18th ICCRTS
“C2 in Underdeveloped, Degraded and Denied Operational Environments”

Title

Challenges in the Development and Evolution of Secure Open Architecture Command and Control Systems

Paper-ID-098

Topics
Architectures, Technologies, and Tools (Primary)
Approaches and Organization (Alternate)

Authors
Walt Scacchi and Thomas A. Alspaugh
Institute for Software Research
University of California, Irvine
Irvine, CA 92697-3455 USA

Point of Contact
Walt Scacchi
University of California, Irvine
Irvine, CA 92697-3455 USA
949-824-4130, 949-824-1715 (fax)
wscacchi@ics.uci.edu

Abstract: We identify challenges that arise during development and evolution of secure Open Architecture (OA) command and control (C2) systems. OA systems are those whose software system components and interconnection mechanisms are either proprietary closed source software offerings with open interfaces (e.g., Application Program Interfaces), open source software, or some architectural configuration of closed and open source elements. Secure OA systems are those where the security of individual software elements may be uncertain, because of the ongoing evolution, poorly understood system integration compromises, or obtrusive software intellectual property licenses, yet where overall OA security must be continuously assured. We present a framework that organizes OA system security elements and mechanisms in forms aligned with stages of the life cycle of C2 for system design, building, and runtime deployment, as well as system evolution. We provide a case study to show our scheme and how it can be applied to C2 system architectures that rely on an OA. Finally, we show how our efforts complement and extend the agile C2 framework that utilizes a new generation of software components and security mechanisms that are engineered/adapted by multiple parties and disseminated within a diverse marketplace ecosystem of software producers, integrators, and consumers.

Introduction

In this paper, we identify and investigate technical and acquisition challenges that arise during the development and evolution of secure Open Architecture (OA) command and control (C2) systems. OA systems are those whose software system components and interconnection mechanisms are either proprietary closed source software offerings with open interfaces (e.g., Application Program Interfaces), open source software (including Government Open Source Software or Defense Open Source Software (OSS) via Forge.mil), or some architectural configuration of closed and open source elements. Secure OA systems are those where the security of individual software elements may be uncertain, because of the ongoing evolution, poorly understood system integration compromises, or obtrusive software intellectual property licenses, yet where overall OA security must be continuously assured [ScA12a, ScA12b, ScA12c].

It is now clear that future C2 systems must resist internal or external attacks on single/multiple system components, interconnection interfaces, or data repositories. No longer can we rely on "air-gap" system barriers, or security through proprietary obscurity, as individual security barriers get compromised through intrusive cyberwarfare software attack vectors or social engineering. Furthermore, current approaches to system security are most often piece-meal with little/no support for guiding the what system security requirements must address across different software system processing elements and data levels, and how those can be manifest during the design, building, deployment, and evolution of OA software systems. Finally, agile C2 efforts seek to transform overall system development and evolutionary adaptation time frames from years to months (or less) [RBC12]. This means fundamentally new approaches to secure C2 system development and evolution must be available.

We present a framework that organizes OA system security elements and mechanisms in forms that can be aligned with different stages of the life cycle of C2 system design, building, and runtime deployment, as well as system evolution. We provide a case study to show our scheme and how it can be applied to centralized C2 system architectures like C2RPC [Gar11, Giz11] or to next-generation decentralized C2 systems [SBN12] that rely on an OA. Finally, we show how our efforts complement and extend the agile C2 framework that utilizes a new generation of software components (apps, widgets, connectors, and encapsulation mechanisms) that are sourced from a diverse marketplace ecosystem of software producers [RBC12].

Open Architectures for Command and Control Systems

Open architecture (OA) software is a customization technique that enables third parties to modify a software system through its exposed architecture, evolving the system by replacing its components, connectors, or configuration. The three military services within the U.S. Department of Defense are pursuing initiatives that encourage the adoption of OA approaches, and OA systems as way to reduce system development costs over the life of a system [ACC13]. Increasingly more software-intensive systems are developed using an OA strategy, not only with proprietary, closed source software components with open application program interfaces (APIs). But also OSS components. However, composing a system with components that are subject to different intellectual property (IP) licenses, increases the likelihood of conflicts, liabilities, and no-rights stemming from incompatible licenses. We call systems whose components are subject to different licenses, heterogeneously-licensed systems [ASA10]. So in our work we define an OA C2 system as a software system for command and control consisting of components that are either OSS or proprietary with open APIs, whose overall system license rights at a minimum allow its use and redistribution, in full or in part. It may appear that using a system architecture that incorporates OSS components and uses open APIs will result in an OA system. But not all such architectures will produce an OA, since the (possibly empty) set of available license rights for an OA system depends on: (a) how and why OSS and open APIs are located within the system architecture, (b) how OSS and open APIs are implemented, embedded, or interconnected, and (c) the degree to which the licenses of different OSS components encumber all or part of a software system's architecture into which they are integrated [ASA10,AAS12, ScA12].

The following kinds of software elements appearing in common software architectures can affect whether the resulting systems are open or closed [BCK03].

Software source code components – These can be either (a) standalone programs, (b) libraries, frameworks, or middleware, (c) inter-application script code such as C shell scripts, or (d) intra-application script code, as for creating Rich Internet Applications using domain-specific languages such as XUL for the Firefox Web browser, “mashups", or their composition into widgets [Fel07, OWl3, SWM11]. Their source code is available and they can be rebuilt. Each may have its own distinct license.

Executable components – These components are in binary form, and the source code may not be open for access, review, modification, or possible redistribution. If proprietary, they often cannot be redistributed, and so such components will be present in the design- and run-time architectures but not in the distribution-time architecture.

Software services – An appropriate network-accessible software service can replace a source code or executable component.

Application programming interfaces/APIs – Availability of externally visible and accessible APIs is the minimum requirement for an “open system" [MeO01]. Open APIs are not and cannot
be licensed, and can limit the propagation of license obligations.

Software connectors – Software whose intended purpose is to provide a standard or reusable
way of communication through common interfaces, e.g. Microsoft.NET, Enterprise Java Beans, GNU Lesser General Public License (LGPL) libraries, and data communication protocols like the HyperText Transfer Protocol (HTTP). Connectors generally limit the propagation of license obligations.

Methods of connection – These include linking as part of a configured subsystem, dynamic linking, client-server connections, and what we call, “interface shims” (abstract interfaces or interface libraries). Methods of connection affect license obligation propagation, with different methods affecting different licenses.

Configured system or subsystem architectures – These are software systems that are used as atomic components of a larger system, or as a reusable or “functional capability,” such that its internal architecture may comprise components with different licenses, affecting the overall system license. To minimize license interaction, a configured system or sub-architecture may be surrounded by what we term a license firewall, namely a layer of dynamic links, client-server connections, license interface shims, or other connectors that block the propagation of reciprocal obligation. Similarly, a configured system or subsystem can be encapsulated within a security mechanism such as a virtual machine [Xen12].

Examples of such elements appear in descriptions and figures presented later in this paper. But the diversity of kinds of elements that appear in an OA system enables the design, development, and evolution of agile C2 systems within an agile and adaptive software ecosystem [RBC12], as we well show.

