NPS-AM-12-C9P06R02-054

EXCERPT FROM THE
PROCEEDINGS

OF THE

NINTH ANNUAL ACQUISITION RESEARCH SYMPOSIUM WEDNESDAY SESSIONS VOLUME I

Addressing Challenges in the Acquisition of Secure Software Systems With Open Architectures

Walt Scacchi and Thomas Alspaugh
University California, Irvine

Published April 30, 2012

Approved for public release; distribution is unlimited.
Prepared for the Naval Postgraduate School, Monterey, CA 93943.

The research presented at the symposium was supported by the acquisition chair of the Graduate School of Business & Public Policy at the Naval Postgraduate School.

To request defense acquisition research or to become a research sponsor, please contact:

NPS Acquisition Research Program
Attn: James B. Greene, RADM, USN, (Ret.)
Acquisition Chair
Graduate School of Business and Public Policy
Naval Postgraduate School
Monterey, CA 93943-5103
Tel: (831) 656-2092
Fax: (831) 656-2253
E-mail: jbgreene@nps.edu

Copies of the Acquisition Research Program's sponsored research reports may be printed from our website (www.acquisitionresearch.net).

Preface & Acknowledgements

Welcome to our Ninth Annual Acquisition Research Symposium! This event is the highlight of the year for the Acquisition Research Program (ARP) here at the Naval Postgraduate School (NPS) because it showcases the findings of recently completed research projects—and that research activity has been prolific! Since the ARP's founding in 2003, over 800 original research reports have been added to the acquisition body of knowledge. We continue to add to that library, located online at www.acquisitionresearch.net, at a rate of roughly 140 reports per year. This activity has engaged researchers at over 60 universities and other institutions, greatly enhancing the diversity of thought brought to bear on the business activities of the DoD.

We generate this level of activity in three ways. First, we solicit research topics from academia and other institutions through an annual Broad Agency Announcement, sponsored by the USD(AT&L). Second, we issue an annual internal call for proposals to seek NPS faculty research supporting the interests of our program sponsors. Finally, we serve as a "broker" to market specific research topics identified by our sponsors to NPS graduate students. This three-pronged approach provides for a rich and broad diversity of scholarly rigor mixed with a good blend of practitioner experience in the field of acquisition. We are grateful to those of you who have contributed to our research program in the past and hope this symposium will spark even more participation.

We encourage you to be active participants at the symposium. Indeed, active participation has been the hallmark of previous symposia. We purposely limit attendance to 350 people to encourage just that. In addition, this forum is unique in its effort to bring scholars and practitioners together around acquisition research that is both relevant in application and rigorous in method. Seldom will you get the opportunity to interact with so many top DoD acquisition officials and acquisition researchers. We encourage dialogue both in the formal panel sessions and in the many opportunities we make available at meals, breaks, and the day-ending socials. Many of our researchers use these occasions to establish new teaming arrangements for future research work. In the words of one senior government official, "I would not miss this symposium for the world as it is the best forum I've found for catching up on acquisition issues and learning from the great presenters."

We expect affordability to be a major focus at this year's event. It is a central tenet of the DoD's Better Buying Power initiatives, and budget projections indicate it will continue to be important as the nation works its way out of the recession. This suggests that research with a focus on affordability will be of great interest to the DoD leadership in the year to come. Whether you're a practitioner or scholar, we invite you to participate in that research.

We gratefully acknowledge the ongoing support and leadership of our sponsors, whose foresight and vision have assured the continuing success of the ARP:

- Office of the Under Secretary of Defense (Acquisition, Technology, & Logistics)
- Director, Acquisition Career Management, ASN (RD&A)
- Program Executive Officer, SHIPS
- Commander, Naval Sea Systems Command
- Program Executive Officer, Integrated Warfare Systems
- Army Contracting Command, U.S. Army Materiel Command
- Office of the Assistant Secretary of the Air Force (Acquisition)

- Office of the Assistant Secretary of the Army (Acquisition, Logistics, & Technology)
- Deputy Director, Acquisition Career Management, U.S. Army
- Office of Procurement and Assistance Management Headquarters, Department of Energy
- Director, Defense Security Cooperation Agency
- Deputy Assistant Secretary of the Navy, Research, Development, Test & Evaluation
- Program Executive Officer, Tactical Aircraft
- Director, Office of Small Business Programs, Department of the Navy
- Director, Office of Acquisition Resources and Analysis (ARA)
- Deputy Assistant Secretary of the Navy, Acquisition & Procurement
- Director of Open Architecture, DASN (RDT&E)
- Program Executive Officer, Littoral Combat Ships

We also thank the Naval Postgraduate School Foundation and acknowledge its generous contributions in support of this symposium.

James B. Greene Jr.
Rear Admiral, U.S. Navy (Ret.)

Keith F. Snider, PhD
Associate Professor

Panel 6. Considerations in Acquiring Open Architecture Software Systems

Wednesday, May 16, 2012	
1:45 p.m. – 3:15 p.m.	**Chair: Captain Joseph J. Beel**, USN, Commanding Officer, Space and Naval Warfare Systems Center Pacific ***A Framework for Reuse in the DoN*** Randy Mactal, *Space and Naval Warfare Systems Center Pacific* Lynne Spruill, *APEO Engineering Support* ***Addressing Challenges in the Acquisition of Secure Software Systems With Open Architectures*** Walt Scacchi and Thomas Alspaugh *University California, Irvine* ***Certifying Tools for Test Reduction in Open Architecture*** Valdis Berzins, *Naval Postgraduate School*

Joseph J. Beel—Captain Joe Beel was commissioned from the U.S. Naval Academy in 1985, earning a Bachelor of Science degree in mechanical engineering. He was designated a Naval Aviator in September 1986. He completed Fleet Replacement Pilot training with HSL-31 in May 1987 and joined the Sea Snakes of HSL-33, flying the SH-2F Sea Sprite until December 1989. He deployed in the USS *Kirk* (FF1067), the USS *Knox* (FF 1052), the USS *Francis Hammond* (FF1067), and the USS *Sterrett* (CG 31), including service in Operation Earnest Will.

He attended the Naval Postgraduate School in Monterey, CA, from 1990 until 1992, earning a Master of Science (with distinction) in operations research. He taught in the U.S. Naval Academy Mathematics Department from May 1992 until May 1995 and served as the Fifth Company Officer from August 1993 until May 1995. He also served as an advanced seamanship and navigation instructor and was designated a craftmaster/yard patrol craft officer-in-charge afloat.

Captain Beel completed Fleet Replacement Pilot training with HSL-41 in February 1996 and joined the Battle Cats of HSL-43, flying the SH-60B Sea Hawk until 1998. He deployed in the USS *Princeton* (CG 59).

From June 1998 until August 1999, Captain Beel served as the training and education program analyst in the Assessment Division (N81), Office of the Chief of Naval Operations. He served in a Federal Executive Fellowship at the RAND Corporation in Santa Monica, CA, from August 1999 to August 2000. From August 2000 until September 2002, he served in the USS *John C. Stennis* (CVN 74), including service in Operations Noble Eagle and Enduring Freedom. He served as officer-in-charge of Navy Warfare Development Command, Detachment San Diego, from October 2002 until August 2003. He served as commanding officer and executive officer, Naval Air Technical Data and Engineering Service Command (NATEC), from September 2003 until September 2006.

Most recently, Captain Beel served four years in the Program Executive Office (PEO), Command, Control, Communication, Computers, and Intelligence (C4I); as PEO chief of staff and deputy for Operations from October 2006 to June 2008; and as deputy program manager of the Navy Tactical Networks Program Office from June 2008 to August 2010.

Captain Beel is a member of the Defense Acquisition Corps and is Level III certified in Program Management, Life Cycle Logistics and Production, and Quality and Manufacturing. He is a certified

Lean Six Sigma Black Belt. He led a continuous process improvement project that was awarded a California Council of Excellence California Team Excellence bronze award and was selected to compete for the American Society of Quality's International Team Excellence Award at the 2011 World Conference on Quality and Improvement.

Captain Beel's awards include the Meritorious Service Medal (three awards), Air Medal (individual award), Navy Commendation Medal (five awards), Navy Achievement Medal, and various unit, campaign, and service awards. He has also received the Sikorsky "Winged-S" Lifesaving Rescue Award.

