System-Aware Cyber Security

A Dissertation

Presented to

the faculty of the School of Engineering and Applied Science

University of Virginia

in partial fulfillment

of the requirements for the degree

Doctor of Philosophy

by

Rick A. Jones

December

2012

APPROVAL SHEET

The dissertation

is submitted in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

Rick A. Jones

AUTHOR

The dissertation has been read and approved by the examining committee:

Barry Horowitz

Advisor

Peter Beling

Stephen Patek

Alfredo Garcia

Barry Johnson

Kevin Skadron

Accepted for the School of Engineering and Applied Science:

[illegible]

Dean, School of Engineering and Applied Science

December
2012

Abstract

As exemplified in the 2010 Stuxnet attack on an Iranian nuclear facility, attackers have the capabilities to embed infections in equipment that is employed in nuclear power systems. In this thesis, a new systems engineering focused approach for mitigating such risks is described. This approach involves the development of a security architectural formulation that integrates a set of reusable security services as an architectural solution that is an embedded component of the system to be protected. The System-Aware architectural approach embeds security components into the system to be protected. The architecture includes services that (1) collect and assess real-time security relevant measurements from the system being protected, (2) perform security analysis on those measurements, and (3) execute system security control actions as required. This architectural formulation results in a defense that is referred to as System-Aware Cyber Security. This includes (1) the integration of a diverse set of dynamically interchangeable redundant subsystems involving hardware and software components provided from multiple vendors to significantly increase the difficulty for adversaries by avoiding a monoculture environment, (2) the development of subsystems that are capable of rapidly changing their attack surface through hardware and software reconfiguration (configuration hopping) in response to perceived threats, (3) data consistency checking services (e.g., intelligent voting mechanisms) for isolating faults and permitting moving surface control actions to avoid operations in a compromised configuration, and (4) forensic analysis techniques for rapid post-attack categorization of whether a given fault is more likely the result of an infected embedded hardware or software component (i.e., cyber attack) or a natural failure. In this thesis we present these key elements of the System-Aware Cyber Security architecture and show, including an application example, how they can be integrated to mitigate the risks of insider and supply chain attacks. In addition, this thesis outlines an initial vision for a security analysis framework to compare alternative System-Aware security architectures. Finally, we *summarize future research that is necessary to facilitate implementation across additional domains critical to the nation's interest.*

Contents

List of Figures

Chapter 1
Introduction

1.1 Motivation

Increasingly systems are being digitized in a drive toward lower costs, improved efficiency, and reduced time to market. However, this drive to digitization also renders these systems susceptible to an increasingly sophisticated and debilitating array of cyber attacks. For example, as shown by Karl Koscher et al. [2010], it is possible for an attacker to embed an infection that is capable of completely disabling an automobile's braking system. Further, in the 2010 Stuxnet attack [Falliere, Murchu, and Chien, 2011], an embedded infection was used to successfully damage up to 1000 centrifuges in Iran (10 percent of the available capacity) [Albright, Brannan and Walrond, 2010]. Such attacks cannot be completely addressed by traditional perimeter security solutions [Wulf and Jones, 2009], as they have been in the past. A new systems engineering focused approach is introduced, integrating fault-tolerant system design concepts with advanced cyber security concepts and the methods developed by the automatic control systems community to address these expanding threats. This involves the development of a security architectural formulation [Bayuk and Horowitz, 2011] based on reusable system security services to create a defense that is referred to as System-Aware Cyber Security [Jones and Horowitz, 2011].

Security services are defined to be security elements that are integrated and embedded as a solution into a system, providing unique security functionality designed and tailored for the specific application. The architecture includes services that (1) collect and assess real-time security relevant measurements from the system being protected, (2) perform security analysis on those measurements, and (3) execute system security control actions as required. These services include (1) significantly increasing the difficulty for adversaries by avoiding a monoculture environment through the integration of a diverse set of redundant subsystems involving hardware and software components provided by multiple vendors, (2) the development of subsystems that are capable of rapidly changing their attack surface through hardware and software reconfiguration (configuration hopping) in response to perceived threats, (3) data consistency checking services for isolating faults and permitting moving surface control actions to avoid continuing operations in a compromised configuration, and (4) forensic analysis techniques for rapid post-attack categorization of whether a given fault is more likely the result of a cyber attack than other causes (i.e., natural failure).

Integrated solutions would be determined through awareness of what the system applications do, how they are designed, what they communicate with as well as how they communicate, what their performance requirements are, what missions they are tied to, what risks are posed through potential attacks, etc. In addition, System-Aware Cyber Security provides the means to hypothesize specific threats in relation to specific application functions. When combined with an understanding of the damage that could occur, this provokes the application of risk sensitive mitigation solutions (i.e., appropriate system security services). This enables systems owners, operators, and regulators to directly link the risk mitigation benefits of specific security services to the cost associated with designing, implementing, and utilizing them. When designing and implementing security solutions, several high level design issues emerge, including (1) selection of the subsystems for redundant diversification; (2) use of moving target solutions for attack avoidance, attack detection, and system restoral functions; (3) selection of which HW/SW components to protect; (4) selection of virtual and/or physical configuration hopping solutions; (5) selection of regimes for physical hopping (local and/or remote); (6) selection of data used to ensure consistency; (7) selection of forensic analysis techniques for rapid categorization of faults; (8) avoidance of interference with normal functioning of the applications; (9) assurance of appropriate isolation of the security solutions; (10) exploitation of opportunities for reuse of existing security solution services; and (11) establishment of administration requirements for control of the security solutions.

System-Aware Cyber Security enhances the security of the system to be protected through the use of security services embedded at the application layer to significantly increasing the level of effort required—i.e., complexity, time, energy, and budget needed to design, develop, and deploy exploits—by an adversary to compromise the system being protected; thereby giving an asymmetric advantage to the defender. This increased level of effort is relative to both the amount of resources necessary to design, develop, and deploy and exploit in the absence of a System-Aware solution and the additional resources necessary to create the System-Aware solution. For example, say that a given System-Aware solution cost $10,000 to create and will increase the cost to develop and exploit by $1,000 dollars. This solution would significantly shift the asymmetry from the attacker to the defender if the exploits defended against had previously been inexpensive (e.g., $100) and the system being protected was critical (e.g., missile defense system). However, the solution would only offer a minimal shift if the system being protected was simple (e.g., a personal computer worth $100) and the exploits being protected against had previously been costly to develop (e.g., $1,000,000).

As System-Aware Cyber Security aims to enhance system security by increasing the complexity and resources necessary to design, develop, and execute successful attacks, it can be difficult to assess using more analytical approaches. For example, the integration of System-Aware security solutions can introduce collateral impacts. These impacts may require additional solutions in order to mitigate their effects on system performance, thereby complicating the overall design of the system to be protected. However, these techniques may also potentially increase the complexity to design, develop, and execute attacks. Is the increased security worth the increased complexity in the system to be protected? This is a decision that will be dependent on the community in question. For example, increased complexity in running a nuclear power plant may be viewed as introducing more risk than the benefits afforded by a given System-Aware solution, due to the significant loss that could result from a malfunctioning reactor. Alternatively, increased security in protecting a ships control systems may be viewed as more important than the increased complexity, due to the potential for an adversary to compromise the ship. In addition, it is difficult to quantify complexity and security. As a result it is also left up to each community to determine which solutions are too complex for the security offered. For example, nuclear power community may decide that the application of moving target solutions for information assurance is too complex. Alternatively the administrator of a cloud computing infrastructure may decide moving target solutions offer minimal complexity compared to the potentially large security gains.

1.2 Background

1.2.1 Information Assurance

It is recognized that perimeter security is the mainstay of the current cyber security solution space [Wulf and Jones, 2009]. This has enabled the system engineering and security communities to respond to perceived risks and threats through the addition of new perimeter security solutions on a responsive basis. The most recognizable of these solutions being the firewall, a device utilized in network security to control access to resources. Another example of perimeter security is utilized to ensure the integrity of the supply chain against insider attacks. In this case interviews, background checks, and constrained purchasing from selected suppliers are used to help prevent individuals with malicious intent from gaining access to resources, facilities, information, etc. In all of these cases the goal of perimeter security is to strictly control access to and from key components.

However, while the focus on perimeter security has provided some advantages, it has also brought with it several disadvantages; disadvantages that have become more significant as the cyber threat has evolved. In particular, there has not been widespread development and application of reusable solutions embedded into the system to be protected. The rising threat of successful attacks warrants the consideration of a top-down systems engineering approach that develops solution strategies regarding cyber security that go beyond the perimeter model. Furthermore, the systems engineering community could be considering tightly coupled system security solutions, starting from the time that a new system architecture is developed and continuing through its entire life cycle. For example, the credit card community has implemented a wide range of techniques to protect against fraud [Kou et al., 2004]. Several of these techniques rely on solutions that are tightly coupled to user behavior. One such technique relies upon the knowledge that it is not possible for someone to spend money in two places at the same time. However, while significant attention has been paid to best practices for dealing with the bottom-up approach for engineering security solutions, including Webber et al. “Applications That Participate in Their Own Defense” [2003] and Cai et al. “Honeygames” [2006], the systems engineering community has yet to develop a corresponding architectural framework for a top-down approach for addressing cyber security. For example, in 1980 the United State’s intercontinental ballistic missile warning system falsely indicated to its operators that a full-scale nuclear attack had been initiated against the U.S. [US Comptroller General, 1981]. After-the-fact analysis revealed that the system was not designed to recognize a condition where there was no missile-related data being received by the system’s sensors but, at the same time, there was data indicating an attack on the screens being observed by operators. Although this event was the result of malfunctioning hardware, it could have just as well been a Trojan horse inserted through the supply chain [Defense Science Board, 2005; DoD, 2009].

1.2.2 Fault Tolerance

Fault-tolerant design enables a system to continue operation, possibly at a reduced level (also known as graceful degradation), rather than failing completely, when some part of the system fails (i.e., unintentional failure). This has resulted in the development of techniques for preventing, isolating, detecting, and restoring systems from faults, including diversity [Avizienis and Kelly, 1984], redundancy, and reliable voting techniques (e.g., Byzantine voting [Lamport, Shostak, and Pease; 1982]). These techniques can also be utilized to prevent or recover from a fault caused by an embedded Trojan horse [Reynolds et al., 2002]. However, the application of fault-tolerant designs to System-Aware Cyber Security poses unique challenges. First, unlike an unintentional failure, an intelligent adversary can actively seek out and take advantage of beneficial asymmetries (i.e., attacks that require exploits that are relatively easy to develop and execute). Thus, rather than simply making a system more resilient, System-Aware security solutions attempt to shift the asymmetry in favor of the defender (i.e., create defenses that are relatively easy to develop, but that make exploit development and execution much more difficult). In addition, analysis efforts traditionally applied in fault-tolerant system design activities must be modified to address cyber security. For example, an intelligent adversary may execute an exploit that requires multiple actions related to multiple sub-systems or system components. Such actions can be viewed as a series of dependent failure events from the perspective of System-Aware security, but as a sequence of independent failure events from a fault-tolerant perspective. For example, the Stuxnet attack damaged centrifuges while masking this damage from the operators of those centrifuges. While such an outcome may also occur as the result of multiple unintentional failures, it would require several such failures to occur independently and concurrently: damaging commands to be sent to the centrifuges, faulty information to be sent to control room operators and automated status monitoring systems, and for that status information to indicate the system is running normally. Thus, from the System-Aware security perspective, such a sequence of events is dependent on the capabilities of an intelligent adversary, while from the fault-tolerant perspective such an outcome is based only on the likelihood of such a sequence of events occurring. This is further emphasized by the fact that an attacker can design exploits that operate only under specific conditions in order to make an attack more difficult to detect. For example, only having an exploit active during a specific time of day or within a specific geographic region. From the perspective of fault-tolerance, such attacks may be viewed as transient faults. Thus, once the component(s) have stopped generating faulty information, they are reintegrated into the system. In contrast, if such faults are the result of a malicious attack, such reintegration may be undesirable. Furthermore, there may be significant ramifications [US Department of Defense, 2011]. Thus, the consequences of false alarms can differ significantly as well.

1.2.3 Automatic Control Systems

Automatic control is the application of control theory for regulation of processes without direct human intervention. System-Aware Cyber Security leverages the methods of this community to provide both automatic restoration from cyber attacks as well as to aid in the detection and deflection of cyber attacks. However, two unique challenges must be addressed in order to apply these techniques to System-Aware security. First, is that reconfiguration must be sensitive to an intelligent adversary. When the system is automatically restored from an unintentional fault [Lin and Chin, 1998] it is not expected that the fault will dynamically adjust. In contrast, when restoring the system from a malicious adversary it is possible that the adversary will continue to actively try and compromise the system. This results in the need to ensure that an automatic restoration from a cyber attack reconfigures the system to a new uncompromised configuration. In addition, as the adversary can actively work to compromise this new configuration, it may be desirable to continue to reconfigure the system in order to prevent the system from being compromised again. This can potentially result in a higher rate of reconfiguration. In addition, as discussed by Horowitz and Pierce [Horowitz and Pierce, 2012] in order to declare that a cyber attack has occurred, there must be a significant enough difference between damaging commands and normal operations. As discussed in section 1.2.2, this is made all the more important due to the differences in responding to a cyber attack then an unintentional fault. This difference in responding also carries through to how reconfiguration happens. For example, when an unintentional fault occurs it is desirable to take action to correct the fault. In contrast, if a cyber attack is detected, it may be desirable to automatically reconfigure; however, it may also be desirable to allow the attack to continue. For example, if it is suspected that an attack has been initiated by an insider, then immediate reconfiguration could alert the insider that she has failed. The insider may then initiate steps to try and cover her tracks. Thus, it might be desirable to delay reconfiguration to allow time to track down said insider.

1.3 System-Aware Solution

System-Aware Cyber Security is aimed at protecting systems whose perimeters have been breached, by providing additional cyber security capabilities to deter possible attackers, detect when the system has been compromised, isolate the sub-systems that have been compromised, and restore the system to an uncompromised state. This makes System-Aware Cyber Security capable of potentially enhancing the overall security of a system over a wide range of threats. In particular, this makes System-Aware Cyber Security well suited to addressing cyber attacks stemming from the supply chain and insiders:

- *Supply chain attack scenario* – A supply chain attack occurs when the hardware or software that composes a system has been embedded with a malicious defect designed to prevent or hinder that system from being able to carry out its stated objectives. In this attack scenario, these components can be compromised at any point before they are integrated into the system. Such attacks can range in complexity from taking over entire sub-systems to simply changing a few critical parameters. In addition, such attacks may be triggered by a variety of mechanisms, including, time, system state, or remote activation. Figure 1 presents a sample supply chain attack scenario for a simplified turbine control systems. For this example, the turbine controller has been embedded with a Trojan horse that simply sends misleading information about the state of the system to the human operators in the main control room and the automated monitoring functions (Health Status Station in Figure 1). This is done to cause inappropriate action to be taken that could potentially result in significant damage to the turbine as well as economic loss.

- *Insider threat* – All systems need to be managed and maintained to ensure both availability and proper functionality. However, this often requires granting the personnel that manage and maintain such systems elevated privileges; privileges that can also be used to deliberately tamper with the hardware or software, inject malicious code into the system, or grant access to unauthorized and external users. An insider attack occurs when a user is able to use their privileges to compromise the security of the system they are managing and maintaining. This compromise can either be triggered immediately or lie dormant until a specific triggering condition is reached; e.g., receiving a specific command, reaching a specific date and time, or the system's entering a particular state. For example, the supply chain attack scenario shown in Figure 1 could have been embedded by an operator at the nuclear power plant. To avoid getting caught the operator might have configured the embedded Trojan horse to only activate after she had left the facility.

As discussed in section 1.2, part of securing a system against supply chain and insider attacks is through a variation of perimeter security solutions. This includes interviews, background checks, purchasing from a limited set of suppliers, and performing analysis on the supplier-provided hardware and software. However, it is not always possible to implement these measures; sometimes system components may need to be purchased from less trusted suppliers (e.g., foreign suppliers) due to a variety of factors, including costs and availability. Furthermore, specific mission objectives may require that systems constructed utilizing rigorous security and oversight may need to be connected with systems constructed with little to no security and oversight. In addition, as these attacks can potentially cause serious damage through very small changes (e.g., changing a single parameter) they can be extremely difficult to detect. Finally, once these systems have been successfully compromised, such methods do not afford any means for restoring the system to an uncompromised state.

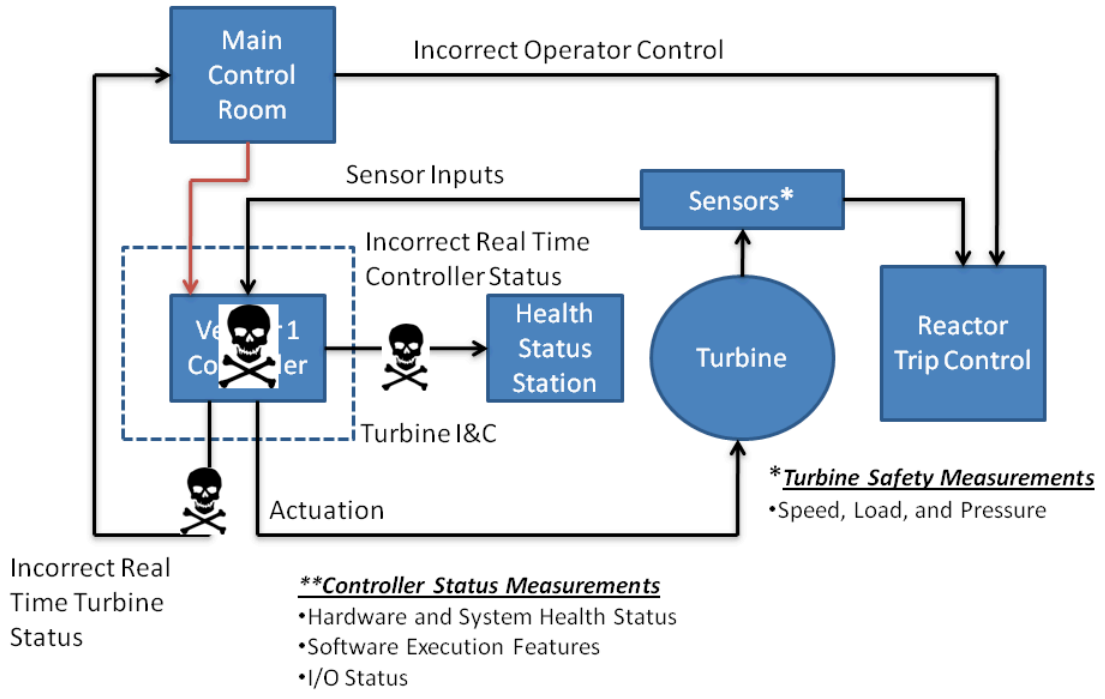

Main Control Room
Incorrect Operator Control
Sensor Inputs
Sensors*
Incorrect Real Time Controller Status
Ve 1 Co ler
Health Status Station
Turbine
Reactor Trip Control
Turbine I&C
*Turbine Safety Measurements
•Speed, Load, and Pressure
Actuation
Incorrect Real Time Turbine Status
**Controller Status Measurements
•Hardware and System Health Status
•Software Execution Features
•I/O Status

Figure 1. Illustration of a hypothetical supply chain based cyber attack on a turbine control system. The controller is embedded with a Trojan horse that can modify, nullify, or replace information sent to the operators in the Main Control Room and the automatic monitoring station. The intended consequence of the attack would be to have the operator(s) believe that the turbine is operating sufficiently outside of specification so as to warrant a procedural action (e.g., shutdown of the turbine), which would induce a plant trip.