Accommodating Agile C2 Development and Evolution
The MITRE Corporation and others in the Defense community seek to embrace the development of agile C2 systems [RBC12]. Such systems are envisioned to arise from the assembly and integration of system elements (application components, widgets, content servers, networking elements, etc.) within a software ecosystem of multiple producers, integrators, and consumers who may supply or share the results of their efforts. The assembly and integration of system elements produces "C2 system capabilities" (C2SCs). C2SCs may be produced, acquired, integrated, shared, or reused by different trusted parties. C2SCs may address a set of ISR data/signal processing components, office productivity components supporting mission planning, or the like. Our purpose is to identify how our approach to the design of secure OA systems can be aligned with their vision for agile C2 systems. Along the way we focus on design of OA system capability involving office productivity components that must be configured as a secure C2SC.

The design and development of agile C2 systems follows from two sets of principals: one set addressing guidelines/tenets for multi-party engineering (MPE) of C2 system components; the other set addressing attributes of agile and adaptive ecosystems (AAE) for producing C2SCs or C2 system elements. For brevity, we simply identify these principals for MPE and AAE, as they are more fully explained elsewhere [RBC12], but we do so in ways that foreshadow and more clearly align with our approach that follows in a later section.

MPE Tenets:

1. Provide small system components that can be rapidly developed, and accommodate different functionally equivalent variants, or functionally similar versions
2. Certify components are consistent with "shared agreements" regarding security requirements, system architecture, data semantics, production and integration processes or process constraints, and other aspects of mission-specific or mission-common domain models
3. Supply diverse C2 system components via a market of component producers or integrators
4. Assemble and integrate C2SCs from components available in the market that are consistent with relevant shared agreements
5. Provide feedback from C2 system users to component producers or capability integrators to improve market efficiency and effectiveness.

AAE Attributes:

1. Encourage and sustain a software ecosystem that is agile (supports assembly and integration C2SC) from components in market, and adaptive (supports substitution of functionally similar

component versions or functionally equivalent component variants), in line with user feedback.
2. Component markets are federated so as to accommodate sharing, reuse, or trading of components across different system integrators or consumer organizations.
3. Shared agreements serve as a basis for enabling multi-party collaboration in system development, integration, and evolution/sustainability.
4. Production, integration, or post-deployment support for components or C2SCs must be viable for small businesses or large, as well as promoting market diversity and effectiveness.
5. Consumer/user organizations seek to manage portfolios of components or C2SCs that collectively improve mission effectiveness, agility and adaptiveness, while reducing costs.

Finally, to help understand what we mean by a software ecosystem, we use Figure 1 to represent where different parties are located across a generic software ecosystem, and the supply networks or multi-party relationships that emerge to enable the software producers to develop and release products that are assembled and integrated by system integrators for delivery to consumer/end-user organizations.

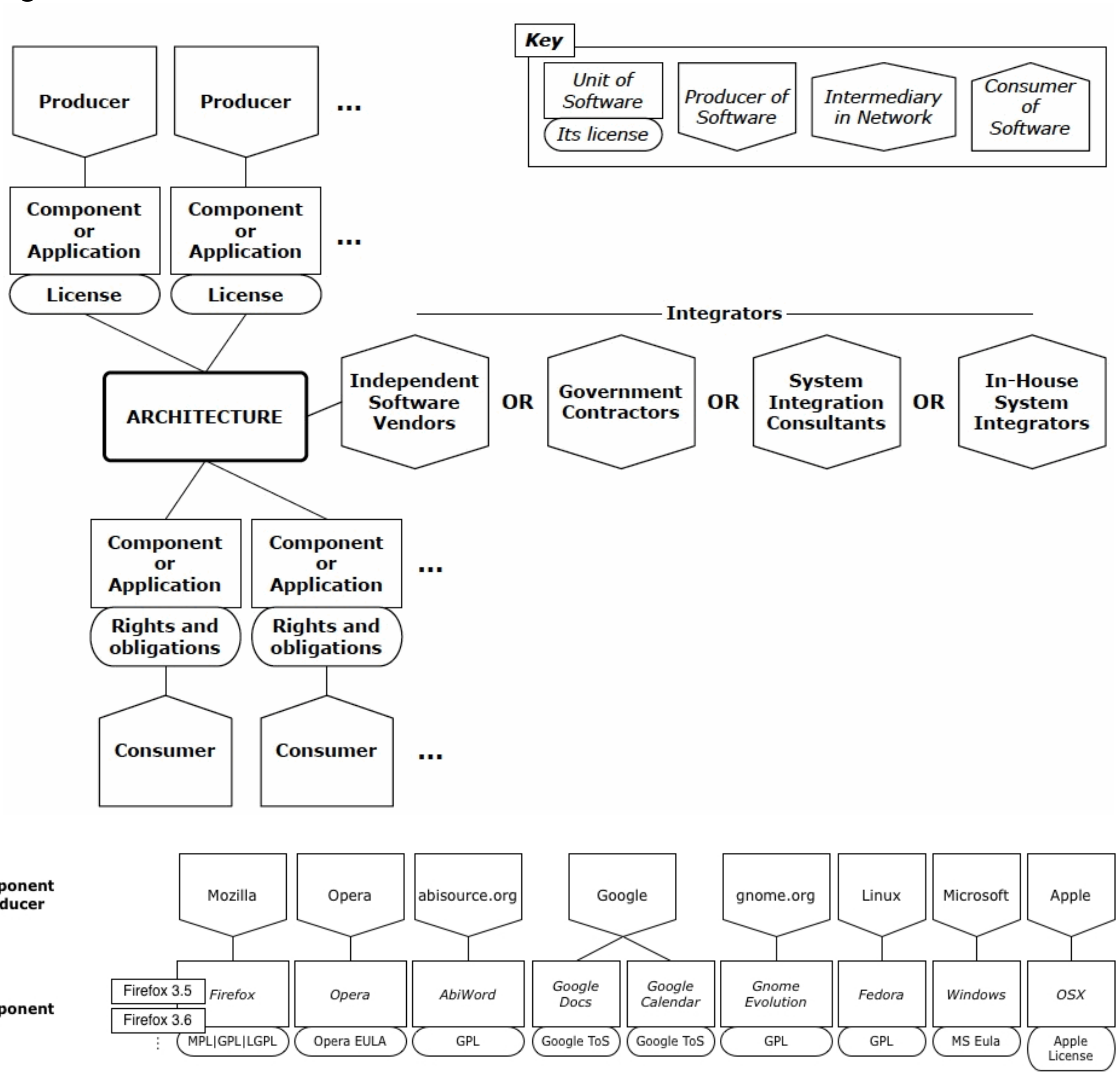


Figure 1. A generic software ecosystem supply network (upper part), along with a sample elaboration of producers, software components, and licenses for an OA system components they employ (lower part) [ScA12].

The lower part of Figure 1 also identifies where elements of shared agreements like IP licenses enter into the ecosystem, and how the assembly of components into a configured system or subsystem architecture by system integrators effectively (and perhaps unintentionally) determines which IP license obligations and rights get propagated to consumer/end-user organizations. Agreement terms and conditions acceptable to consumer/end-user organizations flow back to the integrators. This helps reveal where and how shared agreements will mix, match, mashup, or not at the system architecture level, which is another reason for why we use (and advocate) explicit OA system models.

