Proceedings

8th Layered Assurance Workshop

LAW 2014

Proceedings

8th Layered Assurance Workshop

New Orleans, Louisiana, USA
8–9 December 2014

Sponsored by
Applied Computer Security Associates

Editorial production by Adam Hahn

Table of Contents

Technical Program ..... **v**
Message from the Workshop Chair ..... **vi**
Message from the Program Chair ..... **vi**
Organizing Committee ..... **vii**
Program Committee ..... **vii**

Technical Program

Contributed Papers

Assurance for Defense in Depth via Retrofitting 1
Vinod Ganapathy, Trent Jaeger, Christian Skalka, Gang Tan

Towards High Assurance Platforms for Securing the Cloud with Disparate Domains 11
Sujay Doshi, Roger Schell

A Cost Effective High-Assurance Layered Solution for MLS Test, Training, and Live Virtual Constructive (LVC) 20
James A. Marek

An Integrated Framework for Multi-layer Certification- based Assurance 26
Rajesh Harjani, Marcos Arjona, Javier Espinar, Antonio Maña, Antonio Muñoz, Hristo Koshutanski

Works In Progress

Layered Assurance of Security Policies for an Embedded OS 32
Jia Song, Jim Alves-Foss

Towards a Highly Secure PLC Architecture for CPS 33
Nikhil Mahesh

Mapping Security Policies to Implementations 34
Jim Alves-Foss

Empirical Evaluation of API Usability and Security 35
Sam Weber, Robert Seacord, Forrest Shull, David Keaton, Brad Myers, Michael Coblenz

Message from the Workshop Chair

Welcome to participants in the 8th Layered Assurance Workshop. The LAW organizers strive each year to ensure that your interest in LAW is repaid with benefits. As security and safety are becoming increasing concerns is a wide range of application and cyber-physical systems the relevance of our ongoing work in assurance is becoming increasingly relevant to a broader audience. LAW represents a continuing effort to identify, from the experiences of consumers and producers of critical systems, successful strategies and practices, and as yet unfulfilled needs for more cost-effective assurance.

LAW provides a forum for exchange among customers, developers, evaluators, certifiers and researchers. It is our intent and hope that this effort will yield fruit from the synergies of these communities. We look forward to reports in future editions of LAW of the results of research, development, and practice inspired by past and present editions of LAW that may be adopted into the assurance practices of developers, evaluators and certifiers of future systems. We strive to provide a program for LAW that has a balance of opportunities to present needs, describe ongoing work, report advances, gain insights from thought leaders, interact and make new contacts.

As LAW chair I am most grateful for the steadfast contributions of Peter Neumann, Gabriela Ciocarlie, the program committee members, Adam Hahn proceedings chair, and perennial LAW participants. I hope you will join me in thanking them for our program and proceedings, and for the continuing existence of LAW. Best wishes for an enjoyable and productive Layered Assurance Workshop.

Rance J. DeLong, LAW 2014 Workshop Chair

Message from the Program Chair

It is my pleasure to welcome you to the 8th Layered Assurance Workshop, hosted with ACSAC again in New Orleans, Louisiana. A total of 4 research and 4 work-in-progress papers were accepted (same as last year), with each paper receiving three reviews. I would like to thank the authors for contributing their work and, in what is quickly becoming a LAW tradition, I am looking forward to engaging discussions during the presentations. I would also like to thank our LAW Program Committee members for their continuous support and participation in the workshop. It has been a privilege to work together. Moreover, I would like to thank Adam Hahn, the new LAW Proceedings Chair, for compiling the proceedings and for kindly agreeing to be part of the LAW family. LAW will also continue its tradition in offering exceptional invited talks and panels, thanks to Rance DeLong, the LAW Chair, and Peter G. Neumann, the Law Panel Chair, who manage to always find the perfect composition (pun intended!) for a successful workshop. It has been a pleasure to work together, and thank you for your outstanding contributions.

Gabriela F. Ciocarlie, LAW 2014 Program Committee Chair

Organizing Committee

Rance J. DeLong, Santa Clara University (General Chair)
Gabriela F. Ciocarlie, SRI International (Program Chair)
Peter G. Neumann, SRI International (Panels Chair)
Adam Hahn, Washington State University (Proceedings Chair)

Program Committee

Gabriela F. Ciocarlie, SRI International (Program Chair)
Rance J. DeLong, Santa Clara University
Peter G. Neumann, SRI International
Gordon Uchenick, Coverity, Inc.

Assurance for Defense in Depth via Retrofitting

Vinod Ganapathy
Rutgers University
vinodg@cs.rutgers.edu

Trent Jaeger
The Pennsylvania State University
tjaeger@cse.psu.edu

Christian Skalka
University of Vermont
skalka@cems.uvm.edu

Gang Tan
Lehigh University
gtan@cse.lehigh.edu

ABSTRACT

The computer security community has long advocated *defense in depth*, the concept of building multiple layers of defense to protect a system. Unfortunately, it has been difficult to realize this vision in practice, and software often ships with inadequate defenses, typically developed in an ad hoc fashion. Currently, programmers reason about security manually and lack tools to validate assurance that security controls provide satisfactory defenses. In this position paper, we propose STRATA—a holistic framework for defense in depth. We examine application of STRATA in the context of adding security controls to legacy code for authorization, containment, and auditing. The STRATA framework aims to support a combination of: (1) interactive techniques to develop *retrofitting policies* that describe the connection between program constructs and security policy and (2) automated techniques to produce optimal security controls that satisfy retrofitting policies. We show that by reasoning about defense in depth a variety of advantages can be obtained, including optimization, continuous improvement, and assurance across multiple security controls.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection

General Terms

Design, Experimentation, Management, Security

Keywords

Automated Retrofitting, Defense in Depth, Authorization, Containment, Auditing, Assurance

1. INTRODUCTION

The security community has long encouraged programmers to strive to implement *defense in depth*, where multiple layers of security controls are employed to protect security-sensitive operations. However, programmers almost always focus on functional and economic issues initially, often delaying the introduction of needed security controls until after initial deployment. As a result, programmers often find themselves in the so-called penetrate-and-patch mode, removing vulnerabilities as they are identified by adversaries.

Even when programmers decide to add the security controls necessary to implement defense in depth, they face many practical challenges. First, retrofitting software with a security control requires a comprehensive understanding of the program code and security requirements to integrate the security controls correctly. For this reason, past projects that retrofit software manually for attack containment [55, 29, 57], authorization [76, 72, 20, 4, 70, 52, 33, 12, 58], or auditing [5, 3] often span multiple years. Gaining consensus over whether the implementation of a security control is satisfactory is ad hoc [61, 77, 60], and mistakes have been made [21, 78], some of which were not discovered until years later [67]. Second, retrofitting software with security controls must account for several other factors beyond security. Naive containment implementations can cause tremendous performance overheads and naive authorization mechanisms can result in spurious policy management. As a result, many attack containment projects, particularly for performance-critical software such as operating systems, have been abandoned and some authorization mechanisms have been refined several times after introduction [51]. Third, security controls may interact, making other controls unnecessary or impacting their placement. For example, if authorization prevents data leakage provably (noninterference [27]), then there is no need to audit for data leaks. However, programmers lack tools to reason about such interactions, resulting in unnecessary or misplaced security controls.

Researchers have recognized that programmers need assistance in retrofitting their programs with security controls, but the proposed automated methods still require too much programmer effort, fail to account for factors other than security, and do not reason about multiple security controls. First, researchers have proposed automated methods to detect missing or misplaced authorization controls [78, 21, 67, 63, 65, 53, 66] or even retrofit programs for authorization [26, 46, 39, 64], but these methods require detailed program knowledge, such as security-sensitive data types, variables, and/or statements, or require a partial authorization implementation. Second, methods to retrofit for authorization and attack containment [37, 10] have been naive about the impact of security controls on other factors, such as the performance and management overheads, preventing wide adoption. Third, we are not aware of any prior work that reasons about retrofitting for auditing or for multiple security controls and their interactions.

Our insight is that advances in retrofitting software for security enable the development of a holistic framework for the assurance

of security controls for defense in depth. While retrofitting programs for security is a challenging problem for any security control, recent advances in methods for retrofitting programs for security demonstrate what can be automated and how, distinguishing what can be computed from what intelligence programmers need to supply. The proposed approach takes a comprehensive view of the problem, with an emphasis on *automated* and *interactive* tools that developers can use to identify site-level security goals, explore the design space of security mechanisms, and retrofit legacy code to enforce security policies in a manner that can be machine-verified for assurance.

In this position paper, we examine the unification of three common security mechanisms — containment, authorization, and auditing — to assess how reasoning about defense in depth encompassing these three mechanisms may improve security assurance. First, we find that placing security controls for these mechanisms involves solving a set of common problems, so designing methods to solve those problems and to verify the effectiveness of the solutions may be reused. Second, we find that we can compose the validation of defense in depth for this combination of security controls, enabling assurance for defense in depth. Third, we find that runtime auditing can be leveraged for continuous improvement of the placement of security controls for defense in depth, ensuring that the controls can be optimized for the desired policies. We refer to completed research results where available, but a goal of this paper is to motivate reuse of common ideas across controls and integration of controls for improved security.

The remainder of the paper is structured as follows. In Section 2, we examine the problem of designing programs to control access to program and system resources using multiple security controls. In Section 3, we provide an overview of how to use automated retrofitting of programs to produce validated security controls for defense in depth. In Sections 4 to 6, we explore the challenges in retrofitting programs for containment, authorization, and auditing independently. In Section 7, we outline the problem of unifying retrofitting methods for defense in depth and examine opportunities for assurance of defense in depth, including continuous improvement. In Section 8, we conclude the paper.

2. BACKGROUND

2.1 What Should Retrofitting for Defense in Depth Do?

When program vulnerabilities become too numerous, programmers may be motivated to make fundamental changes to their programs to add security controls. For Sendmail and OpenSSH, programmers found that the typical penetrate-and-patch approach to security was not keeping them ahead of adversaries, leading to complex retrofitting [55] or complete reimplementations [54, 6]. For programs that process resources belonging to multiple clients, such as servers and middleware, programmers often found that simple isolation approaches (e.g., sandboxes) were insufficient to protect data security and provide necessary functionality [16, 22]. We use the simple program below to demonstrate the problems.

```
request_loop (client_data, private_data) {
  read(client_passwd, client_req );
  if (necessary ||
      compare_client(client_passwd,
                     private_data))
    access_object(client_req, client_data);
}
```

The client request loop above is representative of many programs that require retrofitting. This program processes requests from multiple, mutually-untrusting clients (obtained by `read`) by: (1) comparing a client-supplied password (`client_passwd`) to the program's password database (`private_data`) in `compare_client` and (2) processing a client request (`client_req`) to access data managed by the program (`client_data`) in `access_object`. In this discussion, we assume that the program code is benign, but may have flaws that allow client input read by the program to permit unauthorized access. The first operation may cause vulnerabilities if the program allows client input to affect the program's passwords or if some password data is leaked as a result of the comparison. The second operation may cause vulnerabilities if it allows any client unauthorized access to the client data of another client. Many programs perform these two types of operations, including operating systems, middleware, server programs, and even some user applications. For example, operating systems process many client requests (e.g., system calls) and process private operating system data that must not be manipulated by clients. On the other hand, browser applications also run programs from multiple sources (i.e., the browser's clients), so they must control access to browser resources available to those programs and protect their private resources from leakage and unauthorized modification.

In this discussion, we will focus on retrofitting programs to control client access to *security-sensitive operations*, such as those in the program above that use the program's private data and client data.

We examine three kinds of security controls that are commonly used to achieve this goal. First, programmers may use *containment* to place protection boundaries that limit the ways that clients may access security-sensitive data. For example, the program above may be privilege-separated [59] into two modules running in separate processes: (1) one that receives client requests and provides access to client data using `access_object` and (2) another that runs `compare_client` that has access to the private data. Clients can only communicate directly with the first module, limiting the program flows that may reach or leak the private data.

Second, programmers use *authorization* to control access to program data. For example, the program above may be retrofitted with a reference validation mechanism that satisfies the reference monitor concept [7] to ensure correct enforcement of an access control policy governing which clients may access which client data and preventing leakage and unauthorized modification of private data, regardless of the complexity of the code in the `compare_client` and `access_object` functions. Reference validation mechanisms must be designed to enforce the data access policies expected by the programmer, whose goals may include least privilege [59], lattice policies [17], noninterference [27].

Third, programmers use *auditing* to collect information to aid intrusion detection retroactively for authorized operations. For example, clients authorized to run `compare_client` may still cause the private data to be leaked through some program flaw, so auditing could record the values of the authorized operation and the data returned to the client to enable later detection of whether leakage occurred. As can be seen, these security controls form three layers of defense, where containment limits client access at the boundaries, authorization within the program, and auditing follows authorized operations.

2.2 State-of-the-Art in Retrofitting Programs for Defense in Depth

Programmers retrofit programs with containment [55, 29, 57], authorization [76, 72, 4, 70, 20], and auditing controls [5, 3] manually, which presents a variety of challenges. First, programmers

must identify security-sensitive operations from low-level program constructs, such as variables, data types, and statements. While the program above may be simple, real programs have hundreds of user-defined types and thousands of program statements. Despite the availability of several prototype reference monitor implementations, the Linux Security Modules framework [76] still contained several errors [21, 78], even some that were not discovered until years later [67]. Second, programmers must determine where to apply controls to protect those security-sensitive operations, but they must be careful not to introduce high performance and management overheads. Programmers currently balance such functional and security requirements in an ad hoc way. If the variable `n`ecessary in the `r`equest_loop is usually true, then separating `compare_client` may be satisfactory, but otherwise large overheads may be incurred. As a result, retrofitting projects take several years, face delays that bring their purpose into question [61], and may reduce the scope of security controls to only known attacks [55]. Because of these challenges, only a few programs have been retrofit with all three security controls we investigate in this project.

Researchers have recognized the challenges facing programmers, but to date fail to address the most fundamental of those challenges. First, proposed methods to retrofit code still require programmers to identify security-sensitive operations from low-level code constructs, such as code patterns, data types, and variables that correspond to such operations [26, 63, 39, 10]. For example, to automate placement for information flow control, all the relevant variables must be identified and assigned an accurate security label. Second, most current research only addresses functional concerns implicitly if at all. For example, some prior work aims to produce a "minimal" number of security controls [39, 50], but these methods still introduce far more controls than added manually. Recent research has proposed a retrofitting method that uses functional and security constraints as input [30, 31]. However, such constraints are written as traces in terms of program locations, still requiring significant program knowledge to get right (e.g., context sensitivity). Finally, none of the prior methods retrofit software with multiple security controls, possibly introducing spurious security code.

2.3 Goals for Retrofitting Programs for Defense in Depth

The goal is to develop a method that programmers can use to retrofit their programs with security controls for containment, authorization, and auditing that satisfy explicit security goals (e.g., policies to enforce) and are globally optimal relative to functional costs. Thus, we have two broad challenges: (1) develop the theory and techniques to retrofit multiple security controls optimally from program code, security goals, and functional concerns and their costs; and (2) reduce programmers effort for producing security goals and the costs of functional concerns sufficient to achieve desirable security in practice. Based on the limitations of prior research, we highlight the essential questions presented by these challenges:

- Can we design methods for identifying security-sensitive operations and security goals for programs that do not require detailed, manual analysis of program code?
- Can we design methods to retrofit programs with containment, authorization, and auditing security controls that enable verification that each type of security control enforces a program's security goal with respect to that program's security-sensitive operations?
- Can we design methods to optimize the functional cost of satisfying a security goal across containment, authorization, and auditing simultaneously?

Thus, an ideal retrofitting method would extract security-sensitive operations from `request_loop` and relate those operations to security goals, with detailed, manual code inspection or annotation by programmers. Using the security-sensitive operations and security goals, this ideal method should produce a retrofitted program that consumes minimal functional cost and verifiably satisfies those security goals for the security-sensitive operations.

3. APPROACH OVERVIEW

The goal is to retrofit a program to add a series of defensive security controls to protect program data from unauthorized access, specifically containment, authorization, and auditing. While there are many differences among these security controls, we find that retrofitting these security controls into programs requires solving four common problems:

- *Finding security-sensitive operations.* Each security control aims to mediate access to security-sensitive operations for different purposes (e.g., defining protection boundaries or logging such operations). While security-sensitive operations may differ for individual controls, we find that such operations share the ability to direct execution among unsafe choices. We propose a method based on finding the program statements where control and data "choices" are made using input from untrusted sources [50].
- *Relating security-sensitive operations to security goals using retrofitting policy.* We have found that simply mediating every security-sensitive operation creates unnecessary overhead for performance and policy management. Instead, programmers need a way to relate security goals to security-sensitive operations that does not require detailed, manual analysis of program code. We propose a method that programmers use interactively to find relationships between security-sensitive operations based on their impact on satisfying security goals [51], which we call a *retrofitting policy*.
- *Place controls for security goals.* Given a retrofitting policy, the goal is to transform the program to satisfy that policy while minimizing cost. While different transformations are applied for different types of security controls, choosing where to place security controls requires complete mediation of relevant program flows in all cases. We propose to explore use of *program dependence graphs* [24] (PDGs) for reasoning about control and data flows uniformly for all security controls.
- *Verifying correct transformations.* Despite the use of different methods for placing transformation and distinct transformation primitives, each method transforms code to mediate security-sensitive operations for a security goal. We explore how to leverage formal methods, so that high assurance is obtained from our retrofitting framework. We plan to verify the correctness of the transformation methods proposed by building proofs of correctness inside Coq [14].

Figure 1 presents an overview of our proposed STRATA framework, which aims to implement the methods described above. The STRATA framework enables programmers to retrofit their programs with containment, authorization, and auditing security controls in two steps. In the first step, programmers interactively develop retrofitting policies for each of these security controls, leveraging methods to find security-sensitive operations and relate those operations to security goals for retrofitting policy. In the second step, automated methods transform programs with security controls to

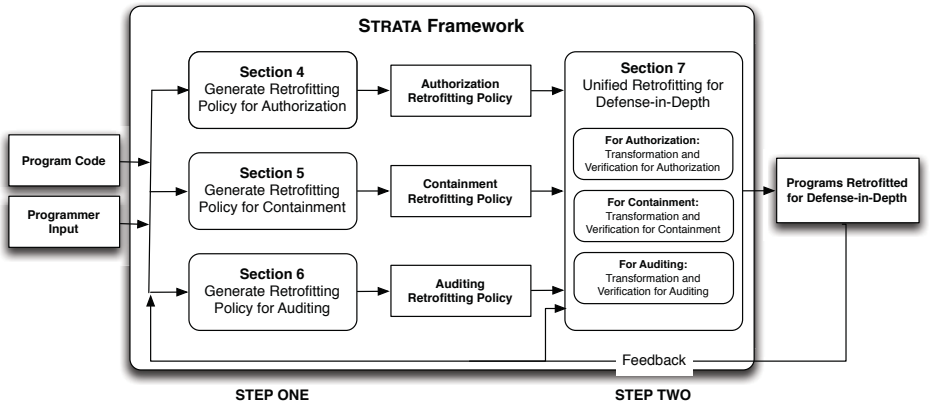

Figure 1: STRATA framework for assurance of security controls for defense in depth

satisfy a composition of those retrofitting policies while minimizing cost and verify the correctness of such retrofitting across all three security controls.

4. RETROFIT FOR AUTHORIZATION

Historically, there are two kinds of related, but distinct, security models for authorizing operations on data. First, *access control* mediates access to a program operation based on who requests the access (subject) and what the data is (object), as well as the operation itself. In the `request_loop`, access control limits which clients can perform which requests (operations) on which objects in `access_object`. Access-control policies can be represented by access-control lists or capabilities of access-control matrices [44], which are intuitive to understand and easy to enforce. As a result, they are widely adopted in programs such as servers and operating systems. The reference monitor concept [7] defines the requirements for enforcing an access control policy correctly: complete mediation, tamperproofing, and verifiability. For example, complete mediation requires the placement of *authorization hooks* in programs that mediate all security-sensitive operations [76, 72, 20, 4, 70, 33, 58, 52]. However, even complete mediation has been difficult to ensure because programmers do not identify security-sensitive operations explicitly.

Second, *information flow* tracks the propagation and release of data through the program. Rather than being focused on program operations, information flow is concerned with how data associated with particular security properties may influence other data values, causing security-critical data to be modified by untrusted data or leaked to untrusted subjects. In the `request_loop`, information flow control limits how private data (e.g., password database) may be used to prevent its leakage when comparing it to client input (e.g., input password) in `compare_client`. These influences are either through explicit data flows or implicit flows [17], in which data is leaked via control structures of programs. For security, hooks must mediate all unauthorized information flows using declassifiers (for secrecy) and endorsers (for integrity) that remove offending data from those flows.

We aim to retrofit software for both access and information flow control. Manual placement of authorization hooks and mediation for information flow control is laborious [32, 38, 61] and error-prone [21, 78, 67], so we propose to develop semi-automated methods for this purpose.

The substantial limitations of previous systems are that they offer little help to programmers for identifying security-sensitive operations or the relationships between security-sensitive operations and security requirements, which we call a *retrofitting policy*; previous systems often assume the retrofitting policy to be a manual input [30, 31, 46]. The reality, however, is that very often programmers have only a rough idea about the security implications of individual program statements and are unwilling to spend time to identify such information manually. We believe that techniques that help programmers discover retrofitting policies are needed.