Addressing Challenges in the Acquisition of Secure Software Systems With Open Architectures

Walt Scacchi—Scacchi is a senior research scientist and research faculty member at the Institute for Software Research, University of California, Irvine. He received a PhD in information and computer science from UC Irvine in 1981. From 1981 to 1998, he was on the faculty at the University of Southern California. In 1999, he joined the Institute for Software Research at UC Irvine. He has published more than 150 research papers and has directed 60 externally funded research projects. In 2012, he serves as general co-chair of the 8th IFIP International Conference on Open Source Systems (OSS2012). [wscacchi@ics.uci.edu]

Thomas Alspaugh—Alspaugh is a project scientist at the Institute for Software Research, University of California, Irvine. His research interests are in software engineering, requirements, and licensing. Before completing his PhD, he worked as a software developer, team lead, and manager in industry, and as a computer scientist at the Naval Research Laboratory on the Software Cost Reduction, or A-7 project. [thomas.alspaugh@acm.org]

Abstract

We seek to articulate and address a number of emerging challenges in continuously assuring the security of open architecture (OA) software systems throughout the system acquisition life-cycle. It is now clear that future system must resist coordinated international attacks on vulnerable software-intensive systems that are of high value, and control complex systems. But current approaches to system security are most often piecemeal with little or no support for guiding what system security requirements must address across different system-processing elements and data levels, and how those can be manifest during the design, building, and deployment of OA software systems. We present a framework that organizes OA system security elements and mechanisms in forms that can be aligned with different stages of acquisition spanning system design, building, and run-time deployment, as well as system evolution. We provide a case study to show our scheme and how it can be applied to common enterprise systems.

Introduction

We seek to research, develop, and refine new concepts, techniques, and tools for continuously assuring the security of large-scale, open architecture (OA) software systems composed from software components that include proprietary/closed source software (CSS) and open source software (OSS). Federal government acquisition policy, as well as many leading enterprise IT centers, now encourage the use of CSS and OSS, and thus OA, in the development, deployment, and evolution of complex, software-intensive systems.

We seek to prototype and demonstrate a new innovative approach and supporting technology that can develop new principles for correctness and security properties for OA systems. This includes developing basic principles to determine the security and performance properties of software systems, the conditions under which these properties hold, and the methods used to prove these properties of interest for systems. Of particular interest are networked OA software systems that are adapted or that evolve to dynamic conditions and threats during their development, deployment, and usage, including those that may rely on new technologies like OA mobile devices (Smalley, 2012; "Security Technical Information Guide," n.d.) or other IT systems relying on open source technologies (Department of Defense [DoD], 2010; Garcia, 2010; Gizzi, 2011; Navy, 2010). In particular, such study may be of value to securing new cyber warfare technologies (DoD, 2011; Scacchi, Brown, & Nies, 2011). Our efforts may also lead to fundamental advancements for secure information sharing between information producers and consumers in order to realize more secure information management, sharing, and interaction.

Challenges of Securing Systems with Open Architectures

Coordinated international attacks on vulnerable software-intensive systems that are of high value and on control complex systems are becoming ever more apparent. As the *StuxNet* case demonstrates, security threats to software systems are multi-valent, multi-modal, and distributed across independently developed software system components (Stuxnet, 2011). Similarly, it is now clear that physically isolated/confined systems are vulnerable to external security attacks via portable storage devices like USB drives, modified end-user devices (e.g., keyboards, mice; "Attack of the Computer Mouse," 2011), and social engineering techniques (Sawyers, 2011). This requires new security measures and policies necessary to defend such systems through new threat prevention and detection methods, as well as appropriate response mechanisms. Thus, what makes a system or system architecture secure changes over time, as new threats emerge and as systems evolve to meet new functional requirements. Consequently, there is need for an approach to continuously assure the security of complex, evolving OA systems in ways that are practical and scalable yet robust, tractable, and adaptable.

However, the best practices for developing OA systems whose components may be subject to differing security requirements (e.g., security rights and obligations) are unclear. Such practices are yet to be identified. This puts IT centers, system integrators, and service providers at a disadvantage when seeking to develop new software-intensive systems whose costs may be lower due to the integration of mature OSS components that are interfaced to pre-existing or new CSS components. OA systems thus present new challenges for assuring software system security.

Software systems security mechanisms for enabling security requirements or policies are often employed on an *ad hoc* basis, because there are not convenient or interactive tools or formal techniques for specifying the security requirements of an OA system or its components. Instead, what is available are disjoint mechanisms for implementing individual system security features (Loscocco et al., 1998; Spencer et al., 1999), such as

- mandatory access control lists and firewalls;
- multi-level security;
- authentication (including certificate authority and passwords);
- cryptographic support (including public key certificates);
- encapsulation (including virtualization and hidden versus public APIs), hardware confinement (memory, storage, and external device [port] isolation; Sun, Wang, Zhang, & Stavrou, 2012), and type enforcement capabilities;
- secure programming practices (including secure coding standards, data type, and value range checking; Seacord, 2008);
- data content or control signal flow logging/auditing;
- honey-pots and traps;
- security technical information guides for configuring the security parameters for applications ("Security Technical," 2011) and operating systems (Smalley, 2012); and
- functionally equivalent but diverse multi-variant software executables (Franz, 2010; Salamat, Jackson, Wagner, Wimmer, & Franz, 2011).

But there is a gap between these mechanisms and any concept of a comprehensive security policy, whether for a system or for any of its components, and no obvious way to integrate and evaluate them as a group. Similarly, it is unclear what relationships arise or

are in place among these different security mechanisms. Further, what guidance is needed regarding which security mechanism to use where, when, why, and how, and how is their usage updated or configured as extant system security policy evolves? The mechanisms are also mostly software implementation choices rather than system architectural choices; no system-specific framework (like an architecture) exists in which they can be pulled together in patterns that can be designed to meet specific security policies and goals. But in an OA system, it may be unclear or unlikely that system integrators will find mature OSS or CSS components that supply all of the system security features that the integrator or the customer requires on a timely, cost-effective basis.

Next, OA systems evolve through more pathways than traditional systems:

- individual components evolve through update revisions (e.g., security patches) made by the component's developers;
- individual components are updated with new, functionally enhanced versions from outside providers;
- individual components are replaced by different components from other sources;
- component interfaces evolve, either due to the system developers or outside sources;
- system architecture and configuration evolve as the developers adapt them to address new functional requirements;
- system functional and security requirements evolve, either due to the system developers, recognized gaps, or outside stakeholders; and
- system security policies, mechanisms, security components, and system configuration parameter settings also change over time.

These additional evolution paths are tied to the benefits of using OA systems with OSS components, but they also present new challenges for security. OA systems are continually evolving, and in our view this fact is fundamentally unaddressed by prior work in security.

Beyond these issues, we must consider the following: How should customers specify what security system features they want their delivered systems to support? How can the history of security failures (vulnerabilities), faults (exploits), possible cyber-warfare attacks (threats), and possible responses (updating system configuration with new elements that resist new threats, close new vulnerability, and prevent newly discovered exploits) guide the evolution of approaches for developing secure OA systems? How can answers to questions like these help formulate a technological innovation element of the DoD strategy for operating in cyberspace (DoD, 2011)? Questions like this remain unresolved at present.

Verification of the usage of security mechanisms in software systems is unclear and often focused either at the whole system (macro) level, or at the program function or coding (micro) level, but generally not at the architectural component and interconnection (meso) level, and not for combinations and alternative configurations of CSS and OSS components with different security histories. We believe that there is a new or under-explored opportunity to address security requirements at the architectural level.

As such, we see the following basic challenges in assuring OA system security:

- how to verify the security of OA system designs throughout system development, deployment, and post-deployment support; and

- how to validate the effectiveness of OA system security measures and feed evolving knowledge of vulnerabilities and exploits back into the ongoing development (continuous evolution) stream for existing and planned systems in an operational, testable form that system designers can use and program managers can assess.

Similarly, we see the following basic challenges in assuring security of OA software systems:

- how best to develop complex OA systems whose OSS or CSS system components may originally come from trusted sources but in which these components, the architectural configuration, and security requirements are subject to multiple sources of adaptation and evolution;
- how to go beyond “many eyes” (a large number of skilled reviewers) to establish a scalable basis for automated or semi-automated verification of software system security properties as the system continually evolves;
- how best to achieve continuous software system security assurance as a system is adapted and evolved to address new security requirements and technology progress;
- how best to protect OA systems through biologically inspired natural defenses that provide adaptive and resilient mechanisms, including agile response, isolation, and fail-soft recovery to immediate attacks, as well as adaptation via dynamic reconfiguration, multi-version mechanisms, (artificial) ecological diversity responses to sustained vulnerabilities or threats (Shrobe, 2011); and
- how to create reference models and security policy requirements that articulate security scenarios appropriate for oversight during system acquisition, as well as during system design, implementation, deployment, and beyond.