The remainder of this section introduces the System-Aware Cyber Security Architecture used to protect systems whose perimeter has been breached, including measurement features for attack detection and tactical forensic analysis, decision functions for system control, and signaling functions for managing system restoration and configuration hopping. These capabilities can be integrated in a variety of ways so as to provide several features for enhancing the security of a system. These features can be designed to deter attackers from exploiting a system; avoid attempts to compromise a system; identify when a system has been compromised, prevent the system from being damaged, isolate the compromised components, and enable restoration of the system to a non-compromised state; and enable operators and administrators to confirm that an event has been caused by a cyber attack and to take the appropriate action(s). Such security solutions will generally require system-specific custom design, resulting in higher costs than off-the-shelf security solutions. This situation calls for architectures that are focused on protecting what are considered to be the most critical system functions. As a result, the combination of perimeter security solutions that aims to protect an entire system combined with System-Aware solutions that provide a second-layer of protection for critical system functions. Several such security features are described below, and three examples integrating these features are outlined in Chapter 3. In addition, several of these security features have been composed into reusable design patterns (see Chapter 2). These design patterns are intended to facilitate in the development of System-Aware solutions across a wide range of domains. Furthermore, it provides a method for documenting and sharing new System-Aware solutions as they are created. Finally, the repository can be continually updated and enhanced as these design patterns are utilized to develop System-Aware architectures.

1.3.1 Diversity

Diverse hardware/software system implementations enhance security by creating distinct, dynamically interchangeable redundant functions in a system (thereby avoiding a monoculture environment). While added redundancy potentially provides additional opportunities for attackers to exploit specific implementations, together, the combination of diversely redundant subsystems potentially deters attackers by increasing the necessary effort that is required to compromise all of the implementations. There are many variations of diversity, each providing a different form of security. For example, having a subsystem assembled by two different vendors makes it more difficult for an attacker to inject a hardware or software Trojan horse during assembly, by forcing the attacker to have insiders at two different companies. As another example, utilizing multiple operating systems can complicate an attacker's activities by requiring the utilization of multiple exploits: one for each operating system. The amount of security gained is directly dependent on the amount and form of diversity integrated into the system. However, while diversity reduces the risk of an attacker being able to compromise a system, as a biproduct, it complicates the design and maintenance for the system. This requires the systems engineer to conduct the necessary tradeoff assessments regarding how and where to best apply diversity.

Diversity can be used in conjunction with configuration hopping (section 1.3.2) and data consistency checking (section 1.3.3) to facilitate in the restoration of a compromised system. Specifically, by having multiple diverse redundant components, a system can be restored to a different configuration than the one that it was in during the time it was compromised. This prevents the attacker from repeating the same exploit and increases the difficulty of completely bringing down the system.

1.3.2 Configuration Hopping

Configuration hopping is a security service that, on a randomized basis within scheduled intervals, enables the dynamic modification of an overall system configuration. This is accomplished through interchanging the modes of operation among diversely implemented redundant components while executing their specified system functions. Interchanges can be accomplished virtually across multiple operating systems, as well as physically across machines that can be either co-located or located over a geographic region (e.g., such as can be employed in cloud computing [Vaquero et al., 2009]). This dynamic interchange provides defense by forcing an attacker to operate within time interval constraints while using a family of coordinated exploits that addresses the complications introduced through diversity. In order to provide this security service, an interchange capability must be developed that does not unacceptably degrade desired system operation. For example, the rate of hopping may have to be constrained due to specific system characteristics that may demand stability. The selection of the specific subsystems to interchange would be determined based upon security and economic considerations.

To offer configuration hopping as a security service to users, two capabilities are needed: tools to configure selected application functions for interchange and tools for controlling interchange subject to specified design criteria. Solutions for a specific system will be derived from (1) the unique system design attributes and estimates of the significance of time constraints on complicating attacker exploits, (2) mission objectives, (3) resource requirements related to making interchanges, (4) system performance costs, and (5) the security risks surrounding the system.

1.3.3 Data Consistency Checking

Data consistency checking is a service that, for the purposes of data integrity, compares data at different points in a system for logical consistency. Consistency violations can be employed in a variety of ways, ranging from informing system operators of a potential problem to stimulating the automatic reconfiguration of a system so as to avoid operating in a compromised state. For example, in the case of a command and control decision support system, inconsistency of internal system measurements, as determined by security-aware decision support applications, can be used as a basis for recognizing a cyber attack, and potentially isolating which subsystems are most likely to have been affected. Isolating faulty components can also be accomplished through the use of voting [Clarkson, Chong, and Myers, 2008; Fujioka and Okamoto, 1992] across a diverse set of redundant components. For example, an automobile brake control system can be diversely and redundantly replicated and the outputs automatically compared as the basis for determining which of the configurations is at fault. This fault isolation also permits restoration control actions to avoid operations in a compromised system configuration. The data consistency checking service should include evaluations of those data elements embedded in operations that impact system functions that are deemed to be related to the critical operation of those applications being secured. It is envisioned that this function would be designed as an agent that interacts with system application functions to query, collect, and analyze required information, and should interact with other System-Aware security services, such as configuration hopping, to provide capabilities designed to avoid operating in a compromised state.

1.3.4 Tactical Forensics

Traditionally, in the context of cyber security, forensics has often been associated with attacks with significant consequences and obtaining legal evidence to present in a court of law [Noblett, Pollitt, and Presely, 2000]. However, in the context of System-Aware Cyber Security, forensics are intended to provide tactical analysis which enables system operators and administrators to rapidly distinguish between those faults caused by a compromised component (i.e., cyber attack) and those resulting from other causes (i.e., natural failure). This distinction is critical, as it can have a significant influence on how system administrators and operators should proceed post-attack. For example, assume that a system utilizes both diversity and data consistency checking to secure a vehicle's braking system. Also assume that, in a particular instance, the System-Aware security system successfully detects and averts a fault that would have disabled the brakes from working. The fault is reported to the owner of the vehicle who now needs to know whether the fault was a result of a failing component or the result of an embedded Trojan horse. If it was a failing component, the owner can have the faulty part replaced. However, if it is the result of a Trojan horse, the owner must report the problem to the relevant parties, for the possibility that many vehicles with the same component are potentially at risk. In the latter case, a forensic investigation is required in order to determine the restoration solution and possibly aid in identifying the culprit.

To separate cyber attacks from other causes of faults, System-Aware Cyber Security architects can utilizes a variety of tools, including decision aids based upon proactive analysis methods, such as Fovino, Masera, and Cian's work [2009] on integrating cyber attacks into fault trees; software and/or hardware embedded in the system specifically designed to identify malicious actions (e.g., a radio frequency spectrum analyzer embedded in a subsystem's hardware chassis, and listening for a wireless triggering command at the time of an actual attack); and application of decision theory to relate evidence to alternative causes.

Finally, it is emphasized that tactical forensics is focused upon rapid attribution and restoration, and would serve to complement traditional forensic analysis techniques, not replace or replicate them.

Chapter 2
System-Aware Design Patterns

2.1 Introduction

One of the strengths of the perimeter security approach is that it offers a set of standardized commercially available products. In contrast, as discussed in sections 3.2, 3.3, and 3.4, System-Aware security solutions are highly customized to the applications to which they are embedded. Thus, there is a need to facilitate reuse of System-Aware security solutions across a diverse set of applications. One approach is to create security design patterns. These security patterns could facilitate the reuse of System-Aware security solutions across additional systems by drawing on the consensus of engineers engaged in building these systems–similar to how they have aided in object-oriented projects [Gamma et al., 1995] and more traditional security technologies [Schumacher et al., 2006]. In addition, these patterns would provide documentation characterizing the sufficient conditions for application as well as suggestions for additional synergistic patterns to enable the engineering community to apply them to new and existing systems.

In order to provide a starting point for the exploration and development of new secure design patterns, three patterns are presented based upon the work outlined in this thesis. The format for these patterns is based upon those used for traditional perimeter security as presented by Schumacher in his book on "Security Patterns: Integrating Security and Systems Engineering" [2006]. However, unlike the patterns presented by Schumacher, these patterns are not based upon implemented solutions but on research cases. Research cases were chosen as, "Patterns support the understanding of problems and their solutions," [Schumacher, 2006] and, "Patterns are generic—as independent of or dependent on a particular implementation technology as need be." [Schumacher, 2006]. Thus, design patterns provide not only a means for recording implemented solutions, but a method for recording research cases so that they can be applied to problems across a wide set of domains. As System-Aware security aims to provide cyber security solutions that are applicable to many domains, design patterns provide an ideal means of recording and presenting such solutions for reuse.

2.2 Diverse Redundancy

Name: Diverse Redundancy

Example of Need: Figure 2 presents a high-level system diagram for a typical steam fed nuclear reactor powered turbine control system. As indicated in Figure 2, the turbine receives actuation commands from a controller, currently available from a variety of vendors (e.g., the GE Mark VI, and Triconex Tricon). Operators located in the main control room of the power plant are responsible for controlling the turbine. These individuals receive status information from the controller that influences their operational actions, which can include stopping the turbine and correspondingly tripping the reactor to stop steam flow into the turbine. In addition to operator actions, the controller receives sensor information (listed in Figure 2) that together influences its automatic control actions. In situations where the turbine operation is such that it is of immediate importance to stop steam flow, the reactor is automatically stopped (i.e., scrammed), with a reactor shutdown process that is supported by the sensor information related to turbine operation.

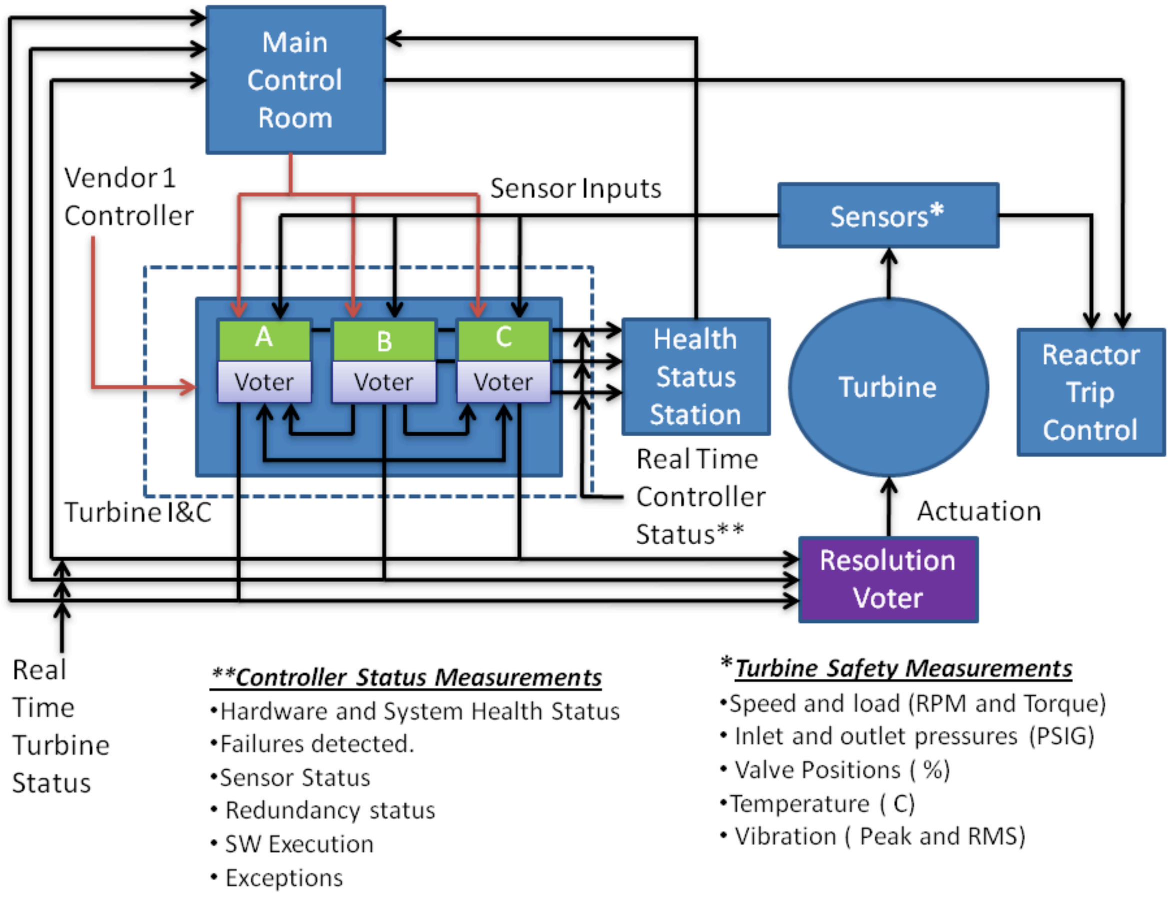


Figure 2. A high-level system diagram for a typical steam fed nuclear reactor powered turbine control system. The turbine controller is designed to meet high reliability and safety standards by employing redundancy and a resolution voter.

Figure 2 also highlights the fact that nuclear power plant turbine controllers are designed to meet high operational reliability and safety standards, and accordingly often employ various types of redundancy. However, there has recently been a rash of insider attacks where a Trojan horse was found to be embedded into the equipment of the supplier of the reactor's controllers. Given the significant economic consequences resulting from serious damage to the turbine, and the need to shut down (trip) the nuclear reactor in the event of a turbine shut-down, how can the reactor's owner continue to maintain high reliability while ensuring her system against a possible supply chain attack?

Context: Ensure that system functions critical for achieving mission objectives and high reliability requirements will be available even if one or more the components that support those functions have been compromised by a cyber attack.

Problem to be Solved: While the use of redundant components in systems is a common way to assure continuity of operation, the use of components that are susceptible to a common source of failure does not provide assurance against a cyber attack that affects all of the common components.

Solving this problem requires one to resolve the following forces:

- For a cyber attack, a single exploit can be developed and used to compromise all of the identically redundant components that might otherwise provide enhanced continuity of operation
- The cyber attack can be embedded into the redundant components through the supply chain or an insider attack, making it difficult to ensure that a cyber attack has not compromised all of the components
- The single exploit may be an extremely minor change (e.g., the change of a single parameter) and triggered remotely or based on a certain condition (e.g., time). As a result detecting that a component or components have been compromised can be extremely difficult.

Solution: Solutions for ensuring that the success of a cyber attack on a critical system function(s) does not result in mission failure can be based upon protection approaches developed by the fault-tolerance systems community. One such technique is to utilize diversely redundant components to ensure that a system is able to carry out its mission objectives even when one of those components breaks down. This assumes that each of the diversely redundant failures is independent; i.e., no common source exists to cause the same fault in all of the components. A cyber attack is one such common source that could put all redundant components at risk, and prevent a system from completing its mission objectives. This solution mitigates the capacity for a cyber attack to successfully compromise all redundant components by utilizing diverse components with a different set of attributes.

Structure:

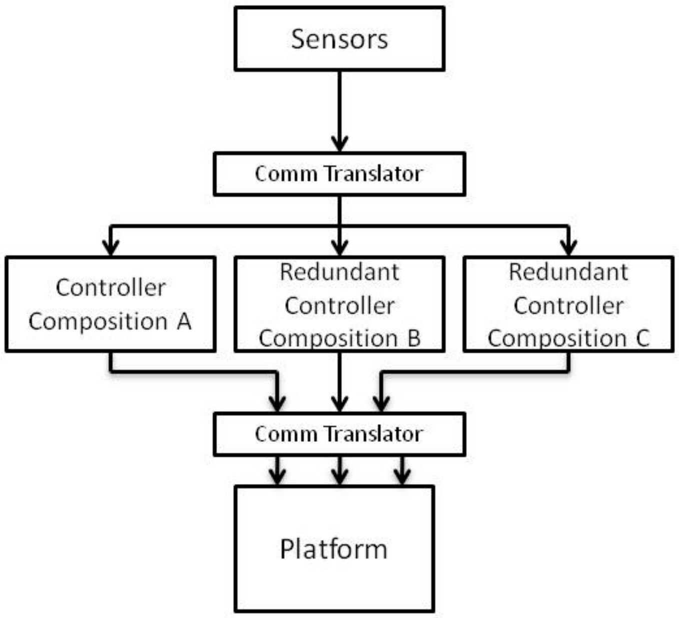


Figure 3. A simple illustration of the structure of *Diverse Redundancy*. In this instance three different controllers are used to receive inputs from a set of sensors and issue inputs to control a platform. Furthermore, each of the controllers is utilizes a diverse set of protocols. Thus, communication translators are included (i.e., the Comm Translators).

Diverse Redundancy requires the following elements:

- Two or more diversely redundant components. These components must be diverse with regards to the common source of the cyber attack. For example, if the common source is a Trojan horse injected via the supply chain, then the common source is the supplier and the components should be procured from independent suppliers.

- Special hardware may be needed to integrate the diverse components into the system. For the structure shown in Figure 3, the diverse components use special communication translators as each of the diverse controllers employs a different communication protocol.

Dynamics: As seen in Figure 3, the diverse components will possibly need to be able to receive input, generate output, and exchange information with other diverse components. Depending on whether the original system employed redundancy or not, additional infrastructure may be needed to transmit information to and from the diversely redundant components, as well as between the diversely redundant components. For example, an additional mechanism might need to be integrated into the system which is used to ensure that only one of the diversely redundant controllers is sending its information along and that the remaining are serving as backups. Alternatively, in order to avoid disturbances in output when it is required to switch components due to a failure, a mechanism could be employed to average or filter the outputs of the diversely redundant components. This result is then utilized as the output of the diversely integrated components.