Approach Overview. We define a retrofitting policy to be a set of *connections* relating basic *constructs*. For access control, basic constructs are the program statements that correspond to security-sensitive operations; an example connection is the operation subsumption between two operations, meaning that authorization of the first operation always implies the authorization of the second. In an information-flow policy, the basic constructs are also security-sensitive operations, which are those program statements that operate on sensitive information at data sinks. Subsumption is also an example of a connection between sinks meaning that the declassification (sanitization) of the first sink always implies declassification (sanitization) of the second.

In our recent work [50], we proposed a technique that infers security-sensitive operations for access control from the least amount of programmer input of any known method. A similar method has since been proposed for Android analysis [47]. Our method requires only the identification of language-specific lookup functions and the sources of untrusted input. Its inference technique is based on a key observation: security-sensitive operations correspond to the deliberate choices the program makes using client input for retrieving data from data collections and for selecting the conditional code paths for processing that data. Therefore, the technique tracks the "choices" made by client requests to automatically infer security-sensitive data and operations. Experiments performed on programs such as `X server` and `postgres` demonstrate this technique is effective at identifying almost all security-sensitive operations. We compared our results with manual hook placements by experts.

However, identifying security-sensitive operations alone is not enough for efficient hook placements. Human experts often remove unnecessary hooks using domain knowledge. As a result, the only way to reduce the number of hooks is to make the domain knowledge explicit. In an ongoing work [51], we make the domain knowledge explicit through the definition of retrofitting policies as connections between security-sensitive operations. We have identified two kinds of connections: operation subsumption and operation equivalence. We have discussed operation subsumption before. Operation equivalence means that two operations are always authorized for the same set of subjects. For multilevel security [9] (MLS), two operations that read the same object are equivalent be-

cause the same set of subjects will also be authorized. These connections are utilized for removing unnecessary hook placements. For instance, if operation one subsumes operation two and if operation one dominates operation two in control flow, then the authorization hook for operation two is unnecessary (assuming an authorization hook for operation one is already there). Preliminary experiments on a variety of software, including the X server, databases, and the Linux kernel, demonstrate our methodology can reduce programmers' effort for discovering their intended policies and can already reduce the number of access control hooks by 30% [51].

To further improve the reduction, more automatic methods are needed to discover retrofitting policies. To help programmers find relevant connections, STRATA must provide security and performance constraints and analyses that find pairs of security-sensitive operations that satisfy those constraints to suggest connections automatically (e.g., must enforce MLS policies, described above). If programmers agree with the high-level constraint, then they can select the resultant connections as a group, rather than one at a time. To date, we have proposed two constraints, one for security and one for performance. Given this experience, we feel it is necessary to investigate the effectiveness of other constraints and other methods for using those constraints. For example, roles [23] define groups of permissions that are authorized for the same subjects, enabling the computation of equivalence relations. Sun *et al.* require role specifications as input [66], but probably role mining [25, 49] will be useful for this problem.

We also feel that this approach applies to the setting of information flow. However, mediators for information flow perform different tasks than access control hooks, so the semantics of their connections will differ. For example, declassifiers allow subjects to receive a subset of the flow's data. Further study is needed to investigate techniques to suggest possible types of declassifiers to programmers, based on types of sources and sinks and types of data that flow between them. One example suggestion is to say that flows between source s1 and sink d1 and between source s2 and sink d1 should use the same declassifier. Given these suggestions, programmers then decide what declassifier to apply and provide the actual declassification code (for example, the code for sanitizing SQL strings).

In addition, Strata needs multiple program analysis and transformation techniques to place authorization hooks automatically in programs. Given a retrofitting policy, the next step is to compute a minimal cost placement for authorization code that satisfies that policy. Previous work on access control hook placement relies on control dependence, whereas previous work on mediator placement relies on data dependence. However, information flow control often leads to fine-grained, non-intuitive mediation requirements, particularly for implicit flows [38]. In general, we believe considering both control and data dependence will be synergistic, but how exactly they interact for better hook placement will be a research question (especially in the case of implicit flows).

We feel that a uniform program representation, called program dependence graphs (PDGs [24]), will help this problem. A program's PDG represents both the control dependence and data dependence of the program and can be extracted using efficient program analysis. We believe the benefit of PDGs is that it will enable a unified framework to compute better hook placements for a variety of security goals, from access control to explicit information flows to implicit information flows. For access control, the framework uses mostly the control dependence to reduce the number of hooks, but data dependence can be used to satisfy complete mediation without blocking authorized operations. For controlling explicit flows, the framework considers data-dependence edges to insert code that tracks data flows and taint checks or declassifiers at appropriate places. Implicit flows can be taken into account by considering both control and data dependence [39].

5. RETROFIT FOR CONTAINMENT

Experience shows that despite our best efforts at improving software security, adversaries may bypass defenses to achieve malicious goals. This is because software is often retrofit for security manually, using ad hoc techniques. These techniques are error-prone, and may leave avenues that adversaries can later exploit [21, 78, 67]. Vulnerabilities may remain despite our best efforts to retrofit software for security, especially if the requirements used during the retrofitting process evolve over time [34, 43]. Robust software assurance must therefore include a layer of defenses that confine adversaries, even if the system is compromised.

Much of today's software is not written with the goal of confining adversaries. Most server applications as well as systems software are written as monolithic artifacts. Vulnerabilities in such systems have been exploited by adversaries to gain control over the entire server [55]. Until about five years ago, web browsers were also designed as monolithic systems. In such designs, the browser kernel, script parsers, renderers, and third-party plugins ran within the same protection domain. This design lead to many security attacks, wherein a vulnerability in a plugin often gave an adversary complete control over the browser [28, 71, 68, 56]. Motivated by such attacks, the browser industry has shifted its focus to *compartmentalized* or *modular* designs, in which browser subsystems execute within different protection domains. For example, Google Chrome uses different OS processes to sandbox web content on each tab, and also creates new OS processes to execute plugins and other third-party browser content.

Strata also follows this approach, and aims to automatically modularize software to enable attacker containment. Our overall goal is to retrofit a monolithic software system to adhere to two basic security principles: (1) *Privilege Separation*, which posits that resources that require different access rights must execute within different protection domains, and (2) *Least Privilege*, which posits that each module, running within its own protection domain, must only receive the privileges that it needs to accomplish its task. Together, these two principles ensure that the attack surface of the modularized software system is minimized, limiting the damage that an adversary can inflict if he were to obtain access to the system.

Strata will develop a number of techniques to retrofit software for attacker containment. First, it will provide a rich interface to specify resources that must be protected. Each of the specified resources will be contained within their own protection domain. Second, Strata will investigate efficient techniques to modularize software. The main performance cost of modularization is that method invocation, which is inexpensive in a monolithic software system, involves crossing protection domains. Strata rectifies this by optimizing for performance within the restrictions of the retrofitting policies. Third, in contrast to prior techniques that primarily used OS processes to define protection domains, Strata will consider a number of alternatives, including language-based protection, lightweight virtual machines, as well as enhanced OS APIs as protection domains, and will develop transformation techniques tailored to these domains. Strata will also include verification techniques that provide assurance on the correctness of the modularized code.

Approach Overview. In a retrofitting policy for software modular-

ization, we identify connections between security-sensitive operations (as done for authorization). However, in this case the concept of a connection has a negative connotation—A connection between two security-sensitive resources requires the two operations to be isolated in individual protection domains. Using the web browser as an example, a developer may identify a connection between the browsing history and network operations because browsing history should not be leaked over the network. The developer can then use the control and data dependencies from the PDG to iteratively identify statements in the software system that are connected and must therefore appear in separate protection domains.

In Strata, a developer additionally provides a performance cost model as part of the retrofitting policy. This model will allow developers to identify code paths that are frequently executed, e.g., using information gathered from runtime profiles, as well as the cost of crossing protection domains. The performance cost model identifies resources that could potentially be placed in the same protection domain to provide good performance in the modularized software system. The connections and cost model together provide a candidate retrofitting policy that is refined iteratively using code analysis. Such performance cost model can potentially be obtained automatically from profile information gathered at runtime.

Strata uses the candidate retrofitting policy identified above, together with static code analysis to identify possible module boundaries in the monolithic software system. This analysis has two goals. First, the modules identified by the analysis must satisfy all the connections, i.e., all program constructs manipulating connected resources *must* be isolated in separate modules. A program construct may be involved in accessing a pair of connected resources; in such cases, it may have to be replicated, with each replica serving one resource. Second, the overall performance cost of the modularization should be small. Program constructs manipulating unconnected resources can be co-located in the same protection domain, provided that the resulting costs of domain crossings is not excessive.

Strata casts the problem of identifying module boundaries as an optimization problem on the inter-procedural control-flow graph (CFG) of the monolithic software system. The edges of the CFG are annotated with weights from the performance cost model. The goal of the optimization then is to partition the graph into subgraphs so that: (a) nodes labeled with connected resources appear in separate subgraphs, and (b) the sum total of the edge weights crossing subgraphs is minimized. Nodes labeled with unconnected resources can appear in the same subgraph, and because there may be several such pairs of unconnected resources, the analysis has some flexibility in identifying module boundaries. Strata formulates this problem as one of finding a min-cost multicut in the directed graph denoted by the CFG [39].

Strata presents a ranked list of such boundaries, together with the estimated cost of the associated modularization, to the developers. The developers may interactively explore this design space before settling upon a retrofitting policy. Once the retrofitting policy has been identified, the developers must also supply a *security policy*, which specifies the permitted message interactions between modules separating a pair of connected resources. Such a policy could be developed interactively. The developers provide a candidate security policy, which will imply a set of security-sensitive operations and connections, resulting in a candidate retrofitting policy. The developer then iteratively refines the policy by observing the runtime behavior of the retrofitted system.

Once the developer has identified module boundaries, Strata transforms the code to enforce those boundaries. The main challenge in this step is to map software artifacts to the specific isolation primitives used, and to generate code to enable communication across protection domains. For example, if OS processes are used to modularize the system, Strata have to generate marshaling and de-marshaling code in each module, together with calls to the OS's IPC primitives to enable communication. Strata will include a variety of isolation primitives, including OS processes (as in PrivTrans), Capsicum sandboxes [73], lightweight VMs and transactions [18], and language-level primitives, such as JavaScript's Harmony modules [1]. Strata's transformation component also generates code to enforce the security policy specified by the developer at module boundaries.

In preliminary work, we have developed a prototype of the above approach for web browser extensions. Such extensions are available aplenty for Mozilla Firefox and Google Chrome and allow end-users to enrich their web browsing experience. Extensions contain code (both JavaScript and native code) that not only interacts with untrusted content on web pages, but also with code that accesses system resources, such as the file system and the network. It is critical to provide adversary containment for such extensions, e.g., to ensure that an adversary that hijacks an extension via a malicious script on a web page is unable to access system resources.

These problems have motivated much work from browser vendors in developing new frameworks for extension development [2, 8] that encourage extension developers to modularize their code. However, few guidelines exist for developers to understand how best to modularlize their code, and the process of creating modules is usually ad hoc. Moreover, browsers such as Firefox, which has only recently adopted modular extension development [2, 36], have legacy extensions that do not benefit from modularization.

We have applied modularization to legacy browser extensions, focusing first on Mozilla Firefox [35]. In this study, we transformed legacy extensions to benefit from the JetPack framework. We plan to extend this tool to also allow such extensions to operate atop Google Chrome, which uses different modularization primitives [73]. We also plan to apply our approach to traditional server software systems, such as the X server, OpenSSH, and the Apache web server.

6. RETROFIT FOR AUDITING

Retroactive security is the enforcement of security, or detection of security violations, after the execution of a process. By contrast, security mechanisms such as access control and information flow control enforce security either before or during execution. Years of experience with securing cyber systems has shown that retroactive security is necessary, in addition to protection-based mechanisms, since not all vulnerabilities can be predicted a priori or managed with prevention alone. Also, retroactive security can be used to hold entities accountable for their actions [45, 75, 74].

Auditing underlies retroactive security frameworks, and has become increasingly important to the theory and practice of cybersecurity and is essential for any defense in depth. However, auditing is error-prone, and difficult to get correct, in at least two ways. First, an audit log produced during execution must be an accurate record of all security-relevant events. Similar to missed access-control checks, it is easy for a programmer to accidentally omit the recording of all relevant events– for example, it has been shown that major health service informatics systems do not log sufficient information in light of guidelines for HIPAA policies [40]. Second, audit logs must be analyzed to detect security violations– a concern is often overlooked during development, resulting in "write only" logs that are never used for security enforcement.

Formal methods have been successful in addressing the second problem, and have been used to establish reliable foundations for

analysis of audit logs [15, 69]. However, little attention has been paid to the first problem: assuring correctness of the audit log. Such assurances are essential for assuring any sort of security analysis based on auditing. Our goals here then are (1) to obtain a *semantics of audit logging* so that assurances can be meaningfully and rigorously obtained for auditing policies, and (2) to define policy-driven retrofitting tools for audit log generation, that provably respect the semantics of input logging policies.

An Illustrative Example. Consider a medical records system at a hospital. Some patients' records are marked as sensitive. To ensure that medical staff has timely access to patient information (e.g., accessing a patient's record when they are admitted to the emergency room), the system allows access to any record by medical staff. The system does not enforce access control restrictions, and allows medical staff to read from medical records, and send the record to others. To ensure that staff do not violate this trust and only use their access appropriately, the system should record in a log when a user reads and subsequently sends a sensitive medical record.

The medical records system must be instrumented to generate the appropriate logs. However, if instrumentation strategies are informal, then the intended policy may not be implemented. For example, developers may just "eyeball" the code to identify where a medical record may be sent and insert code to record this event in the audit log. But this strategy might record false positives if it is not statically known that a secure file is read, prior to the send. It can also lose information, since it may be difficult to statically predict sequences of function calls, especially in the presence of features such as dynamic dispatch.

Observe that the problem is not with the manner in which the audit log is queried, but rather with the way the audit log itself is generated. In particular, the instrumentation of the program does not properly realize the intended logging policy. Here is a more precise textual specification of what should be recorded in the log, which we call LP_H:

> LP_H : *Record in the log information associated with a remote send by a medical staff member, if a sensitive file was read by that staff member prior to the send.*

Subsequently, if system administrators discover that sensitive information is being leaked to some remote location, they may desire to ask the following audit query, which we call AQ_H:

> AQ_H : *Retrieve all destination addresses of remote sends by medical staff in the log file.*

However, note that while administrators expect that the answer to AQ_H will return e.g. *every* relevant potential recipient of sensitive information, this is the case only if LP_H has been instrumented correctly. If logging is incomplete, for example, then some potential recipients may be missed. If logging is overzealous, some legitimate recipients of sensitive information may be erroneously flagged. In other words, the connection between audit queries and log-generating processes is the manner in which programs are instrumented to generate logs, and correctness of logging instrumentation is vital for auditing assurances. By "instrumented", we mean the functionality that is added to code specifically to generate logs.

Approach Overview. We advocate for retrofitting approaches to auditing, since such tools can assure correct audit log generation even for untrusted code. That is, if retrofitting tools can be shown to correctly instrument *any* input program to support some class of logging policies, correctness of generated audit logs is automatically ensured. Clearly, a formal semantics of audit logging is necessary to establish correctness of retrofitting strategies.

We regard an audit log as a piece of information that is a refinement of the information contained in a process. Thus, the proper meaning of an auditing policy is as a kind of operation over information structures. With this view, it is natural to pursue a semantics of auditing based on *information algebra* [42, 41], which is a generalized framework for information systems. Information algebra has been shown to capture systems such as relational algebra, predicate logic, and linear systems. Aside from the philosophical appeal of realizing an auditing semantics in this general theory, an information algebra formulation has a number of technical advantages. For example, relations between distinct information algebras have been established, so the FOL-centric results in this paper can be ported to other systems, e.g. relational databases. Significantly, audit algebras enjoy a partial *information ordering*, denoted $\leq$ that allows comparison of information pieces wrt their information content. This ordering is crucial in relating audit logs with logging policy semantics, and establishing notions of soundness and completeness of retrofitting. Although the former is concrete, whereas the latter is abstract, they can be related by the information they contain.

In more detail, we argue that the semantics of a particular logging policy LP, which is specified in some formal language, be defined as an operation in a complete program trace. That is, for any program $\mathfrak{p}$, the semantics of a logging policy are a refinement of $\mathfrak{p}$'s complete execution trace, denoted $traceof(\mathfrak{p})$. This refinement can be specified as an information algebraic operation we call $genlog$ that takes as input $traceof(\mathfrak{p})$ and LP, so that:

$$genlog(traceof(\mathfrak{p}), LP)$$

denotes the intended semantics of a logging policy LP for a given program $\mathfrak{p}$.

Given this semantic definition, as well as notions of information ordering available in information algebras, we can meaningfully define correctness of retrofitting strategies. Let $\mathbf{retro}$ be some retrofitting strategy, that takes as input programs $\mathfrak{p}$ and a logging policy LP, where we assume that LP is selected from some nonempty set of logging policies $\mathcal{P}$ that the strategy supports. We write:

$$\mathbf{retro}(\mathfrak{p}, LP) \rightsquigarrow \mathbb{L}$$

to denote that the log $\mathbb{L}$ is generated by executing the program $\mathfrak{p}'$ that results from instrumenting the program $\mathfrak{p}$ to support the logging policy LP. We say that $\mathbf{retro}$ is *sound* iff $\mathbb{L}$ represents a subset of information in $genlog(traceof(\mathfrak{p}), LP)$, and $\mathbf{retro}$ is *complete* iff $genlog(traceof(\mathfrak{p}), LP)$'s information content is contained in $\mathbb{L}$'s. More precisely, we have:

DEFINITION 1. *A retrofitting strategy* $\mathbf{retro}$ *is* sound for $\mathcal{P}$ *iff for all* $\mathfrak{p} \in \mathcal{L}$ *and* $LP \in \mathcal{P}$*, we have that* $\mathbb{L} \leq genlog(\mathfrak{p}, LP)$*, where* $\mathbf{retro}(\mathfrak{p}, LP) \rightsquigarrow \mathbb{L}$.

DEFINITION 2. *A retrofitting strategy* $\mathbf{retro}$ *is* complete for $\mathcal{P}$ *iff for all* $\mathfrak{p} \in \mathcal{L}$ *and* $LP \in \mathcal{P}$*, we have* $genlog(\mathfrak{p}, LP) \leq \mathbb{L}$*, where* $\mathbf{retro}(\mathfrak{p}, LP) \rightsquigarrow \mathbb{L}$.

In work so far, we have defined a language of logging policies based on first order logic (FOL), and have formalized a notion of program traces expressed as sets of temporally ordered FOL formulae. These definitions, along with additional constructions, obtain an information algebraic framework in which audit logging can be endowed with a semantics defined in terms of algebraic operations. We have also defined a retrofitting strategy for a core functional calculus that supports an interesting class of logging policies, the so-called surveillance policies. This strategy has been verified to be

sound and complete, in the information algebraic sense described above, using the Coq framework.

In ongoing work, our immediate research targets include developing retrofitting strategies for realistic programming languages with correctness assurances, as well as type-directed optimizations. These optimizations will be based on our previous work on temporally-sensitive typing analyses [62].

7. VALIDATING DEFENSE IN DEPTH

Building on methods to validate security controls for authorization, containment, and auditing, we identify three advantages to reasoning about all three in unison. The first advantage is that we may be able to optimize the use of security controls by eliminating redundant controls. For example, authorization may reliably control all adversary flows from one module to another, which may eliminate the need to separate those modules. The second advantage is that the actual runtime use of the program may motivate changes in security controls that improve security. For example, logging downgraded data could show that the downgrading task is more common and more complex than envisioned, motivating changes in the retrofitting policies to enact more authorization and/or containment. Finally, the third advantage is that assurance can encompass all three controls, providing a comprehensive validation of enforcement. We plan to develop formal verification techniques to certify the correctness of retrofitting, similar to CompCert [13] and Vellvm [79, 80].

Thus, we propose a unified framework for retrofitting programs for defense in depth spanning all three security controls. Figure 1 shows the expected high-level design of the STRATA framework. In this task, we will explore methods for step two, *unified retrofitting*. This step receives retrofitting policies for each of the three security controls, plus the program code and optional feedback from deployed security controls. The output from this step is the program code retrofitted for authorization, containment, and auditing that satisfies the retrofitting policies and is optimal with respect to the costs of the controls.

Unified policy representation. A distinct benefit of designing a multi-layered security framework from the ground up is the opportunity to unify and synthesize policies across layers. For example, authorization and auditing policies can be synthesized to ensure auditing, and thus retroactive accountability, if certain authorization conditions are not met by actors. Such a policy was already suggested in Section 6. A unified policy representation, capturing properties at each layer, can be used to specify this.

Key to enabling unified retrofitting is a uniform denotation of retrofitting policies across STRATA levels. Authorization, containment, and auditing all refer to subjects (to be controlled), objects (that may be accessible to subjects or may have security requirements), program flows (control, data, and traces), and security policies. We argue that it would be beneficial to apply a single language to express retrofitting policies at each layer. One option is to express security controls in terms of automata. For example, *I/O automata* are labeled transition models for asynchronous concurrent systems [48]; they are typically used to describe the behavior of a system interacting with its environment. In I/O-automata-based models of monitoring, the system (node) to be monitored and the monitor are represented as I/O automata, with the input and output actions of each automaton representing their interaction with the environment and each other. Security policies are defined in terms of allowed or disallowed executions (traces). Using I/O automata, we can capture requirements on input (e.g., control of various subjects to that input), output (e.g., the impact of the operation on the security of the object), and trace effects (e.g., logging in particular states). Further, extensions to I/O automata have been proposed to represent probabilistic policies [11] and model cost [19], so this approach could capture a variety of whole-system enforcement scenarios. A remaining challenge is to ensure that I/O automata are at least translatable to resident policy languages at each STRATA layer.