Securing Software Systems

The key ideas in our approach to develop and demonstrate a new solution to the challenges is to specify verifiable security requirements of OA systems using formalized “security licenses” (Scacchi & Alspaugh, 2011) and to use an explicit, evolvable software architecture to mediate and carry the paths of interactions among them. Security licenses must specify the security requirements and access/update rights and obligations within an OA system, its CSS and OSS components, and their interconnections (e.g., APIs, databases, shared files, and communication protocols) that defend against threats and enable appropriate responses to attacks or suspicious/anomalous system behaviors. Subsequently, the goal of our approach is to articulate and refine the ways and means for expressing and verifying that the security requirements of OA system components match up appropriately and together support the security requirements of the entire OA system at architectural design-time while enabling the automated verification of system builds/compositions and deployable, as well as of executable run-time versions of the system.

Software licenses represent a collection of rights and obligations for what can or cannot be done with a licensed software component. Licenses can thus denote both functional and non-functional requirements that apply to software systems or system components during their development and deployment. But rights and obligations are not limited to concerns or constraints applicable only to software as IP. Instead, they can be written in ways that stipulate functional or non-functional requirements of different kinds. Consider, for example, that desired or necessary software system security properties can

also be expressed as rights and obligations addressing system confidentiality, integrity, accountability, system availability, and assurance. This kind of approach provides new principles of correctness for software IP requirements (Breaux & Anton, 2005, 2008).

Traditionally, developing robust specifications for non-functional software system security properties in natural language often produces specifications that are ambiguous, misleading, inconsistent across system components, and lacking sufficient details (Yau & Chen, 2006). Using a semantic model and logic to formally specify the rights and obligations required for a software system or component to be secure (Breaux & Anton, 2005, 2008; Yau & Chen, 2006) means that it may be possible to develop both a "security architecture" notation and model specification that associates given security rights and obligations across a software system, or system of systems. Similarly, it suggests the possibility of developing computational tools or interactive architecture development environments that can be used to specify, model, and analyze a software system's security architecture at different times in its development—design-time, build-time, and run-time. We have already demonstrated how such an approach can work when limiting attention to IP rights and obligations.

The approach we have been developing for the past few years for modeling and analyzing software system IP license architectures for OA systems (Alspaugh, Asuncion, & Scacchi, 2009; Alspaugh, Scacchi, & Asuncion, 2010; Scacchi & Alspaugh, 2008) may therefore be extendable to also address OA systems with heterogeneous software security license rights and obligations (Scacchi & Alspaugh, 2011). Furthermore, the idea of common or reusable software security licenses may be analogous to the reusable security requirements templates Firesmith (2004) proposed at the Software Engineering Institute. Such security requirement templates may simplify and guide the efforts of customers (or contracting officers) to more readily specify workable requirements that can be readily verified through system development, deployment, and post-deployment support.

Security licenses can be specified, modeled, and analyzed continuously from initial system architectural design through post-deployment support and system evolution, with key points for security license analysis occurring at design-time, build/linking-time, and deployment/run-time. Such security licenses can be stated both (a) informally, using restricted natural language for human readability, authorship, and description of non-functional security requirements; as well as (b) formally, specifying functional security requirements in a computer processable form using a logic-based scheme and modeling notation, with automated production of (a) from (b) and automated architecture-mediated inferences using (b). Analysis of a system/s security requirements can therefore be integrated into the software architecture tool used to express and evolve the architecture so that the analysis evolves automatically in parallel with the architecture.

In general terms, a security license is analogous to a software copyright license such as a general public license (GPL; GNU, 2007). Software licenses consist of intellectual property (IP) *rights* granted by the license, and corresponding license *obligations* needed to obtain the rights. Our innovation is to similarly specify the security obligations and rights of OA system components using elements found in known security capabilities, which we can then model, analyze, and support throughout the system's development and evolution, and use to guide system design and instantiation. Our initial investigation of security licenses (Scacchi & Alspaugh, 2011) has identified rights and obligations, such as

- the obligation for a user to verify his/her authority to see compartment T by password or other specified authentication process;
- the obligation for a specific component to have been vetted for the capability to read and update data in compartment T;

- the obligation for all components connected to specified component C to grant it the capability to read and update data in compartment T;
- the obligation to reconfigure a system in response to detected threats when given the right to select and include different component versions or executable component variants;
- the right to read and update data in compartment T using the licensed component;
- the right to replace specified component C with some other component;
- the right to add or update specified component D in a specified configuration;
- the right to add, update, or remove a security mechanism; and
- the right to update security license L.

Further, formally specified OA security licenses are verifiable, as well as grounded in functional and testable system security capabilities.

The security reasoning chains among the security licenses are mediated by the system architecture and evolve automatically with it, much like they can for IP licenses (Alspaugh et al., 2009; Alspaugh et al., 2011; Alspaugh et al., 2010). Each kind of security license details how its obligations are propagated architecturally to other system components. The results of this propagation, coupled with automated identification of gaps, conflicts, and subsumptions, are communicated to analysts as architecturally organized arguments supporting the existence of the identified issues. The arguments provide context-appropriate guidance in terms of the system architecture and the security licenses of the components involved for resolution of security problems through the evolution of the system design.

Our approach neither assumes nor proves that individual elements of an OA system are secure but instead seeks to determine what security rights and obligations are in effect at any time for the overall system architecture as a function of the security rights and obligations of its components. This means that it is possible to configure a secure OA system whose components may be insecure, or not equally secure. Our approach also supports determination of where or how OA system security rights or obligations may be in conflict, mismatch, or subsume one another as individual system components or connectors are adapted to evolve over time. As an organization's security policies (i.e., their security requirements) evolve and adapt, the OA system's security rights and obligations are evolved to match and satisfy them, as long as all security requirements can be expressed through description logic relationships among them.

Security rights and obligations are characterized in terms of enterprise security policies and goals; within that closed world, our approach enables specification of the security properties that an open system architecture must match or satisfy. These security requirements also direct acquisition program managers and architecture analysts attention to problem areas with greatest impact on system security. Where our approach identifies a conflict or mismatch, it indicates an actual, open-world weakness in the security of the OA system under analysis. The chain of reasoning is architecture-mediated, with its units defined piecewise in each component's security license and evolving continuously as the system architecture, configuration, and security requirements evolve. As new kinds or types of vulnerability, threats, or exploits emerge, as well as new categories of effective responses and emerging alternative security mechanisms, we seek to elaborate and demonstrate that this approach can continuously accommodate the specification and analysis of changing security requirements.

Product Lines: Alternatives, Versions, and Variants of OA Elements

In producing a secure OA system in a software product line, there are several levels of variation available for producing artificial diversity among equivalent instances and for selecting and evolving in the face of threats.

At the highest level of granularity, a system developer or integrator can choose among alternative *producers* of similar components, services, and platforms (Sun et al., 2012). For example, we can find *functionally similar* alternatives from software (component) producers of Web browsers like Mozilla (Firefox, Camino, Sea Monkey) versus Google (Chrome) versus Microsoft (Internet Explorer), versus others. Similarly, for word processors, we find alternatives including Microsoft (Word) versus abisoft.com (AbiWord) versus Google (Google Docs, which is a remote Web service rather than a component), versus others. Likewise, for e-mail and calendar applications, we find alternatives like Microsoft Outlook, Gnome Evolution, Google Mail, and Google Calendar, among others. For operating systems, we find Red Hat Enterprise Linux, Microsoft Windows, Apple OSX, and Google Android, among others. Finally, note that some producers produce more than one alternative of the same kind of component or service, such as Mozilla's Web browsers (Firefox, Camino, SeaMonkey), so that a choice among those particular components does not result in a change of producers.

Functionally similar components and services may not be exactly interchangeable, unless their interfaces are similar or identical. As such, it may be necessary to modify, for example, OA system topology or replace connector types, and other architectural measures may be necessary to change from one producer to another, depending on the functionality needed to satisfy functional requirements. However, in general, the overall functionality provided by the system remains substantially the same; but now the diversity among alternative system instances is the greatest: not only is the component, service, or platform distinct between two instances, but its architectural connections in the system will also be distinct, as will be the software development process and organization that produced it; so the chances of a common vulnerability are greatly minimized. Subsequently, when functionally similar components, connectors, or configurations exist, such that equivalent alternatives, versions, or variants may be substituted for one another, then we have a strong relationship among these OA system elements that is called a *product family* (Narayanaswamy & Scacchi, 1987; Bosch, 2006) or a *product line* (Clements & Northrop, 2001).