A Framework for Securing Agile OA C2 Systems

Over the past five or so years, we have been researching, prototyping, and refining an approach to the acquisition, development, and evolution of OA software systems [ASA10, AAS12, ScA12]. Central to our approach is our reliance on explicit models of software system architectures described in an architectural description language [TMD09]. Use of explicit architectural models is key to making OA systems tractable and observable, since we want to be able to make visible where and how a system's architecture is open, and where it is not, during different system development and evolution activities. Explicit architectural representations are also key to coordinating the development, integration, deployment, and evolution of complex OA software systems among a dispersed community of development and user organizations [OPM03]. Our models also draw attention to the identification of a system's elements and their configuration into a system capability or complete system. In addition, our models allow for the system elements to be specified by type or instance (e.g., web browser, Microsoft Internet Explorer), as well as optionally specifying functionally similar versions or functionally equivalent variants thereof (e.g., Internet Explorer 8 and Internet Explorer 9 are similar, while 32-bit and 64-bit variants of IE9 are equivalent)[1]. Finally, we annotate our models with formal expressions that allow us to specify details like IP license or security policy obligations and user rights in ways that are amenable for acquisition contracting and auditing, and compliance practices [ASA10]. Thus, our annotated OA system models form a core of the shared agreements identified as a key element to the development of agile C2 systems within an agile and adaptive ecosystem [RBC12]. The remainder of this section identifies other aspects of our approach that align with the MPE/AAE framework.

Let us consider what needs to be specified during the acquisition of an OA C2 system that allows for a system suggested by use within the command and control rapid prototyping continuum, C2RPC [Gar11, Giz11]. Such a system incorporates both mission-specific components (applications or widgets for processing ISR data, e.g., [GeW12]), and also common office productivity applications that run on a personal computer networked to remote servers. Such a system can include a web browser, word processor, email 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. A web browser like Mozilla Firefox, Microsoft Internet Explorer,

1 Many software producers utilize multi-level numerical identifiers or other nomenclature (e.g., Internet Explorer 10 Release Preview) to distinguish major version releases from minor revision variants (e.g., Internet Explorer 9 versus Internet Explorer 9.1.3) which we also accommodate. Such specificity is required to support system integration and deployment requirements.

Opera, or Google Chrome can further be tailored and invoked through internal scripts to support small, mission-specific widgets, as might be developed using the Ozone Widget Framework [OWF2013].

Figure 2 shows the reference design of an OA system architecture of the office productivity capability associated with a C2 system [cf. Gar11]. This OA system design also accommodates the integration of browser-based remote networked services or scripted widgets. What might a secure software 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?

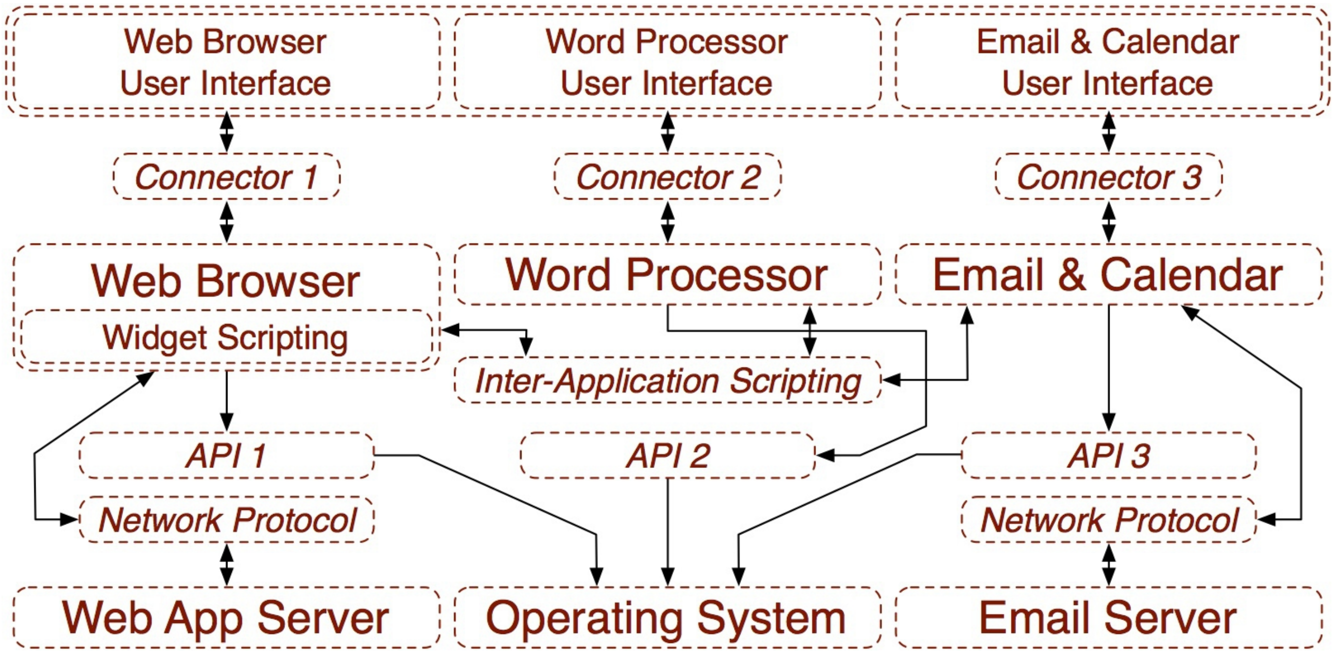


Figure 2. A design-time reference model of an OA system that accommodates multiple alternative software component selections and configurations.

Within our approach, we address non-functional C2 system requirements, such as security, configurability into C2SC, and post-deployment adaptation, These requirements are elaborated at design and integration times by specific functional requirements that explain how and to what degree the non-functional requirements are going to be satisfied at deployment time for consumer/end-user organizations [AAS09, ASA10, ScA12].