Optimization via synthesized transformations. Policies specify the semantics of security mechanisms, but uniform policies will also enable the implementation of policies via program constructs that leverage connections between layers. As a retrofitting policy is defined to be a set of connections among a set of program constructs, the goal of this task is two-fold: (1) produce a single set of program constructs from the three control-specific sets and (2) produce a single set of connections among them from three control-specific sets. While the naive approach to union the three construct and connection sets to form a multi-control retrofitting policies can yield a solution if one exists, it may miss opportunities to find better solutions. For example, if the same constructs are identified for containment and authorization, then a sub-optimal solution that employs containment to isolate the constructs when authorization effectively blocks illegal data can be eliminated. We will explore automated analyses to identify such opportunities. For example, we will explore methods to identify such dominance relations across controls. Recall that programmers produce retrofitting policies interactively with STRATA, so such analyses must be meaningful to programmers. Ultimately, we would like programmers to "program" the retrofitting policy interactively with STRATA.

The problem of transformation takes a program, a retrofitting policy, and a cost function and produces a retrofitted program that satisfies the retrofitting policy for the minimal cost. For individual controls, the cost function focuses on only one dimension at a time, but since different controls apply different cost metrics we must consider transformation as a multi-dimensional optimization where the retrofitting policy implies a set of constraints.

Improving retrofitting policies continuously. The goal of continuous improvement is to use knowledge of how programs are actually run to reduce the risks taken by the trade-off of security with functional concerns proactively. To address this problem, we will leverage the unification of security controls to collect information for guiding improvements to the security controls themselves. The problem is analogous to auditing, except that rather than looking for intrusions we will try to estimate the risks introduced by security controls quantitatively to identify those most in need of revision. This is sometimes called *feedback* in the systems literature. For the auditing example on downgrading, we may estimate risk by the percentage of data to redact or number of decisions necessary to identify the data to redact. Using these quantitative metrics we may identify more (fewer) program constructs in need of control or eliminate (add) connections that are violated (satisfied) in practice, resulting in more (fewer) security controls. In addition, we will explore methods to make retrofitting changes based on such findings automatically, leading to agile retrofitting of programs as they execute.

Verifying transformations. Having retrofitted software for defense in depth, how can we show that the retrofitted system preserve the functionality of the original software system? In fact, we are interested in demonstrating that the behavior of the retrofitted system is a subset of the monolithic original, with omitted behaviors being those excluded by a security policy.

In general, proving correctness of program transformations is a difficult challenge, one that has remained open for several decades,

cf., the quest to produce provably correct compilers and program optimizers. However, over the last few years, there have been impressive developments in this domain, thanks to advances in interactive theorem proving systems and SMT solvers.

We plan to build upon this line of research to build a verified retrofitting pipeline. One approach that we plan to explore is the use of Coq [14] to achieve the goal of verified transformations. Coq is an interactive proof assistant that allows co-development of program transformations, themselves expressed in Coq, together with their proofs of correctness. Transformations can be developed iteratively with their proofs, using the Coq system to debug the transformations or their proofs as errors are discovered. Thus, when the transformation has been fully specified, it is also accompanied with a machine-checkable proof of correctness. This approach was recently used in Vellvm [79, 80] to prove the correctness of several optimizations within LLVM. As has been noted in Section 6, we have already used Coq to verify a retrofitting strategy for auditing in preliminary work.

8. CONCLUSIONS

Even when programmers decide to add the security controls necessary to implement *defense in depth*, they face many practical challenges. First, Defenses are often added manually, using an ad hoc process. Second, each security control typically uses its own policies and mechanisms, so the manual process has to be repeated for each control. Third, it is difficult to prove that a manual deployment of security controls provides an advertised level of assurance. Recent work on methods to retrofit legacy code with security controls has begun to address some of these issues, but these methods still require significant manual effort, do not explicitly map security goals to program code, and they do not reason about multiple security controls. In this paper, we propose the STRATA framework for retrofitting legacy code for authorization, containment, and auditing security controls. The STRATA framework implements a comprehensive view of assurance, with an emphasis on *automated* and *interactive* tools that developers can use to identify site-level security goals, in terms of a *retrofitting policy*, and retrofit legacy code to enforce security policies in a manner that can be machine-verified for assurance. We show how security controls can be retrofit individually by STRATA and how STRATA enables optimization, continuous improvement, and assurance across multiple security controls. We show that by reasoning about defense in depth a variety of advantages can be obtained, including optimization, continuous improvement, and assurance across multiple security controls.

Acknowledgements

We acknowledge the anonymous reviewers of this paper. This material is based upon work supported by the National Science Foundation Grant No. CNS-1408880.

9. REFERENCES

[1] Ecmascript Harmony modules. http://wiki.ecmascript.org/doku.php?id=harmony:modules.
[2] The mozilla jetpack extension development framework. https://wiki.mozilla.org/Jetpack.
[3] Apache security controls and auditing. http://www.isaca.org/Journal/Past-Issues/2003/Volume-5/Pages/Apache-Security-Controls-and-Auditing.aspx, 2013.
[4] F.38. sepgsql. http://www.postgresql.org/docs/9.1/static/sepgsql.html, 2013.
[5] Linux audit-subsystem design documentation for kernel 2.6, version 0.1. http://www.uniforum.chi.il.us/slides/HardeningLinux/LAuS-Design.pdf, 2013.
[6] qmail home page. http://qmail.omnis.ch/top.html, 2013.
[7] J. P. Anderson. Computer security technology planning study, volume II. Technical Report ESD-TR-73-51, HQ Electronics Systems Division (AFSC), October 1972.
[8] A. Barth, A. P. Felt, P. Saxena, and A. Boodman. Protecting browsers from extension vulnerabilities. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2010, San Diego, California, USA*. The Internet Society, 2010.
[9] D. E. Bell and L. J. LaPadula. Secure computer system: Unified exposition and Multics interpretation. Technical Report ESD-TR-75-306, HQ Electronic Systems Division (AFSC), March 1976.
[10] D. Brumley and D. Song. PrivTrans: Automatically partitioning programs for privilege separation. In *Proceedings of the USENIX Security Symposium*, August 2004.
[11] R. Canetti, L. Cheung, D. Kaynar, M. Liskov, N. Lynch, O. Pereira, and R. Segala. Task-structured probabilistic I/O automata. In *Proceedings of 8th International Workshop on Discrete Event Systems*, pages 207–214, 2006.
[12] J. Carter. Using GConf as an Example of How to Create an Userspace Object Manager. *2007 SELinux Symposium*, 2007.
[13] The COMPCERT project. http://compcert.inria.fr/.
[14] The Coq proof assistant. http://coq.inria.fr/, 2008. Version 8.1.
[15] A. Datta, J. Blocki, N. Christin, H. DeYoung, D. Garg, L. Jia, D. K. Kaynar, and A. Sinha. Understanding and protecting privacy: Formal semantics and principled audit mechanisms. In *7th International Conference on Information Systems Security*, pages 1–27, 2011.
[16] D. Dean, E. W. Felten, and D. S. Wallach. Java security: From hotjava to netscape and beyond. In *In Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 190–200, 1996.
[17] D. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–242, 1976.
[18] M. Dhawan, C. Shan, and V. Ganapathy. Enhancing JavaScript with transactions. In *Proceedings of ECOOP'12, the 26th European Conference on Object-Oriented Programming*, volume 7313 of *Lecture Notes in Computer Science (LNCS)*, pages 383–408, Beijing, China, June 2012. Springer.
[19] P. Drábik, F. Martinelli, and C. Morisset. Cost-aware runtime enforcement of security policies. In *Proceedings of the 8th International Workshop on Security and Trust Management (STM 2012)*, pages 1–16, 2013.
[20] D.Walsh. Selinux/apache. http://fedoraproject.org/wiki/SELinux/apache.
[21] A. Edwards, T. Jaeger, and X. Zhang. Runtime verification of authorization hook placement for the Linux security modules framework. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 225–234, 2002.
[22] J. Epstein and J. Picciotto. Trusting X: Issues in building Trusted X window systems -or- what's not trusted about X? In *Proceedings of the 14th Annual National Computer Security Conference*, Oct. 1991. A survey of the issues involved in building trusted X systems, especially of the multi-level secure variety.
[23] D. Ferraiolo and R. Kuhn. Role-based access control. In *In Proceedings of the 15th NIST-NCSC National Computer Security Conference*, pages 554–563, 1992.
[24] J. Ferrante, K. J. Ottenstein, and J. D. Warren. The program dependence graph and its use in optimization. *ACM Trans. Program. Lang. Syst.*, 9(3):319–349, July 1987.
[25] M. Frank, J. M. Buhmann, and D. Basin. On the definition of role mining. In *Proceedings of the 15th ACM Symposium on Access Control Models and Technologies*, SACMAT '10, pages 35–44. ACM, 2010.
[26] V. Ganapathy, T. Jaeger, and S. Jha. Automatic placement of authorization hooks in the Linux Security Modules framework. In *Proceedings of the 12th ACM Conference on Computer and Communications Security*, pages 330–339, Nov. 2005.
[27] J. A. Goguen and J. Meseguer. Security policies and security models. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 11–20, Apr. 1982.
[28] C. Grier, S. Tang, and S. King. Secure web browsing with the op web browser. In *IEEE Symposium on Security and Privacy*, 2008.
[29] C. Grier, S. Tang, and S. T. King. Secure web browsing with the op web browser. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, pages 402–416. IEEE Computer Society, 2008.
[30] W. R. Harris, S. Jha, and T. W. Reps. DIFC programs by automatic instrumentation. In *ACM Conference on Computer and Communications Security (CCS)*, pages 284–296, 2010.
[31] W. R. Harris, S. Jha, T. W. Reps, J. Anderson, and R. N. M. Watson. Declarative, temporal, and practical programming with capabilities. In *IEEE Symposium on Security and Privacy*, pages 18–32, 2013.
[32] B. Hicks, K. Ahmadizadeh, and P. McDaniel. Understanding practical application development in security-typed languages. In *22st Annual Computer Security Applications Conference (ACSAC)*, December 2006.
[33] JDBC: Java Database Connectors. http://java.sun.com/products/jdbc.
[34] P. A. Karger and R. R. Schell. MULTICS security evaluation: Vulnerability analysis. Technical Report ESD-TR-74-193, Deputy for Command and Management Systems, Electronics Systems Division (ASFC), June 1974.
[35] R. Karim, M. Dhawan, and V. Ganapathy. Refactoring legacy browser

extensions to modern extension frameworks. In *Proceedings of ECOOP'14, the 28^{th} European Conference on Object-Oriented Programming*, volume 8586 of *Lecture Notes in Computer Science (LNCS)*, pages 463–488, Uppasala, Sweden, July/August 2014. Springer.

[36] R. Karim, M. Dhawan, V. Ganapathy, and C. Shan. An analysis of the Mozilla Jetpack extension framework. In *Proceedings of ECOOP'12, the 26^{th} European Conference on Object-Oriented Programming*, volume 7313 of *Lecture Notes in Computer Science (LNCS)*, pages 333–355, Beijing, China, June 2012. Springer.

[37] D. Kilpatrick. Privman: A library for partitioning applications. In *Proceedings of the 2003 USENIX Annual Technical Conference—FREENIX Track*, June 2003.

[38] D. H. King, B. Hicks, M. Hicks, and T. Jaeger. Implicit Flows: Can't live with 'em, can't live without 'em. In *Proceedings of Fourth International Conference on Information Systems Security*, Dec. 2008.

[39] D. H. King, S. Jha, D. Muthukumaran, T. Jaeger, S. Jha, and S. Seshia. Automating security mediation placement. In *Proceedings of the 19th European Symposium on Programming (ESOP '10)*, pages 327–344, 2010.

[40] J. T. King, B. Smith, and L. Williams. Modifying without a trace: General audit guidelines are inadequate for open-source electronic health record audit mechanisms. In *Proceedings of the 2nd ACM SIGHIT International Health Informatics Symposium*, IHI '12, pages 305–314, New York, NY, USA, 2012. ACM.

[41] J. Kohlas. *Information Algebras: Generic Structures For Inference*. Discrete mathematics and theoretical computer science. Springer, 2003.

[42] J. Kohlas and J. Schmid. An algebraic theory of information: An introduction and survey. *Information*, 5(2):219–254, 2014.

[43] A. Kurmus, R. Tartler, D. Dorneau, B. Heinloth, V. Rothberg, A. Ruprecht, W. Schoeder-Preikschat, D. Lohman, and R. Kapitza. Attack surface metrics and automated compile-time os kernel tailoring. In *NDSS*, 2013.

[44] B. W. Lampson. Protection. In *5th Princeton Conference on Information Sciences and Systems*, 1971.

[45] B. W. Lampson. Computer security in the real world. *IEEE Computer*, 37(6):37–46, 2004.

[46] B. Livshits and S. Chong. Towards fully automatic placement of security sanitizers and declassifiers. In *POPL*, pages 385–398, 2013.

[47] B. Livshits and J. Jung. Automatic mediation of privacy-sensitive resource access in smartphone applications. In *Proceedings of the 22nd USENIX Security Symposium*, Berkeley, CA, USA, 2013. USENIX Association.

[48] N. A. Lynch and M. R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proceedings ACM Symposium on Principles of Distributed Computing*, 1987.

[49] I. Molloy, N. Li, T. Li, Z. Mao, Q. Wang, and J. Lobo. Evaluating role mining algorithms. In *Proceedings of the 14th ACM Symposium on Access Control Models and Technologies*, SACMAT '09, pages 95–104. ACM, 2009.

[50] D. Muthukumaran, T. Jaeger, and V. Ganapathy. Leveraging 'choice' in authorization hook placement. In *19th ACM Conference on Computer and Communications Security*. ACM, 2012.

[51] D. Muthukumaran, T. Jaeger, and G. Tan. Producing hook placements to enforce expected authorization policies. Technical Report NSRC Technical Report NAS-TR-169-2013, The Pennsylvania State University, September 2013.

[52] D. Muthukumaran, J. Schiffman, M. Hassan, A. Sawani, V. Rao, and T. Jaeger. Protecting the integrity of trusted applications on mobile phone systems. *Security and Communication Networks*, 4(6), 2011.

[53] J. G. Politz, S. A. Eliopoulos, A. Guha, and S. Krishnamurthi. Adsafety: type-based verification of javascript sandboxing. In *Proceedings of the 20th USENIX conference on Security*, SEC'11, pages 12–12. USENIX Association, 2011.

[54] The Postfix mail program. http://www.postfix.org.

[55] N. Provos, M. Friedl, and P. Honeyman. Preventing privilege escalation. In *Proceedings of the 12th USENIX Security Symposium*, Berkeley, CA, USA, 2003. USENIX Association.

[56] N. Provos, D. McNamee, P. Mavrommatis, K. Wang, and N. Modadugu. The ghost in the browser analysis of web-based malware. In *HotBots*, 2007.

[57] C. Reis, A. Barth, and C. Pizano. Browser security: Lessons from Google Chrome. *Communications of the ACM*, 52(8):45–49, Aug. 2009.

[58] R. Sailer, T. Jaeger, E. Valdez, R. Caceres, R. nald Perez, S. Berger, J. L. Griffin, and L. van Doorn. Building a MAC-based security architecture for the xen open-source hyperviso r. In *Proceedings of the 2005 Annual Computer Security Applications Conference*, pages 276–285, Dec. 2005.

[59] J. H. Saltzer and M. D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, September 1975.

[60] Re: Adding support for SE-Linux security. http://archives.postgresql.org/pgsql-hackers/2009-12/msg00735.php, 2009.

[61] SE-PostgreSQL? http://archives.postgresql.org/message-id/20090718160600.GE5172@fetter.org, 2009.

[62] C. Skalka, S. Smith, and D. V. Horn. Types and trace effects of higher order programs. *Journal of Functional Programming*, 18(2):179–249, 2008.

[63] S. Son, K. S. McKinley, and V. Shmatikov. Rolecast: finding missing security checks when you do not know what checks are. In *Proceedings of the 2011 ACM international conference on Object oriented programming systems languages and applications*, OOPSLA '11, pages 1069–1084. ACM, 2011.

[64] S. Son, K. S. McKinley, and V. Shmatikov. Fix Me Up: Repairing Access-Control Bugs in Web Applications. In *ISOC Network and Distributed System Security Symposium (NDSS)*, 2013.

[65] V. Srivastava, M. D. Bond, K. S. McKinley, and V. Shmatikov. A security policy oracle: detecting security holes using multiple api implementations. In *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '11, pages 343–354. ACM, 2011.

[66] F. Sun, L. Xu, and Z. Su. Static detection of access control vulnerabilities in web applications. In *Proceedings of the 20th USENIX conference on Security*, SEC'11, pages 11–11. USENIX Association, 2011.

[67] L. Tan, X. Zhang, X. Ma, W. Xiong, and Y. Zhou. AutoISES: automatically inferring security specifications and detecting violations. In *Proceedings of the 17th conference on Security symposium*, pages 379–394. USENIX Association, 2008.

[68] D. Turner. Symantec internet security threat report: Trends for january-june 07. Technical report, Symantec Inc., 2007.

[69] J. A. Vaughan, L. Jia, K. Mazurak, and S. Zdancewic. Evidence-based audit. In *In Proc. of the IEEE Computer Security Foundations Symposium*, 2008.

[70] E. Walsh. Integrating x.org with security-enhanced linux. In *Proceedings of the 2007 Security-Enhanced Linux Workshop*, Mar. 2007.

[71] Y.-M. Wang, D. Beck, X. Jiang, R. Roussev, C. Verbowski, S. Chen, and S. King. Automated web patrol with strider honeymonkeys: Finding web sites that exploit browser vulnerabilities. In *Proceedings of the 2006 Network and Distributed System Security Symposium (NDSS)*, February 2006.

[72] R. N. M. Watson. TrustedBSD: Adding trusted operating system features to FreeBSD. In *Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference*, pages 15–28. USENIX Association, 2001.

[73] R. N. M. Watson, J. Anderson, B. Laurie, and K. Kennaway. Capsicum: Practical capabilities for UNIX. In *USENIX Security*, 2010.

[74] D. J. Weitzner. Beyond secrecy: New privacy protection strategies for open information spaces. *IEEE Internet Computing*, 11(5):94–96, 2007.

[75] D. J. Weitzner, H. Abelson, T. Berners-Lee, J. Feigenbaum, J. A. Hendler, and G. J. Sussman. Information accountability. *Communications of the ACM*, 51(6):82–87, 2008.

[76] C. Wright, C. Cowan, and J. Morris. Linux Security Modules: General security support for the Linux kernel. In *In Proceedings of the 11th USENIX Security Symposium*, pages 17–31, 2002.

[77] Implement keyboard and event security in X using XACE. https://dev.laptop.org/ticket/260, 2006.

[78] X. Zhang, A. Edwards, and T. Jaeger. Using CQUAL for static analysis of authorization hook placement. In *Proceedings of the 11th USENIX Security Symposium*, pages 33–48, August 2002.

[79] J. Zhao, S. Nagarakatte, M. Martin, and S. Zdancewic. Formalizing the LLVM intermediate representation for verified program transformations. In *ACM Principles of Programming Languages (POPL)*, 2012.

[80] J. Zhao, S. Nagarakatte, M. Martin, and S. Zdancewic. Formal verification of SSA optimizations. In *ACM Conference on Programming Language Design and Implementation (PLDI)*, 2013.

Towards High Assurance Platforms for Securing the Cloud with Disparate Domains

Sujay Doshi
University of Southern California
Los Angeles, California
sujaysad@usc.edu

Dr. Roger Schell
University of Southern California
Los Angeles, California
rrschell@usc.edu

ABSTRACT

Our analysis of the state-of-the-art (SOA) for cloud computing concludes current offerings are fundamentally not able to provide a high assurance of protecting information from disparate security domains sharing a cloud infrastructure. This is especially evident in the face of (even moderately) determined adversaries who are likely to employ software subversion in artifices like Trojan horses and trap doors. An intrinsic limitation is the absence of an explicit security policy for which it is even theoretically feasible to constrain information flow while providing authorized sharing. Although not employed by current cloud offerings the technology SOA does include well-understood mandatory access control (MAC) policy that supports such controlled sharing. Furthermore, currently the cloud is simply not hosted on platforms with hardware and operating system technology that is verifiably trustworthy. Yet the available platform SOA includes high assurance security kernel technology. Our conclusion is that, although not used by current practice, the SOA of current security technology as evidenced in a number of legacy systems reviewd in this paper, can support securing data for big environments with disparate domains in the cloud by providing high assurance.

Keywords

Cloud computing, computer utility, high assurance, security kernel, mandatory access control, TCSEC, TNI, multilevel security

1. INTRODUCTION

Information supremacy wins wars and gains competitive advantage. Such advantage always requires sharing the right information with the right people. Amid this growing unmet need to share information, cloud computing has come to the fore. As fuzzy as this metaphor seems, cloud is becoming the driving force of many enterprises today. Before cloud, each enterprise was a separate security domain having no systematic sharing between these domains[1]. Since the requirement is for controlled sharing in today's world, the air-gapped security domains do not cater to these needs. Mandatory access controls are required for secure sharing of information between these disparate domains. Historically, secure systems have leveraged the power of a sound mandatory access control policy to prevent unauthorized exfiltration.

