and Grid Engineering Laboratory and the Centre for Distributed Systems and Software Engineering. He additionally worked part-time with the Monash e-Research Centre from 2008 as a software specialist. He joined IBM Research in 2013 as a high-performance computing specialist, administering an IBM Blue Gene/Q for the Victorian Life Sciences Computation Initiative, but maintains ties with Monash University on an adjunct appointment. His research, which has evolved over the years, includes network management, network security, distributed and high-performance computing. He has published in the Journal of Grid Computing, Concurrency and Computation: Practice and Experience, and in e-Science, among others.