Implementation: Diversity can encompass a large set of parameters, including hardware, software, vendor, geographical location, administrator(s), etc. Thus, it is important to consider the type(s) of diversity that will be needed to prevent an attack. For example, utilizing multiple diverse operating systems will force an adversary to develop cyber attacks for each of the operating systems, but could leave them vulnerable to an attack embedded in a common hardware component. Diverse components may require special hardware and/or software to ensure interoperability.

Example of Need Resolved: The owner of the nuclear reactor decides to integrate two additional turbine controllers along with *Verifiable Voting* and *Physical Configuration Hopping* (see Figure 4). As the reactor owner was worried about compromised components originating from the supplier, she has decided to integrate three turbine controllers from different vendors. As each of these vendors employs its own communication protocol, additional communication translators are needed to ensure interoperability. *Verifiable Voting* has been utilized to detect and isolate a controller issuing potentially damaging information, as well as to ensure that only one of the controller's command signal reaches the turbine. Finally, *Physical Configuration Hopping* is utilized to both enhance security and select which of the diversely redundant controller's data will be passed to the turbine and which are serving as backups. For a more complete discussion of this example the reader is referred to section 3.2. Additional examples employing diverse redundancy are discussed in section 3.3 and 3.4.

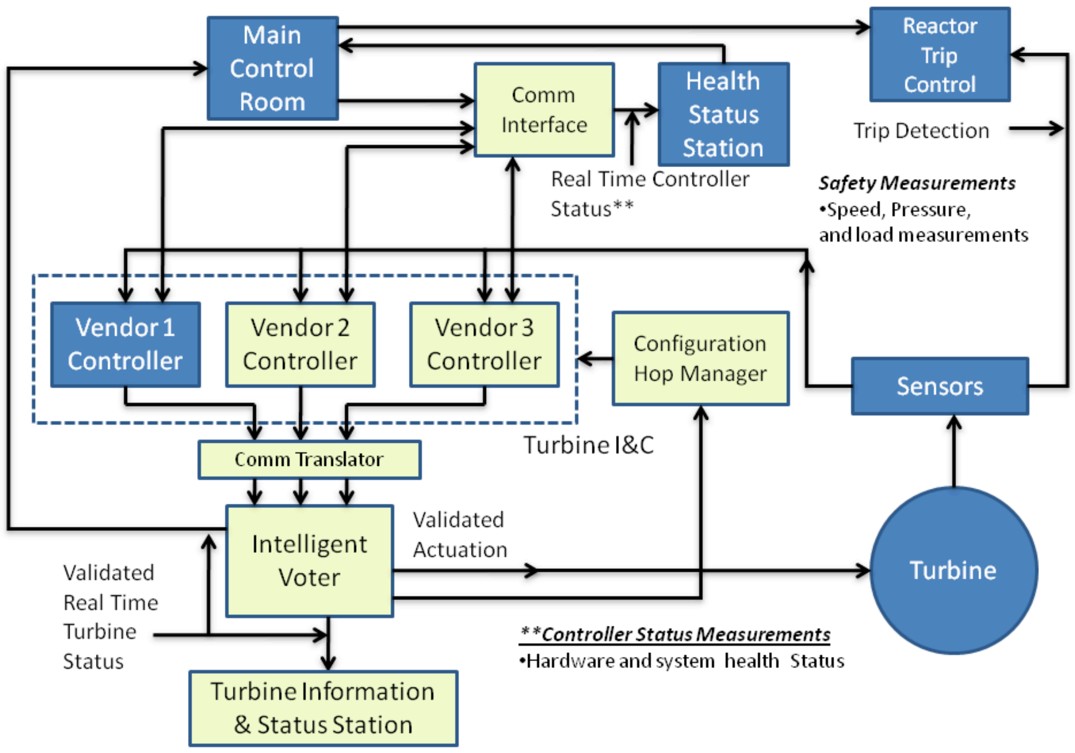


Figure 4. Resolved solution for *Diverse Redundancy*. In this instance, diver redundancy and *Verifiable Voting* have been employed to protect the turbine controller and ensure protection against a supply chain attack.

Variants: A variation includes utilizing redundant components that possess reduced or different capabilities. For example, a GPS-based navigation system can utilize an inertial navigation system as a redundant backup.

Known Uses: [Jones and Horowitz, 2011, Jones, Nguyen, and Horowitz, 2011; Jones and Horowitz, 2012; Babineau, Jones, and Horowitz, 2012]

Consequences: The following benefits may be expected from applying this pattern:

- *Diverse Redundancy* can serve to increase the complexity of an attack that would attempt to compromise all components by forcing the need for cyber attacks with specific capabilities to address each of the diversely redundant components
- In systems without redundant components, *Diverse Redundancy* can potentially increase the systems robustness to faults
- Some systems may already possess diverse components and can possibly make implementation easier

The following potential liabilities may arise from this pattern:

- *Diverse Redundancy* may require additional infrastructure to ensure interoperability with all components
- In systems without redundant components, *Diverse Redundancy* may require new infrastructure to ensure all components receive the appropriate input and that the proper output signals are sent

- As *Diverse Redundancy* requires the components to be diverse with regards to the common source of failure, the amount of commercial off the shelf (COTS) solutions for providing diversity may be limited
- Life cycle costs and training of support staff could increase due to the requirement to service diversely redundant components

Related Design Patterns: *Verifiable Voting* is a mechanism that can be combined with *Diverse Redundancy* to help detect and isolate which of the diversely redundant components have been compromised. *Diverse Redundancy* can also be combined with *Physical or Virtual Configuration Hopping* to dynamically switch which component is engaged in the operational system at any given time in order to both detect a compromised component and minimize the time available for an exploit to affect the system.

2.3 Verifiable Voting

Name: Verifiable Voting

Example of Need: A museum has recently installed a video surveillance system to protect its collection of rare and valuable artifacts. As shown in Figure 5, this system consists of a series of security cameras that transmit their data to a media server and its hot shadowed backup. Security personnel can pull the video streams from the media server to their mobile devices to observe the rooms remotely. In addition, when the museum is closed, the media servers scan all of the incoming video streams for unauthorized personnel. If the servers detect any unauthorized access an alert is sent to the security personnel. The security personnel can then decide to pull the video stream to determine the situation and take appropriate action to apprehend the intruder.

Recently the primary employee responsible for managing and maintaining the media servers was fired under the suspicion that she was planning a heist on the museum. Given the access this employee was afforded to the media servers, the owner of the museum is concerned that the employee may have already tampered with the media servers as part of the planned heist. As a result, the museum owner wishes to employ additional security to protect against a possibly malicious server.

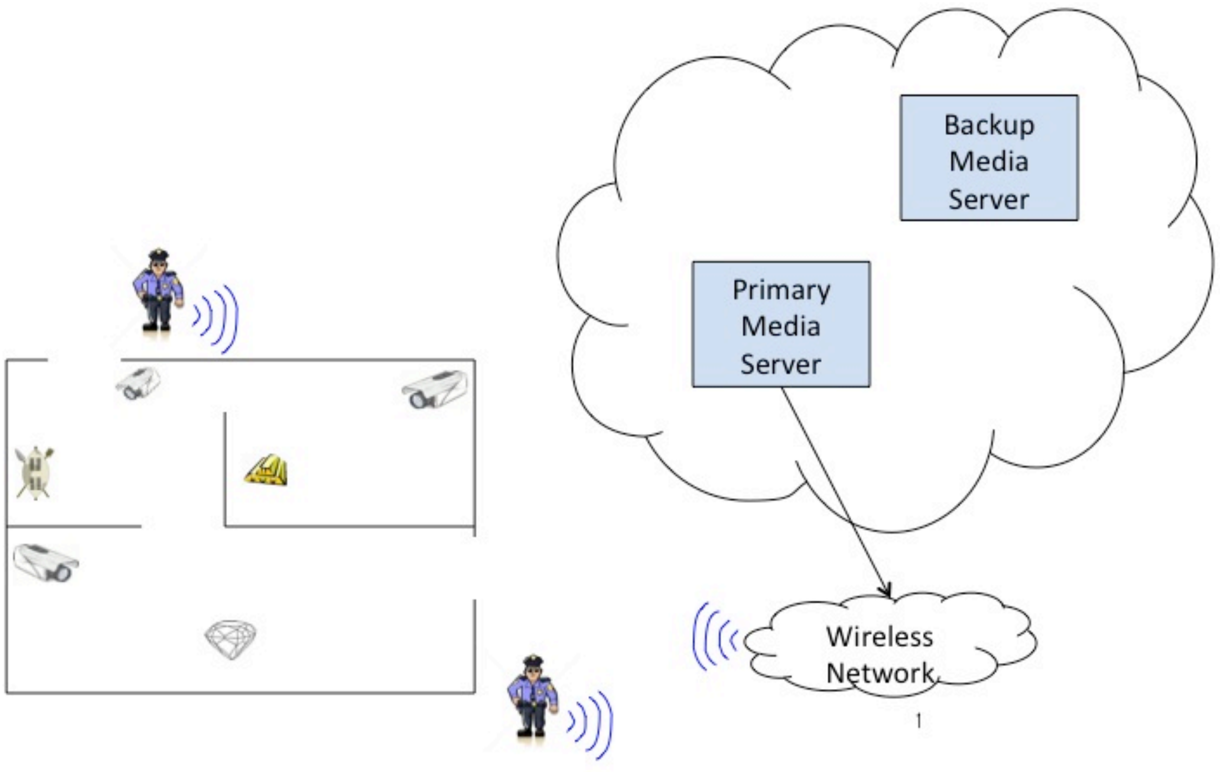


Figure 5. A high-level system diagram of a video surveillance system for a museum. The security cameras send the video surveillance to media servers that distribute the information wireless to security personnel.

Context: Systems often produce information that is critical in determining the appropriate set of actions to be taken to ensure the desired outcome. However, this can potentially result in a significant decline in system performance when there is reason to suspect that the source of information may not always be producing reliable information. This decline can potentially lead to undesired or inferior outcomes whenever the source is producing valid information, but nonetheless is not trusted, or the source is trusted, but producing bad information—such as due to a cyber attack. Thus, a method is needed to be able to detect and/or isolate those components that may be compromised and may be producing faulty information.

Problem to be Solved: How can one continue to utilize (i.e., trust) the outcomes of a critical system when one suspects that the system has been compromised?

Solving this problem requires one to resolve the following forces:

- If the system were compromised by a cyber attack, it could cause considerable damage. However, simply disabling the system is undesirable, as the support it affords is critical to achieving the desired outcomes. Thus, a method is needed to detect when the output of the system is valid and when it is misleading.
- It may be possible to restore a system to working order once a compromise has been detected; however, to do so, it may be necessary to isolate the component responsible for producing the faulty output
- To protect against a cyber attack, the mechanism employed to detect and isolate systems producing faulty information must also be secured. In addition, this mechanism must not impact system performance to the point of preventing the system from functioning properly.

Solution: A voting scheme is typically used to detect and isolate systems that are producing faulty outputs. Voting can also be utilized to detect misleading outputs. However, if the misleading information is being produced as a result of a cyber attack, it is possible that the attack may have been embedded into the component through the supply chain or from an insider. As a result, it is possible that the mechanism used to carry out the voting may be compromised. *Verifiable Voting* is utilized to provide voting in a secure manner. It is based on providing a hierarchy of voters tailored to the specific needs of the system to ensure that components acting maliciously are identified, while not significantly impacting system performance. Each of the voters in the hierarchy is designed based upon a trade-off analysis regarding ease of verifiability—i.e., confidence that it has not been compromised—and ability to perform timely and complex comparisons.

Structure: *Verifiable Voting* is composed of one or more voting mechanisms (e.g., the Byzantine fault-tolerant voter [Lamport, Shostak, and Pease, 1982] or Civitas [Clarkson, Chong, and Myers, 2008]) implemented in hardware or software. This includes an extremely simple voting mechanism, implemented in hardware or software, which is easily verifiable; i.e., known to be secure. However, such a simple mechanism may only be capable of implementing a simple voting scheme. This may result in voting rules that do not include all available information, resulting in an unacceptable degradation of performance compared to a voting scheme that uses more information. Alternatively, using more information may make the voting logic too complex to sufficiently verify its implementation from a security standpoint. As a result, in addition to using the less sophisticated, but more verifiable voters to validate simple, but mission critical machine generated outputs (e.g., fire the gun), they can also be used periodically, as a coarse check on whether a less verifiable voter has been compromised. Finally, *Verifiable Voting* requires that there be multiple redundant systems producing output. The amount of redundancy determines how many of the redundant systems can be compromised before it becomes impossible to detect and isolate potentially compromised components. Figure 6 illustrates one possible hierarchy of voters that assumes only a single redundant system will be compromised at a given moment.

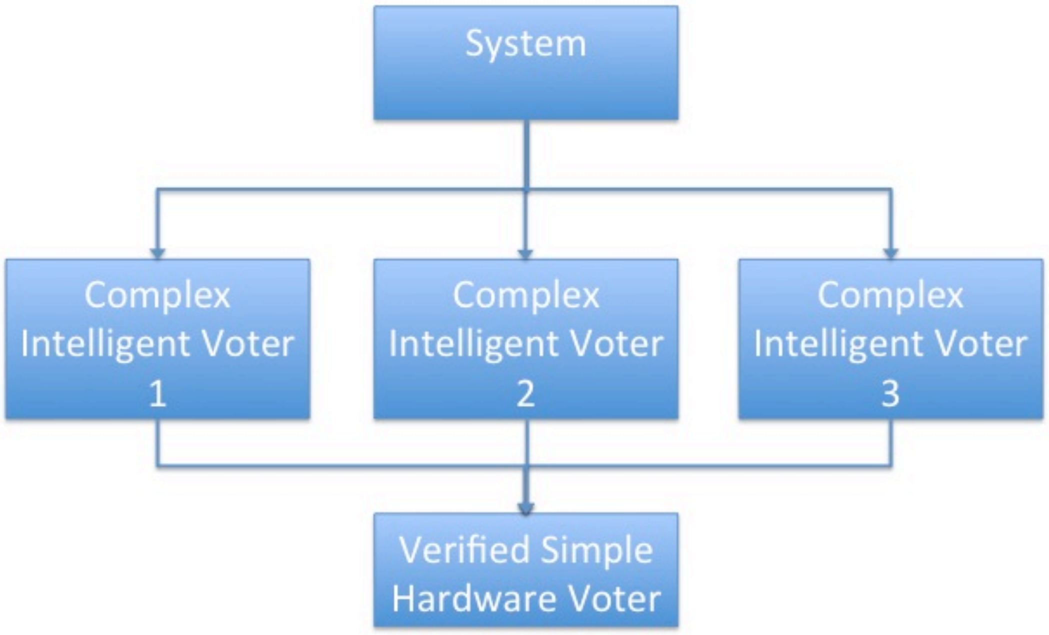

System
Complex Intelligent Voter 1
Complex Intelligent Voter 2
Complex Intelligent Voter 3
Verified Simple Hardware Voter

Figure 6. A simple example of *Verifiable Voting*. This includes three complex intelligent voters that are used to evaluate the information from the system. These results are fed to a simple hardware voter that can be easily verified.

Dynamics: All voters need to be able to receive the necessary outputs for comparison from the multiple redundant systems. It is important that the most verifiable (i.e., secure) of the hierarchy of voters be able to override the decisions of the less secure voters.

Verifiable Voting requires replication of the outputs of the system in order to carry out the vote. If the system already carries the necessary redundancy or the output of the system is small (e.g., a true or false value) then the cost of this replication can be negligible. However, when the outputs being voted on are large (e.g., the output of diversely redundant video streams received over a wireless network for voting) then such voting can add significant overhead. While, this overhead can potentially be mitigated through the use of additional resources, it may also be possible to mitigate it through the use of customized system designs. For example, in Figure 6 each of the three complex intelligent voters is receiving the three inputs simultaneously. However, it is possible to stagger the voting across each voter; i.e., complex intelligent voter 1 receives the three inputs and votes, than complex intelligent voter 2 receives the three inputs and votes, and finally complex intelligent voter 3 receives the three inputs and votes. Once this is done each of the complex intelligent voters can send its simplified results to the simple hardware voter for a final decision (see Figure 6). For the case of a wireless network communicating the information, this scheme of staggered voting can result in a reduction in bandwidth utilization, while also potentially delaying the detection of any modification of data in one of the streams.

Implementation: When implementing *Verifiable Voting* it necessary to determine an appropriate scheme for voting as well as the input that will be voted on. Given this information, it is possible to determine the desired number of redundant system components to achieve detection and isolation. It is also possible to develop an appropriate hierarchy of voters. This hierarchy will depend on the type of information used in voting, the frequency of voting, and the desired security of the *Verifiable Voting* scheme itself. Finally, additional resources or techniques may be needed to ensure that the desired level of system performance is achieved.

Example of Need Resolved: To defend the museums rare artifacts against a possible cyber attack embedded in the media server, the owner decides to implement *Verifiable Voting*. As there are only two media servers, *Verifiable Voting* is only able to provide detection. As the museum has security guards on patrol and possesses the capacity to rapidly lock down the artifacts, it is decided that isolation is not necessary. If the *Verifiable Voter* detects a problem (i.e., cannot reach consensus) it will alert the security personnel who can then place the museum on lockdown.

To ensure that the *Verifiable Voter* will be secured against cyber attacks, it is decided that the *Verifiable Voter* will be deployed onto mobile devices used by the security personnel for alerts. While it is possible that a single guard's device could be compromised, there would still be several additional security guards capable of receiving the information. Thus, an attacker would have to compromise all of the mobile devices used by personnel. From the perspective of the museum owner, this is deemed an unlikely event and thus an acceptable risk.

Finally, each of the guard's devices will perform *Verifiable Voting* on the information coming from the media servers, including the video stream. Due to both the large bandwidth consumed by video and the limited bandwidth available for wireless communications, it is decide to implement a duty cycle voting scheme (see section 3.4) to mitigate the bandwidth cost. See sections 3.2, 3.3, and 3.4 for more detailed examples utilizing *Verifiable Voting*.

Variants: None.