Before the "dotcom" era, the enterprises were mostly separate domains where the systems were logically air-gapped. There was no possibility of unauthorized information to flow from one enclave to other. But at the same time it made it impossible for legitimate information to be shared between these domains. At this time, before cloud computing was a reality; the basic concepts were very well articulated in what was then called a computer utility[2]. Multics, a research project initiated at MIT, was described initially in a set of six papers presented at the 1965 Fall Joint Computer Conference. A computer, its software, and staff set up to provide 24-hour service for all of a community's information needs, the ancestor of the "Information Highway". Multics was envisioned as quite this, "**Mult**iplexed **I**nformation and **C**omputing **S**ervice", from which it took its name. The envisioned service was present, reliable, powerful, and all that is needed as an information resource for a large number of people, all the time. This was radical at the time, when computers were mostly restricted to dedicated functions.

The seeds of this concept were laid out in an article *The Computers of Tomorrow* by Martin Greenberger of MIT in 1964. He posed a very important futuristic question on "How computers would be used in its third decade?". By achieving reliability along with capability, computers have won broad commercial acceptance. But was there a plateau just above the horizon? He asserted that *"computing services and establishments will begin to spread throughout every sector of American life, reaching into homes, offices, classrooms, laboratories, factories, and businesses of all kinds"* [13]. The article then goes on to make an analogy to electricity and discusses the concept of information utility at length. Security was a core value proposition for the computer utility, as evident in its requirement that one of its core properties is **Sufficient control of access to allow selective shar-**

[1]**Security Domain** here is analogous to a physical encalve wherein the information is air gapped from other enclaves.
[2]FJ Corbato, JH Saltzer and CT Clingen described Multics as a *computer utility* in their paper *"Multics- The first seven years"*

ing of information. But the innovative computer utility concept did not have an immediate or widespread adoption, which is not surprising as Internet (i.e. the World Wide Web) was not a reality then. However, several decades later the concept of the "cloud" was born out of some of the similar emerging requirements of sharing of information and resources stimulated by widespread use of World Wide Web (www).

Some writers have opined that the cloud metaphor seems to have arisen from graphics in which the complexity of a computing network was replaced by an abstraction for a cloud, in order to avoid the complexity of drawing all the lines. It is not a very helpful metaphor except perhaps in the sense that the cloud interface obscures the details behind it. Interest in cloud computing has rapidly grown in recent years due to the advantages of greater flexibility and availability of computing resources at lower cost. NIST defines cloud computing as [15]:

> *"A model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or cloud provider interaction."*

While aspects of these characteristics have been realized to a certain extent, cloud computing remains a work in progress. Cloud computing services benefit from economies of scale achieved through versatile use of resources, specialization, and other practicable efficiencies. However, cloud computing is an emerging form of distributed computing that is still in its infancy. In our analysis we did not find strong evidence that security is at this time a core value proposition for the cloud, in sharp contrast to a computer utility.

2. OVERVIEW OF MAJOR PRESENT DAY CLOUD PROVIDERS

Cloud,today, has changed the way the conventional IT enterprises functioned. It has revolutionized the way IT is managed and maintained. But with all such advancements, one primary issue of providing high assurance security has not been addressed. We provide a brief overview of 2 leading cloud providers and the security mechanisms implemented in their cloud solutions. While the two examples are not exhaustive of the present day offerings, they are illustrative of the security approaches.

2.1 Microsoft

Microsoft Azure (formerly Windows Azure before 25 March 2014) is a cloud computing platform and infrastructure, created by Microsoft, for building, deploying and managing applications and services through a global network of Microsoft-managed datacenters. It supports many different programming languages, tools and frameworks, including both Microsoft-specific and third-party software and systems to provide various services. With Azure, Microsoft hosts data and programs belonging to customers. Microsoft Azure must therefore address information security challenges above and beyond traditional on or off-premises IT scenarios. Microsoft Azure is designed to abstract much of the infrastructure that typically underlies applications (servers, operating systems, web and database software, and so on) so that developers can focus more on developing applications[19].

Customers manage the storage and applications through subscription, which broadly can be considered a discretionary access control method. SMAPI (Service Management API), a programtical way to access Azure, authentication is based on a user-generated public/private key pair and self-signed certificate registered through the Microsoft Azure Portal. The certificate is then used to authenticate subsequent access to SMAPI. Azure implements Identity and Access Mangement as one of its service supplemented with encryption to protect the control channels. To align with the principle of least privilege, customers are not granted administrative access to their VMs, and customer software in Microsoft Azure is restricted to running under a low-privilege account by default.

2.2 Amazon

Amazon Web Services (AWS) is a collection of remote computing services (also called web services) that together make up a cloud computing platform, offered over the Internet by Amazon.com.Such is the popularity of AWS that many enterprises like Dropbox, Netflix have their business operated on AWS! The most central and well-known of these services are Amazon EC2 and Amazon S3. The service is advertised as providing a large computing capacity (potentially many servers) much faster and cheaper than building a physical server farm.

The AWS network has been architected to permit you to select the level of security and resiliency appropriate for your workload. Network devices, including firewall and other boundary devices, are in place to monitor and control communications at the external boundary of the network and at key internal boundaries within the network. These boundary devices employ rule sets, access control lists (ACL), and configurations to enforce the flow of information to specific information system services. All of the AWS APIs are available via SSL-protected endpoints which provide server authentication. Amazon EC2 AMIs automatically generate new SSH host certificates on first boot and log them to the instance's console. You can then use the secure APIs to call the console and access the host certificates before logging into the instance for the first time. Amazon EC2 instances cannot send spoofed network traffic. The AWS-controlled, host-based firewall infrastructure will not permit an instance to send traffic with a source IP or MAC address other than its own. AWS provides a number of ways for you to identify yourself and securely access your AWS Account. AWS also provides additional security options that enable you to further protect your AWS Account and control access: AWS Identity and Access Management (AWS IAM), key management and rotation, temporary security credentials, and multi-factor authentication (MFA)[1].

2.3 Is this the solution?

Based upon a brief analysis of some of the security techniques adopted by Amazon and Microsoft, it is fair to ask if we can adopt the above methodologies to assert authoritatively that the cloud is secure with disparate domains. But

unfortunately that is **NOT** true. The methods employed have a couple of huge gaps for security: They don't include any:

- Systematic mandatory access controls that are essential to addressing disparate domains and,
- They offer no basis for high assurance that would significantly mitigate the threat of subversion by a witted adversary.

Cross VM side-channel attacks as mentioned in [24], [26] clearly shows that the problem is far from being solved. The recent Apple iCloud hack [3], [2] is a wake up call for the cloud vendors to re-visit their security practices because all it takes is one vulnerabiltiy for a determined adversary to expose the entire system[5]. The clear **LOUD** message in c**LOUD** security is requirement of data isolation, controlled sharing and high assurance above all.

3. REQUIREMENTS FOR SECURE CLOUD WITH DISPARATE DOMAINS

Computer utility, as described before, had 3 requirements namely controlled sharing, isolation and high assurance. Drawing parallels from it, the present day cloud scenario strictly demands all these requirements when dealing in a multilevel environment as is cloud because no cloud architecture can emphatically address the issue of subversion without the above three properties. Thus, implementing an architecture keeping in mind the computer utility goals provides a sufficiently high assurance to resist subversion from witted adversaries.

3.1 Cloud Isolation

The cloud is a vast area with many types of service models, heterogeneity of hardware, and support many types of use cases. Isolation is the first of the three dominant requirements for securing the cloud with information from disparate domains. A domain, in the context of this paper, means the information with a common basis for access authorizations, such as that belonging to an enterprise at a given sensitivity level. Long before the advent of cloud computing, all the enterprises had their information within their security perimeter as these enclaves were physically air gapped. This ensured that no unauthorized information from one enterprise flow to the other. Isolation, as this air-gapped arrangement is called, ensured that if the systems are not connected to each other then the information (unauthorized) does not flow.

A virtual machine (VM) is a software implementation of a machine (e.g., a computer) that executes programs like a physical machine. Virtual machines operate based on the computer architecture and functions of a real or hypothetical computer. The piece of software that provides this abstraction of virtual machines is known as Virtual Machine Monitor (VMM). One of the earliest examples, the VM/370 system enables a single IBM System/370 to appear functionally as if it were multiple Independent System/370's (i.e., multiple "virtual machines") [21]. Thus, a VMM can make one computer system function as if it were multiple physlcally isolated systems(a typical realization of air-gapped arrangement). A VMM accomplishes this feat by controlling the multlplexing of the physical hardware resources in a manner analogous to the way that the telephone company multiplexes communications enabling separate and, hopefully, isolated conversations over the same wires [21].

VAX VMM, a high-assurance VMM (actually it is a security kernel) developed by DEC ran on the VAX 8530, 8550, 8700, 8800, and 8810 processors [18]. It created isolated virtual VAX processors, each of which can run either the VMS or ULTRIX-32 operating system. If desired, virtual machines running each of the operating systems can run simultaneously on the same computer system.[3] VAX VMM is an example of MAC enforcing VMM which provided high assurance security - in particular high assurance isolation. It provides virtual machines and virtual disks as an abstarction of subjects and objects respectively.

In 1998, VMware figured out how to similarly run multiple instances of operating systems on the x86 platform and created the market for their form of x86 virtualization. Be it VMWare's virtualization technique or Xen's paravirtualization technique[4] or Microsoft Virtual Server, which is Microsoft's solution for desktop virtualization facilitating the creation of virtual machines on Windows, the hypervisor (VMM) has been leveraged to provide nominal isolation between the virtual machines by virtualizing the CPU, memory etc. But in all variants, the inherent property of nominal isolation stays, i.e, in all the solutions above the very meaning of isolation between the domains is maintained which ensures restriction of unauthorized information flow. Thus, there exists a state-of-the-art technology for cloud isolation, although (except for the VAX VMM) there is little basis for assurance in this isolation from a security perspective.

3.2 Cloud Sharing

Since the enterprises had the storage and processing of information restricted to their server room before advent of cloud, the controlled sharing of information between these disparate domains was generally not regarded as practical. These isolated physical enclaves in different domains needed a way to share information with high assurance such that the information does not flow from one domain to another.

The VMM technology nominally provides isolation between the virtual machines but fails to provide a solution for controlled sharing of information. Each virtual machine is isolated from other virtual machines running on the same hardware. Although virtual machines share physical resources such as CPU, memory, and I/O devices, a guest operating system on an individual virtual machine cannot detect any device other than the virtual devices made available to it as shown in Figure 1. But in the cloud, the information of different domains from different enterprises resides in one place. In the cloud controlled sharing is one of the most important requirement for securing the cloud with disparate domains. One common approach is to make a copy in the high domain, using techniques like one-way data diodes between multiple independent domains. This is depicted in Figure 2 where all

[3]The VAX architectue was not virtualizable and hence some extensions were added to support virtualization.

[4]As opposed to full hardware self-virtualization techniques, paravirtualization changes the guest OS by making hypercalls to the hypervisor in case of sensitive OS instructions.

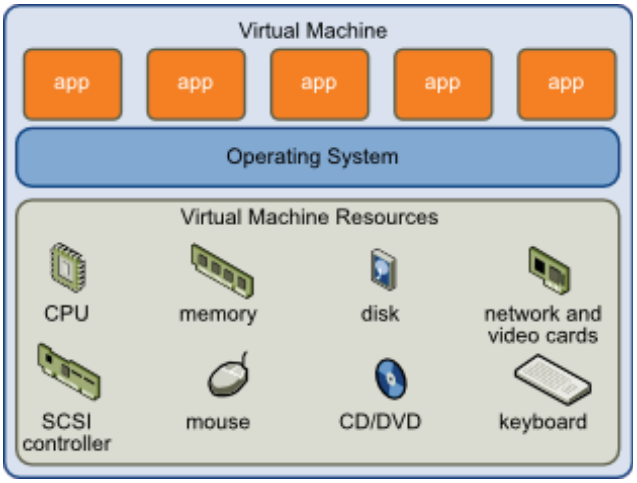

Figure 1: Virtual Machine Isolation

the readable data from a domain is copied to another domain. This can provide high security but is inefficient and even impractical for the massive amounts of data needed from the cloud. That problem is further exacerbated and grows in an combinatorial manner as more domains are involved in network-based operations.

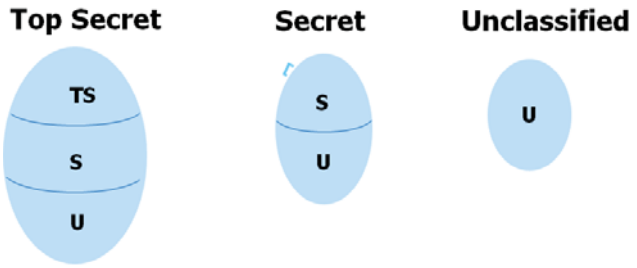

Figure 2: Massive copies of same information in disparate domains

Cross-domain solutions (CDSs) provide another way to fulfill this requirement when the information passes from one domain to another. But again presently there are few if any high assurance CDS solutions available. Network based orchestration relies on untrusted application to connect the end-points on disparate domains. Emory Anderson created a subversion demonstration with an attack on the Network File System in the Mandrake version of Linux. The artifice, consisting of a total of eleven lines of source code, was distributed in several locations within the operating system, and was activated and deactivated by a single corrupted UDP packet[7]. This shows us that it merely takes good programming expertise t subvert a system if underlying platform does not provide high assurance which is in the case of this attack on NFS server on Linux. Any protection mechanism can be rendered ineffective by modifying it so that the protection mechanism is bypassed. With the risk that the artifice can be planted in the kernel, no application layer security solution can be counted on to provide the protection for which it is designed[6]. Computer security is a general term which can be used to describe defenses against everything from wire tapping to sophisticated software attacks using subversion, like "Trojan horses" and "trap doors". Firewalls, cryptographic mechanisms and other software based security tools would provide protection to information in transit but not while the information is at rest or during computation.

On Multics, as on many systems, the first line of defense is a set of tables which lists users and their access rights to data- what is commonly termed an access control list (ACL). These tables are scanned by the operating system on each user's reference to a block of data. This might seem to be a simple and unbreachable defense. But in reality, a fundamental and intrinsic limitation of discretionary access control policy (DAC) is that it is not possible to prevent unauthorized information flow [16]. To prevent unauthorized flow between disparate domains, the more powerful mandatory access control (MAC) policy is required. The Bell-LaPadula model [8] gave a verifiable path to secure systems by proposing a set of rules to be followed. The model provides a proof that the the security is maintained in every state (i.e through inductive nature). The model was based on the concept of labels encompassing the user clearance and object's classification. Based on a well-defined dominance relation, the labels are compared and the access to the information is mediated. This provides a secure way by which information is shared in a controlled fashion. Though the model does not address the problem of covert channels explicitly, but the secure systems like GEMSOS, VAX VMM, SCOMP, Multics Guardian, which have been interpretations of BLP model, demonstrates a framework to effectively analyze and address the covert channels as evident from [20], [18].

The computer utility approach provides controlled sharing as it is one of its specifically enumerated requirements for a system to be qualified as a computing utility. The Access Isolation Mechanism, the extended access control system with a composition of discretionary controls and non-discretionary (i.e. MAC) mechanisms used on some Multics systems was one of the first attempt to achieve this [16]. This computer utility was in use at Pentagon and General Motors which allows us to fairly assert that it can be used for controlled sharing of the information. But our analysis could not find this controlled sharing requirement inherently implemented in or provided by any present cloud architectures.

3.3 Assurance

The current approaches to the needed controlled assured sharing have serious limitation for both security and efficiency reflecting that security is not a core-value proposition. Present day cloud architectures primarily use incomplete techniques like testing. Unfortunately they fail to address one core problem that is essential to a secure cloud: the problem of subversion by a witted (determined) adversary. A witted adversary needs to just find (or introduce via subversion) one flaw in the system. That witted adversary does not care if the system is secure against other attacks. This is one of the reason there is a compelling requirement for Class A1 assurance systems (evaluation class in TCSEC/Orange Book) which provide verified protection so that the system substantially addresses the threat of subversion of TCB.

Security is not an add-on. The systems to which security en-

forcement mechanisms have been added, rather than built-in as fundamental design objectives, are not readily amenable to extensive analysis since they lack the requisite conceptual simplicity of a security kernel. This is because their TCB[5] extends to cover much of the entire system. Hence, their degree of trustworthiness can best be ascertained only by obtaining test results. Since no test procedure for something as complex as a computer system can be anywhere near exhaustive, it is always likely that a subsequent penetration attempt would succeed. It is for this reason that such systems must fall into the lower evaluation classes [27].

SELinux is a modified version on Linux kernel with security specific module added to it. It is interesting to compare the complexity and size of SELinux with that of Multics which achieved much of its security by structuring of system and minimizing complexity. The ring 0 of Multics of 1973 occupied about 628K bytes of executable code and read only data. This was at that time considered to be a large system- in fact far too large to be high assurance without being restructured to have a much smaller security kernel. By comparision, the size of just the SELinux module with an example policy code and read-only data has been estimated to be 1767K bytes. This shows that just the example security policy of SELinux is more than 2.5 times bigger than the entire 1973 Multics kernel and that does not count the size of Linux kernel itself [17]. The reason for comparing the size of the security kernel in terms of the lines of code is the overall complexity of the system which increases as the TCB becomes larger and more complex. TCB is the totality of the hardware and software which provides implmentation of a kernel, and it needs to be minimized so as to assure that the attack surface for the determined adversaries is less. Given that complexity is the single biggest enemy of security, it becomes evident the SELinux designers probably failed to carefully examine whether or not there is a complexity problem to be addressed. Given that security related module (known as Linux Security Module(LSM)) is added on, it is not surprising that the assurance with which it was evaluated was the low assurance EAL 4+ according to Common Criteria evaluation classes.

SEVMS VAX is another example which was the security enhanced version of Digital's OpenVMS VAX Version 6.1 operating system. SEVMS VAX was designed for secure data processing environements that requires MAC, multiple levels and categories of classifiesd data, and users of varying security clearances [9]. Not surprisingly, SEVMS VAX version 6.0 was considered evaluable by the NCSC as Class B1 secure operating system, using TCSEC.

From the two examples above, it can be inferred that security if added on later on fails to address the problem of subversion and provides low assurance. For Multics to meet the security requirements for a computer utility, the security features for mandatory access controls (MAC) were added on later. These were known as Multics AIM (Access isolation Mechanism). Our analysis did not find any current cloud service offerings with a suitable architecture for security. An architecure which provides MAC, let alone high assurance, needs to be based on the basic builiding block required for contolled access and sharing, i.e., Mandatory Acces Control policy. To efficiently achieve MAC policy with high assurance it is required to leverage hardware that provides segmentation and protection rings. Figure 3 indicates the segmentation implemented in Multics. Here the Core refers to the place where the segment resides in the memory. A CPU can directly address the memory using segments where the access rights annotated as "acc" facilitate the controlled sharing.

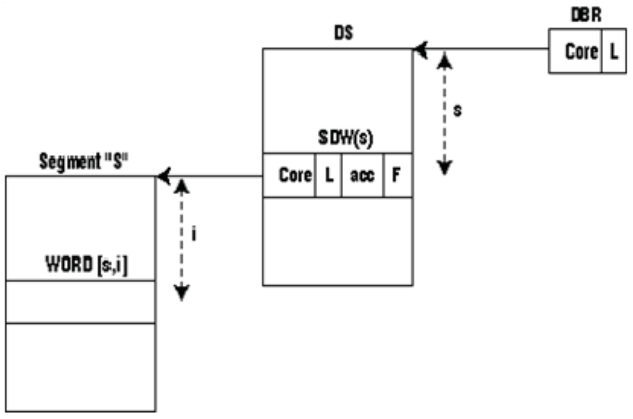

Figure 3: Segmentation in Multics

According to the Orange Book evaluation classes, a distinguishing difference between Division B and Division A is the substantial protection from subversion provided with Class A1 assurance level even though both divisions require support of MAC policy. Mitigation of subversion at the TCB itself is primarily what makes a system secure in the face of a witted (determined) adversary. With TCSEC as a metric to evaluate a system for assurance, there is no evidence of SELinux being evaluated at level higher then B1 or SEVMS VAX and Multics AIM higher then B2. Thus, any architecture with such low to moderate assurance systems at the base fails to address the problem of subversion. On the flip side, considering security architectures with systems like GEMSOS, VAX VMM at its base provides an effective protection from subversion with having high assurance security ingrained in it from beginning of its design. Trusted Networks Interpretion (TNI) gives a consistent set of design principles for laying out a Network Security Architecture Design (NSAD) that can be adopted to build a high assurance cloud architecture. The incipient architecure that we have suggested here in Figure 6 and Figure 7 leverages these principles and this architecture can be further built into a complete NSAD. Figure 7 represents different clients interacting with the MLS verifiable security kernel. The architecture makes no comments about clients being at different levels, but even so, this architecture assures that there would not be any exfiltration of unauthorized information from one enclave to other. On the server side, where the cloud is conceptually realized, there is a persistent storage which intracts with various SaaS services and the NFS file server. Though the details have been abstracted but there has been a successful effort in past in form of GARNETS [14] to leverage the high assurance GEMSOS/GTNP base

[5]TCB: Trusted Computing Base is the totality of hardware, software and firmware which provides implementation of a security kernel.

to provide a file system implementation. Interfacing it with the standard Linux middleware, one can create a high assurance NFS server implementation over a verifiably secure MLS platform. The present day architectures fairly lack high assurance security of this order.