As described previously, a shift from one alternative to another ordinarily requires a change in architecture, software connectors, and other measures. Changes between some alternatives will also produce a change of producers, while others will not. However, when components or connectors provide alternative implementations of the functionality they provide, then these are designated as versions. For example, most Linux operating systems support multiple file systems for data storage, though developers or integrators select their preferred file system for inclusion at either design-time or build-time. Similarly, for connectors to remote Web servers, developers or integrators may specify unencrypted (e.g., HTTP) or encrypted (e.g., HTTPS) data communication protocols for use in a Web-based enterprise system. Next, at the OA system configuration level, selection of alternative components or connectors, or of different versions of components or connectors, result in different overall system versions that conform to a system product line. Further, recent advances in source code compilation now allow for creation of *functionally identical* variants of software components, though each variant has a different run-time image in the computer, through code randomization techniques (Franz, 2010; SJWWF11). Last, software product lines can be bound to a network of software producers, system integrators, and

system users/consumers through a software ecosystem (Bosch, 2009), such that secure systems can be realized through composition or configuration at the software ecosystem level (SA12). Consequently, we now have a complete and robust basis for specifying OA systems that can include components, connectors, or application systems from alternative producers, or with different versions or variants included. This is now our basis for moving forward to address the challenges of creating secure OA systems through secured software product lines.

Given the basis for software product lines for OA systems, we now address how to frame and align software system architectures with software security mechanisms. We use the following scheme to address this, as shown in Table 1.

Table 1. Different System Security Elements Whose Rights and Obligations Depend on Capabilities Supported by Lower Level Elements

<table>
<tr><td colspan="2">Security policies</td></tr>
<tr><td>Developers, system integrators and users</td><td rowspan="2">Persistent data</td></tr>
<tr><td>System configurations</td></tr>
<tr><td>Components</td><td rowspan="2">Ephemeral data</td></tr>
<tr><td>Connectors</td></tr>
<tr><td>Platforms</td><td>User I/O data</td></tr>
</table>

System security policies provide the overall context for what kinds of security mechanisms or capabilities (e.g., mandatory role-based data access control) that a particular systems requires. The requirements must be realized through multiple levels of system composition that span a processing space from people to processing platforms, and through data/content space that is processed during system usage/operation.

Aligning system security elements with security mechanisms gives rise to the following associations:

Platform—base technological elements that constitute the computer environment that hosts the target system:

- *hardware*: specifies hardware confinement constraints needed to securely operate the software system configuration, potentially to address memory, storage, and external device port isolation (see SecureSwitch [Sun et al., 2012)]). Hardware may be configured as an embedded processor, mobile computer (e.g., smartphone or tablet), personal computer, multi-processor computation server, or multi-server data center;
- *virtual machine*: a software layer that can isolate and confine the operating system, component applications, or application services from direct control of system hardware, network operations, or operating system processes. OSs, software systems, components, or connectors can each run within their own virtual machine, in alternative configurations, as long as they are completely confined at a higher level of system security and do not overlap virtual machine boundaries (Spencer et al., 1999; Smalley, 2012);

- *network*: message filtering and access control firewalls for data/control flows that move across external hardware system security boundaries; and
- *operating systems*: mandatory access control (Loscocco et al., 1998; Spencer et al., 1999), capability type enforcement (Smalley, 2012), OS configuration parameters (“Security Technical,” n.d.), and run-time audit logs, all currently coded and managed by system integrators/administrators.

Connectors—software mechanisms that implement secure communication mechanisms within and across system boundaries. Connectors enable security mechanisms providing

- data cryptography (encryption/decryption) before/after data transfer;
- component-connector-specific firewalls that can be implemented via (pre-conditions) constraints on in-bound data flow and plug-in/helper application invocation, or on out-bound data flow and external program invocations (post-conditions); and
- multi-version connector configurations between components that allow for artificial diversity and dynamic reconfiguration potential through functionally similar versions.

Components—software mechanisms that implement application functionality required for the targeted system to operate as intended. Components enable security mechanisms providing

- access/usage authentication control obligations (e.g., login with authorized identification and password) for which people in what roles (e.g., developer, system integrator, system administrator, system user) have the specified set of rights to view/update data, data control flow invocations, or external program invocations;
- encapsulate components as services within virtual machines to confine potential exploits while mitigating their propagation;
- alternative versions that increase artificial diversity and enable dynamic replacement with functionally similar alternatives;
- multiple versions that allow for changes in vulnerability space, including concurrent versions with replicated input data, but different out data connector (routing) configurations; and
- multiple variants that reduce vulnerability to component version attacks.

System configuration—the composition and interrelationship of components and connectors that together realize the system architecture at design-time, build-time, or run-time. System configuration (or composition [Bo06]) enables security by providing

- the ability to host multiple (one or more) alternative, version, or variant system configurations on one or more processors (either single-core [Sun et al., 2012], multi-core, multi-blade, or multi-site) that can be dynamically selected in response to security policy directives or in response to detected threats;
- the ability to host concurrently running multiple (one or more) alternative, version, or variant system configurations on one or more processors (either multi-core, multi-blade, or multi-site) that can be dynamically selected in response to security policy directives or in response to detected threats; and

- the ability to (formally) specify system configuration as an open architecture at design-time, build-time, and deployment run-time, along with automated tools that can verify the consistency, completeness, and traceability.

Developers, system integrators, and users—denote the people authorized and trusted to work on or with the configured systems or their elements over time, depending on their externally assigned role(s):

- developers should employ software development environments, tools, or processes that reinforce security-safe software coding practices of components or connectors they implement as products (Seacord, 2008);
- developers should produce multiple, unique, executable variants of the components or connectors they produce and distribute;
- system integrators design OA system architecture;
- system integrators build OA system configurations that select from one or more component or connector alternatives, versions, and variants;
- system integrators deploy one or more run-time system configuration variants that can be readily installed and appropriate parameters entered by system administrators or end users;
- system integrators or system administrators, or automated mechanisms under their control must be able to monitor and access system execution audit logs, to determine if threats or anomalous system behaviors are detected, and to dynamically reconfigure system configuration or security parameters in order to move the executable system into a more trusted operational state;
- users must be provided with online identifiers or identification methods that enable them to access security controlled systems via one or more alternative authentication mechanisms in place.

In parallel with these processing security spaces are data security spaces:

User I/O data—data that may exist only as it passes across communication channels. Examples are keystrokes and mouse movements communicated from a keyboard or mouse to a processor, voice data from microphones and to speakers, Wi-Fi packets, and so forth. This data may be discarded or incorporated into ephemeral data.

Ephemeral data—data that exists in memory for a brief time before being either discarded or incorporated into persistent data. Examples are Web forms that have been filled out but not submitted, and data in various sorts of hardware buffers.

Persistent data—data that exists for a substantial time on local disks or solid-state storage devices, USB memory sticks, DVD-ROM, or server storage.

Security policies—provide overall guidance and requirements for what security mechanisms and regimes are to be designed, implemented, and satisfied during the deployment, operation, and evolution of a specified system. Security policies

- should provide *non-functional requirements* regarding the membership, structure, and behavioral specifications of each of the proceeding categories of security elements at minimum, or further specification of security sub-elements within each category, as per the security exposure of the system being addressed;

- non-functional requirements may only specify rights provided when corresponding obligations are fulfilled that cannot be automated or verified in lower level security elements;
- non-functional requirements should be expressible in human-readable and computer-processable forms within the system security policy license; and
- must provide *functional requirements* regarding the membership, structure, and behavioral specifications of each of the proceeding categories of security elements at minimum, or further specification of security sub-elements within each category, as per the security exposure of the system being addressed;
  - functional requirements are those that can be formalized, automated, and verified by corresponding automated mechanisms available at lower level security elements;
  - functional requirements may only specify rights provided when corresponding obligations are fulfilled that must be automated or verified in lower level security elements; and
  - functional requirements should be expressible in human-readable and computer-processable forms within the system security policy license.

The case study that follows describes where these different system security elements appear in forms that can be available for review by authorized program acquisition personnel.

Case Study of a Secure Product Line for an Enterprise System

Let us consider what needs to be specified during the acquisition of an enterprise system that incorporates common office productivity applications that run on a personal computer networked to remote servers. Such a system can include a Web browser, word processor, e-mail, and calendaring applications that are configured to operate on a personal computer, where the PC's operating system, Web browser, and other applications need to be configured to access remote data/Web content servers. Figure 1 shows part of the system ecosystem of software producers and the components they can provide for our enterprise system.