Known Uses: [Jones and Horowitz, 2011, Jones, Nguyen, and Horowitz, 2011; Jones and Horowitz, 2012; Babineau, Jones, and Horowitz, 2012]

Consequences: The following benefits may be expected from applying this pattern:

- Can both detect misleading output as well as isolate the offending component
- Voting mechanism can be implemented in a more secure manner
- Offers a flexible implementation to trade off desired level of security with cost, complexity, and performance impacts

The following potential liabilities may arise from this pattern:

- Detection and isolation require the introduction of multiple redundant components with the attendant liabilities (see the design pattern for *Diverse Redundancy* in section 2.2)
- Depending on the information being voted upon, it can result in an increase in complexity and cost to ensure that solution meets the desired goal
- Can be defeated if enough of the redundant devices are compromised to form a majority (what constitutes a majority will depend on the voting scheme utilized)

Related Patterns: This pattern can be combined with *Diverse Redundancy* to potentially increase the difficulty in compromising all redundant components—e.g., through an insider or supply chain attack.

2.4 Physical Configuration Hopping

Name: Physical Configuration Hopping

Example of Need: Modern ships are equipped with a wide set of systems to monitor and control (e.g., engine, propulsion, fire suppression, and climate control). A company wishes to produce a lower cost ship by consolidating the network between the monitoring consoles and the physical systems into a single COTS network switch. To improve the reliability of the design, a redundant network switch is installed to resume operations in the event the primary switch fails. However, consolidating all network connections also leaves the entire ship vulnerable to any cyber attacks embedded into the primary network switch:

- Send potentially misleading information to the monitoring systems
- Could disable the ship through a denial of service attack by dropping all communications
- Modify or inject commands to the physical systems in order to damage, disable or misdirect the ship

Context: Ensure that critical system components that have been infected with a cyber attack will be unable to actively disrupt, damage, or misdirect systems operations.

Problem to be Solved: Techniques exist to detect, isolate, and disable system components that are behaving in a manner to cause harm to the system. However, a system component compromised by a cyber attack has the potential to disrupt and possibly damage critical system components before such methods are successfully able to disable the offending component. In addition, such methods may be unable to prevent cyber attacks aimed at passive monitoring or more sophisticated attacks that attempt to cause disruptions and damage more subtly (e.g., Stuxnet attack).

Solving this problem requires one to resolve the following forces:

- Ensure that a cyber attack is not given enough time to cause damage or disrupt system operations; this time may be less than the time needed to detect and isolate the compromised component
- Prevent a cyber attack exploit from reading enough information to form a coherent data set for use by the attacker
- Security solution must not compromise the systems mission objectives by significantly impacting system performance

Solution: Solutions for preventing compromised system components from taking potentially malicious action can be based on techniques developed by the cyber security community. One such technique is moving target defense; a technique that aims to dynamically switch functionality across multiple resources. *Physical Configuration Hopping* builds on this technique by continuously shifting control between multiple redundant physical system components in order to disrupt a cyber attack before it can cause permanent damage.

Structure: As seen in Figure 7, *Physical Configuration Hopping* requires multiple redundant components to be dynamically interchanged (two in Figure 7). This dynamic reconfiguration determines which component(s) is in control at any given time. In addition, there is a mechanism to the control the frequency of the dynamic readjustment as well as determine which component is in control—in Figure 7 it is the configuration hop manager. Finally, their needs to be a mechanism in place to control the switching between components; this includes the frequency of hopping, as well as the order of hopping from one component to another (pertinent to cases of higher orders of redundancy).

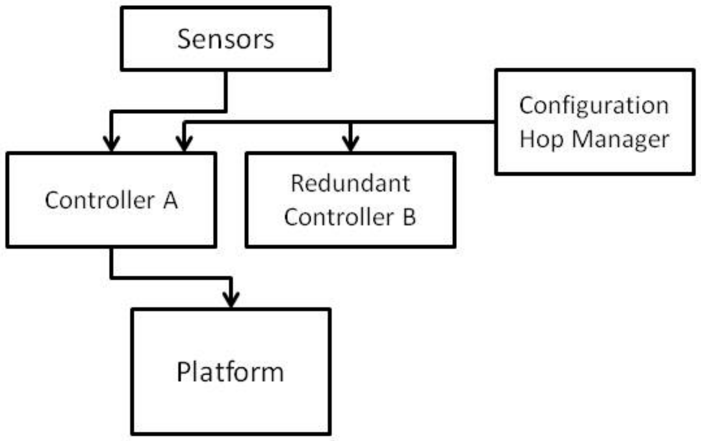


Figure 7. A simple *Physical Configuration Hopping* setup. This instance includes dynamic reconfiguration across two redundant controllers. Controller A is currently set to the active controller.

Dynamics: *Physical Configuration Hopping* requires that all redundant components be able to receive and generate output to the appropriate systems, as control will need to be dynamically switched between those components. In addition, it may be necessary to ensure that the dynamic switching between components is bumpless. For example at the time of switching the multiple redundant components may be in different states; thus, the switch between components results in an unintended switching of states.

Implementation: When implementing *Physical Configuration Hopping* it is important to consider the time it will take for a compromised component to cause damage. For example, a turbine in a nuclear reactor can potentially be damaged in a matter of seconds. Alternatively, it may take several minutes or even hours to steer a ship far enough off course to be considered damaging. In addition, the sophistication involved in switching between redundant system components depends on the sophistication of the cyber attack to be prevented. For example, switching between redundant components in a round robin fashion may disrupt a cyber attack that is just trying to transmit damaging commands quickly. However, a more sophisticated attack may be able to detect the switching patterns. This information could then potentially be used to issue commands that ultimately cause damage through controlled thrashing that occurs every time a switch from the compromised component to a non-compromised component occurs. It is also important to decide how much control is given to administrators to change the frequency of hopping as well as alter the algorithm used to control the switching order and specific, perhaps pseudo-randomized, timing.

Example of Need Resolved: The ship building company decides to combine *Physical Configuration Hopping* with *Diverse Redundancy* in order to protect the ship from a compromised network switch. The company decides to purchase two switches from different vendors in order to help prevent a scenario where both switches are compromised via the supply chain. The company then determines that it is not worried about a Trojan horse being embedded in the new system component used for monitoring the information, as control and status information between systems is not of direct value to an attacker; however, it is worried about a compromised switch causing denial of service or injecting false and/or damaging commands. It is then determined that it would take at least five minutes before a compromised network switch could cause any permanently damaging actions. Finally, the dynamic switching has the potential to cause some status information to be lost; however, the amount of information lost is small relative to the frequency of updates; i.e., no additional resources are needed for bumpless control. See sections 3.2 and 3.3 for more complete examples involving *Physical*

Configuration Hopping.

Variants: *Virtual Configuration Hopping*

Known Uses: [Jones and Horowitz, 2011, Jones, Nguyen, and Horowitz, 2011; Jones and Horowitz, 2012; Babineau, Jones, and Horowitz, 2012]

Consequences: The following benefits may be expected from applying this pattern:

- Prevent a system component compromised by a cyber attack from being able to compromise the mission objectives; prevention can occur independently, and faster than methods used for detection, isolation, and restoration
- Makes the development of cyber attacks more difficult by introducing time as an element

The following potential liabilities may arise from this pattern:

- Requires multiple redundant components with the attendant liabilities of the *Diverse Redundancy* design pattern
- Introduce the need for methods to ensure bumpless control
- Defeated if the frequency of hopping is too slow, or the algorithm for switching is predictable

Related Patterns: Can be combined with *Diverse Redundancy* to potentially mitigate the risk that multiple redundant components will be compromised.

2.5 Virtual Configuration Hopping

Name: Virtual Configuration Hopping

Example of Need: An e-commerce business stores customer credit card information in a secure facility equipped with a video surveillance system. This video surveillance is maintained and routinely inspected by a private contractor to ensure that it is operating properly. Recently the company has learned that several of the companies that also use this private contractor have been the victims of theft. An investigation of each of the sites has revealed that each of the systems responsible for receiving and displaying the streams to security personnel was infected with a Trojan horse to perform a simple replay attack. Furthermore, it is suspected that an employee of the private contractor did the theft. The e-commerce site has invested significant resources in building the secure facility as well as the video surveillance system and desires a solution to secure the video surveillance system against a possible insider attack.

Context: Ensure that critical system functions that have been infected with a cyber attack will be unable to actively disrupt, damage, or misdirect systems operations.

Problem to be Solved: Techniques exist to detect, isolate, and disable system functions that are behaving in a manner to cause harm to the system. However, a system function compromised by a cyber attack has the potential to disrupt and possibly damage critical system functions before such methods are successfully able to disable the offending functions. In addition, such methods may be unable to prevent cyber attacks aimed at passive monitoring or more sophisticated attacks that attempt to cause disruptions and damage more subtly (e.g., Stuxnet attack).

Solving this problem requires one to resolve the following forces:

- Ensure that a cyber attack is not given enough time to cause damage or disrupt system operations; the time to cause damage or disruption may be less than the time needed to detect and isolate the compromised function

- Prevent a cyber attack exploit from reading enough information to form a data set for use by the cyber attack

- Security solution must not compromise the systems mission objectives by significantly impacting system performance parameters

Solution: Solutions for preventing compromised system functions from taking potentially malicious action can be based on the techniques developed by the cyber security community. One such technique is moving-target defense that aims to dynamically switch functionality among multiple resources. *Virtual Configuration Hopping* builds on this technique by continuously shifting control between multiple redundant virtualized system functions in order to disrupt a cyber attack before it can cause permanent damage.

Structure: As seen in Figure 8, *Virtual Configuration Hopping* requires multiple redundant functions to be dynamically interchanged (two in Figure 8). This dynamic reconfiguration determines which function(s) is in control at any given time. In addition, there is a mechanism utilized to the control the frequency and exact timing of the dynamic readjustment as well as determine which function is in control—in Figure 8 it is the configuration hop manager. Finally, their needs to be a mechanism in place to control the switching between function; this includes the frequency of hopping, as well as the order of hopping from one function to another (pertinent to cases of higher orders of redundancy).

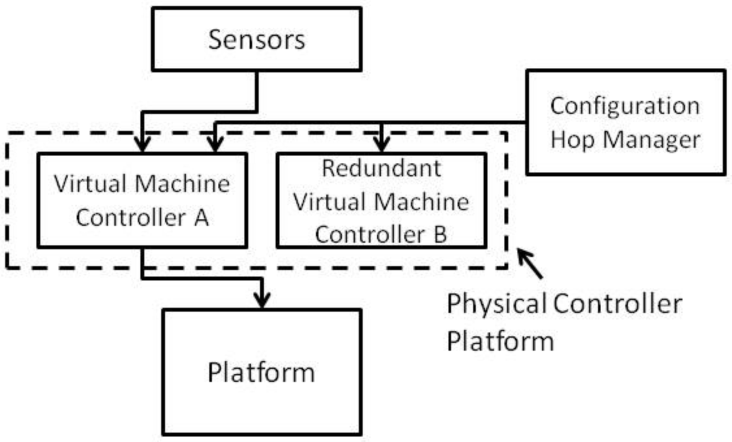

Sensors
Configuration Hop Manager
Virtual Machine Controller A
Redundant Virtual Machine Controller B
Physical Controller Platform
Platform

Figure 8. A simple *Virtual Configuration Hopping* setup. This instance includes dynamic reconfiguration across two virtually redundant controllers located on the same physical platform. Controller A is currently set to the active controller.

Dynamics: *Virtual Configuration Hopping* requires that all redundant functions will be able to receive and generate output to the appropriate systems, as control will need to be dynamically switched between those functions. In addition, it may be necessary to ensure that the dynamic switching between functions is bumpless. For example at the time of switching the multiple redundant functions may be in different states; thus, the switch between functions results in an unintended switching of states.

Implementation: When implementing *Virtual Configuration Hopping* it is important to consider the time it will take for a compromised function to cause damage. For example, a turbine in a nuclear reactor can potentially be damaged in a matter of seconds. Alternatively, it may take several minutes or even hours to steer a ship far enough off course to be considered damaging. In addition, the sophistication involved in switching between redundant system functions depends on the sophistication of the cyber attack to be prevented. For example, switching between redundant functions in a round robin fashion may disrupt a cyber attack that is just trying to transmit damaging commands quickly. However, a more sophisticated attack may be able to detect the switching patterns. This information could then potentially be used to issue commands that ultimately cause damage through controlled thrashing that occurs every time a switch from the compromised function to a non-compromised component occurs. It is also important to decide how much control is given to administrators to change the frequency of hopping as well as alter the algorithm used to control the switching order and specific, perhaps pseudo-randomized, timing.

Example of Need Resolved: The concerned e-commerce business determines that the system responsible for receiving and displaying information can be virtualized quickly at minimal costs and decides to use *Virtual Configuration Hopping*. The e-commerce site sets up a virtualized environment to run multiple copies of the system. In addition, the e-commerce site obtains a video surveillance application from another vendor and adds that into its virtual environment. Once this has been set up, the e-commerce business determines that it should be concerned regarding the possibility of the credit card information stored at the protected site being stolen. It then determines that it would take an intruder at least 10 minutes to download all of the credit card information. The system is then set-up to hop between the virtualized system functions every 5 minutes. However, during switching the video feed appears to exhibit some slight distortions (i.e., it is bumby). To mitigate this effect, *Virtual Configuration Hopping* system is updated to provide a smooth (i.e., bumpless) stream. See section 3.4 for more detailed examples utilizing *Virtual Configuration Hopping*.

Variants: *Physical Configuration Hopping*.

Known Uses: [Jones and Horowitz, 2011, Jones, Nguyen, and Horowitz, 2011; Jones and Horowitz, 2012; Babineau, Jones, and Horowitz, 2012]

Consequences: The following benefits may be expected from applying this pattern:

- Prevent a system component compromised by a cyber attack from being able to compromise the mission objectives; prevention can occur independently, and faster than methods used for detection, isolation, and restoration
- Makes the development of cyber attacks more difficult by introducing time as an element

The following potential liabilities may arise from this pattern:

- Requires multiple redundant functions with the attendant liabilities of the *Diverse Redundancy* design pattern
- Introduce the need for methods to ensure bumpless control
- Defeated if the frequency of hopping is too slow, or the algorithm for switching is predictable

Related Patterns: Can be combined with *Diverse Redundancy* to potentially mitigate the risk that multiple redundant functions will be compromised.

Chapter 3
Applications of System-Aware Cyber Security

3.1 Background

Three example systems were developed to explore the development of the design patterns in Chapter 2, as well as evaluate how a System-Aware security architecture utilizing those design pattern solutions would deter and/or defend against potential cyber attacks. While the integration of a set of security services is a general solution approach, these examples serve to illustrate the genesis for potential architectures to establish specific solutions.

3.2 Nuclear Power Plant

This section provides examples of potential cyber attacks on a nuclear power plant, and illustrates how a System-Aware Cyber Security architecture would potentially deter and/or defend against such attacks. While the integration of a set of security services is a general solution approach, the examples serve to illustrate the genesis for potential architectures to establish specific solutions. For the examples, the turbine control subsystem for the power plant is selected as the target for cyber attacks. This selection is based on the significant economic consequences of serious damage to the turbine, and the need to shut down (trip) the nuclear reactor in the event of a turbine shut-down. It is assumed that the attacking mechanism to be defended against is embedded in the equipment that is part of the turbine control subsystem. Furthermore, it is assumed that the actual attack may either be triggered through a pre-established protocol that is built into the infected equipment and deployed through the maintenance process, or through a power plant insider communicating the attack initiation in real time via a built-in communications channel, which is part of the infected equipment. The projected solutions have impacts in the pre-attack stage (deterrence), trans-attack stage (defense and temporary restoral), and the post-attack phase (longer-term restoral and threat reduction for power plants with similar equipment to the attacked plant) of a cyber attack. Each of these phases of attack is discussed for each of the attack scenarios.

Representative Model of a Turbine Control Subsystem

This section describes a model of a turbine control system that is used as the basis for postulating specific supply chain related cyber attacks on such a system, and to address both the potential impacts of attacks and possible System-Aware Cyber Security architectures to either reduce the consequences or possibly eliminate the attacks.

Figure 9 presents a high-level system diagram for a typical steam fed nuclear reactor powered turbine control system. As indicated in Figure 9, the turbine receives actuation commands from a controller, currently available from a variety of vendors (e.g., the GE Mark VI, and Triconex Tricon). Operators located in the main control room of the power plant are responsible for controlling the turbine. These individuals receive status information from the controller that influences their operational actions, which can include stopping the turbine and correspondingly tripping the reactor to stop steam flow into the turbine. In addition to operator actions, the controller receives sensor information (listed in Figure 9) that together influences its automatic control actions. In situations where the turbine operation is such that it is of immediate importance to stop steam flow, the reactor is automatically stopped (i.e., scrammed), with a reactor shutdown process that is supported by the sensor information related to turbine operation.