4. CLOUD PLATFORMS

The security of a system as a whole is equal to that of the weakest link of the system. Making some components more secure than others does not serve the purpose. The system needs to be built with a policy in mind and every component should with consistent assurance implement the policy elements allocated to it by the security architecture of the cloud system. The overall cloud security architecture needs to address all the three requirements mentioned in previous section: isolation, controlled sharing, and high assurance. These should have an implementation of the reference monitor because one cannot utter high assurance MAC and controlled sharing without a reference monitor as the foundation. The security kernel, which implements the reference monitor, should be high assurance so as to thwart subversion from a witted adversary. From our analysis, the computer utility provides a mature and proven technological base for the cloud.

Multics (Multiplexed Information and Computing Service) is a multi-user operating system begun in 1965 and used until 2000. The system was built as a part of Project MAC at MIT where the design intentions were to build it as a fully support commercial products that could support the computer utility concept. Multics was designed to be secure from the beginning, and incorporated MAC in the course of its development. The Multics memory architecture divides the memory into segments and uses paged memory in the manner pioneered by the Atlas system [4]. Segmented addresses generated by the CPU are translated by hardware from a virtual address to a real address as illustrated in Figure 3. As opined by David Jordan in an article on which the paper Multics Data Security is based:

> *"When Multics was developed, an attempt was made to design a system, including security mechanisms, which could grow without system reorganization. The designers recognized that it would be impossible, at the design stage, to anticipate all the problems which would crop up when the software was written. Therefore, if problems arose as a module of the system was implemented, it, was redesigned, a process which served to reduce the convolution and complexity of the final software system. In addition, provision was made to allow functions to be added to the system as subsystems rather than as modifications of the operating system itself."*

Initially Multics was not built with mandatory access controls but in response to clear and demanding requirements from both their governement and commercial customers for deployment in multi level environments, MAC was made an integral part of its security capabilities. Project Guardian grew out of the ARPA support for Multics and the potential sale of Multics systems to both the US Air Force and the auto industry, e.g., General Motors. USAF wanted a system that could be used to handle more than one security classification of data at a time. Project Guardian led to the creation of the Access Isolation Mechanism (AIM), the implementation of the labelling and Bell-LaPadula star property support in Multics [28]. AIM was implemented in response to General Motors' demands to separate and protect engineering data while providing controlled sharing of less sensitive administrative data, and a Pentagon request for a mechanism which would enforce military security policy. Multics AIM was later evaluated at Class B2 according to Orange Book.

On systems which use both the ACL and AIM mechanisms, the user's effective access to a segment is determined by the most restrictive of the two. The DoD Orange Book was influenced by the experience in building secure systems gained in Project Guardian. The main aim of Project Guardian was to produce a high assurance minimized kernel for Multics, as complexity is the single largest enemy of security, and produce an auditable version of Multics. The implementation of Multics[6] in the 60s and 70s preceded the ideas of a minimal security kernel; several facilities in Ring 0 were included for efficiency, even though they didn't need to be there for security. In 1974, Ring 0 comprised about 44,000 lines of code, mostly PL/I, plus about 10,000 lines of code in trusted processes [28]. Several design documents were produced in shrinking the size of the kernel which entailed moving non-security functions out of Ring 0. Air Force ESD was directed to discontinue computer security research by the Office of the Secretary of Defense in 1976 which led to Multics guardian project getting cancelled without shrinking Ring 0.

At around the same time Honeywell initiated a project called SCOMP (Secure Communications Processor) to create a secure front-end communications processor for Multics. SCOMP included hardware specially designed for security and formally verified software design. Although Project Guardian design for a Multics security kernel was not implemented, the use of Multics features to provide multilevel security was pursued in the SCOMP effort. The hardware base was a Honeywell Level 6 minicomputer, augmented with a custom Security Protection Module (SPM), and a modified CPU whose virtual memory unit obtained descriptors from the SPM when translating virtual addresses to real. All authorization was checked in hardware. As illustrated in Figure 4, SCOMP had a security kernel and 4 hardware rings [11]. The SCOMP OS, called STOP, was organized into layers: the lowest layer was a security kernel. A Formal Top Level Specification (FTLS) for the system's function was verified to implement DoD security rules [11]. In 1983 SCOMP was evaluated to Class A1, the highest class according to the Orange Book. However, it did not maintain the Class A1 rating beyond the initially evaluated version.
The Gemini Multiprocessing Secure Operating System (GEMSOS), a Gemini Computers Inc. product, was an Class A1 evaluated trusted operating system based on a security ker-

[6]The security mechanisms provided in initial Multics were somewhat ad hoc. The lack of a simple model of security meant that even if an auditor were to undertake the previously mentioned overwhelming task of understanding every line of code, that auditor would lack a systematic specification of what to look for [25].

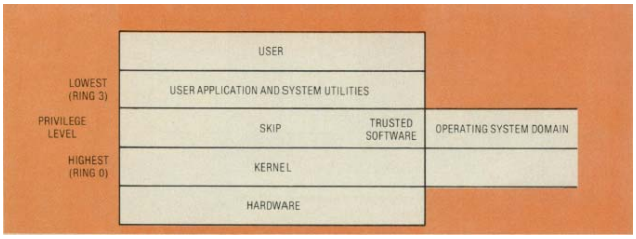

Figure 4: SCOMP software architecture

nel. The GEMSOS Final Evaluation Report [10] discusses, at several places, using GEMSOS as a Virtual Machine Monitor (VMM) to implement virtual machines on top of it. GEMSOS[7] is a multiprocessor real time operating system (RTOS) that leverages the Intel IA-32 processor architecture to implement a Reference Monitor that verifiably enforces Mandatory Access Control (MAC) policies. GEMSOS delivers full multilevel security (MLS) capabilities with the highest assurance, as confirmed by previous NSA Class A1 evaluations and deployments. The GEMSOS security kernel design was significantly influences by the Project Guardian Multics kernel design. Consistent with that, with GEMSOS controlled sharing, sensitive processing can directly access less sensitive data without massive write-up copies. Yet strong separation of processes and protection domains allow strict isolation where required. In a cloud environment where we recognized isolation too is a dominant requirement alongwith controlled sharing, GEMSOS provides both with high assurance. High assurance is delivered "out of the box" with no need for trusted application code.

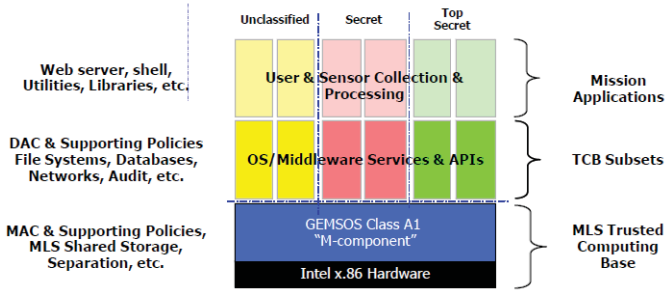

Figure 5: GEMSOS delivers MLS Sharing "Out of the Box" among strongly separated partitions

5. RELATED WORK

Historically, many efforts have been made to secure information in a multi-level environment. The baseline approach of all the successful efforts was implementing the underlying security policy correctly with high assurance by conformance to a formal security policy model such as Bell-LaPadula. HSRP (Headquarters System Replacement Program) [12] was a large scale Management Information System acquisition for the USAF 7th Communication Group (7CG) located at Pentagon. The primary requirement of the project was to replace the Honeywell Multics systems with AIM, when the vendor was acquired and discontinued the product. The existing system was to be replaced by an initial COTS Class C2 implementation which would conceptually evolve to Class B2 and finally a Class A1 implementation as defined in TNI (Trusted Network Interpretation) of TCSEC. The Class B2 and eventual Class A1 implementation is based on introduction of a Secure Communications Processor Environement(SCPE) which would be based on GEMSOS security kernel.

SCOMP, as discussed in the previous section, was evaluated at Class A1 but it did not maintain this evaluation class. A distant relative of SCOMP on different hardware and software is presently available in the trusted operating system, STOP, is today distributed at a relatively low assurance level of EAL 4+ on the XTS-400 platform. BAE XTS-400 platform is used by Naval Postgraduate School for Monterey Security Architecture (MYSEA), an MLS cloud [22] research project. Its a federated architecture with multiple MLS servers which jointly enforce system wide MLS policy. The platform is designed to be evaluated against Common Criteria EAL 5. It consists of TPE (Trustted Path Extension), conjoined to every workstation to provide trusted path to MYSEA cloud, and TCM (Trusted Channel Module), to provide labelling of network traffic from multiple single-level networks. These two components are based on Least Privilege Seperation Kernel. But since by its very definition a seperation kerne inherently does not implement a reference monitor, there is little basis for assurance that it mitigates against subversion.

Gemini Computers (currently a subsidiary of Aesec Corporation), the developers of verifiably secure GEMSOS-based platfroms , is developing a MLS File Server that will support a controlled access and controlled cross-domain file sharing solution (CDS), including for cloud storage. Prior to that development the current high-assurance approach to maintaining the security of multidomain environments is to duplicate the hardware for each domain, which is expensive and which complicates sharing in many applications that depend on it. The Gemini MLS File Server is designed to support the Network File Service (NFS) file protocol where workstations, servers, and applications process information of a single classification level while allowing data sharing between the various levels. The prototype server will have separate Network Interface Cards (NICs) for each security domain. No trusted code outside the GEMSOS TCB is required in order to provide low-to-high controlled sharing via MLS read-down from higher domains to lower domain files and directories.

6. CONCLUSION AND FUTURE WORK

This paper focuses on the secure system implemetations that have been done in the past and how the methodologies adopted at that time be used today to address security in a cloud with disparate domains. Even if the cloud was not a buzzword then, the properties of sharing, assurance and isolation were still prominent as evidenced from *computer utility* requirements. Thus, this paper does not intend to innovate in terms of a new cloud technology but rather aims to apply the security constructs codified in TCSEC/TNI to analyze the current cloud security. This makes clear that

[7]GEMSOS and GTNP (Gemini Trusted Network Processor) are used interchangeably in this paper since GTNP used GEMSOS security kernel.

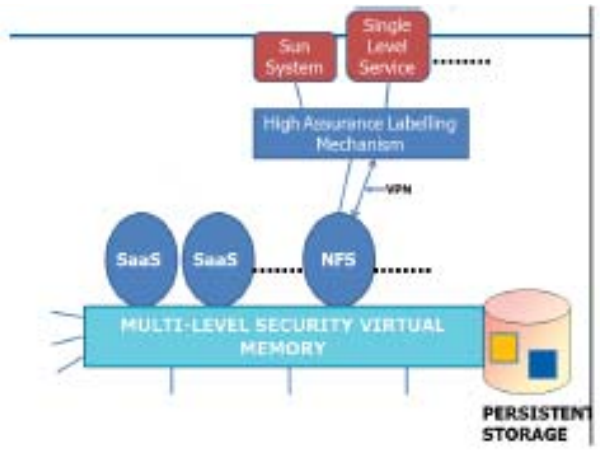

Figure 6: Server-Side Architecture

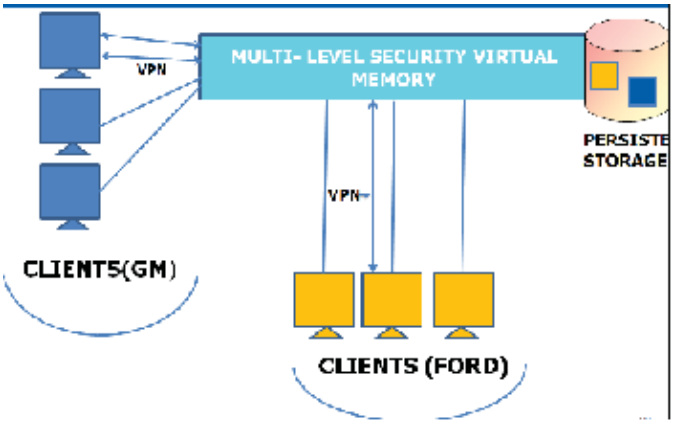

Figure 7: Client-Side Architecture

current practice is far short of providing the required sharing, assurance and isolation. In contrast these are evident in the computer utility formulation, that have been demonstrated for high assurance mandatory access control to provide effective sharing and isolation in the face of attacks like subversion by a witted adversary.

Through a comprehensive analysis of available highly secure kernels shown in this paper, we propose an initial blueprint of the cloud security architecture as shown in and Figure 6 and Figure 7 in its most rudimentary level. This architecture can be used as a basis to provide MLS verified protection in a cloud with disparate domains. The architecture is laid out by applying the principles codfiied in TCSEC/TNI by the experienced investigators in this field. We aim to build an assured system from assured components.

One of the most prominent IT analysts, Gartner, identified cloud computing as the primary source of growth in IT spending, increasing over 20 percent year-on-year to global revenues of 56 billion dollars in 2009 and surging to over 130 billion dollars by 2013 [23]. This demonstrates the importance of cloud computing today. But presently available cloud architectures, have not addressed the Advanced Persistent Threats (APT), that are illustrated by the threat of system subversion evident in attacks like Stuxnet and the one demonstrated by Anderson[6]. A recent Microsoft survey found that: *"58 percent of the public and 86 percent of business leaders are excited about the possibilities of cloud computing. But more than 90 percent of them are worried about security, availability, and privacy of their data as it rests in the cloud"* [23].

Multi-tenant clouds with data from disparate security domains require strong separation,controls on data access, configuration and executable sharing, and a system-based approach to assuring customers their data will remain private to their domain unless and until the system policy explicitly authorizes them to share it. Although the VMMs used today are not even close to high assurance needed for this isolation, not to mention their inability to support controlled sharing, the the technological state-of-the-art suggests that this is possible, all that is required is a sound architetural design implementing MAC and providing high assurance. The design principles of a computer utility, as Multics was espoused, can be leveraged while designing the current cloud product because they are largely encompassed by the computer utility vision of future. A verifiable MLS security kernel, like GEMSOS, should be used to implement applications on top of it which provides controlled sharing of information between different levels with high assurance. The GEMSOS product is available today as a commercial off-the-shelf (COTS) product under a proven OEM business model.

7. REFERENCES

[1] Amazon web services: Overview of security processes. `https://media.amazonwebservices.com/.../AWS_Security_Whitepaper.pdf`.

[2] Apple's icloud cracked: Lack of two-factor authentication allows remote data download.

[3] How did hackers steal celebrities' private icloud photos? `http://www.dailymail.co.uk/sciencetech/article-2739764/`.

[4] Multics history. `http://www.multicians.org/history.html`.

[5] A wakeup call for the cloud. `http://www.networkworld.com/article/2366862/iaas/a-wakeup-call-for-the-cloud.html`.

[6] E. A. Anderson. *A Demonstration of the Subversion Threat: Facing a Critical Responsibility in the Defense of Cyberspace*. PhD thesis, Monterey, California. Naval Postgraduate School, 2002.

[7] E. A. Anderson, C. E. Irvine, and R. R. Schell. Subversion as a threat in information warfare. Technical report, DTIC Document, 2004.

[8] D. E. Bell and L. J. LaPadula. Computer security model: Unified exposition and multics interpretation. *MITRE Corp., Bedford, MA, Tech. Rep. ESD-TR-75-306, June*, 1975.

[9] S. BiGtcky, K. Lynch, and S. Lipner. Se/vms: Implementing mandatory security in vax/vms. *c IIV% Oeoov*, page 47.

[10] N. C. S. Center. Final evaluation report for the gemini trusted network processor. 1995.

[11] L. J. Fraim. Scomp: A solution to the multilevel security problem. *Computer*, 16(7):26–34, 1983.

[12] D. Gambel, Fordham, and M. Walter. Hsrp-ai'ing a large-scale management information system.

[13] M. Greenberger. *The computers of tomorrow*. Atlantic Monthly, 1964.

[14] C. E. Irvine. A multilevel file system for high assurance. In *Security and Privacy, 1995. Proceedings., 1995 IEEE Symposium on*, pages 78–87. IEEE, 1995.

[15] W. Jansen, T. Grance, et al. Guidelines on security and privacy in public cloud computing. *NIST special publication*, 800:144, 2011.

[16] D. Jordan. Multics data security. *SCI. HONEYWELLER.*, 2(2):44–53, 1981.

[17] P. A. Karger and R. R. Schell. Thirty years later: Lessons from the multics security evaluation. In *Computer Security Applications Conference, 2002. Proceedings. 18th Annual*, pages 119–126. IEEE, 2002.

[18] P. A. Karger, M. E. Zurko, D. W. Bonin, A. H. Mason, and C. E. Kahn. A retrospective on the vax vmm security kernel. *Software Engineering, IEEE Transactions on*, 17(11):1147–1165, 1991.

[19] C. Kaufman and R. Venkatapathy. Windows azure security overview. *go. microsoft. com*, 2010.

[20] T. E. Levin, A. Tao, and S. J. Padilla. Covert storage channel analysis: A worked example. *Proc. National Computer Security Converence*, pages 10–19, 1990.

[21] S. E. Madnick and J. J. Donovan. Application and analysis of the virtual machine approach to information system security and isolation. In *Proceedings of the workshop on virtual computer systems*, pages 210–224. ACM, 1973.

[22] T. D. Nguyen, M. A. Gondree, D. J. Shifflett, J. Khosalim, T. E. Levin, and C. E. Irvine. A cloud-oriented cross-domain security architecture. In *MILITARY COMMUNICATIONS CONFERENCE, 2010-MILCOM 2010*, pages 441–447. IEEE, 2010.

[23] J. Rhoton. *Cloud Computing Explained*. Recursive Press, 2009.

[24] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 199–212. ACM, 2009.

[25] M. D. Schroeder, D. D. Clark, and J. H. Saltzer. The multics kernel design project. *ACM SIGOPS Operating Systems Review*, 11(5):43–56, 1977.

[26] D. X. Song, D. Wagner, and X. Tian. Timing analysis of keystrokes and timing attacks on ssh. In *USENIX Security Symposium*, volume 2001, 2001.

[27] D. TCSEC. Trusted computer system evaluation criteria. *DoD 5200.28-STD*, 83, 1985.

[28] T. V. Vleck. B2 security evaluation. `http://www.multicians.org/b2.html`, 2014.

A Cost Effective High-Assurance Layered Solution for MLS Test, Training, and Live Virtual Constructive (LVC)

James A. Marek
Rockwell Collins
M/S 182-100 400 Collins Rd. NE
Cedar Rapids, IA 52498
319-295-4225
jim.marek@rockwellcollins.com

ABSTRACT
Modern warfare has increasingly evolved to include multiple variants of platforms and participants, including coalition partners, which are hosting and exchanging information spanning a wide range of classification/caveat/compartment levels. This has led to a difficult Multi-Level Security (MLS) challenge that must be addressed both technologically and operationally in order to effectively test and train modern warfighters. The ongoing evolution to include Live, Virtual, and Constructive (LVC) within the modern training curriculum is exceeding the current capabilities of existing security solutions, resulting in the need for a cost effective solution to address the problem. Rockwell Collins has developed a layered solution for high-assurance MLS test, training, and LVC to address this challenge. The solution takes advantage of several Multiple Independent Levels of Security (MILS) technologies along with a focus on modularity and composability across the solution. This has led to a very efficient and flexible solution that is an ideal case study for layered assurance.

Categories and Subject Descriptors
H.1.1 [**Information Systems**]: Models and Principles Systems and Information Theory – *General systems theory, Information theory, Value of information.*

General Terms
Design, Security.

Keywords
Test Range Infrastructure, Training Range Infrastructure, Live Virtual Constructive (LVC), Multi-Level Security (MLS), Multiple Independent Levels of Security (MILS), Simulation and Training Systems, Cost effective MLS implementation

1. INTRODUCTION
In looking at the solution space to address modern test and training and LVC interoperability issues, it becomes apparent that range instrumentation is an ideal solution space to leverage the principles of layered assurance. This space includes significant security-related scope as well as safety issues which must be addressed. High assurance/robustness of the solution is critical to address these issues. However, cost and schedule viability are a serious consideration as with any modern military development effort. This paper describes a solution which is based on the principles of layered assurance and composability at multiple levels in the architecture. All of the elements described herein have completed their individual certification efforts (e.g. NSA Type 1, Common Criteria, etc.). The system-level solution is rapidly maturing and is in the process of completing accreditation as part of its fielding.

2. OVERVIEW OF THE PROBLEM SPACE

2.1 Testing
Modern military ranges are testing with a variety of platforms that can each host and share information at a wide range of classification/caveat/compartment levels. For example 5th generation platforms (e.g. F-35) and 4th generation platforms (e.g. F-18) have communications links, weapons, and sensors that are processing data at different levels of security thus requiring different keys and infrastructure at the range to support tests with each platform. In addition, the humans working with and in these platforms also have limits on clearance and "need to know". Multiple test missions may be simultaneously occurring at the range and different sets of personnel will be accessing different sets of the information which is flowing around the range on common infrastructure.

2.2 LVC Training
A typical training scenario will include 3rd, 4th, and 5th generation platforms, as well as Unmanned Aerial Systems (UAS), each with different security levels and requirements for exchange. The weapons being simulated also have a wide range of classification levels. Datalinks and sensors have a variety of performance parameters and capabilities which also span a number of classification levels. Add to that, the desire to interoperate with coalition partners and the requirements continue to expand. The current state of training exercises results in segregated information flows that lead to a reduction in safety as well as training effectiveness by keeping necessary information from being shared between the participants who need it the most.