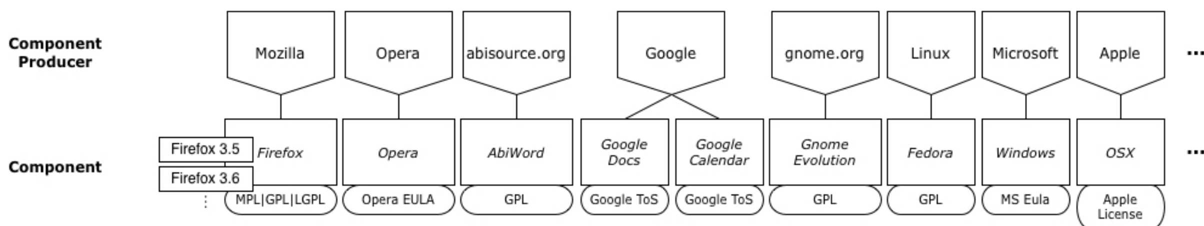

Figure 1. A Partial View of a Software Ecosystem of Producers and the Software Components for an Enterprise System They Produce

Figure 2 shows the design-time architecture of such an enterprise system. What might a secure product line for a system like this involve, and how might it provide benefits and security qualities to be specified for design-time, build-time, and run-time? How can its OA and product-line characteristics contribute to security throughout the acquisition system life-cycle?

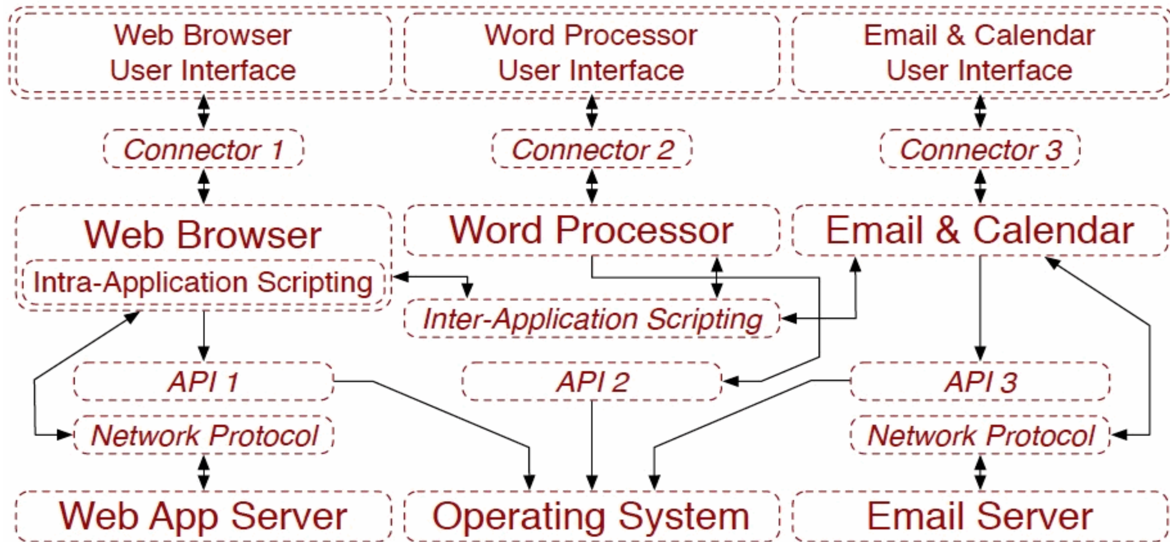

Figure 2. A Design-Time Reference Model of an OA System That Accommodates Multiple Alternative System Configurations

We envision an approach in which non-functional requirements, such as security, reliability, and evolvability requirements at acquisition time are elaborated at design- and build-times by specific functional requirements that explain how and to what degree the non-functional requirements are going to be satisfied at run-time. Analogous to our previous work with IP licensing, we envision that these requirements are structured in the same logical forms as IP licenses (with specific rights that are obtained only by fulfilling specific obligations) and managed through the architecture by the same approach of calculating which obligations are satisfiable, in what way, and as a result what rights are available (Alspaugh et al., 2009; Alspaugh et al., 2010; Scacchi & Alspaugh, 2011).

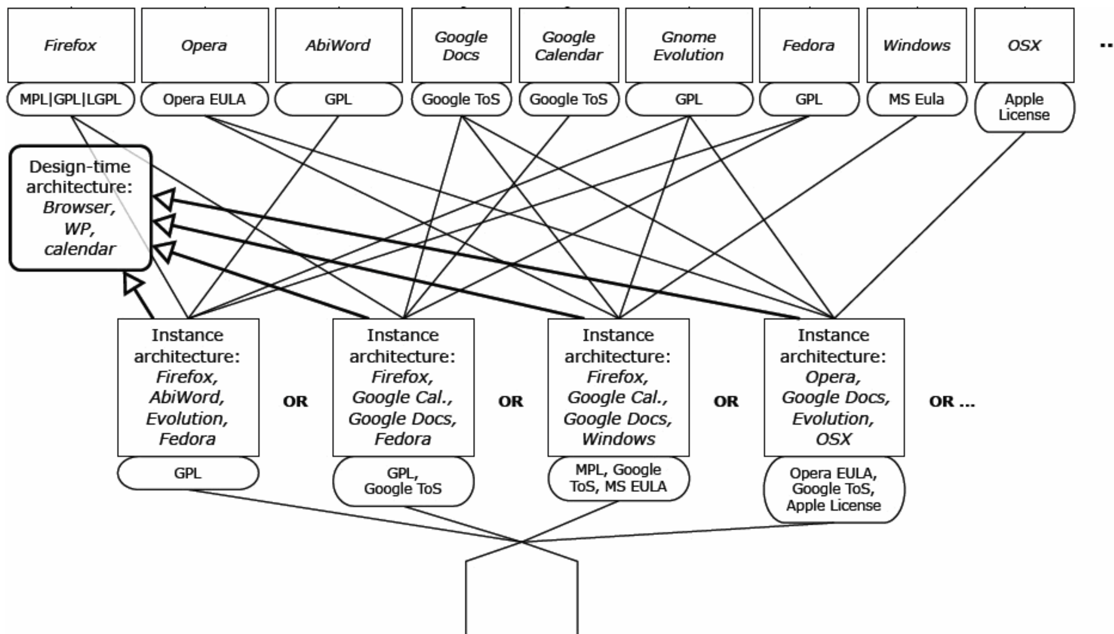

Figure 3. A View of an OA Software Ecosystem That Provides Alternative, Functionally Similar Components Compatible With the Reference Design-Time Architecture

Figure 3 illustrates a possible OA software ecosystem for this product line. Here, a number of possible producers and alternative components have been placed into play, and four specific instance architectures (produced in four specific ecosystems) have been sketched. With appropriate architectural topologies, and appropriate shim components and connectors inserted between the major components, each of these four instance architectures can support the same functionality. It is also possible to achieve different nonfunctional qualities, including security qualities through the four choices, for example, by requiring that OS be an appropriate security-enhanced version of Linux, or by requiring that the network protocol connector be HTTPS.

Within the overall ecosystem of Figure 3, Figure 4 shows one possible instance ecosystem involving specific producers (Mozilla, abisource.org, gnome.org, Red Hat) and specific alternatives (Firefox, AbiWord, Evolution, Fedora).

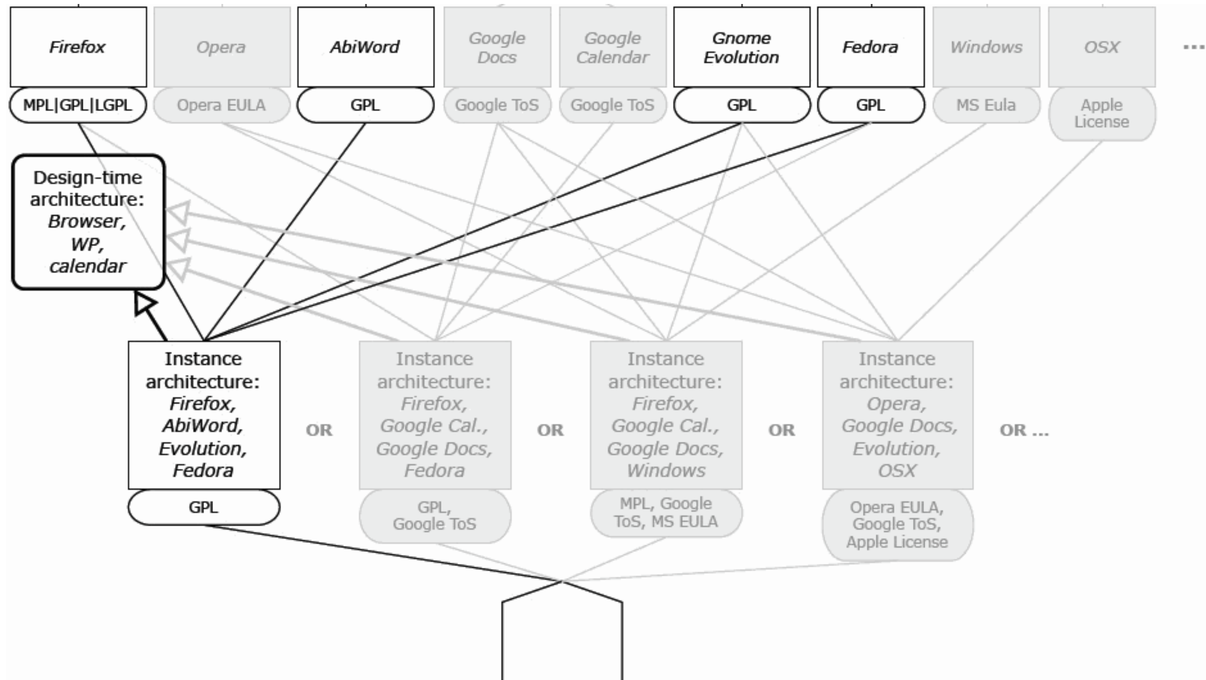

Figure 4. A Selection Among Alternative Components That Can Be Included at Build-Time to Produce an Integrated System Compatible With the Design-Time Reference