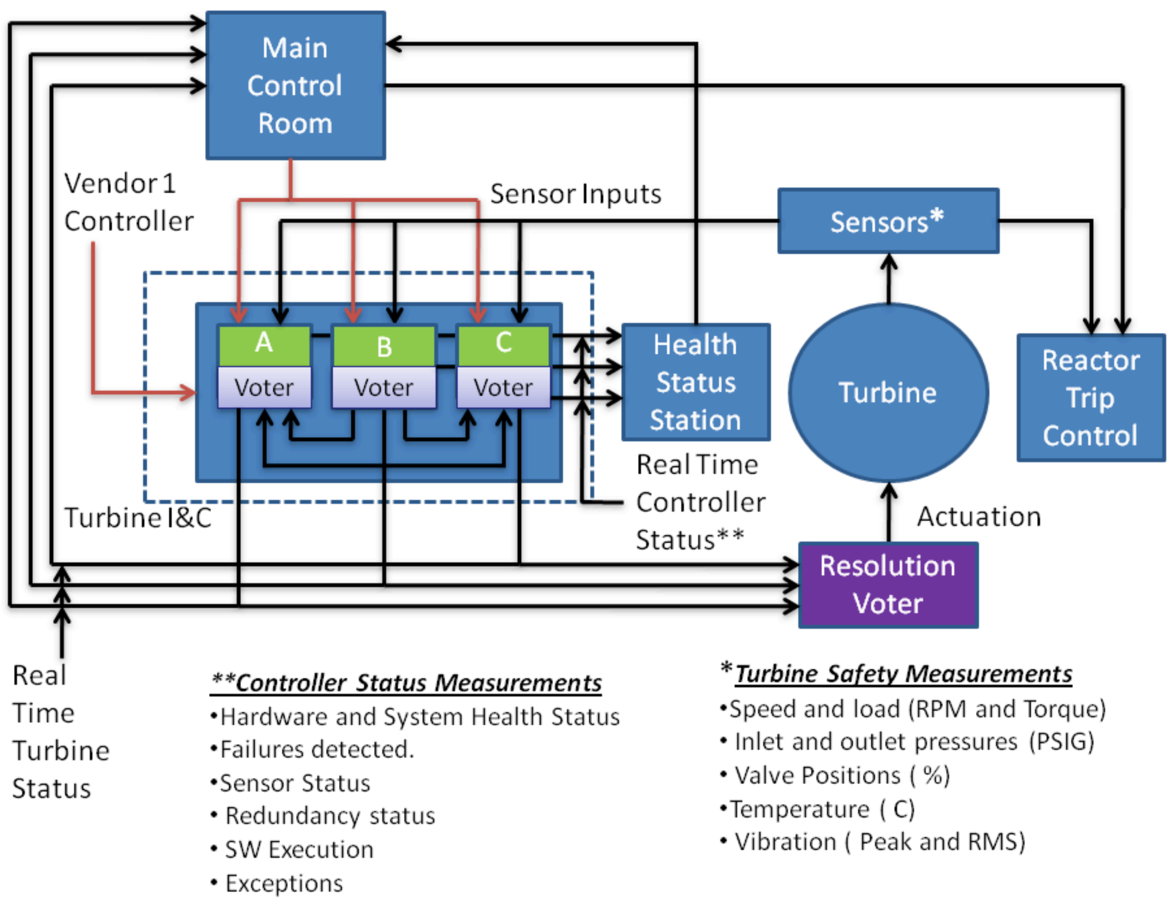


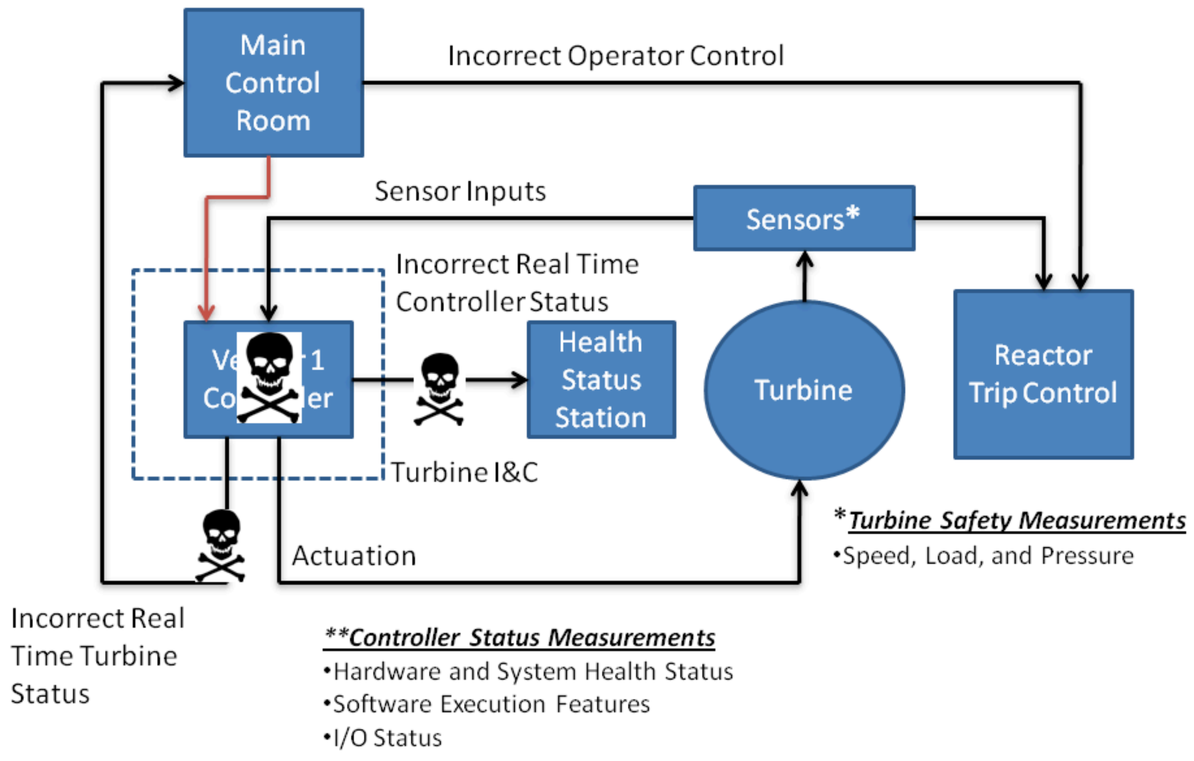
Figure 9. A high-level system diagram for a typical steam fed nuclear reactor powered turbine control system. The turbine controller is designed to meet high reliability and safety standards by employing redundancy and a resolution voter.

Figure 9 also highlights the fact that nuclear power plant turbine controllers are designed to meet high operational reliability and safety standards, and accordingly often employ various types of redundancy. Controller replication is a prevalent application for redundancy, and is depicted in Figure 9 as channels A, B, and C. In this example, the employment of a distributed voting scheme among control elements provides fault tolerance against randomly occurring hardware faults in the redundant controllers. The distributed voters typically exchange and vote on how to utilize the individually derived input sensor information, internal controller state information, and output commands. The results of the distributed voting process are forwarded to a master voter that resolves the output controller commands to the actuation system in the turbine. The master voter function is typically integrated as part of the vendor provided controller platform design. While the system diagram in Figure 9 is representative of a typical fault-tolerant controller, different vendors may employ different architectural principles and implementation strategies to manage redundancy and realize the desired level of fault tolerance.

Potential Supply Chain Related Cyber Attacks

This section describes two hypothetical cyber attacks that relate to current turbine control systems. Both attacks could result in the turbine operation being halted and the reactor being tripped. The attacks are structured along similar lines as the Stuxnet attack referred to earlier in the paper.

Figure 10 provides a diagrammatic representation of an attack where the turbine controller is infected with a Trojan horse. The Trojan horse implementation can involve a mixture of hardware and software manipulations. The Trojan horse is designed such that it can modify, replace, or nullify information that is forwarded to control room operator(s). The intended consequence of the attack would be to have the operator(s) believe that the turbine is operating sufficiently outside of specification so as to warrant a procedural action (e.g., shutdown of the turbine), which would induce a plant trip. The success of this attack depends on the attacker having knowledge of those operator procedures that demand rapid shutdown decisions. In general, this information is known to experienced nuclear power plant designers, integrators, and operators, and is thus likely to be readily available to attack designers. Furthermore, the attack demands that the Trojan horse be designed to circumvent the controller's security processes and internal voting process for fault tolerance. This could occur if the redundant elements all contain the same technology infection across all channels, which could occur if the Trojan horse were to be embedded by a common supplier of the hardware and/or software for the turbine controller.



Main Control Room
Incorrect Operator Control
Sensor Inputs
Sensors*
Incorrect Real Time Controller Status
Ve 1 Co er
Health Status Station
Turbine
Reactor Trip Control
Turbine I&C
*Turbine Safety Measurements
•Speed, Load, and Pressure
Actuation
Incorrect Real Time Turbine Status
**Controller Status Measurements
•Hardware and System Health Status
•Software Execution Features
•I/O Status

Figure 10. Illustration of a hypothetical cyber attack on a turbine control system. The controller is embedded with a Trojan horse that can modify, nullify, or replace information sent to the operators in the Main Control Room and the automatic monitoring station. The intended consequence of the attack would be to have the operator(s) believe that the turbine is operating sufficiently outside of specification so as to warrant a procedural action (e.g., shutdown of the turbine), which would induce a plant trip.

Figure 11 provides a diagrammatic representation of a more aggressive attack, based on a more sophisticated Trojan horse than the one required for the attack represented earlier. The intended consequence of the attack would be to create severe damage to the turbine, as well as to trip the reactor. For this case, the Trojan horse sends disruptive control signals to damage the turbine (e.g., misguided oil or temperature control commands), as well as manipulating attack-revealing information being sent to the control room operators. This would need to include sensor and feedback information regarding turbine status. The success of the attack depends on the assumption that the control room operator(s) would not be able to recognize, in a short period (1's of seconds), through independent means of the normal data paths that the turbine must be brought to a fail safe stop.

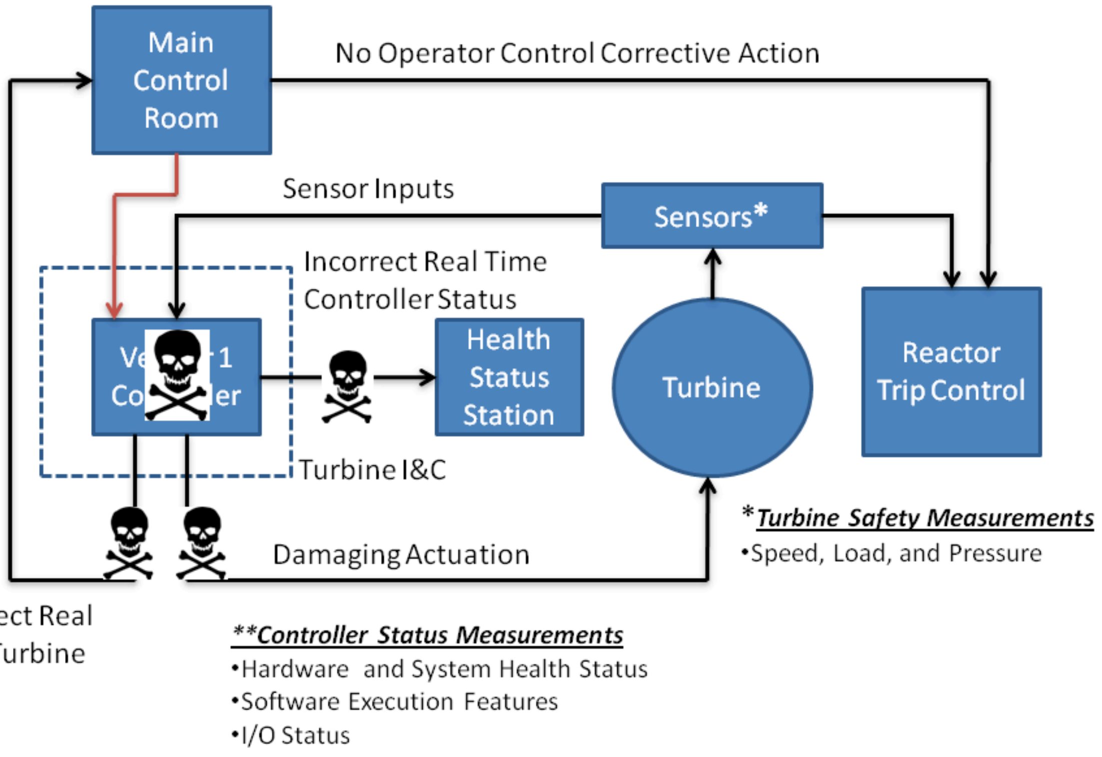

Main Control Room
No Operator Control Corrective Action
Sensor Inputs
Sensors*
Incorrect Real Time Controller Status
Ve 1 Co er
Health Status Station
Turbine
Reactor Trip Control
Turbine I&C
*Turbine Safety Measurements
•Speed, Load, and Pressure
Damaging Actuation
Incorrect Real Time Turbine Status
**Controller Status Measurements
•Hardware and System Health Status
•Software Execution Features
•I/O Status

Figure 11. Illustration of a hypothetical cyber attack on a turbine control system. The controller is embedded with a Trojan horse that can modify, nullify, or replace information sent to the operators in the Main Control Room, Health Status Station, and the actuation commands sent to the Turbine. The consequence of a successful attack would be to create severe damage to the turbine, as well as to trip the reactor by sending disruptive control signals to damage the turbine and manipulating control information sent to the Main Control Room.

System-Aware Cyber Security Architecture

To address attacks of the nature described in the previous section, a System-Aware security solution is embedded within the infected turbine control system. Figure 12 represents the turbine control system integrated with the four System-Aware Cyber Security design patterns outlined in Chapter 2.

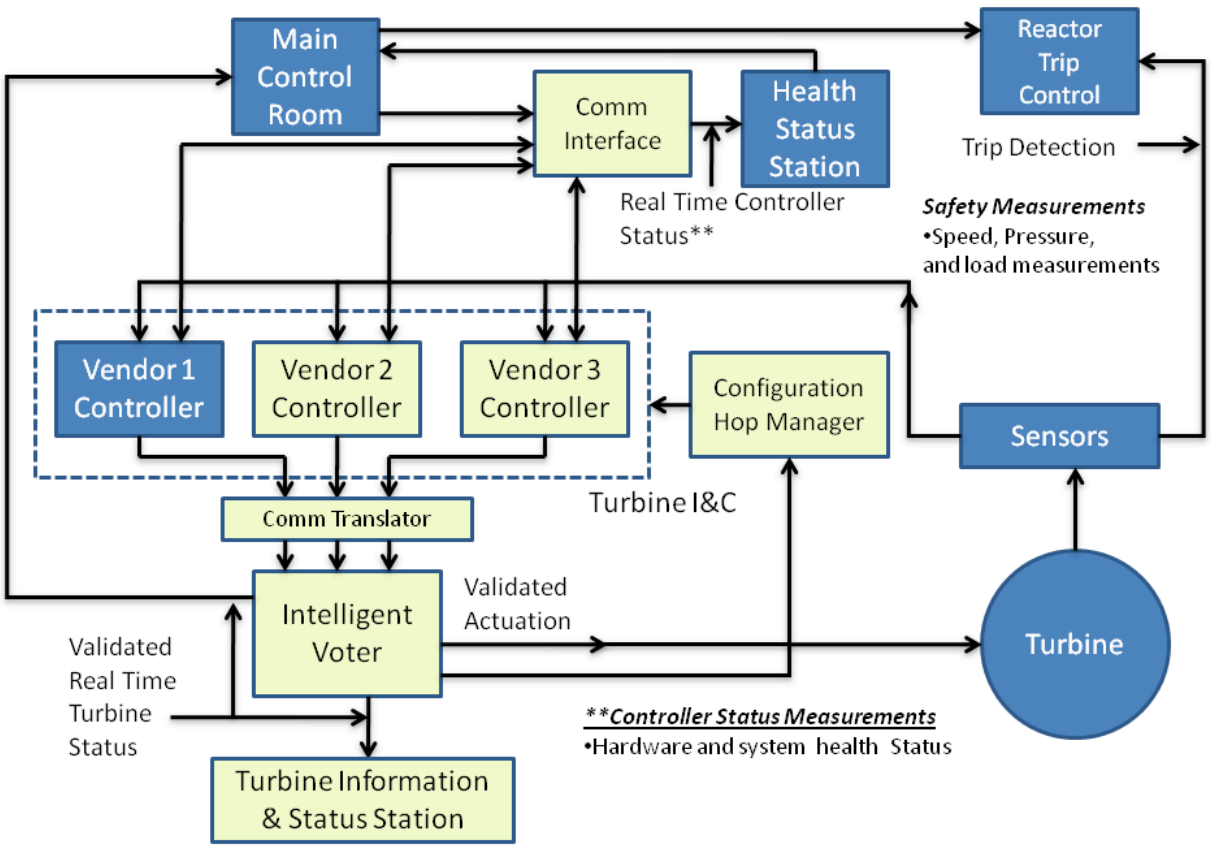


Figure 12. Possible System-Aware security architecture to address the threats illustrated in Figure 10 and Figure 11. This includes three controllers in the system architecture and an intelligent voting process to rapidly disable a differentiated controller from carrying out its turbine actuation functions.

Diversely Implemented Redundancy of Subsystems and Components

For each of the previously outlined examples, an attacker must be able to embed a Trojan horse in the turbine control system. As outlined in the *Diverse Redundancy* design pattern (see section 2.2), this threat can be addressed by the turbine system integrator selecting diverse vendors to supply multiple controllers; the underlying principle being that attackers would find it increasingly difficult to design and embed coordinated Trojan horses into diverse controllers. In addition, the overall control system can be designed to integrate the outputs from these diverse controllers through a cyber security sensitive intelligent voter.

Figure 12 presents one possible implementation. This includes three controllers in the system architecture and an intelligent voting process to rapidly disable a differentiated controller from carrying out its turbine actuation functions. During turbine operation, should a specific controller be discovered as a potential source of a cyber attack, the remaining controllers could continue to operate utilizing a new logic to assure that the two controllers' outputs are compatible. If the outputs are not compatible, because of the ambiguity regarding which is the flawed controller, the turbine would need to be shut down, and the reactor would need to be tripped. Figure 12 also includes two communication translator subsystems providing communications between elements of the secured turbine control system. The purpose of these subsystems is to perform the necessary protocol translations that enable communications from the diverse controllers to be integrated for voting or other system related purposes. Currently, a variety of products exist to perform integration across different communication buses; products of this sort would be required as part of the overall turbine control system as a response to introducing diversity for cyber security.

Returning to Figure 9, it is useful to note that an alternate security architecture from the architecture in Figure 12, but one that also builds on diversity, would incorporate diversity within a particular vendor's controller. While it is likely that organizing for a single vendor with diverse components would provide less difficulty to an attacker, an assessment of the cost and risk reduction differences would be required in order to draw a conclusion regarding architecture selection.

Physical Configuration Hopping

As shown in Figure 12, *Physical Configuration Hopping* (see section 2.4) would be managed within the configuration hopping manager subsystem. For this security service the turbine control system dynamically switches among controllers, during normal operations, so as to time-vary which controller would actually be sending its actuation outputs to the turbine at a given time. The hopping would occur in a randomized manner so as to create uncertainty for attackers about when the infected controller might actually be designated to control the turbine. As a result, during a given time interval, the remaining two controllers would only be used for voting purposes until called upon by the configuration hopping manager to take physical control of the turbine. The hopping process would need to be designed to assure that none of the controllers had physical control of the turbine for a sufficient time so as to be able to cause significant damage (i.e., 1's of seconds). When combined with *Diverse Redundancy*, *Physical Configuration Hopping* would serve to complicate matters for the attacker regarding the best time to initiate a desired attack, because the infected controller might not ever have the opportunity to actually control the turbine post-attack initiation. This complication occurs because the infected controller is likely not to be in control of the turbine at the time of attack initiation; enabling the diversity voting process, which is continuously searching for anomalies, to enable the security system to take action(s)—after attack initiation—to prevent the infected controller from taking control of the turbine. Note that this approach treats an infected controller in much the same way as a failed controller. This means that the additional costs for this form of protection are mitigated.