2.3 Common Problem
All of this leads to the need for an MLS information exchange solution that enforces the operational security requirements, but enables testing and training to be performed efficiently and successfully. In order to test or train effectively in modern environments, one must be able to affordably control the flow of information among the players while ensuring that all information (which must be exchanged) is supported in a timely manner. In addition, the ranges often want a low impact scalable solution that supports system-high (at a single-level) through certified MLS operation.

3. GUIDANCE FOR THE SOLUTION SPACE

The following general guidelines have been developed as a means to bound the problem and direct the solution space toward scalable solutions that can be cost effectively implemented and technically useful for addressing the current test, training, and LVC functional requirements.

Table 1: General MLS Guidelines

Guideline
Perform all data processing in the lowest possible (level/caveat) security enclave.
Follow relevant Defense Information Systems Agency (DISA), National Security Agency (NSA), National Information Assurance Partnership (NIAP), and Department of Defense (DoD) services guidance documents for secure configuration of applications and devices in the MLS LVC environment.
Where possible use a MILS-based approach to support Multiple Single Level (MSL) and MLS information processing.
Where the MILS-based approach is insufficient and the MLS requirements do not drive spanning more than 2 levels (e.g. Top Secret-to-Secret (TS-S) or Secret-to-Unclassified (S-U)), use a traditional label-based MLS approach (nesting MLS within a MILS partition is also an option to reduce SWaP).
MLS processing implementation should be scalable to support multiple system-high / single-level partitions with no cross domain information flow through all-way guarding between enclaves.
Safety channel or allocation in the training range data link to support low latency high priority safety traffic.
Utilize Black-side only training data link, as opposed to a data link that spans both Red and Black sides.
Operational security mechanisms (non-technological processes) are an essential part of a comprehensive security solution.
Ensure solution is capable of supporting operation at System-High at a single-level through certified MLS operation with a single equipment set.

Table 2 Cross Domain Solution (CDS) and Encryption Guidelines

Guideline
Bind classification/caveat/compartment labels to information.
Include meta-data where possible on which element of a data item drives the classification label level.
Utilize encryption systems that support CIK-less operation and remote ignition and key distribution/loading.
Encryption system should support single channel and multi-channel system-high operation with key agility and the ability to scale to support MLS with minimum impact to accreditation, thereby allowing customers to gradually transition to MLS.
Encryption systems should not be the bottleneck for training data link throughput. Determine the aggregate throughput required for the training data link and size the encryption system to support that rate.
Participant package should include MLS encryption capable of supporting recording storage and retrieval of relevant sensitive training exercise information (data-at-rest)
Participant package should include MLS encryption capable of supporting communication of relevant sensitive training exercise information over-the-air (data-in-transit)
Ground infrastructure to support MLS encryption capable of supporting communication of relevant sensitive training exercise information over-the-air (data-in-transit)
Utilize MILS or MLS cryptographic systems which include support for a range of classification levels, caveats/compartments, and key agility.
Utilize cross domain guards (transfer CDS) which support the “all-way” guarding function.
Utilize cross domain guards (transfer CDS), virtual where possible, physical where not, to mediate cross domain information flow.
Where possible use “access CDS” for MLS common operational picture when required.
Label low information and utilize low overhead one-way data diodes to pass information from lower enclaves to all higher enclaves.
Outside of traditional malware/virus and otherwise network attack-based detection avoid use of “smart” cross domain functions for low-to-high traffic flow.
Leverage smart cross domain functions to perform high-to-low filtering and downgrading.
Assume a minimum of three enclaves are needed, but ensure that the architecture supports the flexibility to address more when/if necessary.

4. DETAILS OF THE SOLUTION

Rockwell Collins has broken the solution down into several architectural elements which are targeted at providing a modular solution with an eye toward reusability across the ground infrastructure as well as airborne/mobile assets. The solution also takes advantage of previously evaluated elements which are composed in a layered manner to provide high assurance at the system level based on these underlying building blocks.

For a typical test or training application, the system of systems can generally be divided into two parts: the ground-based elements (generally stationary), and the airborne or mobile elements. The airborne or mobile elements are usually attached to or installed in participant packages. These modular reusable elements have been packaged in a variety of form factors and can be re-packaged as needed to address a wide range of platforms.

4.1 Airborne/Mobile Elements

4.1.1 Element 1: MLS Participant Interface Module (PIM)

The PIM is a Multi-Level MILS processing environment supporting configurable interfaces (Fibre Channel, 1553, Ethernet, Serial, etc.). The separation kernel is a Common Criteria Evaluation Assurance Level (EAL) 6+ certified MILS Real-Time Operating System (RTOS) that hosts the multiple single-level enclaves and a modular NSA-evaluated Cross Domain Guard (CDG) supporting TS-U information processing and flow control. The solution supports system-high operation with single or multiple channels as well as certified MLS operation for military users who do not need MLS or only want a path to MLS in the future. The fundamental building block of this element is the processor board which is one of the key building blocks of the system and provides mandatory access control along with other security and information flow policy enforcement. As shown in Figure 1 below, the component is a MILS based implementation using a separation kernel to provide the separation policy enforcement. Independent of that, the Filter Engine adds another layer to the composable security policy enforcement providing the cross domain filtering. Finally, a labeler and label checking function (shown as the purple box with an "L") provides the ability to bind labels to information packets within the system or check labels of packets. This module works synergistically with the End Cryptographic Unit (ECU) and the Multi-Channel MLS Mission/Debriefing Room Cross Domain Guard (MMMDR-CDG) to provide appropriate information flow policy enforcement at the system level. The enclave applications represent a range of options from processing of participant information from the host platform to execution of weapon or threat simulations. The Filter Engine enables information flow across the domains such as command/control, Built-In Test (BIT)/Status, Real-Time Kill Notification (RTKN), etc. One additional key aspect of this implementation is that all of the enclaves, as well as the guarding rules, are configurable in conjunction with the setup of the ECU. This benefits the user by providing a layer of flexibility that allows the same hardware and core software to support a wide range of both participant platforms as well as I/O interfaces which lowers total lifecycle cost. Each of the external interfaces is assigned to a partition (security level) based on the configuration. All of this capability is contained in a 1.5"x 3.5" x 6.4" package, making it easy to embed in a wide range of platform level packages, thus demonstrating that MLS can be packaged in a constrained form factor.

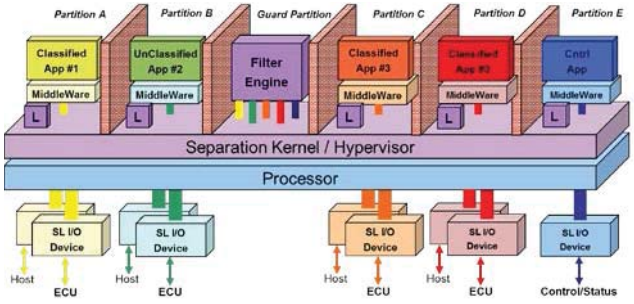

Figure 1: PIM Architecture

4.1.2 Element 2: Multi-Channel MLS NSA Type 1 End Cryptographic Unit (ECU)

The ECU is a MILS-based multi-channel MLS encryption unit that abstracts the encryption from the datalink. This provides datalink-agnostic encryption at the IP networking layer. It operates much like an IP router. It also provides data-at-rest encryption for on board storage of mission data in support of post mission processing. In addition, the encryption unit also provides another layer of assurance for the labeling provided by the PIM by checking the labels for each channel against a pre-defined (at configuration time) table on a packet-by-packet basis. Thus, if an error occurs in the PIM, it will be detected at the ECU. The ECU supports single channel and multi-channel system-high operation with key agility along with the multi-channel MLS capability so customers can gradually transition to MLS if desired. The core of the ECU is the Janus encryption engine which is based on the Rockwell Collins Gemini Encryption Engine and the MILS certified AAMP7 hardware separation kernel processor. These two devices provide MILS based MLS encryption that has completed NSA Type 1 certification supporting simultaneous key handling of TS and below (TSAB) key material, as well as simultaneous encryption/decryption of multiple differing classification levels (one in each channel). Five Red side and 3 Black side interfaces are included to support a variety of enclaves on the Red side and over the datalink, local encrypted storage on the Data Recorder Device (DRD), as well as a local wired connection for the Portable Test Set (PTS) or other wired test equipment on the Black side. All of this capability is contained in a 1.5" x 3.5" x 6.4" package which typically consumes 12W of power when using all channels. Figure 2 depicts the ECU architecture highlighting the MILS approach and layered architecture.

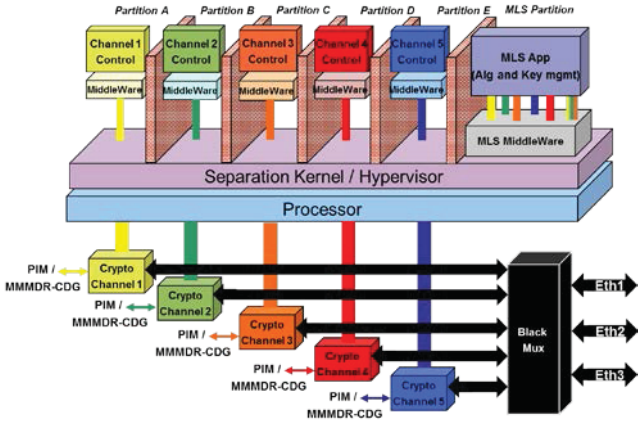

Figure 2: ECU Architecture

4.1.3 Element 3: High-Throughput, Mobile Ad-Hoc Networking (MANET) Data-Link (HT-DL)

The datalink network employs uplink, downlink, and peer-peer crosslink services with packet rates roughly 4 to 5 times greater than legacy pod-based range instrumentation. Relay routes are self-forming, out to 4 hops. Manual routing can also be managed between user-selected nodes. Datalink range for a single-hop route is 100 nmi air-air, and 130 nmi air-ground.

The datalink at 6.6 lbs. is miniaturized to roughly half the weight of existing equipment. The datalink is partitioned into a Transceiver Modem (TRM) module and a Power Amplifier (PA)

module, measuring 6.6" and 11" in length, respectively, and each having a 3.5" x 1.4" cross section.

The datalink can be configured to support built-in Type 3 encryption for sensitive but unclassified operations. Because the Type 1 encryption is provided external to the datalink, the system is more modular to enable alternate datalinks to be utilized for specific applications or for easy upgrade of datalink functionality without requiring recertification with the NSA.

4.1.4 Element 4: High Accuracy Time Space Position Information (HA-TSPI)

Although not a part of the security functionality or layering of security, the TSPI is a critical element of any test/training system. The information feeds testing exercises as well as on-board weapon and threat simulations. Much like the datalink, by isolating this function to an independent element, the modularity, and composability of the system is enhanced. The HA-TSPI leverages the Rockwell Collins high accuracy miniature Selective Availability Anti-Spoofing Module (SAASM) GPS and state-of-the-art Inertial Measurement Unit (IMU) technology tightly coupled together to provide Real-Time Horizontal (x, y) and Vertical (z) position accuracy of 0.5 meters RMS, Real-Time Horizontal (x, y) and Vertical (z) velocity accuracy of 0.03 m/sec RMS, and Real-Time Attitude accuracy of 0.1 degrees RMS.

4.1.5 Element 5: User Interface (UI) with high capacity Data Recorder Device (UI-DRD)

The UI-DRD provides a modular UI which can support remote key loading and zeroization for encryption and GPS key material within the ECU and HA-TSPI as well as hosting a solid state storage media for mission data recording. The storage media is also able to store configuration data and files in addition to the over-the-air configuration option.

4.2 Ground-Based Elements

For the most part, all ground elements are 19" rack mounted units; however, significant module-level reuse is achieved based on the airborne elements.

4.2.1 Element 1: Multi-Channel MLS Mission/Debriefing Room Cross Domain Guard (MMMDR-CDG)

Based on the core processor and underlying Common Criteria (CC) EAL6+ RTOS and NSA evaluated CDG used in the PIM, the MMMDR-CDG includes a scalable, rack mounted version of the PIM which is configured to support the filtering of information for mission and exercise debriefing rooms. This filtering is currently based on classification level/caveat as well as mission number. Figure 3 below depicts the software architecture for the MMMDR-CDG. Significant commonality was achieved between the PIM and the MMMDR-CDG.

In order to ease system testing and debug, the PIM and MMMDR-CDG can be directly connected thereby isolating the ECU's and Datalinks from the infrastructure. In the same manner, the ECU's can be directly connected on the Black side thereby isolating the datalinks from the infrastructure. The open and modular nature of this design approach enables much easier integration and testing.

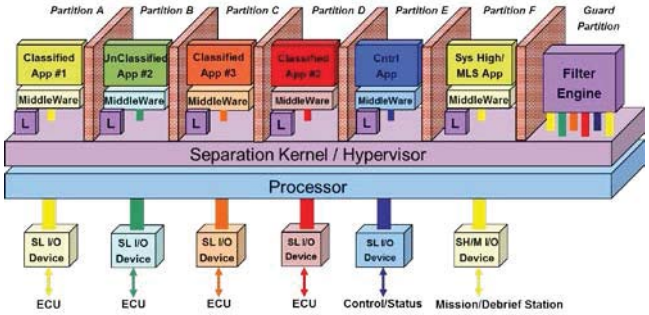

Figure 3: MMMDR-CDG Architecture

4.2.2 Element 2: Multi-Channel MLS NSA Type 1 End Cryptographic Unit (ECU)

This is an identical module as used in the airborne application which has been rack mounted and connected between the Data Link Controller and the MMMDR-CDG. Generally, in a ground based application, this device can also be used to perform over-the-air key distribution and rekey. It also supports two different modes: one mode where it can utilize a Crypto Ignition Key (CIK), and another where it supports CIK-less operation for remote participant ECUs based on over-the-air authentication to a CIK-ignited ECU. Generally this module performs the communication to/from the MMMDR-CDG and the ground-based array of datalinks.

4.2.3 Element 3: Data Link Controller (DLC)

The DLC is a commercial computing platform hosting software that enables management of both ground-based and participant package datalink modules, as well as datalink network, including information flow to and from ground and airborne nodes. Each DLC hosts an EAL4 certified OS and conforms to DISA Security Technical Implementation Guide (STIG) guidelines for cyber security. Thus they play a part in the security role of the layered security architecture.

4.2.4 Element 4: System Controller Workstation (SCW)

The SCW is a commercial computing platform hosting mission and participant management software, including configuration for ground and airborne elements, as well as key distribution. Each SCW hosts an EAL4 certified OS and conforms to DISA STIG guidelines for cyber security. Each is allocated a port on the MMMDR-CDG which filters range traffic to and from the SCW. The SCW generally operates in a "blind administration" mode (not typically accessing range participant traffic, but primarily focused on command and control functions that manage the range assets participating in exercises). The SCW also plays a part in the layered security architecture, as it is capable of managing encryption keys as well as performing other airborne/mobile and ground element configuration and control functions.

4.2.5 Element 5: Mission/Debrief Room Workstation (MDRW)

The MDRW is a commercial computing platform hosting mission management and debrief application software. Each MDRW hosts an EAL4 certified OS and conforms to DISA STIG guidelines for cyber security. Each is allocated a port on the MMMDR-CDG which filters range traffic (live, recorded playback or a hybrid). The MDRW also benefited from reuse of the functionality from the SCW which enabled cost and schedule efficiencies. The

MDRW adds another independent layer of security to the system based on its isolation, independent configuration and management with respect to the other elements of the system.

4.2.6 Element 6: Portable Test Set (PTS)

The PTS is essentially a man-portable miniature ground sub-system capable of being carried around the range to support wired remote operations, as well as configuration, test, and debug of airborne equipment spread across the range locale. Each PTS includes a ruggedized laptop computer that hosts an EAL4 certified OS and conforms to DISA STIG guidelines for cyber security. The PTS also leverages some common software from the SCW and MDRW providing a cost effective functional and security implementation.

4.2.7 Element 7: Remote Ground System (RGS)

The RGS includes a datalink and an optional weather sensor, along with the ability to remotely power manage the RGS elements. One or more RGS datalinks are used to provide area coverage (diversity) for the ground-based DLC (Data Link Controller). The weather sensor augments accuracy when operating an RGS in a GPS-denied mode. Although not a significant security element, the RGS does provide a critical function in the ability to expand the reach of the ground-based systems when needed.

4.2.8 Element 8: Range Gateways

A variety of flexible computing and networking equipment and associated protocol translation and formatting software/firmware are also key elements of the range system to enable inter/intra-range operability and bridging to legacy systems.

4.3 Bringing it all together

As shown in the figures below, the various layers of security all work together to provide a cyber-hardened multi-level information processing, storage, transmission, and handling system for test and training/LVC applications. As previously stated, the system is unique in that it is one of the few systems in deployment that is leveraging the MILS principles across-the-board to support trusted separation of multiple security levels, trusted labeling, trusted cross domain guarding both high-to-low and low-to-high, trusted encryption and decryption, and trusted routing. As stated, the system was designed to support both single channel and multi-channel system-high operation and key agility to enable customers to transition toward MLS over time if desired. In addition to being very modular with open standards based interfaces, the system is designed to be highly configurable such that the processing, labeling, storage, I/O, encryption, and guarding elements are all boot-time configurable for the specific classification level or cross domain flow policy assigned to them. Further, some of the configuration elements can be updated at run-time to address dynamic system reconfiguration requirements. All of these features and capabilities have been evaluated as part of the system accreditation and component certification efforts.

Figure 4 below depicts the context for a participant instrumentation package sub-system. The primary information sources are from the host participant interfaces and the Time, Space, Position information (TSPI) provided by the included TSPI module. These interfaces of the host platform and the TSPI module directly connect to the PIM. The PIM hosts mission or training applications in partitions along with the guard and health/control partition as detailed in Figure 1. Information from the host participant is passed through the PIM interface into a partition based on the classification level of the information. The information is then processed and, if necessary, it is passed to the filter engine within the PIM for cross domain transfer. Otherwise, the information flows in its classification level "swim lane" between the PIM and the ECU. The PIM and ECU have "swim lanes" dedicated to individual security enclaves which are directly connected between them. The ECU is further detailed in Figure 2.

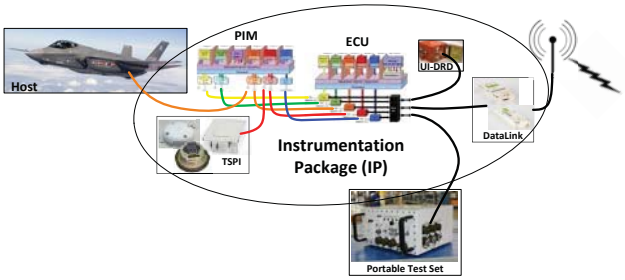

Figure 4: Participant Sub-System

At the ECU interfaces additional checks are performed for information integrity before the payload is encrypted and then passed to the Black side of the ECU where it is routed to the appropriate external interface based on an IP address. The ECU essentially provides a low overhead IP routing function with NSA Type 1 encrypted payloads through IP/Ethernet interfaces on the Red and Black sides. This implementation provides a very low overhead approach which is quite beneficial for over-the-air communications, leaving much more bandwidth available for the data versus alternative approaches such as SSL, IPsec, or HAIPE.

The Black side Ethernet connections support communication through the high throughput datalink as well as Black side recording. The data is passed to the recorder for Data-At-Rest (DAR) storage on the UI-DRD or to the HT-DL for Data-In-Transit (DIT) communication with other participants in the network or the ground control sub-system(s). The Black side recorder DAR capability supports hosting encrypted applications and configuration information for the modules within the instrumentation package.

Additionally, the data can be flowed to the PTS for testing and configuration or to a portable debrief station as necessary, through an available third Ethernet port.

Finally, the instrumentation package hosts a datalink which supports communication to/from other participants and any ground control systems. As discussed earlier, the datalink also has integrated Federal Information Processing System (FIPS) 140-2 certified (Type 3) encryption where sensitive but unclassified information is being communicated between participants and Type 1 encryption is not desired.

Figure 5 below depicts the context for a ground control sub-system. The ground control system functions very similar to the participant instrumentation package sub-system. Communications with participants is performed via the datalink.

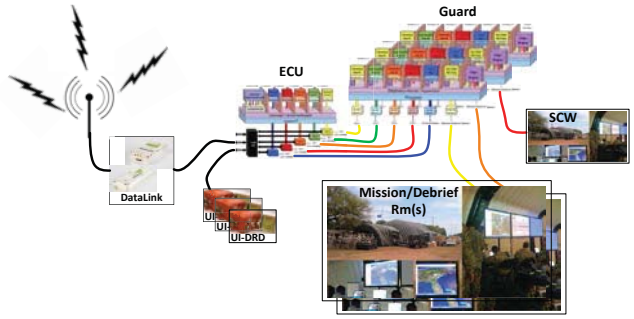

Figure 5: Ground Control Sub-System (GCS)

The information flow on the ground sub-system from the datalink network side is as follows.

The RGS and DLC are utilized as part of the ground infrastructure to support a wide range of participants and datalinks communicating to multiple potential ground based entities across a very broad range. The datalink controller (DLC) fronts for one or more datalinks and routes encrypted data to/from the ECU via one of the Black side Ethernet interfaces. The ground ECU (being the same as the participant ECU) supports recording and playback of recorded information as well as encryption and decryption of application and configuration files for ground sub-system elements via the UI-DRD.