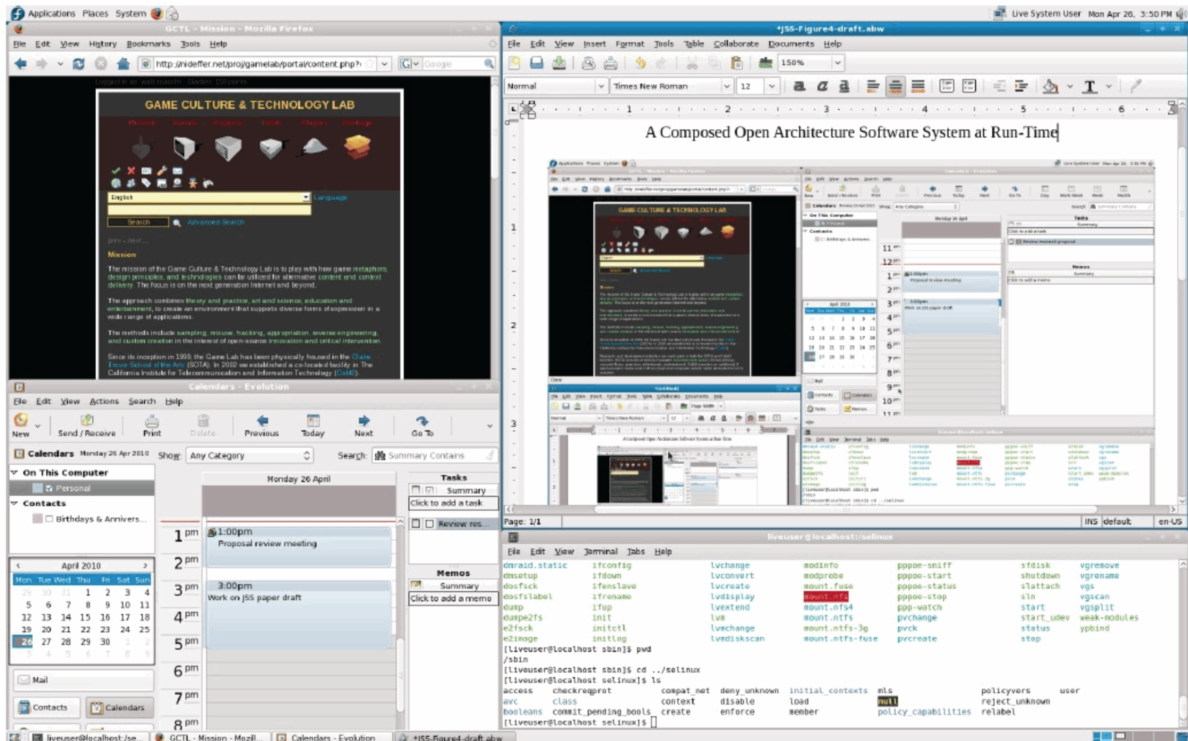

Figure 5. An End-User Run-Time Version of the Selected Alternative Components That Fulfills the Design, Where the Red Hat Enterprise Linux Operating System (Lower Right Corner) Can Utilize the Security Modules Library, SELinux, for Coding and Enforcing Mandatory Access Control on Programs/Data and Other Security Capabilities

Acquisition-time requirements, such as the use of SE Linux and the use of HTTPS, could be satisfied by this choice; with an appropriate architecture, the IP licensing obligations could also be satisfied. At design-time, the functional requirements would need to be satisfied by appropriately specified shims inserted among the principal components, and if such shims could be designed then this would be the proof that the acquisition-time nonfunctional requirements could also be satisfied. Figure 5 shows a run-time view of this instance architecture, resulting from the specific OA ecosystem and instantiating the overall ecosystem of Figure 3 and the software product line of which the software system is an instance.

This instance architecture has both a manageable IP license regime that ensures its openness and a manageable security regime. For IP, in this architectural instance, all component versions can be selected to use permissive licenses (Web browser, Web server) or reciprocal GPL licenses (word processor, e-mail, calendar, and operating system), They are cleanly separated by dynamic run-time links, which are types of connectors that do not transmit IP obligations or rights, though they do allow for control flow integration and data flow interoperation.

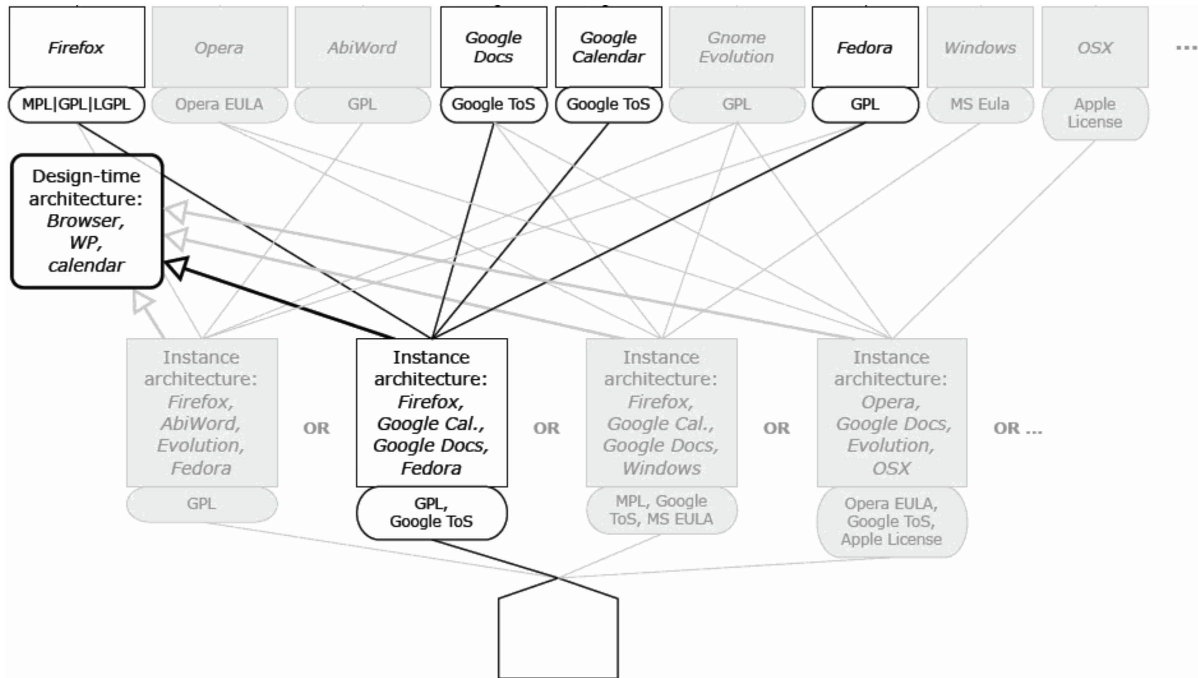

Figure 6. A Second System Configuration Using Alternative but Functionally Similar Components

Figure 6 outlines an alternative system configuration and the instance ecosystem that produces it. This instance architecture substitutes services for components in the case of Google Docs for the word processing functionality and Google Calendar for the calendar functionality. With appropriate shims and changes to the architectural topology this combination of major components could also support the system's functional requirements, and because the services are accessed through client-server connections, which block the propagation of most license obligations, there are a number of ways to satisfy the IP constraints imposed by the component and service licenses.

This alternative configuration also highlights possible acquisition-time concerns and the nonfunctional requirements and security license issues that follow from them. For example, a remote service, such as Google Docs, provides benefits and imposes costs with respect to a compiled component, such as AbiWord. On the one hand, the remote service makes some qualities easier to achieve (data sharing, backup, etc.), but on the other hand may make some qualities harder to achieve (data security over a network connection and in the "cloud," up-time of the service, little or no control over when new versions of the service are used compared to complete control over when new versions of a component are integrated).