Data Consistency Checking

The application of data consistency checking, as suggested earlier, is that while an in-progress cyber attack can be successfully masked by manipulating the most prominent data, it would be extraordinarily difficult for an attacker to adjust all system data that might give indication of the attack in progress. The use of *Verifiable Voting*, as described in section 2.3, is the most direct application of data consistency checking. However, there could be checks across a broader set of system components that would also be revealing, and could become part of the dynamic management of system configuration. For example, in a nuclear power plant there are numerous measurements that occur for safety and operational reasons. It is possible to combine these measurements with turbine system measurements to reveal discontinuities, contributing to a more robust basis for signaling a cyber attack. The application of System-Aware security to a nuclear power plant would need to include an exploration of this opportunity as part of the system design process.

Tactical and Strategic Forensics

Should a subsystem in the turbine control system fail (i.e., be voted out of operation), the question would remain for the system owner as to whether the differentiation from its diversified peers was purposefully caused as part of a cyber attack, or was the result of a natural fault. This question must be addressed for cases where the consequences are limited (e.g., a component being voted out of service with no impact on turbine operation) as well as for cases where the consequences are much more significant (e.g., the turbine being damaged). There are three time constants associated with receiving answers to this question. One time constant is relatively short, and relates to the processes for immediate restoration of the failed component(s) or subsystem. This short period is referred to as tactical. Another time constant relates to management strategies for sustaining operations at other nuclear power plants that are using the same equipment as the potentially compromised system. A third time constant relates to issues surrounding the supplier and the use of equipment containing components from the supplier in question. These latter two cases are referred to as strategic.

This use case focuses on the tactical case addressing a turbine control system. For this case, one can implement specific hardware and software forensic capabilities that can be brought to bear quickly for restoral decisions. Normally, in current operations, a technician would find the part of the system that failed and, when needed, would take immediate action(s) to return the system to its normal operational mode. However, in the event that the failure was actually caused by a cyber attack, such action(s) might not be desirable. Instead, one might choose to restore the system by using one of the other vendors' controllers. This would reduce diversity, but would increase security during a period required for deeper analysis. In any event, one can develop a logical decision process for forensic analysis that could work as follows: (1) if a subsystem is voted out of operation, and (2) if at the time of the vote that component was producing signals that were within the normal specifications for equipment performance (i.e., the component was not “broken”), and (3) if the voted out signals had been applied to the turbine at the time of the vote they would have caused significant consequences, then a forensic analysis supporting the hypothesis of a cyber attack would be considered as conceivable. Specific forensic tools could provide additional information confirming that the software in operation at the time of the failure was identical to the software that was believed to be in operation. Another forensic tool could look for external signaling from an attacker to initiate the attack. This could potentially be accomplished by embedding a radio spectrum frequency analyzer into the hardware for the controller for the purpose of observing wireless communications signals. It may also be possible to discover data insertions into the protected hardware through the various data entry ports that are part of the hardware (e.g., serial ports, USB ports). Regardless of the mechanism, forensic tools can provide a useful contribution by adding confirming evidence in a situation that has the potential for being a cyber attack.

Solution Assessment

This section summarizes the benefits and cost of the overall System-Aware security solution for turbine control during pre-attack, trans-attack, and post-attack phases.

Pre-Attack

The usage of *Diverse Redundancy* as part of the System-Aware security architecture forces an attacker to create a network involving a larger number of suppliers than otherwise would be called for. It also requires the attacker to learn about more subsystem designs than otherwise would be necessary in order to design a successful attack. The usage of *Physical Configuration Hopping* as part of the solution requires the attacker to consider the impact of timing on the attack. This requires the attacker to possess greater knowledge about the influence of system dynamics on the technique employed for attack. It also points to the need for managing the attack initiation with greater precision than otherwise called for. Utilizing techniques such as *Verifiable Voting* as part of data consistency checking across broad segments of the entire system forces the attacker to consider more sophisticated data manipulation designs. Finally, the tactical forensic analysis for restoration support raises concerns about being detected even when the cyber attack fails. Together these elements could serve as a significant deterrent to attacks being directed at, in this case, the turbine control system. In addition to the cyber security benefits, the power system would be more reliable due to the added redundancy and diversity.

Negative consequences include an increase in system costs (one-time and life cycle), added complexity in system validation, and the possible need for more skilled technical support staff at the plant. In addition, design and evaluation for achieving "bumpless" performance for a turbine control system that incorporates diverse controllers will require design and evaluation activities before incorporating this approach.

Trans-Attack

The various elements of the System-Aware security architecture together increase the likelihood of identifying and disabling an attack. The suggested architecture can also provide the basis for post-attack forensics to play a role, by time stamping voting results and storing information that can help identify when a cyber attack has occurred.

Negative consequences could include the complications of keeping plant operators sufficiently informed about the system configuration and security outputs both in normal operation and during a possible attack.

Post-Attack

The application of tactical forensic information would guide system restoration actions so as to avoid a follow up attack. It would also influence strategic decision-making regarding the supplier and other on-going use of the problematic equipment.

Negative consequences include the development of confident methods for tactical forensic analysis that can be used by technicians at the plant, and could require significant training time and effort.

3.3 System-Security Aware Network Switch

This section introduces a possible System-Aware architectural solution for protecting a communications switch. This solution both explores how System-Aware security could be utilized to provide protection, as well as serves as a basis to explore the potential impacts of *Physical* and *Virtual Configuration Hopping*.

For a substantial number of ships, a communications network switch is central to the ship control system design. Networks offer the ability to easily distribute feedback measurements and control signals among diverse elements of the overall system, including Navigation, Steering, Propulsion, Electrical Power Distribution, Fire Detection, and Fire Suppression. However, the use of networks has created a potential vulnerability associated with automated control systems. For example, a denial of service attack on the network could result in the inability to control the engine or rudder, endangering the vessel and preventing mission essential functions. Such denial of service attacks are frequently levied against other systems [Lennon, 2011].

Potential Cyber Attacks Against Control Networks

As a preamble to discussing specific cyber attack scenarios, a simplified representation of a ship control system, including its embedded network, is introduced. Consider a system consisting of two operator control stations, two network switches, two propulsion controllers, and two engines under control. This simplified control system representation consists of eight components and is illustrated in Figure 13.

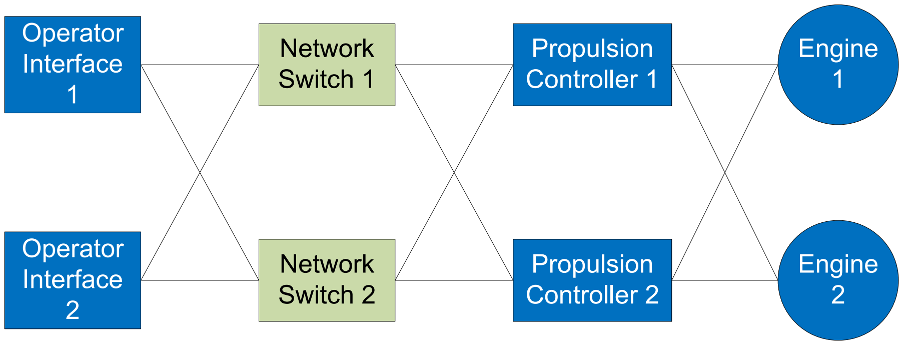


Figure 13. Simplified representation of a ship control system. This includes two operators controlling two engines over a network. For this simplified example, the network is composed of two a network switches to provide fault tolerance; i.e., on of the switches is acting solely as a backup.

For this simplified system, the propulsion controller provides local control of each engine. It acts upon commands from the network switch to change speed or perform other high level functions such as start or stop. Once the controller receives a command from the network, it performs the detailed control of the engine to achieve the command. Furthermore, it is assumed that Network Switch 2 will assume the roles of Network Switch 1 in the event of a failure to that switch.

Now consider that the network switch has been subjected to a cyber attack, disabling the ability to communicate control requests for changes to engine speed. Since the system has been designed with a second switch, Network Switch 2, if uncompromised, would take over, thereby avoiding immediate adverse impacts to system performance. However, in redundant designs the replicated components usually have the same design, implementation, and supplier. This provides economic advantages for the purchase and integration of a control system, and also results in reduced spares and training for maintenance personnel. However, in this example, it becomes possible for both switches to be successfully attacked with a single exploit, resulting in a situation where the control system would be disabled and no further control actions could be performed. In addition, it is possible that the switches have been compromised in such a way that the operator is unaware of the failure. The result to the ship might be increased uncertainty and result in a longer time to address the issue, putting the ship at greater risk.

A System-Aware Cyber Security Solution

To address this class of vulnerabilities, a System-Aware Cyber Security solution is suggested. For the purposes of this example, the design patterns of *Diverse Redundancy* (section 2.2) and *Physical Configuration Hopping* (section 2.4) are utilized in the suggested architectural solution.

The suggested architectural solution is shown in Figure 14. In order to make it more difficult for the attacker to compromise all of the network switches, *Diverse Redundancy* is introduced by substituting an alternate implementation for Network Switch 2. This configuration requires an attacker to develop separate exploits to accomplish the same result as discussed earlier. In addition to *Diverse Redundancy*, *Physical Configuration Hopping* is introduced into the security design through the implementation of a supervisory process which causes the active communication switch to be dynamically hopped between Network Switch Model A to Network Switch Model B. However, as approximately two-thirds of all communication is done via UDP, there exists the possibility that this dynamic hopping will result in information being lost. If enough information is lost it could adversely impact system performance. Thus, the frequency of hopping is a balance between the overhead and dropped messages from hopping too frequently and the risk exposure to not hopping fast enough.

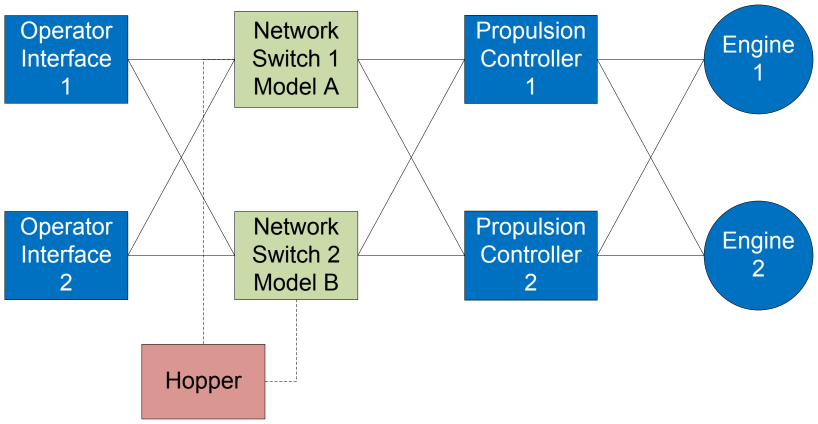


Figure 14. Simplified network switch configuration shown in Figure 13, augmented with a System-Aware architectural solution. This solution includes the usage of *Diverse Redundancy* and *Physical Configuration Hopping* applied to the network switches. *Physical Configuration Hopping* is controlled via the Hopper.

Impact of *Physical Configuration Hopping* on the Performance of the Network

To determine the feasibility of the architectural solution, analysis was performed to address whether the hopping between switches would have an adverse impact on system performance. System performance would be adversely impacted if the control functions were disrupted to the point that they were compromised. To establish whether control functions were compromised, the following criteria were developed, including (1) critical messages could not be lost, (2) information critical for control functions could not be delayed beyond the point that the control function was adversely impacted (<25ms), (3) message loss be limited (<1%), and (4) information critical for review by an operator could not be delayed beyond the point that the operator would lose confidence in the control system (<1000ms). Furthermore, in order to provide additional security to the system it is desirable that configuration hopping be as fast as possible while sustaining control over the system.

To ensure that critical messages are not lost, one can consider the fact that message loss can occur even in a configuration where no hopping is occurring due to normal network activity such as message collision. For this reason, control systems are designed to ensure that critical messages are retransmitted. Therefore, it was determined that the first criteria would not be impacted by hopping unless the messages were also delayed; i.e., the second criteria.

To address the second criteria, one can consider the types of systems to be controlled over the network and the typical rates at which they operate. Consider the example of a ships heading which is used in a closed loop control by the autopilot to control the rudder for steering. The commercial specification for ships heading only requires heading to be updated at 1-10 HZ for large vessels and 40 HZ for a high speed craft [International Maritime Organization, 1995]. Thus, in the worst case (High Speed Craft) messages could not be delayed beyond 25ms.

To address the third criteria, one can considered that systems are designed to handle loss of messages. For non-critical messages such as those containing information about continuous measurements, such as temperature, the loss of a single message is not critical. In fact, implementations often use a non-guaranteed method of delivery for these types of messages such as UDP. For messages like temperature measurement, using something similar to UDP, we considered losses up to 1% to be acceptable. For the messages requiring ensured receipt, such as offered by Transmission Control Protocol (TCP), any message loss will be automatically handled by the transmission protocol and resent.

To address the fourth criteria, one can consider that in multiple systems, the operator interface was only updated at a 1 HZ rate. Therefore information for the operator cannot be delayed more than 1 second. This is a relatively slow rate for the systems controlling ships and is not a major factor since the automatic control requires much smaller delays.

Experimental Results

To explore the relationships between hopping, lost data, and delayed data, a laboratory experiment was conducted using a combination of emulation and simulation to model a simple ship control system. The decision to focus on the messages of the automated control functions and operator messages was due to the use of UDP to transmit information. As explained above, the UDP protocol does not ensure delivery, and thus carries the potential for packet loss. In addition, the vendors of network switches can employ a variety of hardware, software, and configuration settings to optimize the performance of their product offerings. This can possibly result in different rates of forwarding packets. Thus, when the system dynamically reconfigures from one switch to another, this can result in packets being dropped and information being lost. This can possibly lead to negative impacts on the performance of the automated control systems and operator interfaces as messages are lost, which, in turn, can lead to updated status information being delayed.

Emulation was used to analyze the performance of diversely redundant network switches while simulation was used to model the traffic generated by the automated control subsystems and operator messages. The network switches were emulated to capture all of the possible effects due to a vendor's design choices. Simulation was used to generate the network traffic as it was not necessary to accurately model the specific information sent in each packet nor emulate the actual control actions taken by the automated control systems or operator interfaces.

For this experiment, a typical traffic load for an automated ship control system operating under normal conditions was generated. This involved the simulation of 100 separate channels of information, each generating traffic at a rate of 250 Kbps and sending data packets of 1KB in size; i.e., a total traffic load of ~25 Mbps. In addition, approximately one third of the traffic is TCP and two thirds of the traffic load is UDP. Thus, 33 of the 100 channels sent their information using the TCP, and 67 of the 100 channels sent their information using the UDP. This traffic load of ~25 Mbps was sent over two diversely redundant gigabit network switches for one hour (i.e., network switch one had a load of ~25 Mbps and network switch two had a load of ~25 Mbps). It is noted that these switches are over specified for the traffic load;

however, this is typically the case for automated ship control systems. Over the course of the hour the sending and receiving time of every packet was recorded. This information was then used to estimate the number of packets that would have been lost if various configuration hopping rates had been applied.

Figure 15 shows the number of UDP packets lost per 10,000 for a series of ten separate experimental runs for configuration hopping at rates of five, ten and twenty seconds. A reconfiguration rate of five seconds was chosen as being sufficiently fast enough to detect the erroneous behavior of a network switch before permanent damage could be caused to the ships systems. Ten and twenty second reconfiguration rates were selected to assess the relationship between performance impact of reconfiguration and its impact on performance.

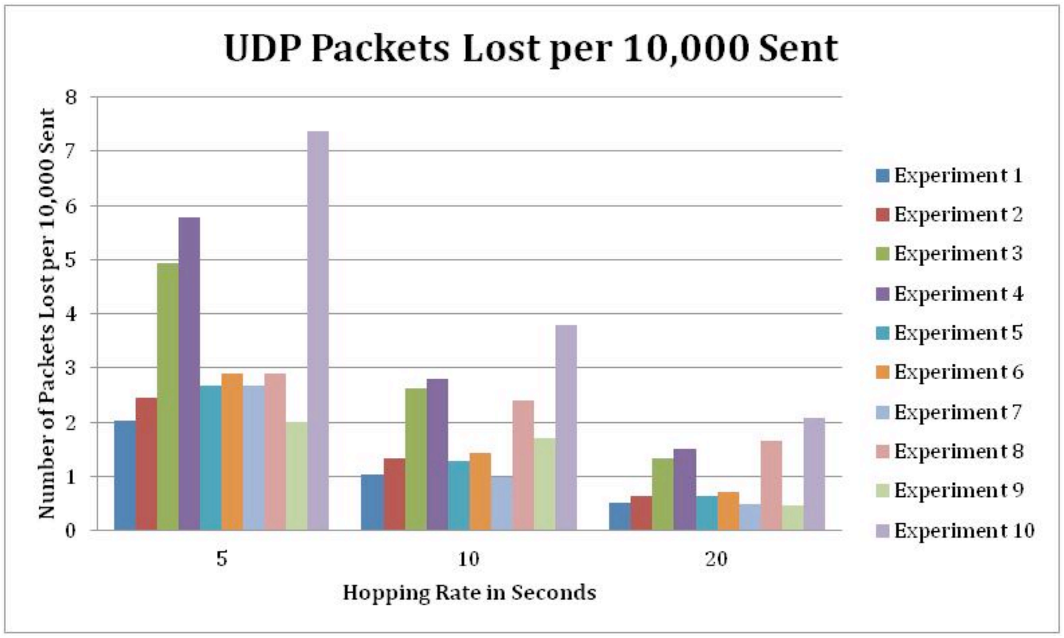


Figure 15. UDP packets lost per 10,000 sent due to configuration hopping for a set of 10 experiments. The packet losses for each experiment are shown for reconfiguration rates of five, ten, and twenty seconds.

As seen in Figure 15, the number of packets lost due to configuration hopping is minimal. Thus, the impact of packet loss on operator messages and its corresponding affect on the delay of automated control system messages is minimal. In addition, it can be observed that the number of packets decreases approximately linearly as the rate of reconfiguration increases. This linear relationship is expected, as the rate of reconfiguration is relatively large compared to the rate at which information is being generated, and the traffic load is low compared to the network switch capacity. As a result, each instance in which the system reconfigures can be considered independent of the last. Finally, it is noted that the average packet delay is ~10ms across all experiments.