The "swim lanes" between the ground guarding function and the ECU are identical to that of the Instrumentation package. They are allocated to individual security enclaves as required for the application of the day. Enclave switches support connectivity to as many MMMDR-CDGs as needed to support the number of mission and/or debrief rooms and associated SCWs and MRDWs.

If passed through the MMMDR-CDG, the information is processed by the label checking function and passed to the filter engine where the guard rules are applied to determine if the information is to be routed to the mission/debrief workstation. Of course, range level command and control information would be flowed to/from the SCW rather than the MDRW.

Figure 6 below provides a context for the system in a typical test or training range application.

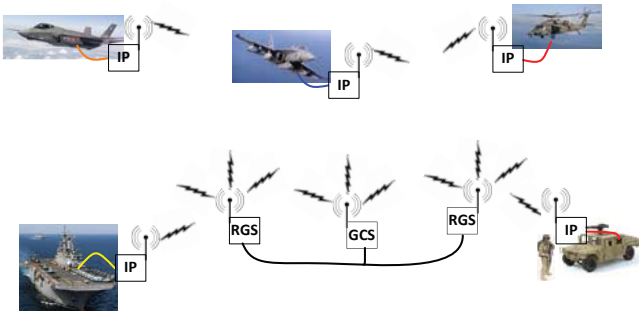

Figure 6: System Diagram for Typical Application

Figure 7 adds (to Figure 6) some key elements for a Live, Virtual, and Constructive (LVC) training application namely virtual participants, virtual Electronic Warfare (EW) elements, and constructive participants. In addition, supplemental weapon and threat simulations would be hosted at the appropriate classification levels within the various participant instrumentation packages. This may include hosting simulations at classification levels above that of the host platform.

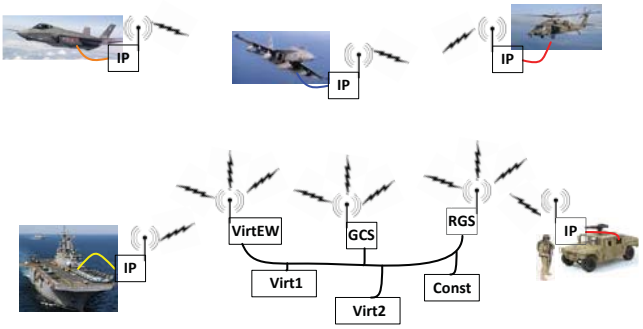

Figure 7: System Diagram for Typical LVC Training Application

5. SUMMARY

A layered approach to solving the MLS problems associated with modern test, training, and LVC issues has been described. This solution takes advantage of layering and abstraction with a focus on modularity and composability to enable cost effective implementation for a wide range of applications. The solution also takes advantage of several MILS building blocks to reduce Certification & Accreditation (C&A) cost, schedule, and risk. And, the solution supports a range of other modes of operation as needed by users (e.g. system-high, MSL, MLS). Finally, the solution described is not simply a proposed concept but is founded on Technology Readiness Level (TRL) 6+ products and technologies that have completed certification and are finishing accreditation through deployment for both domestic and international applications to solve MLS test, training, and LVC challenges.

6. ACKNOWLEDGMENTS

I would like to thank all those in the layered assurance community for the years of support and education I have received. In particular, Mark Van Fleet and Bill Beckwith who I have enjoyed many deep discussions with on security related topics and how to solve the world's problems. I would also like to thank all the test and training range personnel that I have had the pleasure of interacting with and for the breadth of knowledge I have received in range operations. I would like to thank Dr. Angus McLean who has tutored me on many things related to military training operations as well as simulation technologies and architectures. Finally, I would like to thank the team at Rockwell Collins (too numerous to list) with whom I have been working on development of our MLS Test, Training, and LVC solution; it has been a pleasure working with so many highly skilled and innovative experts who operate with the utmost professionalism and integrity.

An Integrated Framework for Multi-layer Certification-based Assurance

Rajesh Harjani
University of Malaga
Malaga (Spain)
rajesh@lcc.uma.es

Marcos Arjona
University of Malaga
Malaga (Spain)
marcos@lcc.uma.es

Javier Espinar
University of Malaga
Malaga (Spain)
espinar@lcc.uma.es

Antonio Maña
University of Malaga
Malaga (Spain)
amg@lcc.uma.es

Antonio Muñoz
University of Malaga
Malaga (Spain)
amunoz@lcc.uma.es

Hristo Koshutanski
University of Malaga
Malaga (Spain)
hristo@lcc.uma.es

ABSTRACT

Complexity, dynamism and overlays in networks and systems are some of the main challenges we face nowadays when reasoning on systems' assurance and behavior. Security certification has shown to be a solid foundation to provide assurance and trust about system properties. This paper presents a certification framework for composite, layered and evolving systems, such as cloud systems or cyber physical systems. The framework's certification-based methodology defines a solid ground to provide security assurance aspects of these systems. The framework integrates two main domains of research: *(i)* certification, models and mechanisms (based on testing, monitoring, trusted computing, and hybrid evidences) for providing assurance of the system components and attesting properties of the composite systems; and *(ii)* software engineering, process, methodology and tools to enable developers engineer cloud applications with strong awareness and requirements on security assurance of underlying cloud platforms and services.

Keywords

Assurance, Security, Multi-layer Certification, Engineering Process, Monitoring, Testing, Trusted Computing.

1. INTRODUCTION

Most of the current trends and paradigms in computing systems (notably cyber-physical systems, Internet of Things or cloud computing) share a series of characteristics that greatly complicate the tasks of guaranteeing their behavior, especially in terms of security, dependability, privacy, etc. Among these characteristics, we highlight three essential ones:

Dynamism. Systems are not static anymore. At design time, system engineers do not have all the information they would need to design systems that fulfill their requirements, especially the non-functional ones like security, dependability, performance, etc. Moreover, they have to design systems that have both adaptation capabilities (involving short term reaction to better fit the current context and the system state) and evolution capabilities (involving long term reactions to keep the system aligned to its design goals and the external situation). In this situation there is no permanent and complete system implementation that can be used to apply thorough and rigorous code reviewing, testing, formal analysis, and other techniques to verify (ensure its quality and correctness) and validate (ensure fulfillment of requirements) these systems as a whole.

Composition: Most of the new computing paradigms and trends follow a component-based approach. Systems are created by integrating components both statically at development time and dynamically at runtime. These components are frequently coming from different providers, and sometimes remain under the control of such providers instead of the system owner. The evolution of these components is decoupled from the evolution of the systems in which they are used. Composition in these systems happens both vertically (between what we normally call layers) and horizontally (between components at the same layer).

Complexity: We have already mentioned that systems are larger, include more functionalities, require more guarantees, are interconnected to other systems forming Systems-of-Systems, are in continuous evolution, etc. The combination of these characteristics results inevitably in levels of complexity scaling up quickly.

Providing a practical approach to support assurance in complex, composite, layered and evolving systems requires the combination of different elements in a coherent and integrated way. In particular, a practical assurance approach for these types of systems requires at least mechanisms: (i) to provide static assurance based for the system components; (ii) to attest the dynamic state of system components (including the supporting hardware infrastructure); and (iii) to derive properties of the composite system based on the state and properties offered by components. In addition to these, we also believe that any successful approach must be complemented with engineering processes, methods and tools to support developers of such systems to take full advantage of the approach.

Recent extensions and improvements to several existing technologies like certification, trusted computing, monitoring and reconfiguration provide a solid basis to develop an integrated layered assurance framework to support the assurance of complex, composite, layered and evolving systems in practice. In this paper, we present such approach, show how it is applied to cloud computing and discuss the challenges of future application to other types of systems, with a particular focus on cyber-physical systems.

2. INTEGRATED CERTIFICATION-BASED ASSURANCE FRAMEWORK

A common approach in enhancing assurance and reducing risks in the light of such uncertainties is to rely on the certification of the

different components and artifacts that constitute the potentially complex and fast changing nature of our target systems. Therefore, the main goal of the proposed approach is to develop an integrated framework of models, processes and tools supporting the certification-based assurance of security properties for layered computing infrastructures.

Assurance of cloud-based applications and services allows service consumers and providers to ascertain that the service properties provided in the certificates guarantee continuous compliance with their own requirements [1][2][3]. This increases consumers' and providers' confidence that their required level of assurance is being kept, before becoming involved in service design, deployment, and access on cloud.

With this purpose, the framework relies on multiple types of security evidences (e.g., testing, monitoring, trusted computing) used for certificate issuing, and includes relevant mechanisms for generating the evidence supporting a security property and for the secure communication of these evidences between different components within the certification infrastructure [2][3]. This evidence communication is supported by Trusted Computing (TC) [4] mechanisms providing means to establish integrity (authenticity) of evidence, and subsequently verify if the captor integrity holds (can be trusted). Whenever possible, evidence gathering is build upon existing standards and practices (e.g., interaction protocols, representation schemes etc.) regarding the provision of information for the assurance of security in clouds.

Furthermore, the framework supports the generation of hybrid certificates based on the combination of different types of evidences, including testing and monitoring data, and trusted computing platform proofs [5][2]. Hence, it supports decision making in business and societal contexts, which, due to existing legislation, established societal and business practices or individual preferences, might require and accept evidence of specific degrees of formality regarding a security property of a cloud service before this service can be used. This leads to cover security properties to an unprecedented extent and increase the overall confidence in the use of cloud computing.

To address the aforementioned security problems, several partners from European science and industry have joined efforts in the CUMULUS[1] (Certification infrastrUcture for MUlti-Layer cloUd Services) research project to investigate how to improve assurance, security and trustworthiness of multi-layer cloud services facing end users.

In its current implementation, the integrated framework allows service users, service providers and cloud suppliers to work together with certification authorities in order to use security certificates for deriving dynamic assurance evaluations in the ever-changing cloud environment. To achieve this, the proposed approach focuses on the following tasks:

- Definition, development and realization of advanced models for certification-based assurance of security properties based on evidences drawn from service testing and operational monitoring, as well as on trusted computing platform proofs. This facilitates the task of how to address a layered assurance framework given the complexity of interactions of cloud services.
- Development of an interoperable certification infrastructure for generating, maintaining and using certificates according to the different types of the certification models developed, so that to make them available to cloud providers and cloud customers.
- Development of an engineering process supporting the development of (i) cloud services in a way that facilitate their certification through a semi-automatic process and (ii) applications taking advantage of those services.
- Evaluation of the certification framework to ensure its technical soundness and industrial applicability, in particular for SmartCities and eHealth domains.
- Delivery of an interoperable certification solution and contribution to existing standards (e.g., interaction protocols, representation schemes etc.) regarding the provision of information for the assessment of security in clouds.

As a framework that aims to support the certification-based assurance of security properties in clouds at infrastructure, platform and software application layer services, the CUMULUS architecture is structured as an infrastructural overlay of the monitored payload system. The overlay is implemented by components, which provide the hooks to the monitored cloud system. Figure 1 shows the CUMULUS multi-layer certification-based assurance and infrastructure high-level overview including several conceptual layers and artifacts:

- Certification Aware Service/App Engineering Tools: providing means for supporting the engineering of cloud services and applications that can make use of the framework. This is a tool capable of interacting with the Infrastructure, and in particular with the different repositories, in order to take advantage of Certified Services.
- Certification Infrastructure: producing test, monitoring and trusted computing based multi-layer, incremental and hybrid certificates. The Certification Manager will realize this by making use of different Certification Models containing the necessary requirements and guidance to support the generation of certificates.
- Evidence Generation and Communication: for the provision of the certification evidences, it is where the components producing core test, monitoring and trusted computing based evidences are deployed.

The multi-layer certification-based assurance starts from the physical platform's Trusted Platform Module (TPM) [6] where platform integrity measurements are stored. The physical platform assurance (TC assurance) provides the basic building block over which the compositional layered assurance of the higher levels of the cloud system is built upon. Each of the higher levels of a cloud system has it own certification-based assurance provided by test, motoring, TC or hybrid certification models, which provide an assurance building block for next (higher) level of the cloud system and corresponding certification models.

3. CERTIFICATION-BASED ASSURANCE BUILDING BLOCKS

To achieve the proposed goals, namely providing means for evidence generation, communication and combination for assurance, we have developed different certification-based assurance building blocks [2][5][7]. These building blocks are based on testing, monitoring and trusted computing methods.

[1] http://www.cumulus-project.eu

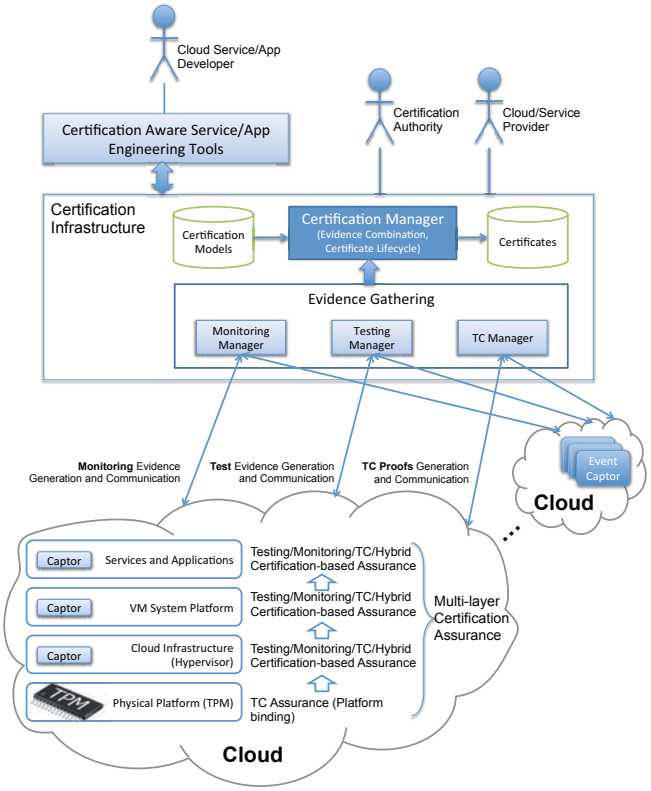

Figure 1. CUMULUS multi-layer certification-based assurance

By using testing mechanisms we can obtain static and dynamic evidences, however monitoring and trusted computing proofs (based on TPM) are clearly focused on collecting dynamic evidences. We make use of both, static and dynamic certificates, as well as TPM remote attestation as the elements for secure evidence communication. For each specific case a certification authority is responsible of the evidence combination and encodes the resulting properties in a certificate. It is important to notice that the combination is not automatic; it is the responsibility of an authority to perform it case by case. The Certification Authority analyses all involved components in the combination with their respective properties, and the resulting one from the combination is also particularly analyzed for it. This analysis can be done by different ways: checking the code, testing, monitoring, etc. The certification models are designed to allow the combination of different properties, making unnecessary the use of external rules to check the validity of the certificates.

Test-based certificates rely upon the results that are extracted of executing tests on the targeted services/software [8][9] as well as on composite services [7]. The Certification Authority guides the tests according to the target of certification in order to know if the software holds a certain property. This kind of certificates can be based on static or dynamic proofs; the static tests are performed offline while the dynamic tests are executed once the software is in a production environment. Therefore, these certificates may include both dynamic and static evidences according to the kind of tests used to extract the proofs. Software testing is performed by testing agents/captors their main task consists of injecting the test cases and collecting the corresponding results to compose the evidences. Testing Captors implement functionalities for both static and dynamic collection of evidences and they can be running on the same system or in another external to the software we want to certify.

Monitoring-based certificates are clearly focused on extracting dynamic evidences [10][11]; the monitoring operations are, by definition, continuous and they have to be performed once the software is deployed and accessible to user [12]. The proofs that will be included in such certificates can cover contextual conditions (e.g., co-tenant software, optimization strategies, network status...) that might not be possible to extract in a pre-production environment. The entities responsible for the monitoring process are the monitoring agents/captors; they capture all events and check if these events are compliant with the assertion included in the certification model [10].

The third type of certificates is based on Trusted Computing mechanisms that are used to provide hardware-based support for securing computing platforms, allowing certification authorities to verify that only authorized code runs on a system [13][14]. A TC mechanism usually is implemented using a TPM chip, which is integrated into the hardware of a platform (such as a PC, a laptop, a PDA, a mobile phone). TPM can be accessed directly via TC commands or via higher layer application interfaces (the Trusted Software Stack, TSS).

We consider two main scenarios for TC-based certification; in these scenarios the trusted computing mechanisms are used to protect both the integrity of the software and the underlying platform (including software and hardware) [14]. In the first scenario, the TC certification model is not used as an independent model but we make use of this building block to provide the trust that is needed for the validation of the Monitoring and Testing based certification. This means that the TC is not employed to directly certify a security property but instead used to increase the trust in other types of certifications. For instance, in a test-based certificate, it can be used to prove that the platform configuration at runtime is the same as the one used during testing in a pre-production environment, or it can also be used to ensure that the agent or monitoring captors have not been modified, meaning that they are extracting the proofs correctly. In the second scenario, the TC certification model will be used as an independent model to certify either platforms or services that are running in distributed systems. We would like to remark that this second use case is a generalization of the first scenario in such a way that allows to protect the integrity of more heterogeneous platforms and services, instead of only the monitoring and testing agents involved in the other CUMULUS certificates.

Each of the three types of certificates complies with the CUMULUS Meta-Model. This Meta-Model has a modular structure to represent: the property vocabulary, the certificate itself (including assertions, evidences, context...) and the certification model.

The combination of different evidences, to cover all components in a heterogeneous distributed system, can be carried out by a certification authority easily since all types of certificates conform to the same Meta-Model. This common structure fosters the combination of different properties to compose new ones avoiding the use of external rules to check the validity of the certificates. In addition, the dynamic testing, monitoring and trusted computing techniques allow these certificates to provide dynamic assurance of the properties they contain. These certificates are able to transit from one state to another, in their lifecycle, based on the dynamic evidences; for example, a valid certificate can be revoked if the monitoring agent captures an incompatible event with the

included assert, or if the trusted computing mechanisms are no longer able to prove the integrity of the software and/or the underlying platform.

4. ENGINEERING PROCESS

On the basis of the insights exposed, both the foundations and the powerful assurance artifacts sustaining the capabilities of this proposal, we need a solid platform ready to combine sources of security knowledge with the means to create assurance building blocks close to the end-users terms and requirements. Such complexity cannot be faced as a whole with a single threaded approach but instead it has to be addressed as a compositional set of engineering tools and activities that converge together into a coordinated and security-enriched process, giving solution to these multidisciplinary issues.

This Service Engineering Process (SEP) [15] sets out different guidelines to drive all the activities during the complete lifecycle of service development. The most prominent virtues and features allow *(i)* to interact with the different security areas and experts to gather security knowledge into machine processable data; *(ii)* to express this information in form of security requirements close to customers, developers and cloud system engineers,*(iii)* to define adequate compositions of techniques, certified services and building blocks to provide validated assurance solutions for those requirements and *(iv)* to support the deployment, usage and integration of these security solutions into cloud services and applications.

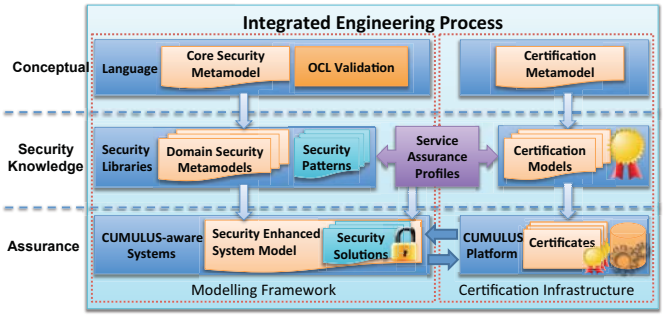

Figure 2. Integrated Engineering Process

Therefore the main objective of the SEP is to support the development of services and applications compliant with predefined certification models, in order to flexibly integrate certified security properties along with the necessary security knowledge to assure the target systems. Figure 2 shows the big picture of the Engineering workflow, defining several layers of abstraction (horizontal) and different elements at same level (vertical). Following, we use this figure as a basis for a brief description of the SEP and its main activities.

As we have stated, the SEP requires a very thoughtful approach to the multiple activities involved in the assurance methodology. Therefore a hierarchical strategy has been defined based on previous research and use cases experiences [16][17] defining three abstraction layers, each one composed of different artifacts:

- Conceptual: The top layer of the process stack lays down the common modeling language to express security knowledge, along with a collection of OCL rules [18] to validate the usage of the language. On the other hand, the certification metamodel and their schematics are conceptually described at this level.
- Security Knowledge: The middle layer uses the concepts of the abstract layer to express the knowledge and experience of security experts into libraries and to elaborate certification models aiming the production of certified services for cloud applications. Both artifacts operate across the Service Assurance Profiles explained later below. The actions of this layer are intended not only to help cloud app developers to integrate security knowledge by design, but also to make the system a better candidate for certification-based assurance.
- Assurance: The bottom layer reflects the final stage of the SEP, where all the security libraries and their solutions are used to transform or reengineer legacy cloud applications into certification-aware systems. This stage implies several transformations based on the security knowledge, and the resulting application will be able to interact with the CUMULUS platform deployed in the cloud infrastructure in order to retrieve or reconfigure certified services. Those services, as explained in the previous section, have been created and accredited by certification authorities, which guarantee the chain of trust.

The layered approach of SEP requires several modules to interact at horizontal level in the same abstraction level but also defines the workflow to move and export resources and knowledge between different vertical layers. This significant amount of activities are not perceived by users because these processes are managed and performed by the different supporting tools of the SEP, depending on their goal and stage in the overall methodology. All of these together make an integrated framework that covers all expected functionalities, dividing the methodology in two different responsibility areas, the Modeling Framework and the Certification Infrastructure. Both connected by Service Assurance Profiles (SAP).