- Who in the ecosystem of human actors for this system has the right to make the decisions to use a service in place of a component, or one component version in place of another? What obligations are they required to satisfy first? These questions are of concern at acquisition time and, we claim, are addressable by *acquisition licenses* that restrict rights and impose obligations important to system acquisition officers, just as IP licenses do for IP rights and obligations important to software producers.
- When can these decisions be made? In traditional development processes, these would occur at design-time; but in the larger view we propound here, such

decisions, or rather the policies or acquisition licenses that control them, are perhaps more properly considered at acquisition time. As we see in Figure 7, it is also possible that in order to achieve specific security qualities, they might be made at build- or run-time, in response to specific threats.

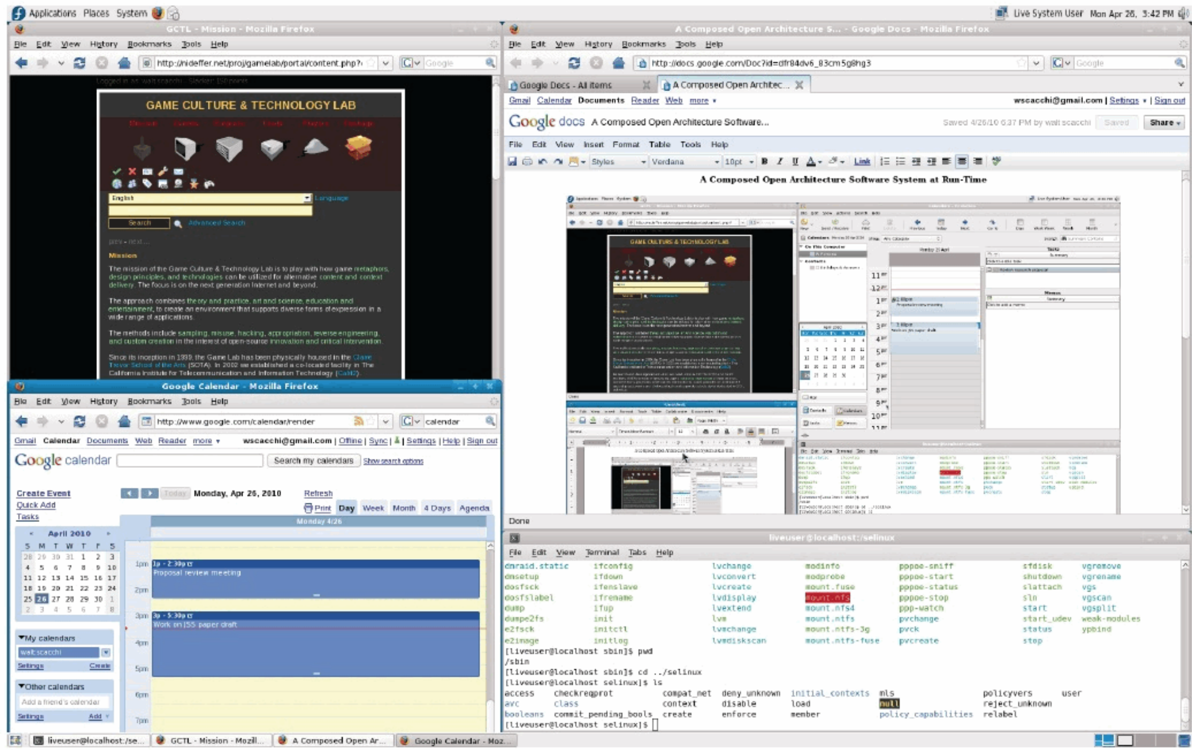

Figure 7. An End-User View of the Alternative Run-Time System Configuration

Figure 7 shows a run-time view of this alternative configuration. To the end user, this system appears quite similar to the one in Figure 5, and the differences might scarcely be noticed, which raises the next set of possibilities.

Both these instance architectures specify specific alternatives for the major components, for example, Firefox for the Web browser component. But which version of Firefox? For example, it is quite possible that both the instance architectures discussed above could be implemented using either Firefox 10 or Firefox 11, satisfying all the functional requirements with no change to the instance architecture and no revision of software shims. Who has the power to decide to use version 10 rather than version 11? How late in the software process can this decision be made? For example, could it be made as late as system startup time by a system user, in response to a particular security attack on the previous configuration?

At the conceptually lowest level, the advent of code randomization and multi-variant software executables leads to the possibility of substituting essentially equivalent variants of the same component, most obviously at build-time. The decision to substitute one variant for another, or the decision to allow the substitution, can be made through the entire range of development times from acquisition time to run-time. The substitution can be put into effect by a human actor or by a software monitor following a security policy, either randomly or in response to specific events in the environment.

Finally, an orthogonal consideration is the use of containment vessels to encapsulate components or subsystems within a virtual machine, to monitor and control interactions

among components and subsystems in order to block attacks and protect vulnerable parts of a system. Figure 8 shows a screenshot in ArchStudio of a design-time architecture utilizing eight containment vessels, seven for individual components and connectors and the eighth for the group of components and connectors associated with the OS.

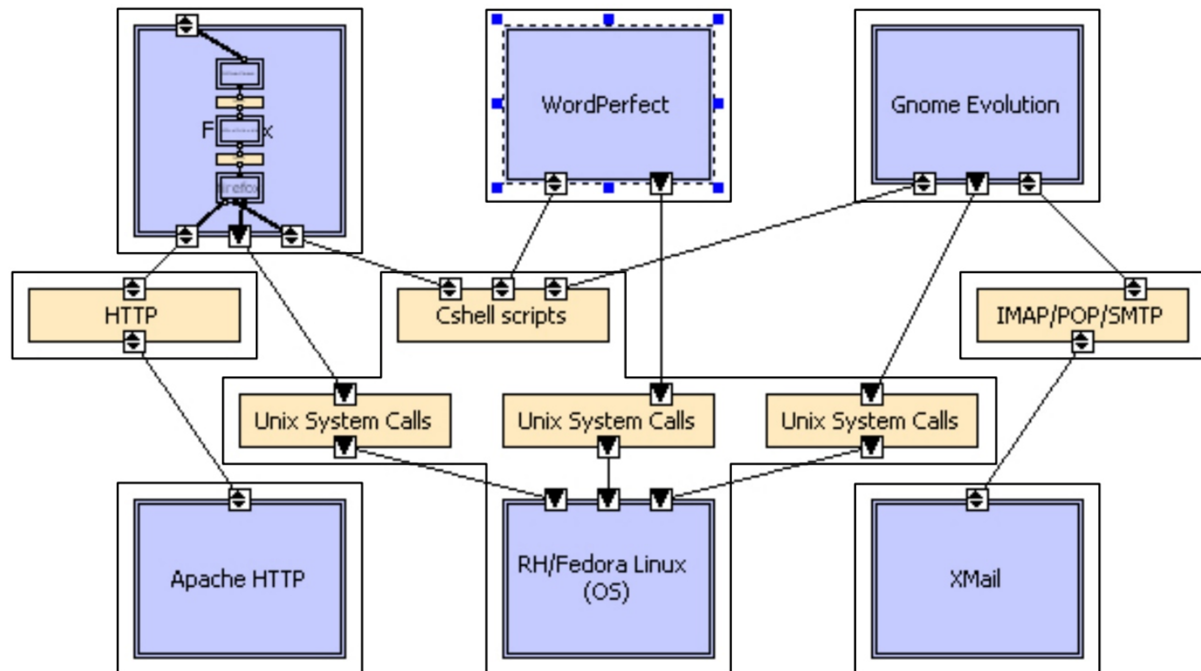

Figure 8. A Security Configuration Alternative for the Run-Time Configuration Instance That Encapsulates OA System Components and Connectors Within Different Virtual Machines (e.g., using the "Xen Hypervisor Project," 2012)

For security, the GPL'd Fedora can employ the SELinux capabilities to restrict all shell/operating systems commands through mandatory access control and type enforcement (see Figure 8), while other components can all be contained within one (for minimal security confinement) or more (for increased security confinement on a per component basis) Xen-based virtual machines (again, see Figure 8). The interoperability of SELinux and Xen is now a common feature of many large Linux system installations (e.g., Amazon.com now has more than 500K Linux systems running Xen; "SELinux on Xen," 2012; "Xen Hypervisor Project," 2012).