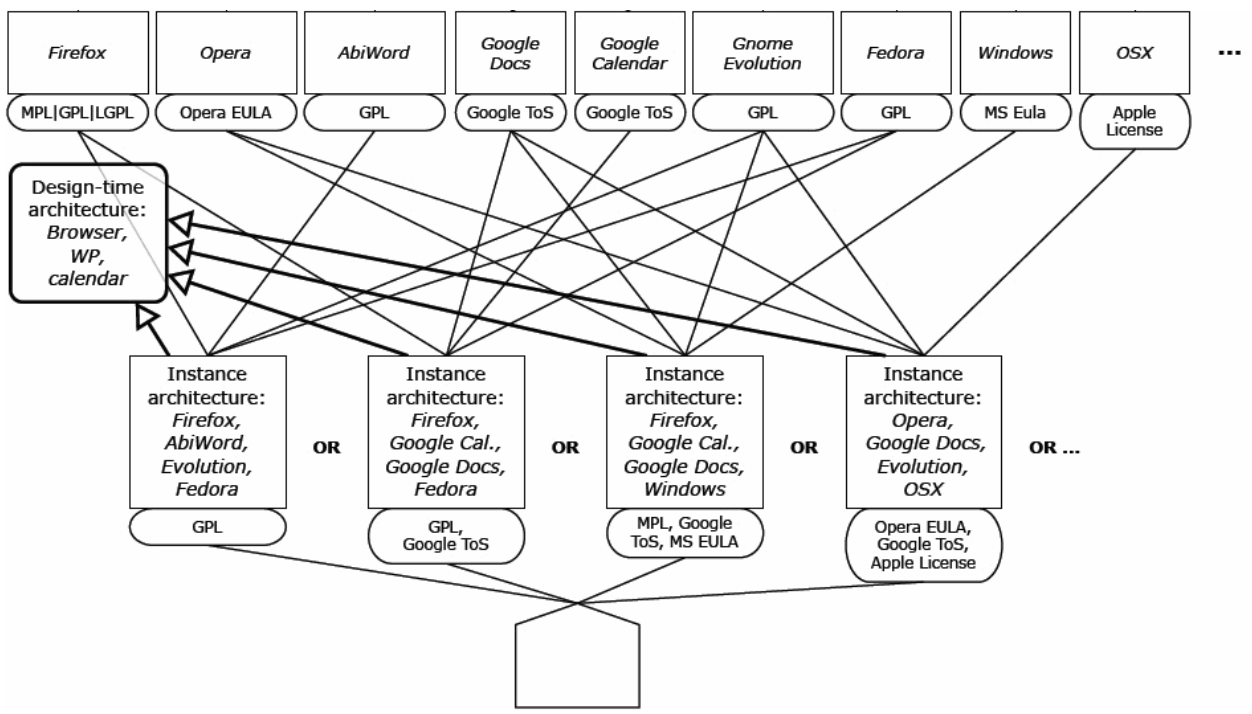

Firefox
Opera
AbiWord
Google Docs
Google Calendar
Gnome Evolution
Fedora
Windows
OSX
...
MPL|GPL|LGPL
Opera EULA
GPL
Google ToS
Google ToS
GPL
GPL
MS Eula
Apple License
Design-time architecture: Browser, WP, calendar
Instance architecture: Firefox, AbiWord, Evolution, Fedora
OR
Instance architecture: Firefox, Google Cal., Google Docs, Fedora
OR
Instance architecture: Firefox, Google Cal., Google Docs, Windows
OR
Instance architecture: Opera, Google Docs, Evolution, OSX
OR ...
GPL
GPL, Google ToS
MPL, Google ToS, MS EULA
Opera EULA, Google ToS, Apple License

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 software product line that follows from the OA software ecosystem shown in Figure 2. Here a number of possible producers and the components they produce and license come into play, within four specific instance C2SC architectures. With appropriate architectural topologies, and appropriate shim components and connectors inserted between the major components, each of these four instances support the same general functionality of the office productivity C2SC that can support mission planning. This means that it becomes possible to offer support for rapidly switching from one OA system configuration to another by substituting compatible (functionally similar) components. This gives us the ability to adapt the C2SC in ways that sustain its overall operational requirements, while allowing multiple parties to independently maintain or evolve the component configuration they choose.

Last, it is also possible to achieve different nonfunctional requirements addressing support for security policies through the four architectural choices; for example, by requiring that computer operating systems on which such a capability be hosted must support an appropriate mandatory access control and type enforcement mechanism, such as is provided in the Security-Enhanced Linux protection service library (e.g., for computers running the RedHat Enterprise Linux or Fedora operating systems), or by requiring the use of secure network protocol connectors like HTTPS which provide basic network data encryption functionality.

A Case Study for Securing OA C2 Systems

We utilize a case study to describe and analyze our approach to design secure OA C2 systems or C2SCs. Such a study serves to help identify issues that arise pertaining to our support of MPE/AAE elements, which in turn drive the development or evolution of such systems, whether they are deployed in a fully developed environment, or whether they are envisioned to address challenges that arise in underdeveloped, degraded or resource limited operational environments.

Our study is divided into two parts, the first addressing design of a simple centralized C2 system with an OA for use in a fully developed environment, and the second addressing a similar but decentralized C2 system with an OA, which may be appropriate for experimental studies.

Centralized C2 system architecture

Traditional C2 systems are designed to support a centralized system deployment. In such a situation, all core system elements or capabilities are located in a single facility, though such a facility may be mobile (e.g., airborne or shipboard).

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

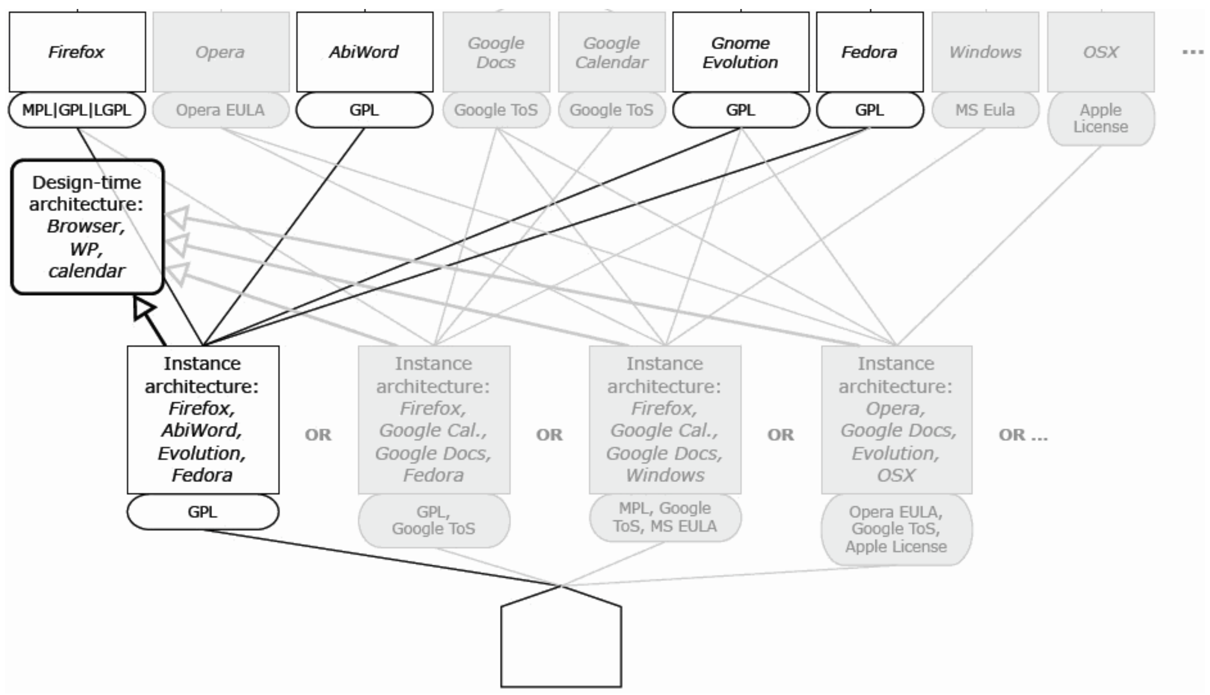
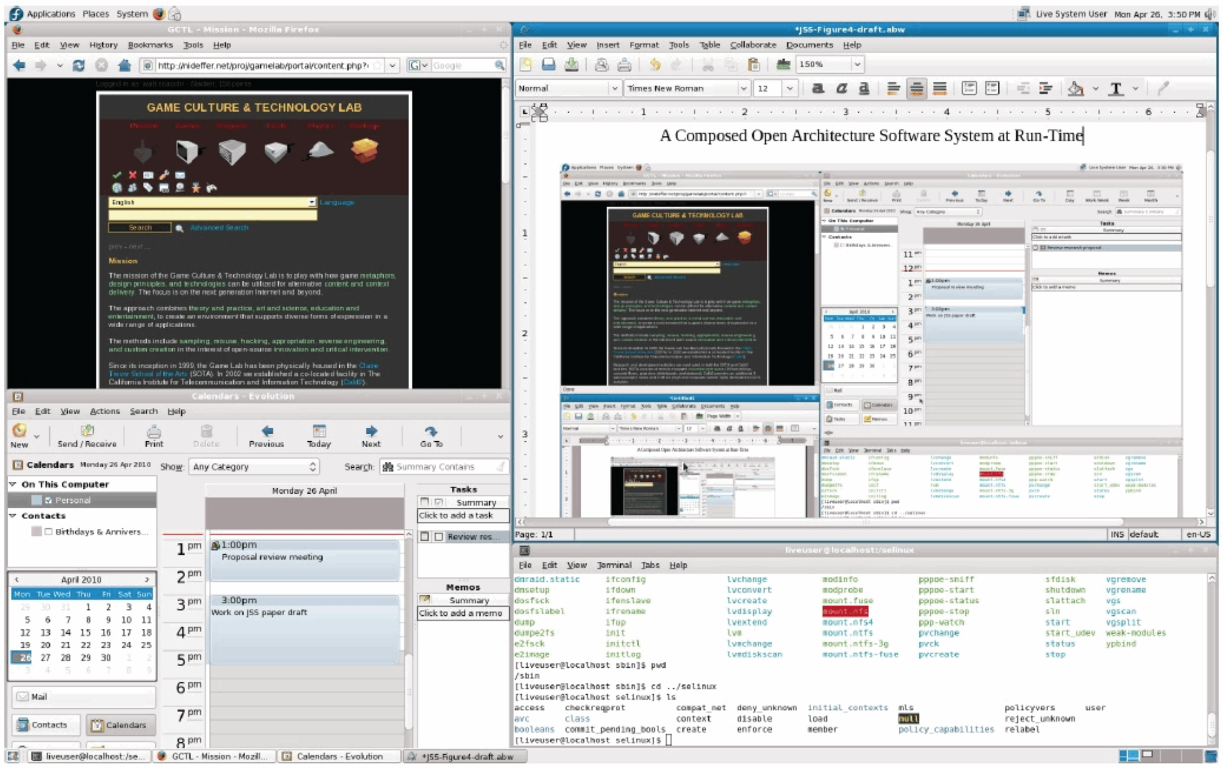