The Modeling Framework has been implemented as a plugin of MagicDraw [19] to support all the designing and modeling activities required in the SEP along with the transformation and validation of the intermediate stages. The results of this software are the sum and composition of the following elements in the three levels of progression:

- The Core Security Metamodel (CSM) and the OCL Validation rules establish the proper and valid definition of UML elements to represent the security knowledge, defining the language, characteristics and mandatory attributes to describe the information for further automatic machine processing.
- Domain Security Metamodels (DSM) and Security Patterns [20] express security knowledge and solutions respectively to fulfill security requirements using the previous language. Security experts create those security libraries gathering security knowledge in compliance with the environment of their applications (company policies, standards, etc.) for a particular domain and they include well-known solutions and solvers in form of security patterns.
- Security Enhanced System Models are obtained once security libraries have been applied to improve the cloud system models with the security measures and solutions, fulfilling the security requirements of the system by means of certification requirements and claiming remote certified services from the CUMULUS platform.

With respect to the certification assurance operations, an interoperable Certification Infrastructure for generating, maintaining and using certificates according to the different types of certification models have been described in Section 3. To

achieve this goal the SEP requires a deployed CUMULUS platform in the cloud infrastructure allowing the interaction with CUMULUS-aware systems to offer and respond to service request by the CUMULUS enhanced systems.

The SAPs are auxiliary artifacts designed to establish the anchorage point between the two categories of security knowledge feeding the engineering process. SAPs aim to link specific details of certification requirements included in the security libraries, with the wide spectrum of certification mechanisms and certified services, most of these with similar goals, certification models, evidences or assurance goals, registered by different certification authorities. Therefore, SAPs have been created to introduce a discrimination of the expected security necessities and preferences, as security experts have the freedom to select the most proper assurance approaches and entities for their security solutions, based on reliability, trustworthiness and efficiency parameters these experienced users have obtained.

5. OUTLOOK AND FUTURE WORK

Cyber Physical Systems (CPSs) – systems created as a federation of smart, cooperative, sensing devices – are starting to play an important role in the everyday life of citizens, connected both to ICT systems and to the physical world. While the notion of sensors gathering data is not new, the sheer amount of new devices, the amount of data they can now gather, their data processing capabilities and the fact that they are all becoming connected to the Internet of Things, enables exciting, new services. The components of a CPS, whether ICT or non-ICT ones, may operate under distributed ownership and control, and within uncontrolled and unprotected physical environments, characterized by changing operational conditions and constraints (e.g., changing temperatures, physical damage, changes to power supply etc.). They may also operate within the remit of different and not always harmonized jurisdictions and transfer data across them. Furthermore, the ICT components of CPSs may have diverse computational features and roles. As a consequence of these factors, CPSs may often:

- Be vulnerable to security attacks and adverse operating context conditions that can compromise the availability and security of some of their components (e.g., local sensors, network components, application level components etc.);
- Generate, make use of and inter-relate massive personal data in ways that can potentially breach legal and privacy requirements;
- Experience frequent and unpredicted changes in the components and infrastructures that they rely on, which can compromise the security, resilience and availability of their operations and/or the service(s) that they offer.

Preserving quality, security and privacy (QSP) properties in CPSs under the above circumstances is a particularly challenging scientific and engineering problem. Hence, engineering CPSs in ways that can simultaneously guarantee all QSP properties of interest becomes a challenging problem requiring an integrated CPS design, development, monitoring, and adaptation approach.

A CPS engineering approach needs also to be aware of and support effectively the composition of software and physical components. This composition is inherently different from traditional software-based/service-based systems compositions. This is because software centric approaches to composition (e.g., software service/components orchestration workflows) are often not appropriate for CPS systems or at least the physical layer of their implementation stack due to their intrinsic complexity that often cannot be supported in the case of embedded systems.

An assurance-oriented engineering approach like the one presented in this paper, that could be applied to the development of CPS would clearly represent an important advance. Based on the current results obtained from the application of our approach to cloud systems, and its capacity to solve several of the problems we have just mentioned, we have started to develop extensions and adaptations to deal with the engineering of CPS. In particular, we have already extended the concept of security pattern to be able to represent solutions for CPS [21]. Ongoing work focuses on the representation of the dual nature of CPS by using layers that allow us to provide different engineering views for the physical and cyber layers of a CPS while maintaining their interrelations.

6. CONCLUSIONS

We have presented a layered assurance framework for certification of security properties of complex, layered and dynamically evolving systems, such as Cloud-based systems. The framework's certification-based assurance methodology provides a solid ground to manage security aspects of these systems. The cornerstones of the framework are the certification models (based on testing, monitoring, TC, and hybrid evidences) which drive certificates lifecycle, and the evidence gathering and composition for certification.

The framework provides to certification authorities a comprehensive tool set to enable effective cloud certification on a number of relevant and important security properties; and to Cloud app developers a comprehensive methodology and tools to engineer cloud applications with strong awareness and requirements specification on security assurance of underlying cloud platforms and services.

The security engineering methodology provides not only a source of specialized structured security knowledge for system engineering, but also compliance to specific certification models used to certify systems' properties. Thus, following the engineering process, a system designer will not only embed system's security aspects by design, but will also embed modular security and assurance of system's components for more effective system certification.

Cyber-physical systems with their complex and composite nature can well leverage on the framework's certification models, mechanisms and methodology to successfully handle security assurance of their (network of) interactive elements. The framework's certification-based assurance is envisaged to provide an important foundation towards a more comprehensive cyber security framework.

7. ACKNOWLEDGMENTS

This work was undertaken as part of the CUMULUS project funded by the European Union's Seventh Framework Programme for research, technological development and demonstration under grant agreement no 318580.

REFERENCES

[1] A. Sunyaev, S. Schneider. Cloud Services Certification. *Communications of the ACM, Vol. 56 No. 2*, 2013, Pages 33-36.

[2] G. Spanoudakis, E. Damiani, A. Mana. Certifying Services in Cloud: The Case for a Hybrid, Incremental and Multi-layer Approach. Proc. of 2012 *IEEE 14th International Symposium*

on High-Assurance Systems Engineering (HASE), Omaha, NE, pp. 175-176, 2012.

[3] S. Cimato, E. Damiani, R. Menicocci, F. Zavatarelli. Towards the certification of cloud services. Proc. of *IEEE 2013 International Workshop On Security and Privacy Engineering, Assurance, and Certification (SPEAC 2013),* Santa Clara, CA, pp. 100-105, 2013.

[4] Chris Mitchell, Trusted Computing, Institution of Electrical Engineers, 2005.

[5] S. Katopodis, G. Spanoudakis, K. Mahbub. Towards Hybrid Cloud Service Certification Models. Proc. of 11th *IEEE International Conference on Services Computing*, USA, 2014.

[6] Trusted Platform Module (TPM) Specifications. Trusted Computing Group. http://www.trustedcomputinggroup.org/resources/tpm_main_specification

[7] M. Anisetti, C.A. Ardagna, E. Damiani. Security Certification of Composite Services: A Test-Based Approach. Proc. of the 20th *IEEE International Conference on Web Services (ICWS 2013)*, San Francisco, CA, USA, June--July, 2013.

[8] S. Hanna and M. Munro. An approach for specification-based test case generation for web services. In: Proc. of the *IEEE/ACS International Conference on Computer Systems and Applications (AICCSA 2007*). Amman, Jordan, May 2007.

[9] M. Anisetti, C. A. Ardagna, E. Damiani and F. Saonara. A Test-based Security Certification Scheme for Web Services, *ACM Transactions on the Web, Volume 7 Issue 2*, May 2013.

[10] M. Krotsiani, G. Spanoudakis, K. Mahbub. Incremental Certification of Cloud Services. In *7th Int. Conf. on Emerging Security Information, Systems and Technologies (SECURWARE 2013)*, Barcelona, pp. 72-80, 2013.

[11] H. Foster, G. Spanoudakis and K. Mahbub. Formal Certification and Compliance for Runtime Service Environments. In 9th *IEEE International Conference on Service Computing*. 2012.

[12] Ghezzi C., Guinea S. (2007), Runtime Monitoring in Service Oriented Architectures, In Test and Analysis of Web Services, (eds) Baresi L. & di Nitto E., Springer, 237-264, 2007.

[13] N. Santos, K.P. Gummadi, and R. Rodrigues. Towards trusted cloud computing. In HotCloud, 2009.

[14] A. Muñoz, A. Maña. Bridging the GAP between Software Certification and Trusted Computing for Securing Cloud Computing. In *IEEE International Workshop on Security and Privacy Engineering, Assurance, and Certification (SPEAC 2013)*, June, 2013.

[15] M. Arjona, R. Harjani, A. Muñoz, and A. Maña. An Engineering Process to Address Security Challenges in Cloud Computing, 3rd *ASE International Conference on Cyber Security*, 2014.

[16] J. F. Ruiz, A. Maña, M. Arjona, and J. Paatero. Emergency Systems Modelling using a Security Engineering Process. *3rd International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH)*. SciTePress, 2013.

[17] J.F. Ruiz, A. Rein, M. Arjona, A. Maña, A. Monsifrot, and M. Morvan. Security Engineering and Modelling of Set-Top Boxes, 2012 *ASE/IEEE International Conference on Biomedical Computing (BioMedCom)*.

[18] M. Arjona, C. Dania, M. Egea and A. Maña. Validation of a security metamodel for development of cloud applications. *14th International Workshop on OCL and Textual Modeling Applications and Case Studies (OCL 2014)*. 17th International Conference on Model Driven Engineering Languages and Systems (MODELS'14).

[19] MagicDraw Modelling Tool. http://www.nomagic.com/products/magicdraw.html

[20] M. Arjona, , J.F. Ruiz and A. Maña. Security Patterns for Local Assurance in Cloud Applications. *International Workshop on Engineering Cyber Security and Resilience ECSaR'14*. The Third ASE International Conference on Cyber Security 2014.

[21] A. Maña, E. Damiani, S. Gürguens, G. Spanoudakis. Extensions to Pattern Formats for Cyber Physical Systems. Proceedings of the *31st Conference on Pattern Languages of Programs (PLoP'14)*. Monticello, IL, USA. Sept. 2014.

Layered Assurance of Security Policies for an Embedded OS

Jia Song
Ctr Secure & Dependable Systems
University of Idaho
Moscow, ID 83844
song3202@vandals.uidaho.edu

Jim Alves-Foss
Ctr Secure & Dependable Systems
University of Idaho
Moscow, ID 83844
jimaf@uidaho.edu

ABSTRACT
A layered assurance argument requires a mapping of lower level security mechanisms, and their corresponding policies, into the support for higher level policies. This WIP presents a security policy for security tagged architectures and highlights key features that need to be addressed in a lauered assurance framework.

Categories and Subject Descriptors
D.4.6 [**Operating Systems**]: Security and Protection – a*ccess control, information flow control.*

General Terms
Experimentation, Security, Verification.

Keywords
Keywords are your own designated keywords.

1. INTRODUCTION
A layered assurance argument requires a mapping of lower level security mechanisms, and their corresponding policies, into the support for higher level policies. This work in progress (WIP) presentation discusses current research focused on simplifying that mapping process.

The project discussed in this WIP uses a proposed hardware-based security tagged architecture (STA) as the foundation for operating system security services. The intent of the STA is to provide mechanisms to enable the operating system to provide services for application level security policies. As is true with many lower level mechanisms, there is not a direct mapping to the higher level security policies, and there are ways to misuse the mechanism.

In the STA, a security tag is associated with each 32-bit word of memory and each memory register. The hardware-based tag checking and propagation controls access and information flows in the system. This protection includes stack protection, user isolation and enforcement of least privilege. Individual modules of the embedded OS are classified into different security levels, which are stored in the security tag and supported by the STA hardware, and software-level tag and user management modules.

Hardware-based tagging can be used to check for memory type mismatches, to compare security levels of tags, control information flow, etc. Because hardware-based tagging is based on a defined hierarchy of security labels, and is only aware of the security tags and associated domains, not all security policies can be directly enforced by the STA. To make the tagging system working properly, and to support a wider range of security policies, software level tagging is needed to initialize the tagging system and help deal with some special conditions on tag checking and propagation.

Proofs of the correctness of the tagging rules and implementation at the hardware level indicate proper tag propagations and correct behaviors of the STA. These proofs can then be used to easily validate some higher level security policies. However, because of the existence of complex tag checking and propagation rules implemented at the software level, which can alter information flows in the system, the verification and validation of rules that enforce more complex security policies are more complicated.

To simplify the verification of these more complex policies we are classifying the verification issues and policy concerns into sets that satisfy specific characteristics. These characteristics apply to system configurations, properties of system modules and other implementation issues. If a specific system is shown to satisfy these characteristics, then we can use the hardware proofs to verify the preservation of the higher level security policies. To ensure that all of the higher level policies are correctly implemented, the characteristics defining specific system configurations and issues that may cause violations of the policies need to be identified and corresponding verifications have to be conducted.

In this WIP presentation, we will discuss the STA security policy, highlighting key features that can generate problems in a layered assurance framework. We then discuss the configuration, mechanisms, and verifications that are needed to provide the assurance that the higher level policy is correctly implemented. Although we use an STA, we believe the techniques we discuss are generalizable to a wider range of systems..

Towards a Highly Secure PLC Architecture for CPS

Nikhil Mahesh
Student, Viterbi School of Engineering
University of Southern California
nmmahesh@usc.edu

In recent decades, there have been numerous attacks on Cyber-Physical Systems (CPS) that are part of the critical infrastructure. CPS components have many vulnerabilities with the ability to bring down systems in their entirety. This has made institutions and people realize the importance of CPS security which was absent when the systems were initially built. In the past decade there are several examples that demonstrate the vulnerabilities that exist in the current system. These vulnerabilities can be devastating if a terrorist attacks and brings down our critical infrastructures. After the discovery of the Stuxnet attack based on software subversion malware, several PLC producing companies are working towards making the PLC secure, whereas before, security of PLCs received little attention. Such attacks also make clear that the real time operating systems that are used are vulnerable to exploits. The adoption of commercial software, firmware, and hardware as the CPS platform has led to vulnerabilities of the commercial environment. We need to deal with these systems in a different manner, i.e., we need to develop highly secure critical infrastructure systems with existing features. The present systems are susceptible to subversion, illustrated by Stuxnet, as they are not of high assurance. To address subversion we need verifiable protection which is systematically codified, such as done by the TCSEC and TNI. It is important to note that a system is secure only with respect to a policy. The present systems often have no explicit security policy, and in particular there is no MAC policy implement in a trusted computing base, which is essential for any system to deal with subversion. To provide this we propose to integrate an existing security kernel and associated hardware into a reusable Trusted Device, reminiscent of a reusable commercial computer motherboard as the CPS platform. There are three components to cyber physical system: the PLC, HMI and software development environment. The PLC has several basic stages of operation also as called as scan cycle. The scan cycle is essential for controlling the steps in execution on the PLC. We should note that the steps each perform a different task that is important for the PLC to work. The security relationship of the steps can be represented as a MAC integrity policy. And they can be implemented in different process so the MAC policy can be implemented with integrity access classes for different subjects. These should be communicating processes, and there must be synchronization between them. We suggest using assured pipelines so that each process communicates with just those processes it needs. An assured pipeline limits the communication within a sequence of steps so that each process only communicates with an adjacent process in the scan cycle. Any other process outside the pipeline cannot interfere with data in the pipeline. Constrained by the MAC integrity policy the interrelated processes maintain the secure decorum between the different processes. These use the divide and conquer technique to build a secure application. Introducing the reusable Trusted Device as the platform at the core of the architecture makes it significantly agnostic to the specific application context, while retaining its properties for substantial reduction in time and resources.

This is only the one aspect of the cyber-physical systems. There are many security facets of this problem. In order to address them we need to have more researches in this field.

Mapping Security Policies to Implementations

Jim Alves-Foss
Ctr Secure & Dependable Systems
University of Idaho
Moscow, ID 83844
jimaf@uidaho.edu

ABSTRACT
There is often a disconnect between the high level abstract security policies and the actual policy implemented by lower level mechanisms. However a layered assurance argument requires a mapping of lower level security mechanisms, and their corresponding policies, into the support for higher level policies. This WIP presents work towards resolving that mapping between high level security policies and actual implementations within a network configuration.

Categories and Subject Descriptors
C.2 [Computer-Communication Networks] General – *security and protection*

D.4.6 [**Operating Systems**]: Security and Protection – a*ccess control, information flow control.*

General Terms
Experimentation, Security, Verification.

Keywords
Keywords are your own designated keywords.

1. INTRODUCTION
This work in progress (WIP) presentation discusses current work to define a mapping between high level security policies and actual implementations within a network configuration.

At the most abstract level, a classic security policy specifies events that may occur and the security actions that happen for those events. An event is specified in terms of a subject, object and action. The subject is the active entity in the system attempting to perform some action on a passive entity, the object. At the highest levels of abstractions, the subject is typically defined as a user, the object is some type of data resource and the action is a generic activity such as read, write, etc. Associated with each entity are a set of attributes, which could be as simple as a security classification, and as complex as characteristics of data releasability tags, location of the request, time, history of previous requests, source of the data, etc. The security mechanisms may allow the requested action, deny it, log the activity, modify the request, or trigger additional actions.

To complicate matters, a subject specified at the highest level of abstraction, may not have a direct corresponding entity in the implementation. For example, a firewall in a network is configured with rules based on the contents of network packets, specifically packet headers. The firewall, one of the security mechanisms implementing the policy, is aware of things such as ip addresses, ports, and networks. It is not aware of specific users or objects in the system.

Verification efforts require a formal way to specify the entities within each security mechanism of the system, the entities within the security policies, and a mapping between them. Currently, this is typically done on a per-system, ad-hoc basis. This WIP discusses a project to formalize this mapping, providing a theoretical basis for the mapping that ensures correctness and completeness. The theorems of the mapping can be instantiated for specific systems and reused -- simplifying the overall verification effort.

This WIP discusses progress made in the development of this formalism, and current issues.

Empirical Evaluation of API Usability and Security*

Sam Weber, Robert Seacord,
Forrest Shull, David Keaton†
Software Engineering Institute

Brad Myers, Michael Coblenz‡
CMU

ABSTRACT

The aim of our project is to gather empirical evidence on the security impacts of language and Application Program Interface (API) design. Ultimately, the cause of cybersecurity failures is flawed code written by programmers. Our philosophy is that programmers are people, and we need to study how to design APIs which are usable by programmers — APIs with which it is easy to develop secure code.

It is well-known that API design can have a large impact on security, and this barrier is difficult, if not impossible, to overcome by training alone. For example, buffer overflows were understood and documented as early as 1972, but are still one of the most common vulnerabilities. Furthermore, APIs are typically designed by a small number of experienced developers but have an extremely long life-span, and therefore the impact of poor API design can have far reaching consequences.

There has been some previous work on the usability of APIs, but so far this work has restricted itself to other software quality attributes, such as learnability. A relevant example of such work is Stylos et al [1], which studied the relative usability of different styles of constructing objects. The results were rather dismaying from a security point-of-view: programmers strongly preferred a style which would cause contructed objects to be mutable, whereas the security community generally considers mutability a source of security problems. One of our tasks will be to investigate and measure this apparent trade-off between traditional usability and security.

We should make clear that we are not targeting just APIs with security-relevant functionality, such as libraries that support authentication. Ordinary libraries — including but not limited to string, file, and XML processing and network libraries — pose more interesting problems because programmers using them are not actively considering security and are consequently more likely to be susceptible to the misconceptions, unstated assumptions, and flawed usage patterns that underlie most vulnerabilities.

API design is a broad domain to research, so we are focusing on a few select areas that research has shown to have security implications such as concurrency and design patterns like immutability. We expect that usability and security will be aligned with respect to concurrency APIs (that is, more usable APIs will also be more secure), but as mentioned above, they will be opposed with respect to mutability. Our research methodology relies on a mix of corpus review (to understand how these issues are dealt with in contemporary code bases) and studies with human subjects under more controlled conditions.

We are extracting typical design patterns by which development teams deal with concurrency API calls by analyzing code repositories that use concurrency. We are augmenting these corpus reviews with field observations and surveys using contextual inquiries of professionals, and surveying alternate concurrency standards and approaches. From this information, we will identify specific hypotheses about the usability differences between alternate styles of presenting concurrency to programmers. This will be used to design and conduct user-studies.

We will use two different populations, students and experts, with a balanced within-subjects design to control for individual programmer differences as well as ordering effects. The evaluation will include quantitative measures (such as number of errors, code length, completion time) and qualitative measures (such as rationales for design decisions) collected through a think-aloud protocol and questionnaires. This task will produce data as to the types and frequency of errors that programmers can be expected to make using alternate concurrency APIs.

In parallel, we will also conduct programmer studies of the impact of mutability on both security and usability.

The ultimate aim of our project is to produce experimentally-validated and specific guidance to API designers on how to create systems which are less prone to security vulnerabilities.

*This work is partially supported by NSF award 1423054

†email: {samweber|rcs|dmk}@cert.org, fjshull@sei.cmu.edu

‡email: {bam|mcoblenz}@cs.cmu.com

1. REFERENCES

[1] J. Stylos and S. Clarke. Usability Implications of Requiring Parameters in Objects' Constructors. *29th International Conference on Software Engineering (ICSE'07)*, pages 529–539, May 2007.