Discussion and Conclusions

Our goal in this study was to develop and demonstrate a new approach to address challenges in the acquisition of secure OA software systems. Program managers, acquisition officers, and contract managers will increasingly be called on to provide review and approval of security measures that are employed during the design, implementation, and deployment of OA systems. We seek to make this a simpler and more transparent endeavor. This requires security policies that are appropriate for review and approval during acquisition by people who may not be expert in the specifics of how best to ensure that secure systems will result. Our view is to address this need by investigating how best to specify or model system security in ways that can accommodate security as a continuous process that must be supported throughout the system acquisition life-cycle for OA systems (Scacchi & Alspaugh, 2008, 2011).

Our efforts reported here reveal that it is possible to employ a scheme through which complex OA systems can be designed, built, and deployed with alternative components and connectors into functionally similar system versions in ways that allow for overall system security through the use of multiple security mechanisms. We described a scheme for how to realize and specify such OA system configurations in ways that are inherently compatible with existing security mechanisms, and this scheme does not assume that individual system elements must be secure before inclusion into the secured system's configuration. Central to our scheme is the incorporation of software product line concepts that are integrated with security mechanisms in a coherent way that is amenable to automated support and acquisition management. We also provided a case study that reveals where and how we specify a secure OA enterprise system product line in ways that can accommodate the diverse needs of software producers, software developers, system integrators, users, and acquisition managers. What remains as an important next step for this line of research effort is to more fully articulate how to simply and transparently specify OA system security using streamlined security policies using the kind of system security licenses we anticipate (Scacchi & Alspaugh, 2011), as well as designing and developing a prototype automated system that can support the modeling and analysis of OA system security policies, alternative version OA system configurations, and different OA security licenses.

References

Alspaugh, T. A., Asuncion, H., & Scacchi, W. (2009). Intellectual property rights requirements for heterogeneously licensed systems. In *Proceedings of the 17th IEEE International Requirements Engineering Conference* (pp. 24–33).

Alspaugh, T. A., Asuncion, H., & Scacchi, W. (2011). Presenting software license conflicts through argumentation. In *Proceedings of the 23rd International Conference on Software Engineering and Knowledge Engineering*.

Alspaugh, T. A., Scacchi, W., & Asuncion, H. (2010, November). Software licenses in context: The challenge of heterogeneously licensed systems. *Journal of the Association for Information Systems*, *11*(11), 730–755.

Attack of the computer mouse. (2011, June 29). Retrieved from The H Online Security website: http://h-online.com/-1270018

Bosch, J. (2006, December). The challenges of broadening the scope of software product families. *Communications of the ACM 49*(12), 41–44.

Bosch, J. (2009). From software product lines to software ecosystems. In *Proceedings of the 13th International Software Product Line Conference* (pp. 111–119).

Breaux, T. D., & Anton, A. I. (2005). Analyzing goal semantics for rights, permissions, and obligations. In *Proceedings of the 13th IEEE International Conference on Requirements Engineering* (pp. 177–188).

Breaux, T. D., & Anton, A. I. (2008). Analyzing regulatory rules for privacy and security requirements. *IEEE Transactions on Software Engineering*, *34*(1), 5–20.

Clements, P., & Northrop, L. (2001). *Software product lines: Practices and patterns.* New York, NY: Addison-Wesley.

DoD Open Source Software (OSS). (2010). Frequently asked question regarding open source software (OSS) and the Department of Defense (DoD). Retrieved from http://cio-nii.defense.gov/sites/oss/Open_Source_Software_%28OSS%29_FAQ.htm

DoD. (2011, July). *Department of Defense strategy for operating in cyberspace*. Retrieved from http://www.defense.gov/news/d20110714cyber.pdf

Falliere, M., et al. (2011, February). *W32.Stuxnet Dossier, version 1.4.* Retrieved from http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf

Firesmith, D. (2004, January–February). Specifying reusable security requirements. *Journal of Object Technology*, *3*(1), 61–75.

Franz, M. (2010, September). E unibus pluram: Massive-scale software diversity as a defense mechanism. In *New Security Paradigms Workshop.* Concord, MA.

Garcia, P. (2010). Maritime C2 strategy: An innovative approach to system transformation. In *Proceedings 15th International Command & Control Research & Technology Symposium* (Paper 147). Santa Monica, CA.

Gizzi, N. (2011). Command and control rapid prototyping continuum (C2RPC) transition: Bridging the Valley of Death. In *Proceedings of the Eighth Annual Acquisition Research Symposium* (Vol. 1). Monterey, CA: Naval Postgraduate School.

GNU. (2007). GNU general public license. Retrieved from http://www.gnu.org/licenses/gpl.html

Loscocco, P., Smalley, S., Muckelbauer, P., Taylor, R., Turner, S., & Farrell, J. (1998). The inevitability of failure: The flawed assumption of security in modern computing environment. In *Proceedings of the 21st National Information Systems Security Conference* (pp. 303–314).

Narayanaswamy, K., & Scacchi, W. (1987). Maintaining configurations of evolving software systems. *IEEE Transactions on Software Engineering*, *13*(4), 323–334.

Navy. (n.d.) *Navy open architecture guidelines*. Retrieved from https://acc.dau.mil/oa

Navy. (2010). PEO IWS releases open architecture contract guidebook update. Retrieved from http://www.navy.mil/search/display.asp?story_id=53661

Salamat, B., Jackson, T., Wagner, G., Wimmer, C., & Franz, M. (2011, July). Run-time defense against code injection attacks using replicated execution. *IEEE Transactions on Dependable and Secure Computing, 8*(4).

Sawers, P. (2011, June 28). US govt. plant USB sticks in security study, 60% of subjects take the bait. *TNW: The Next Web.* Retrieved from http://thenextweb.com/industry/2011/06/28/us-govt-plant-usb-sticks-in-security-study-60-of-subjects-take-the-bait

Scacchi, W., Brown, C., & Nies, K. (2011, July). *Investigating the use of computer games and virtual worlds for decentralized command and control* (Grant #N00244-10-1-006). Irvine, CA: University of California, Irvine, Institute for Software Research. Retrieved from http://www.ics.uci.edu/~wscacchi/ProjectReports/NPS-Reports/DECENT.pdf

Scacchi, W., & Alspaugh, T. (2008). Emerging issues in the acquisition of open source software within the U.S. Department of Defense (NPS-AM-08-036). In *Proceedings of the Fifth Annual Acquisition Research Symposium* (Vol. 1, pp. 230 –244). Monterey, CA: Naval Postgraduate School.

Scacchi, W., & Alspaugh, T. (2011, May). Advances in the acquisition of secure systems based on open architectures. In *Proceedings of the Eighth Annual Acquisition Research Symposium.* Monterey, CA: Naval Postgraduate School.

Seacord, R. (2008). *The CERT C secure coding standard.* New York, NY: Addison-Wesley.

SELinux on Xen. (2012). Retrieved from http://wiki.prgmr.com/mediawiki/index.php/SELinux_on_Xen

Shrobe, H. (2011, November). *Secure computing systems.* Presentation at the Darpa Colloquium on Future Directions in CyberSecurity, Arlington, VA. Retrieved from http://www.darpa.mil/WorkArea/DownloadAsset.aspx?id=2147484460

Security technical information guide, Android 2.2 (Dell). (n.d.) Retrieved from http://iase.disa.mil/stigs/net_perimeter/wireless/smartphone.html

Smalley, S. (2012). *The case for security enhanced (SE) android*. Retrieved from https://events.linuxfoundation.org/images/stories/pdf/lf_abs12_smalley.pdf

Spencer, R., Smalley, S., Loscocco, P., Hibler, M., Andersen, D., & Lepreau, J. (1999). The flask security architecture: System support for diverse security policies. In *Proceedings of the Eighth USENIX Security Symposium* (pp. 123–139).

Stuxnet. (2011). Overview. Retrieved from http://en.wikipedia.org/wiki/Stuxnet

Sun, K., Wang, J., Zhang, F., & Stavrou, A. (2012). SecureSwitch: BIOS-assisted isolation and switch between trusted and untrusted commodity OSes. In *Proceedings of the 19th Annual Network and Distributed System Security Symposium.*

Xen Hypervisor Project. (2012). Retrieved from http://www.xen.org/products/xenhyp.html

Yau, S. S., & Chen, Z. (2006). A framework for specifying and managing security requirements in collaborative systems. In *Proceedings of the Third International Conference on Autonomic and Trusted Computing* (pp. 500–510).

Acknowledgements

Research described in this report was supported by grant #N447602-12-1-0004 from the Acquisition Research Program at the Naval Postgraduate School and from grant #0808783 from the National Science Foundation. No review, approval, or endorsement is implied.

ACQUISITION RESEARCH PROGRAM
GRADUATE SCHOOL OF BUSINESS & PUBLIC POLICY
NAVAL POSTGRADUATE SCHOOL
555 DYER ROAD, INGERSOLL HALL
MONTEREY, CA 93943

www.acquisitionresearch.net