Firefox
Opera
AbiWord
Google Docs
Google Calendar
Gnome Evolution
Fedora
Windows
OSX
...
MPL|GPL|LGPL
Opera EULA
GPL
Google ToS
Google ToS
GPL
GPL
MS Eula
Apple License
Design-time architecture: Browser, WP, calendar
Instance architecture: Firefox, AbiWord, Evolution, Fedora
GPL
OR
Instance architecture: Firefox, Google Cal., Google Docs, Fedora
GPL, Google ToS
OR
Instance architecture: Firefox, Google Cal., Google Docs, Windows
MPL, Google ToS, MS EULA
OR
Instance architecture: Opera, Google Docs, Evolution, OSX
Opera EULA, Google ToS, Apple License
OR ...

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 then shows what a deployed run-time instantiation of this OA C2SC might look like from the perspective of the system's end-users. Here we see the Firefox web browser (upper left corner), AbiWord word processor (upper right corner), Gnome Evolution email+calendaring application (lower left corner), and the Fedora operating system (lower right corner). Though the visual detail in this example is limited, the Red Hat Fedora Linux operating system (lower right corner) is shown utilizing the *SELinux* security protection library, for coding and enforcing mandatory access control on programs/data, and other security enforcement functions.



Mission

The mission of the Game Culture & Technology Lab is to play with how game metaphors, design principles, and technologies can be utilized for alternative content and context delivery. The focus is on the next generation Internet and beyond.

The approach combines theory and practice, art and science, education and entertainment, to create an environment that supports diverse forms of expression in a wide range of applications.

The methods include sampling, misuse, hacking, appropriation, reverse engineering, and custom creation in the interest of open-source innovation and critical intervention.

Since its inception in 1999, the Game Lab has been physically housed in the Claire Trevor School of the Arts (SOTA). In 2002 we established a co-located facility in The California Institute for Telecommunication and Information Technology (Calit2).

```
dmraid.static    ifconfig      lvchange      modinfo          pppoe-sniff    sfdisk      vgremove
dmsetup          ifdown        lvconvert     modprobe         pppoe-start    shutdown    vgrename
dosfsck          ifenslave     lvcreate      mount.fuse       pppoe-status   slattach    vgs
dosfslabel       ifrename      lvdisplay     mount.nfs        pppoe-stop     sln         vgscan
dump             ifup          lvextend      mount.nfs4       ppp-watch      start       vgsplit
dumpe2fs         init          lvm           mount.ntfs       pvchange       start_udev  weak-modules
e2fsck           initctl       lvmchange     mount.ntfs-3g    pvck           status      ypbind
e2image          initlog       lvmdiskscan   mount.ntfs-fuse  pvcreate       stop
[liveuser@localhost sbin]$ pwd
/sbin
[liveuser@localhost sbin]$ cd ../selinux
[liveuser@localhost selinux]$ ls
access    checkreqprot          compat_net  deny_unknown  initial_contexts  mls                   policyvers     user
avc       class                 context     disable       load              null                  reject_unknown
booleans  commit_pending_bools  create      enforce       member            policy_capabilities   relabel
[liveuser@localhost selinux]$ 
```

Figure 5. An end-user run-time version of the selected alternative components that fulfills the OA C2SC system design.

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 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).