3.4 Facility Protection Surveillance System

To provide a proof of concept of the System-Aware security solutions introduced in Chapter 2, a flexible prototype system was developed. As seen in Figure 16, this prototype system was designed to provide surveillance for a given region through the use of distributed sensors that provided continuous monitoring. The data collected by the sensors is streamed to a server that analyzes the data and automatically determines whether an unauthorized intrusion has occurred. Furthermore, when the server detects an unauthorized intrusion it issues an alert to security personnel, as well as directs the sensors' data stream to the appropriate security personnel over a wireless communication network. The security personnel can then utilize this information to make decisions about how to best respond to deal with the possible intruder.

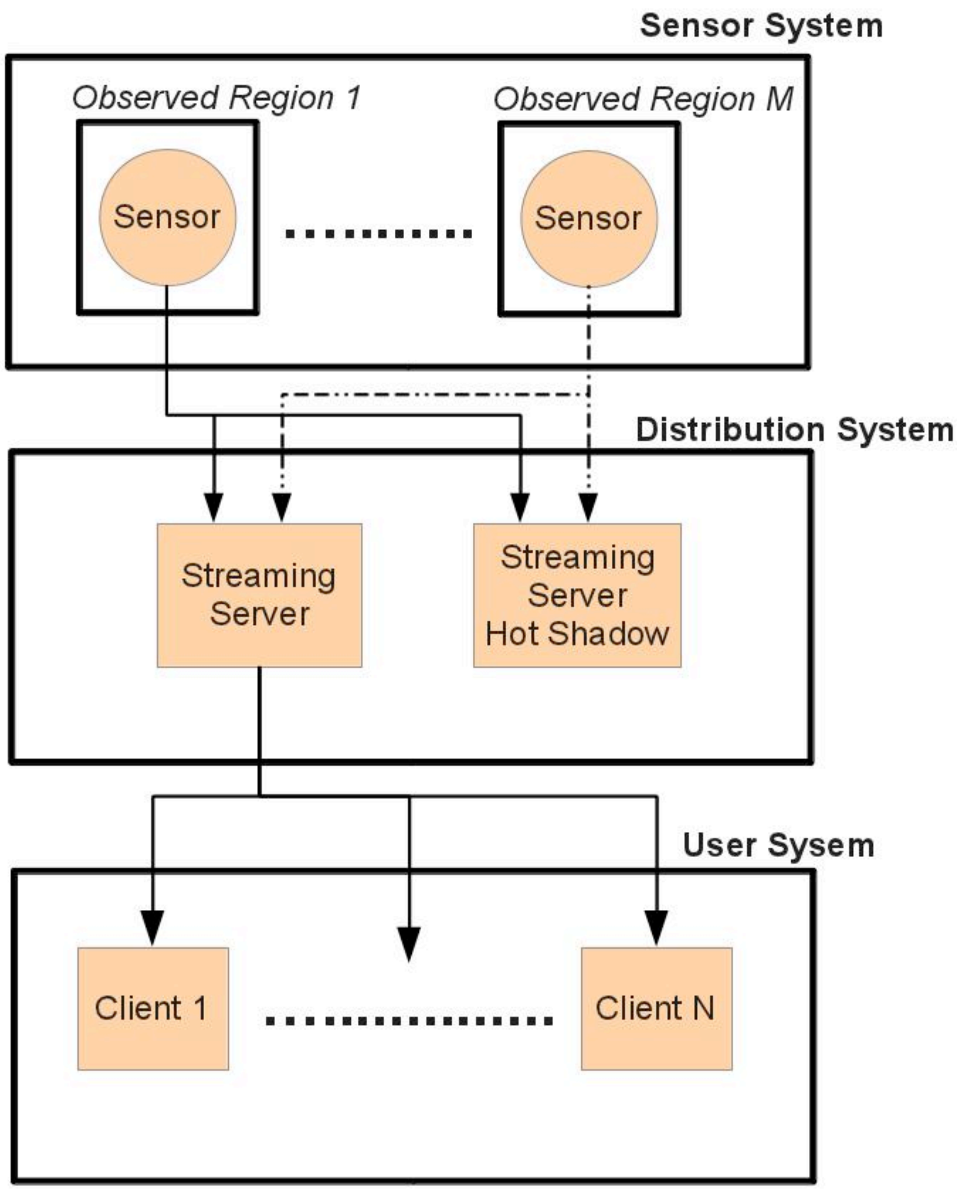


Figure 16. An architectural diagram for a prototype surveillance system. The surveillance system is composed of *M* streaming sensors sending data to a set of redundant servers. This data is analyzed, augmented, and forwarded to mobile clients via a wireless communications network.

For this prototype system, it was assumed that an adversary was attempting to circumvent the system's security through the usage of an embedded Trojan horse that was capable of performing a replay attack. Specifically, it was assumed that the Trojan horse could be embedded into any one of the subsystems; i.e., the distributed sensors; the servers used for receiving information from the sensors, detecting unauthorized access, issuing alerts to security personnel, and forwarding the sensors data to the security personnel; or the end user devices used by security personnel to receive alerts and view the information sent by the distributed sensors. The assumed exploit would be able to perform a replay attack; i.e., take previously recorded sensor information and resend it in place of the live stream. For example, the attack could continuously send a thirty-second loop of video of an empty region in order to make that region appear as if there were no intrusions in progress. In addition, for the prototype system, it was assumed that an attacker could activate the Trojan horse remotely.

To protect the prototype system from an embedded Trojan horse, a System-Aware security solution was integrated into each of the subsystems. This included the introduction of multiple redundant (i.e., three or more) diverse streaming sensors and servers, as well as diversifying the client devices. *Diverse Redundancy* (see section 2.2) forces an attacker to create a network involving a larger number of suppliers than otherwise would be called for. It also requires the attacker to learn about more each of the subsystems designs than otherwise would be necessary in order to design a successful exploit. An intelligent *Verifiable Voting* (see section 2.3) process was integrated into the client devices as well as the streaming servers in order to detect any misleading information. Voting at the client devices was utilized to detect a compromised streaming server. Voting at the streaming server was utilized to detect a compromised streaming sensor. *Physical Configuration Hopping* (see section 2.4) was integrated into the streaming sensors to time-vary which of the streaming sensors would be sending its information to the servers at a given time. *Virtual Configuration Hopping* (see section 2.5) was integrated into the streaming servers to time-vary which of the servers would be sending information to the client devices at any given time. Configuration hopping was utilized in order to require the attacker to consider the impact of timing on the attack, as well as force the attacker to possess greater knowledge about the influence of system dynamics on the technique required for a successful attack. The integrated solution design can be seen in Figure 17.

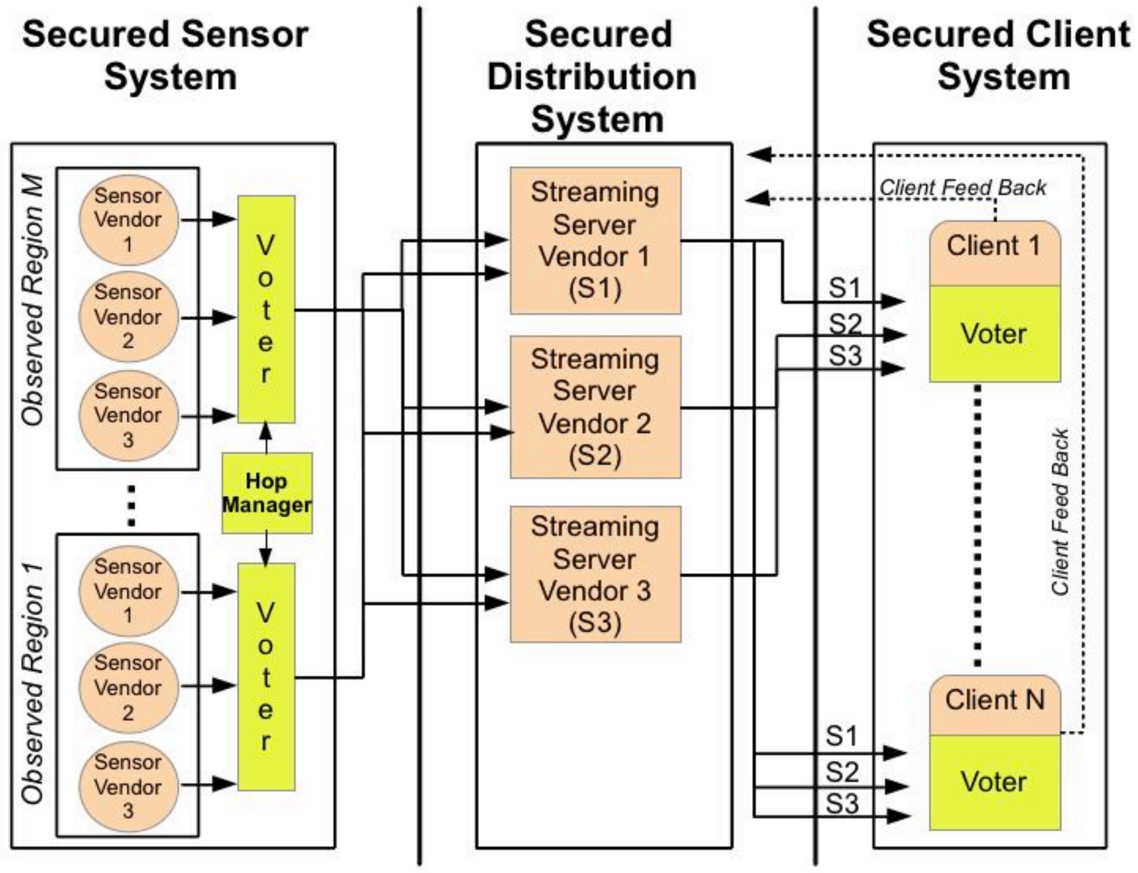


Figure 17. An example System-Aware architectural solution for protecting the system shown in Figure 16. This solution includes *Diverse Redundancy, Configuration Hopping (Physical and Virtual), and Verifiable Voting.*

In order to explore the potential collateral impacts of such a solution, an instantiation of the prototype system was implemented. This system utilized live video surveillance in order to provide security for a facility; i.e., two web cameras for the streaming sensors, two virtual media servers for the steaming servers, and a combination of laptops and cell phones for the client devices. The web cameras were connected to the media servers over a wired network, while the media servers were connected to the client devices via a standard 802.11g wireless network. The decision to utilize live video as the streaming data was made due to its potentially demanding resource requirements on the handheld devices carried by the responders.

During the testing of the implementation, the System-Aware security solution was able to successfully detect when the Trojan horse (i.e., replay attack) was activated and hop to a new configuration. However, testing also revealed that the *Verifiable Voting* has the potential to have a significant impact on system performance; in order to ensure that a client device was able to detect a media server transmitting a misleading stream(s), each responder's handheld device had to receive at least two streams simultaneously. As shown in Figure 18, this requirement to send multiple video streams to each user device can result in a significant amount of bandwidth consumption. This in turn, can lead to a drastic reduction in either the number of regions that can be observed or the quality of the video sent to each user device.

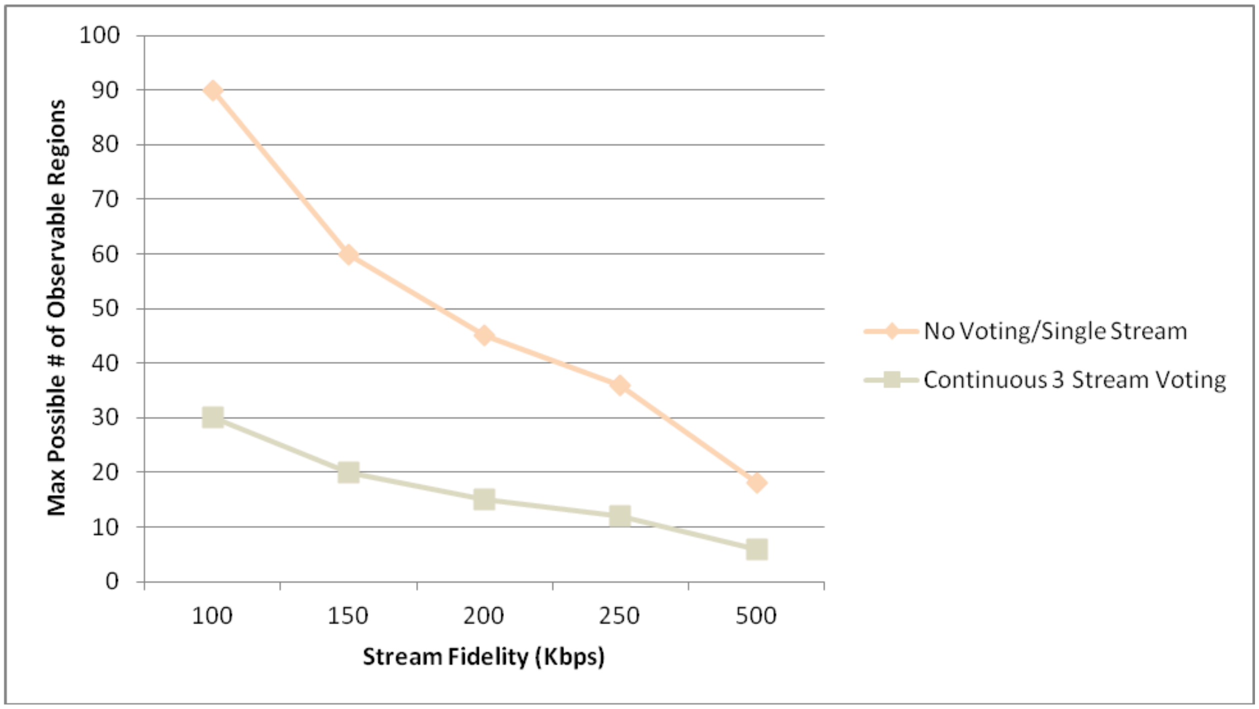


Figure 18. Shows the number of regions that can be observed for a given sensor stream fidelity with voting and without voting.

For the prototype system, it was decided that the extra bandwidth needed between the streaming servers and the responders' devices was deemed unacceptable due to the limited bandwidth available over the wireless network (11 Mbps)—the wired network provided more then sufficient bandwidth for *Verifiable Voting* between the sensors and the streaming servers. Furthermore, it was desirable to provide a solution that did not rely on the usage of additional hardware to provide more bandwidth or segregate the devices among multiple channels. This was due to the fact that, it may not always be possible to integrate additional hardware to increase the bandwidth, and a solution not dependent on specific hardware has the potential to be applicable to wide range of systems that could potentially benefit from *Verifiable Voting.* To mitigate the impact of *Verifiable Voting,* a modified voting scheme was introduced. This scheme aimed to limit the bandwidth consumed due to the introduction of *Verifiable Voting* by only allowing a sub-set of the client devices to perform *Verifiable Voting* for a fixed amount of time; i.e., separate voting among the devices in a time division multiple access (TDMA) manner. An example of this duty cycle voting scheme is shown in Figure 19, where only one device is allocated to perform *Verifiable Voting* at a time (client device one in Figure 19). Once the time allocated for voting had expired, a different set of devices could be selected to perform *Verifiable Voting*. As seen in Figure 19, the division of time among the client devices is managed by introducing a new Duty Cycle Manager. This manager is responsible for deciding both when a specific device will be allowed to vote, as well as for the amount of time available for collecting the streaming data that will be voted on. The amount of time available for collecting information is important as the data is being streamed; thus, as the amount of time allocated to collection of information is increased, so is the amount of data available for voting on the streams in order to detect and isolate a misleading stream. Streams of longer duration potentially make the detection of a corrupted stream more likely, while simultaneously reducing the likelihood of a false alarm. However, as the amount of time allocated to each device to collect information is increased, so is the amount of time it will take to rotate through all of the devices. For example, if there are 10 devices and each device is given 1 second to collect information, it will take 10 seconds before every device has gathered the necessary information to perform *Verifiable Voting*. Alternatively, if there are 10 devices and each device is given 6 seconds to collect information, it will take 1 minute before each of the devices has collected enough information to perform *Verifiable Voting*. Thus, while such a scheme can potentially result in a significant amount of bandwidth saving (see Figure 20), it also introduces the need for additional analysis:

- How many devices should be allowed to perform verifiable voting at once
- What length of time should be allocated to perform verifiable voting
- Does this amount of time yield a desirable detection rate
- Does it yield a desirable false alarm rate

- What is longest duration of time before a device will have the opportunity to detect a misleading stream

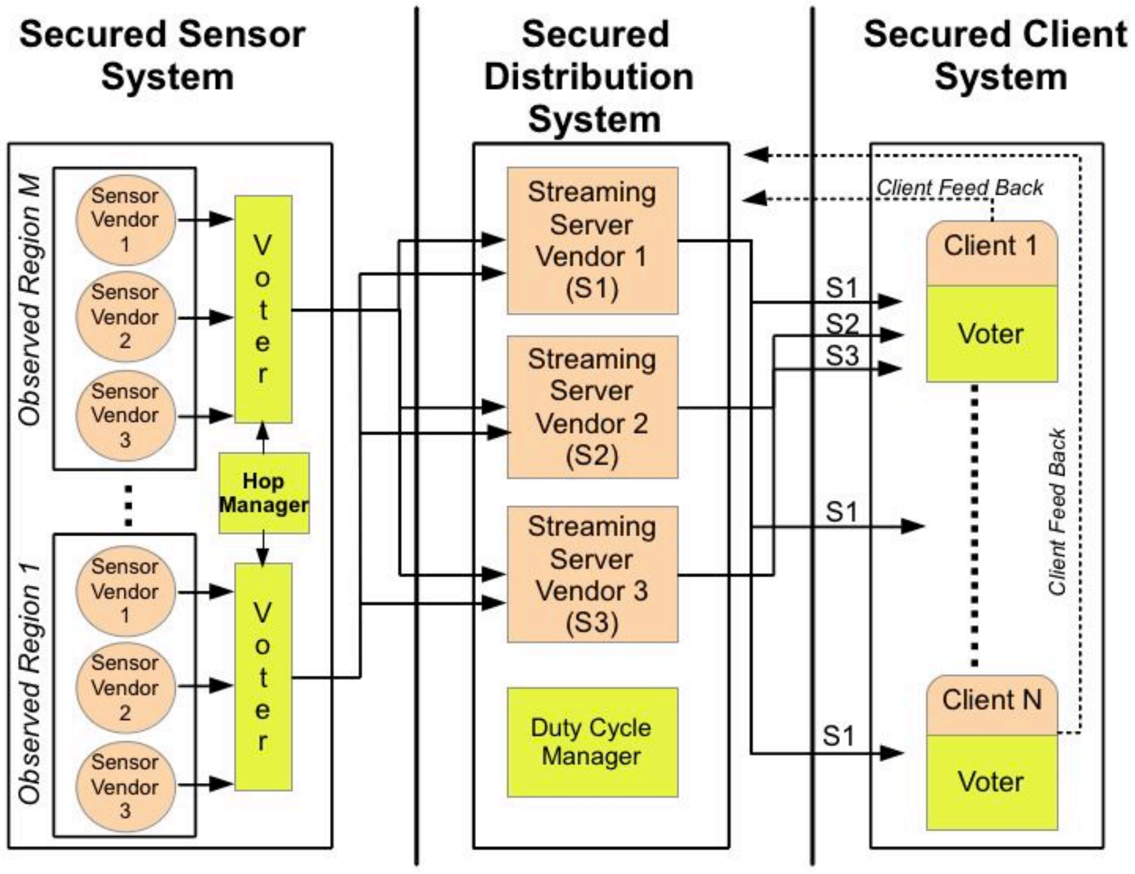


Figure 19. Shows the System-Aware solution shown Figure 17 augmented with a duty cycle voting scheme. For this particular example, the duty cycle voting is done only between the streaming servers and the client devices. In addition, this process is managed by a special Duty Cycle Manager.