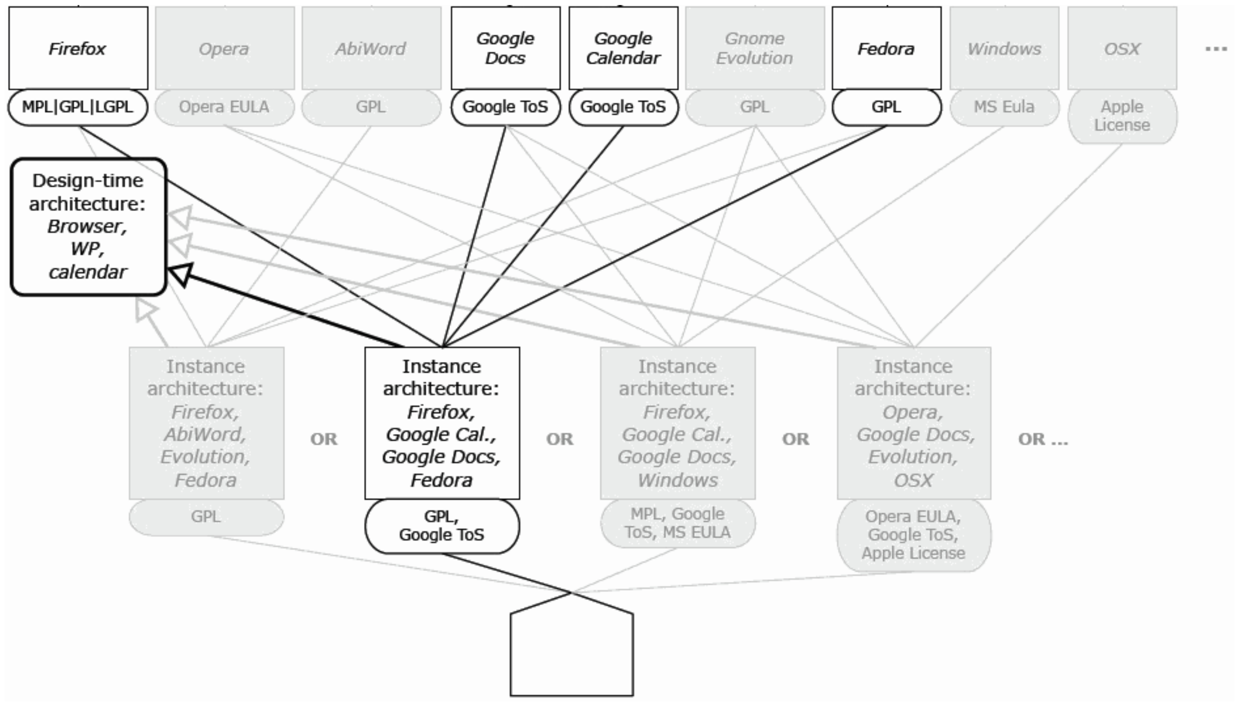
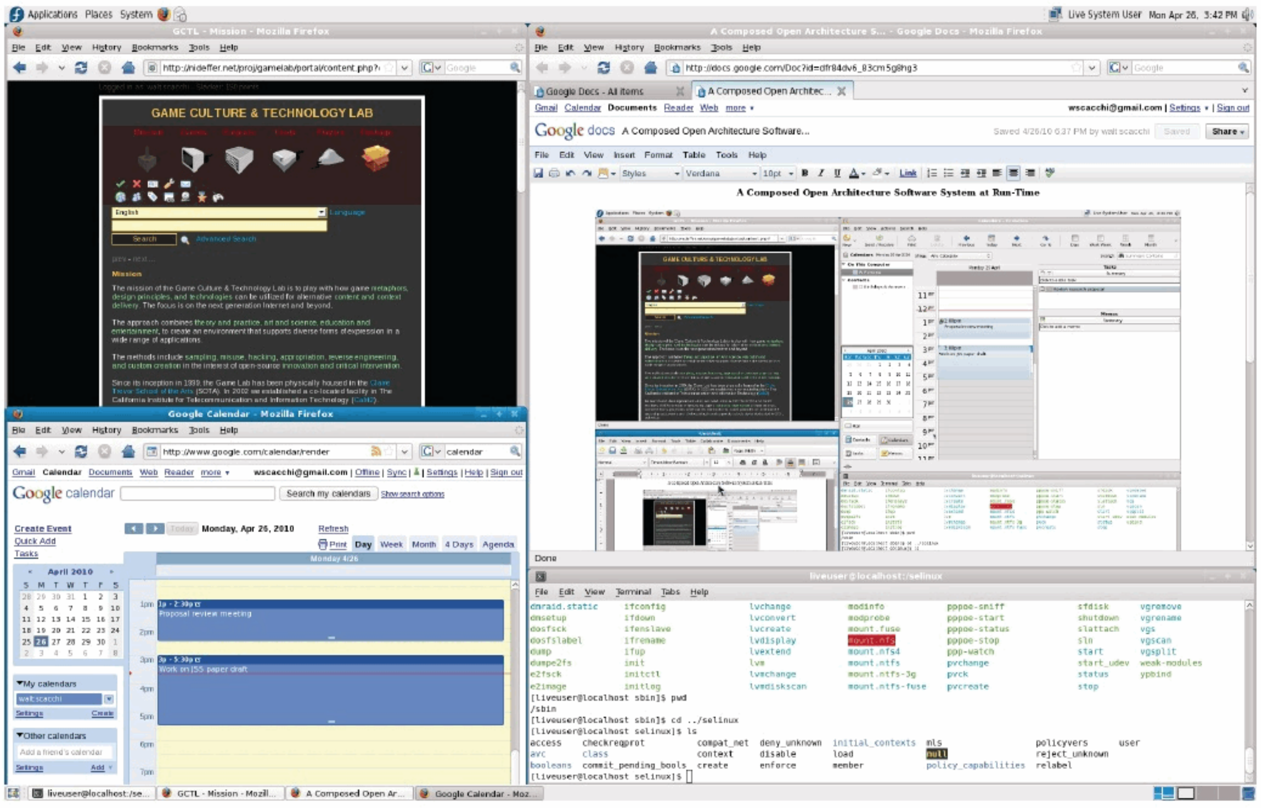

Firefox
Opera
AbiWord
Google Docs
Google Calendar
Gnome Evolution
Fedora
Windows
OSX
...
MPL|GPL|LGPL
Opera EULA
GPL
Google ToS
Google ToS
GPL
GPL
MS Eula
Apple License
Design-time architecture: Browser, WP, calendar
Instance architecture: Firefox, AbiWord, Evolution, Fedora
OR
Instance architecture: Firefox, Google Cal., Google Docs, Fedora
OR
Instance architecture: Firefox, Google Cal., Google Docs, Windows
OR
Instance architecture: Opera, Google Docs, Evolution, OSX
OR ...
GPL
GPL, Google ToS
MPL, Google ToS, MS EULA
Opera EULA, Google ToS, Apple License

Figure 6. A second configuration of the OA C2SC example, using alternative but functionally similar components.

This ability to rapidly and conveniently substitute components (agility) to adapt to different end-user needs or required components aligns with the MPE/AAE elements. However, we still need to address how other elements in the shared agreements, like IP licenses, enable/constrain whether such an alternative configuration, though technical possible, meets other organizational requirements. This concerns raises the following kinds of questions:

- Who in the ecosystem of human actors (the multiple parties) for this system has the right to make the decisions to use a remote/networked computational service in place of a locally hosted software system 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 through explicit acquisition policies that stipulate desired rights and acceptable obligations by the acquiring organization, where such policies are important to system acquisition officers, just as IP licenses do for IP rights and obligations important to software producers. Our shared agreements need to provide guidance for what to do here.
- When can these decisions be made? In traditional development processes these would occur at design time, but in the larger view we examine here such decisions, or rather the shared agreements that control them, are perhaps more properly considered at component, C2SC, or system acquisition time. As we will see below, it is also possible that in order to achieve specific security qualities they might be made at system integration time or at deployment time, in response to specific end-user organization needs.



```
dmraid.static   ifconfig      lvchange     modinfo          pppoe-sniff   sfdisk      vgremove
dmsetup         ifdown        lvconvert    modprobe         pppoe-start   shutdown    vgrename
dosfsck         ifenslave     lvcreate     mount.fuse       pppoe-status  slattach    vgs
dosfslabel      ifrename      lvdisplay    mount.nfs        pppoe-stop    sln         vgscan
dump            ifup          lvextend     mount.nfs4       ppp-watch     start       vgsplit
dumpe2fs        init          lvm          mount.ntfs       pvchange      start_udev  weak-modules
e2fsck          initctl       lvmchange    mount.ntfs-3g    pvck          status      ypbind
e2image         initlog       lvmdiskscan  mount.ntfs-fuse  pvcreate      stop
[liveuser@localhost sbin]$ pwd
/sbin
[liveuser@localhost sbin]$ cd ../selinux
[liveuser@localhost selinux]$ ls
access    checkreqprot          compat_net  deny_unknown  initial_contexts  mls                   policyvers      user
avc       class                 context     disable       load              null                  reject_unknown
booleans  commit_pending_bools  create      enforce       member            policy_capabilities   relabel
[liveuser@localhost selinux]$ 
```

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 displayed in Figure 5 and Figure 7 specify specific alternatives for the major components, for example, Mozilla 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 18 or Firefox 19, 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 18 rather than version 19? 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? or in response to a particular security attack on the previous configuration?

Finally, an orthogonal consideration is the use of software-based access control 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 computer's operating system.

For security, the Fedora operating system can employ the SELinux capabilities to restrict all shell/operating systems commands through mandatory access control and type enforcement, 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 (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) [Xen12]. So it is possible for shared agreements to call for the use of multiple software-based security mechanisms to protect a OA system or C2SC, while still accommodating the MPE/AAE elements. This is an important accomplishment.

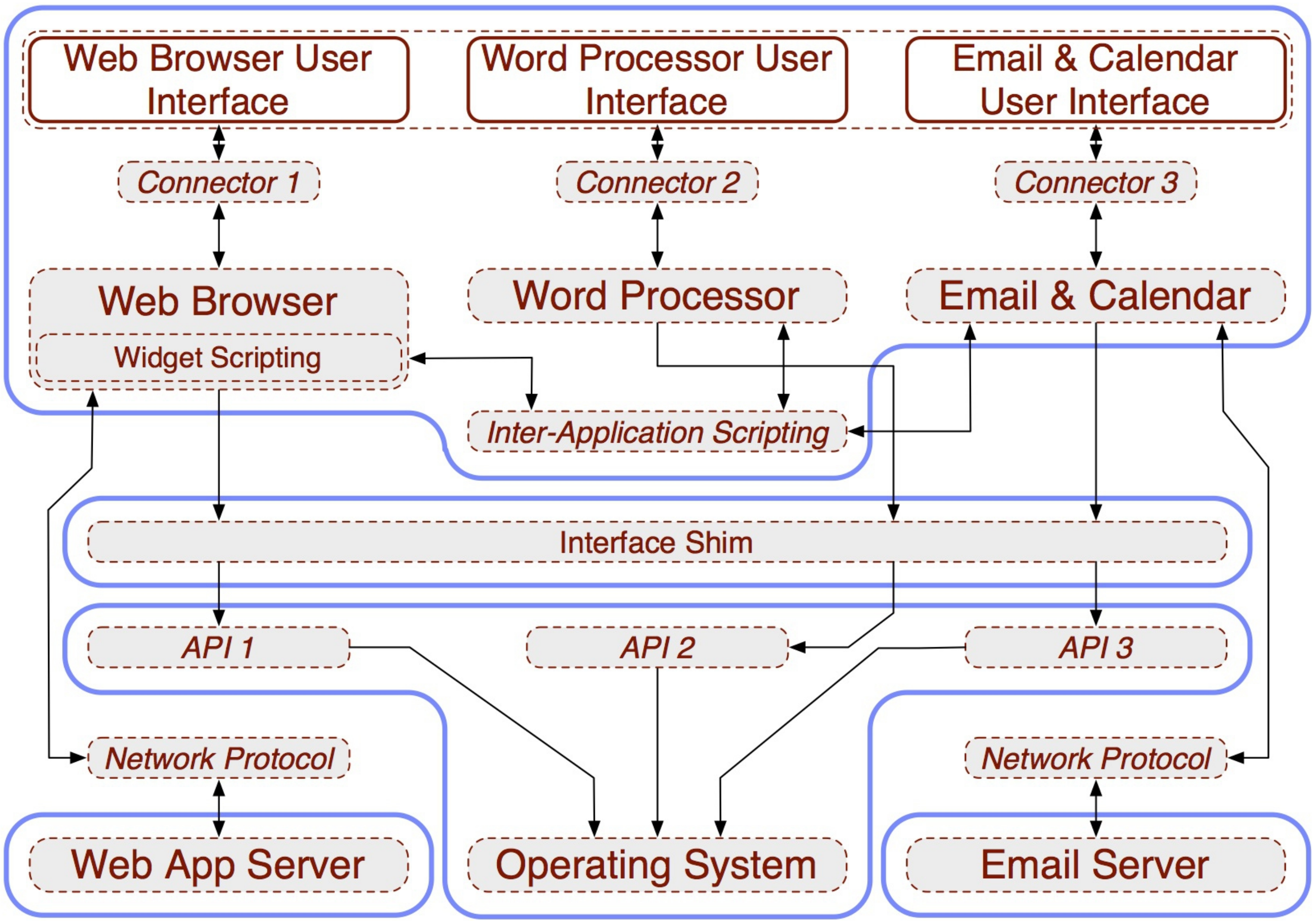


Figure 8. A secure OA system configuration alternative that encapsulates multiple system capabilities within different virtual machines (e.g., using [Xen12]), where each system capability may be under the purview of a different organizational authority.

Decentralized C2 system architecture

Decentralized C2 systems can employ OA that accommodate their deployment and usage in degraded and limited operation environments. Decentralized C2 systems can have a very small physical footprint, and mission planners/commanders may be located potentially anywhere in the world. So decentralized OA C2 systems can serve as appropriate candidates for experimentation or training in underdeveloped, degraded, or denied operational environments.

In our previous work, we have investigated and prototyped a C2 system called DECENT [SBN12] that provides an immersive 3D virtual world for experimenting with decentralized C2 activities. DECENT is designed to run in a wide-area network environment that supports 3D browsers (or conventional Web browsers with 3D world viewer plug-ins) as clients, and remote servers to provide content and other services accessible through the browsers. DECENT operates on the Internet, but not on the Web, relying instead on a separate decentralized network grid architecture called the *HyperGrid* [Lop11]. The HyperGrid infrastructure in turn allows for the establishment and operation of distinct or interconnected hypergrids as separate administrative authorities for research, experimentation, or training applications. The MOSES hypergrid is such an example [Max12, Mos13].

The Ozone Widget Framework [OWF13] supports a different form of decentralized OA. OWF provides support for the development of Web-compatible widgets as lightweight applications that access pre-specified kinds of content from remote servers. This allows the creation of virtual private networks offering Web-like applications and services through network-managed security capabilities. However, in an underdeveloped operational environment, such networking capabilities may not be available or reliable, so other means must be utilized to realize secure communications. Alternatively, applications or widgets that rely on signals from known types of sensors may also not be available or reliable, so other widget versions may be need to be provided.

Subsequently, one strategy for developing a decentralized C2 system would combine the OA for a centralized C2 system, but allow for the system components (applications, widgets, content servers, operating systems, etc.) or C2SCs to be interconnected to local/remote servers through encrypted network connectors (e.g., secure data communication protocols like HTTPS or TLS, or robust dynamic capability connectors, like those used in the COAST [GST12]) that enable data, control, or signals to flow across (or tunnel beneath) virtual software defined networks. In other words, the user interface would still rely on a Web browser modality, yet in a simple implementation, be able to securely access hypergrid worlds in one window, with other widget-based content services located in other browser window or user sessions, depending on overall security policy. So we have a basis for developing a secure decentralized C2 system that allows a consistent OA system model whose components or connectors may be centralized or decentralized by design choice or security policy. However, compiling and deploying such a decentralized C2 system are the system development activities when the security components will be integrated.

Overall, a decentralized C2 system can be developed using system elements (components, connectors, or embedded C2SCs) that may reside on local computer or remote networked computers that are accessed through software client applications that may reside within a C2 virtual world. Such choices may be most appropriate for OA C2 systems that are intended to support experimentation in C2 mission planning activities. Such activities may be most relevant where underdeveloped or degraded system elements are employed, in order to help train mission commanders to learn how to articulate their requirements for new system components that can be rapidly developed, integrated and deployed. It also can serve to reveal other practical advantages and constraints that arise when end-user organizations follow the MPE/AAE guidelines for adapting and evolving their OA C2 systems.

Conclusions

In this paper, we identified and investigated technical and acquisition challenges that arise during the development and evolution of secure Open Architecture (OA) command and control (C2) systems. OA systems are identified as those whose software system components and interconnection mechanisms are either proprietary closed source software offerings with open application program interfaces, open source software, or some architectural configuration of closed and open source elements. Secure OA systems are identified as those where the security of individual software elements may be uncertain, because of the ongoing evolution, poorly understood system integration compromises, or obtrusive software intellectual property licenses, yet where overall OA security must be continuously assured.

We presented a framework that organizes OA system security elements and mechanisms in forms that can be aligned with different stages of the life cycle of C2 system development. We focused attention to the design of OA C2 systems or C2 system capabilities using commonly available software components that provide office productivity capabilities that support C2 operations.. We provided a case study to show our scheme and how it can be applied to C2 system architectures that rely on an OA. Finally, we showed how our efforts complement and extend the agile C2 framework that utilizes a new generation of software components and security mechanisms that are engineered/adapted by multiple parties and disseminated within a diverse marketplace ecosystem of software producers, integrators, and consumers.

Acknowledgements

The research described in this report was supported by grant #N00244-12-1-0004 from the Acquisition Research Program at the Naval Postgraduate School, Monterey, CA, and by grant #1256593 from the National Science Foundation. No endorsement, review, or approval implied. This paper reflects the views and opinions of the authors, and not necessarily the views or positions of any other group, enterprise, or government agency.

References

[ACC13] Acquisition Community Connection, *Naval Open Architecture and DoD Open Systems Architecture*. https://acc.dau.mil/oa , accessed 1 February 2013.

[ASA10] Alspaugh, T.A., Scacchi, W., and Asuncion, (2010. H. Software licenses in context: The challenge of heterogeneously-licensed systems. *Jour. Assoc. Information Systems*, 11(11), 730-755.

[BCK03] Bass, L., Clements, P., and Kazman, R., (2003). *Software Architecture in Practice*, 2nd Edition, Addison-Wesley Professional, New York.

[Fel07] K. Feldt, K. (2007). *Programming Firefox: Building Rich Internet Applications with XUL*. O'Reilly Media, Inc., Sebastopol, CA.

[Gar11] Garcia, P. (2011). Command and Control Rapid Prototyping Continuum (C2RPC): The Framework for Achieving a New C2 Strategy, *16th. Intern. Command and Control Research and Technology Symposium* (ICCRTS), Paper-033, Fairfax, VA, June 2011.

[GeW12] Gerschefske, K. and Witmer, J. (2012). Wired Widgets: Agile Visualization of Space

Situational Awareness, *Advanced Maui Optial and Space Surveillance Technologies Conf.* (AMOS'12), http://www.amostech.com/TechnicalPapers/2012/POSTER/GERSCHEFSKE.pdf

[Giz11] Gizzi, N. (2011). Command and Control Rapid Prototyping Continuum (C2RPC) Transition: Bridging the Valley of Death, *Proceedings 8th. Annual Acquisition Research Symposium*, Vol. 1, Naval Postgraduate School, Monterey, CA.

[GST12] Gorlick, M.M., Strasser, K. and Taylor, R.N. (2012). COAST: An architectural style for decentralized on-demand tailored services. In *Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture (WICSA-ECSA '12)*, 71–80.

[Lop11] Lopes, C.V., (2011). Hypergrid: Architecture and Protocol for Virtual World Interoperability, *IEEE Internet Computing*, 15(5), 22-29, Sept-Oct 2011.

[Max12] Maxwell, D. (2012). Military Grid Hits Performance Goals, *Hypergrid Business*, 26 November 2012. http://www.hypergridbusiness.com/2012/11/military-grid-hits-performance-goals/ , accessed 3 February 2013.

[MeO01] Meyers, B.C. and Oberndorf, P. (2001). *Managing Software Acquisition: Open Systems and COTS Products*. Addison-Wesley Professional, 2001.

[Mos13] MOSES, (2013), *Military Open Simulator Enterprise Strategy*, http://brokentablet.arl.army.mil/ accessed 4 February 2013.

[RBC12] Reed, H., Benito, P., Collens, J., and Stein, F. (2012). Supporting Agile C2 with an Agile and Adaptive IT Ecosystem, *17th. Intern. Command and Control Research and Technology Symposium* (ICCRTS), Paper-044, Fairfax, VA, June 2012.

[OPM03] Ovaska, P., Rossi, M., and Marttiin, P. (2003). Architecture as a Coordination Tool in Multi-Site Software Development. *Software Process—Improvement and Practice*, 8(4), 233-247.

[OWF13] Ozone Widget Framework (2013). *Ozone Widget Framework Readme*, https://github.com/ozoneplatform/owf/blob/master/README.md , accessed 4 February 2013.

[ScA12a] Scacchi, W. and Alspaugh, T. (2012a). Addressing the Challenges in the Acquisition of Secure Systems with Open Architectures, *Proc. 9th. Annual Acquisition Research Symposium*, Naval Postgraduate School, Monterey, CA, Volume 1, May 2012.

[ScA12b] Scacchi, W. and Alspaugh, T. (2012b). Understanding the Role of Licenses and Evolution in Open Architecture Software Ecosystems, *Journal of Systems and Software*, 85(7), 1479-1494, July 2012.

[ScA12c] Scacchi, W. and Alspaugh, T. (2012c). Developing Secure Systems using Open Architectures with Open Source and Closed Source Components, *Proc. 8th. IFIP International Conf. Open Source Systems*, IFIP Advances in Information and Communications Technology, Vol. 378, 144-159, Hammamet, Tunisia, September 2012.

[SBN12] Scacchi, W., Brown, C. and Nies, K. (2012). Exploring the Potential of Virtual Worlds for Decentralized Command and Control, *17th. Intern. Command and Control Research and Technology Symposium* (ICCRTS), Paper-096, Fairfax, VA, June 2012.

[SWM11] Soylum A., Wild, F., Modritscher, *et al*. (2011). Mashups and Widget Orchestration, *Proc. Intern. Conf. on Management of Emergent Digital EcoSystems* (MEDES'11), 226-234.

[TMD09] Taylor, R.N., Medvidovich, N. and Dashofy, E. (2009). Software Architecture: Foundations, Theory, and Practice, Wiley, New York.
[Xen12] Xen.org, *What is the Xen Hypervisor?* http://xen.org/products/xenhyp.html , accessed 1 November 2012.