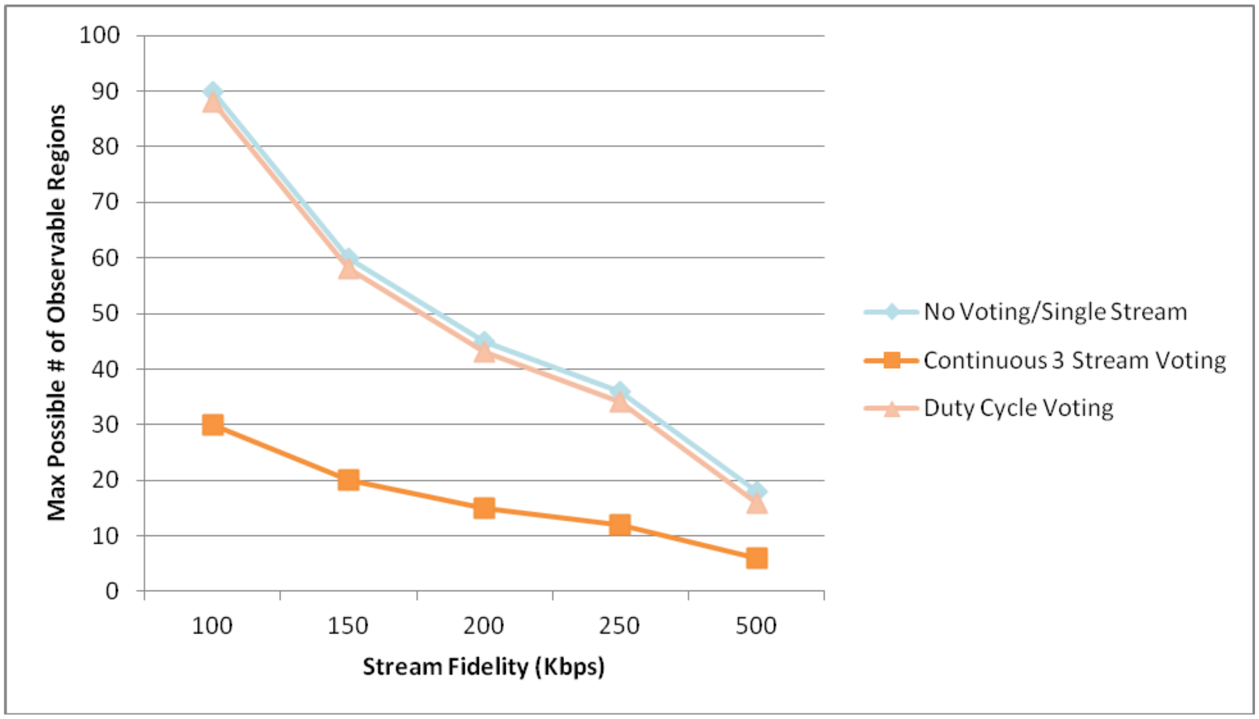


Figure 20. Shows the number of regions that can be observed for a given sensor stream fidelity with voting, without voting, with a duty voting scheme where only one device is voting at any given instant.

Chapter 4
Metrics

4.1 Introduction

Chapter 1 and Chapter 2 discussed how System-Aware Cyber Security can address the threats of infections embedded in mission critical systems. This was illustrated in Chapter 3 through the use of three examples showing how System-Aware security services could be used to mitigate specific threats in a nuclear power plant turbine control system, network switch, and surveillance system for protecting facilities. This included a discussion of how these security services would increase the difficulty to an adversary and provide the basis for a high-level assessment of the benefits and cost. However, as detailed in Chapter 2, multiple System-Aware security design patterns could be candidates for an integrated security system architecture for addressing threats. Thus, there is a call for security analysis methodologies that are able to compare alternative system security architectures accounting for the selection and integration of security design patterns as well as the details of specific implementations. The desired methodology must include an array of methodological elements for assessing: (1) the level of security afforded by a given System-Aware security architecture (e.g., how much more difficult has this solution made an adversary's task and how rapidly can system restoration be accomplished); (2) the collateral impacts on the system as a whole (e.g., performance, ease of operation, and cost); (3) identifying those system functions that warrant the most protection from a risk perspective; (4) evaluating, from a security perspective, the hardware/software implementation of the System-Aware security architecture (e.g., avoidance of buffer overflow opportunities in the implementing software); and (5) evaluating the life-cycle management plan for the System-Aware security architecture, including approaches for responding to discovered security solution design flaws and to overall system design enhancements that occur over the life cycle. With regard to risk based security protection, standards exist for conducting risk based security analysis [ISO/IEC, 2009; Stoneburner, Goguen, and Feringa, 2002]. Similarly, industrial methodologies exist for software implementation and software patching over the life-cycle of a security solution [Howard, 2002; Stackpole and Hanrion, 2007]. However, to completely address the five elements outlined above, the System-Aware security approach requires a supporting methodology for the assessment of the level of security potentially afforded by a given security solution. This section presents an initial outline for a scoring methodology to assess and compare System-Aware security architectures, building upon existing work in the field of safety.

4.2 Background

Just as digitization has created new challenges in the field of cyber security, it has also created new challenges in the field of safety systems. One such challenge is causally related failures of redundant or separate equipment. When systems were analog, these common-cause failures (CCFs) were typically caused by slow moving processes such as corrosion. However, as systems increasingly became digital, software design flaws and bugs arose as a new source for CCFs. The nuclear safety community mitigated the risk posed by this increasing threat by employing multiple types of diverse redundancy, including design diversity, human diversity, and software diversity. This new solution created a call for a methodology for comparing alternative designs and assessing the level of CCF risk mitigation provided by a given design.

One methodological solution is employed by the Nuclear Regulatory Commission (NRC), as documented in NUREG/CR-6303. This methodology provides system designers and implementers with a guide to achieve resilience to CCFs through use of a procedure related to mitigation of consequences. This is achieved by assessing the amount of diversity offered by a particular system's design—based on the principle that more diversity in a system results in less susceptibility to CCFs. A system's diversity is evaluated through the application of a weighted rank ordering process that utilizes a wide range of criteria, such as differing technologies, similar technologies within different architectures, and different manufacturers of fundamentally different designs.

A weighting scheme has been determined based upon assumptions and principals derived from designers' experiences with avoiding CCFs. While the method is not supported by an underlying mathematical theory, the results permit the NRC and system designers to engage in a constructive dialogue regarding the attention paid in a specific design regarding the avoidance of CCFs.

4.3 Unique Challenge Posed by Cyber Security

As shown in section 1.3.1, security solutions must address cyber attacks that can concurrently exploit common redundant components, leading to diverse redundancy as one of the security services employed by System-Aware Cyber Security. Recognizing the employment of diverse redundancy utilized by safety engineers for addressing CCFs, one can look to the significant efforts of the safety community as a source for an initial scoring methodology that can be enriched for cyber security application. However, there are several critical challenges unique to cyber security that requires a CCF-based foundation for scoring to be enriched before it can be effective. First, while CCFs can be addressed solely through diverse redundancy, as indicated earlier in the discussion of design patterns, security solutions must include additional solution components, that go beyond the application of diversity, in order to fulfill its functions. Second, unlike CCF solutions, cyber security solutions attempt to deter, deflect, and restore a system against an intelligent adversary exploiting available vulnerabilities, including the capability to assess the cause of failure indeed being a cyber attack. Finally, a variety of design patterns, including D*iverse Redundancy*, can be integrated into solutions, thereby requiring a scoring methodology that establishes criteria for assessing and comparing the value contributed by the individual elements of the broader solution space.

4.4 Outline for a Possible Scoring Model

Using NUREG/CR-6303 (outlined in section 4.2) as a starting point, and given the unique challenges outlined in section 4.3, this section provides an outline of an initial methodology for developing scores for comparing and assessing System-Aware security architectures. This scoring methodology involves addressing three key factors: (1) for each of the individual security services within a System-Aware security architecture, identifying the potential contribution and its importance to the overall security being offered, (2) determining the potential effectiveness of each security service within a particular System-Aware security architecture, and (3) evaluating the cost and collateral impacts of the solution services on the system's normal operations. Multiple methods can be utilized to evaluate the resulting value of an integrated set of security services combined into a System-Aware architecture. For the purposes of this thesis, a simple linear model is introduced for combining the values of individual security services into a resulting score.

4.4.1 Identifying the Security Contribution of Individual System-Aware Security Services

Every System-Aware security architecture is composed of an integrated set of design patterns. Each of these patterns enhances the security of the system being protected by (1) deterring the attacker (pre-attack); (2) identifying, isolating, and preventing malicious attacks (trans-attack); and/or (3) aiding in the restoration of the system to a non-compromised state (post-attack). For example, in the turbine control example outlined in section 3.2, *Physical Configuration Hopping* is a pre-attack, trans-attack, and post-attack design pattern that contributes to restoration, deflects attacks by preventing them from taking potentially damaging action(s), and deters an attacker by making the attack more complex to design and develop. *Diverse Redundancy* is a pre-attack and post-attack pattern that deters an attacker by making the attack more difficult to design and execute, and also aids in restoration through the employment of diverse elements that have not been compromised. The desired methodology should include the classification of every design pattern in a given System-Aware security implementation in terms of its contribution to security and the stage(s) of attack for which it is designed to operate. This enables system owners and operators to analyze the tradeoff between the type of security offered by a given System-Aware security architecture, and the impacts this has on their system. For example, a system owner concerned about interfering with the normal performance of their system may select a given System-Aware security architecture that provides a significant amount of low interfering post-attack capabilities and no high interfering pre-attack and trans-attack capabilities. In addition to providing restoration, this solution provides a higher likelihood that an attacker will be caught and potentially a significant amount of deterrence, while maintaining a minimum impact on the system's performance. In contrast, the architects and operators of a more mission critical system may choose an implementation that offers a significant number of pre-attack, trans-attack, and restoration services to ensure maximum availability, even if this somewhat degrades or increases the cost of the normal operations of the system.

4.4.2 Assessing the Potential Effectiveness of Individual Security Services

As shown above, it is possible to evaluate a given System-Aware security architecture in terms of the contribution to security offered by a particular design pattern, as well as which phases of the attack it is effective. However, there still remains the issue of assessing the efficacy of a particular architecture. One possible means for achieving this objective proceeds as follows. First, it would be necessary to determine the mission critical functions to be protected in a particular system. One such method is discussed in section 4.5.

However, it is not enough to ensure that a particular System-Aware security architecture would provide some amount of defense; it is also necessary to evaluate the potential efficacy of particular solutions. For example, as described in section 2.4, *Physical Configuration Hopping* forces an attacker to operate within a given time interval. This could add value to security whether or not other security services are employed by a particular System-Aware security architecture. However, the efficacy of *Physical Configuration Hopping* in the absence of other security services depends upon the predicted attack execution difficulties imposed on an adversary. If it is believed that a principal difficulty to an adversary is the exact timing of the attack, then implementation of *Physical Configuration Hopping* as an isolated security service might prove to be highly effective. However, if triggering the potential attack(s) is predicted to be simple, and does not include stringent timing requirements, then configuration hopping will be less effective and contribute most to system restoration functions.

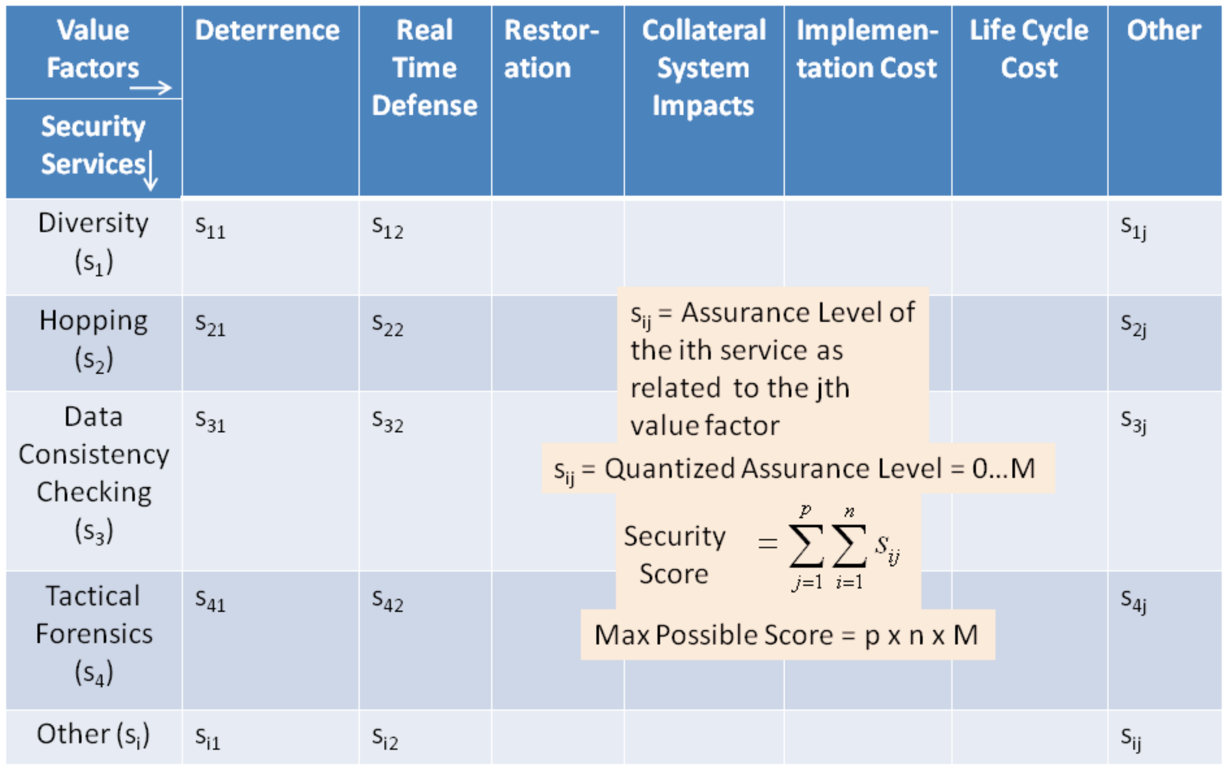
4.4.3 Impacts

It is generally expected that security solutions will increase the implementation and life cycle cost of a system. It is also expected that security solutions will require additional resources—CPU, memory, bandwidth, etc.—and in some cases could potentially degrade the overall performance and ease of operation of the system to be protected. In addition, since System-Aware security architectures provide solutions embedded into the system to be protected, they may also introduce intricate collateral impacts on the system. For example, as outlined in section 3.2, a potential solution to protect a nuclear power plant turbine control system involved dynamically switching control between a diverse set of vendor controllers. This solution improves the security of the system by making it harder for an adversary to gain control of the system, but also introduces the need to make the handoff between controllers bumpless in order to ensure the proper functioning of the turbine. Another example of the possible collateral impacts introduced by a System-Aware security architecture was illustrated in section 3.3, where *Physical Configuration Hopping* was performed between communication switches. As in the case of the turbine controller, hopping across diverse communication switches reduces the possibility that an attacker will gain control of the system; however, it can also result in information loss if the switches being hopped are not synchronized. This loss of information is a collateral impact that requires additional analysis—how much information is expected to be lost and how important is that information—and a solution—should the system owners and operators just accept the loss of information or implement solutions that better ensure switch synchronization.

Collateral damage need not be purely technical. In the turbine control example tactical forensics were utilized to help technicians distinguish faults from malicious attacks (see section 3.2). This not only requires new tools and techniques, but also requires the introduction of new policies and procedures. This in turn, introduces the need for additional training for technicians on how to properly utilize these new tools, techniques, policies, and procedures to distinguish faults from malicious attacks.

4.4.4 Architectural Scoring Framework

Figure 21 is a possible representation of what an architectural scoring framework containing the elements outlined in section 4.4 would produce: a table where each row contains a score regarding the value of a design pattern and each column a specific value factor. As outlined in Chapter 2, each of these design patterns enhances the overall security of the system, while also affecting the system's performance and cost to a varying degree. These effects are represented in the table as value factors. For example, Figure 21 utilizes four System-Aware design patterns: diversity, configuration hopping, data consistency checking, and tactical forensics. Each of these patterns can enhance the system's security through increased deterrence, greater real-time defense (i.e., deflection), and/or improved restoration capabilities. In contrast, each design pattern may negatively affect a system through collateral impacts, increased implementation cost, and/or increased life cycle cost. Finally, each pattern may provide positive collateral impacts, such as reduced CCFs and improved reliability.



Value Factors → / Security Services ↓	Deterrence	Real Time Defense	Restor-ation	Collateral System Impacts	Implemen-tation Cost	Life Cycle Cost	Other
Diversity (s_1)	s_{11}	s_{12}					s_{1j}
Hopping (s_2)	s_{21}	s_{22}					s_{2j}
Data Consistency Checking (s_3)	s_{31}	s_{32}					s_{3j}
Tactical Forensics (s_4)	s_{41}	s_{42}					s_{4j}
Other (s_i)	s_{i1}	s_{i2}					s_{ij}

s_{ij} = Assurance Level of the ith service as related to the jth value factor

s_{ij} = Quantized Assurance Level = 0…M

$$\text{Security Score} = \sum_{j=1}^{p} \sum_{i=1}^{n} s_{ij}$$

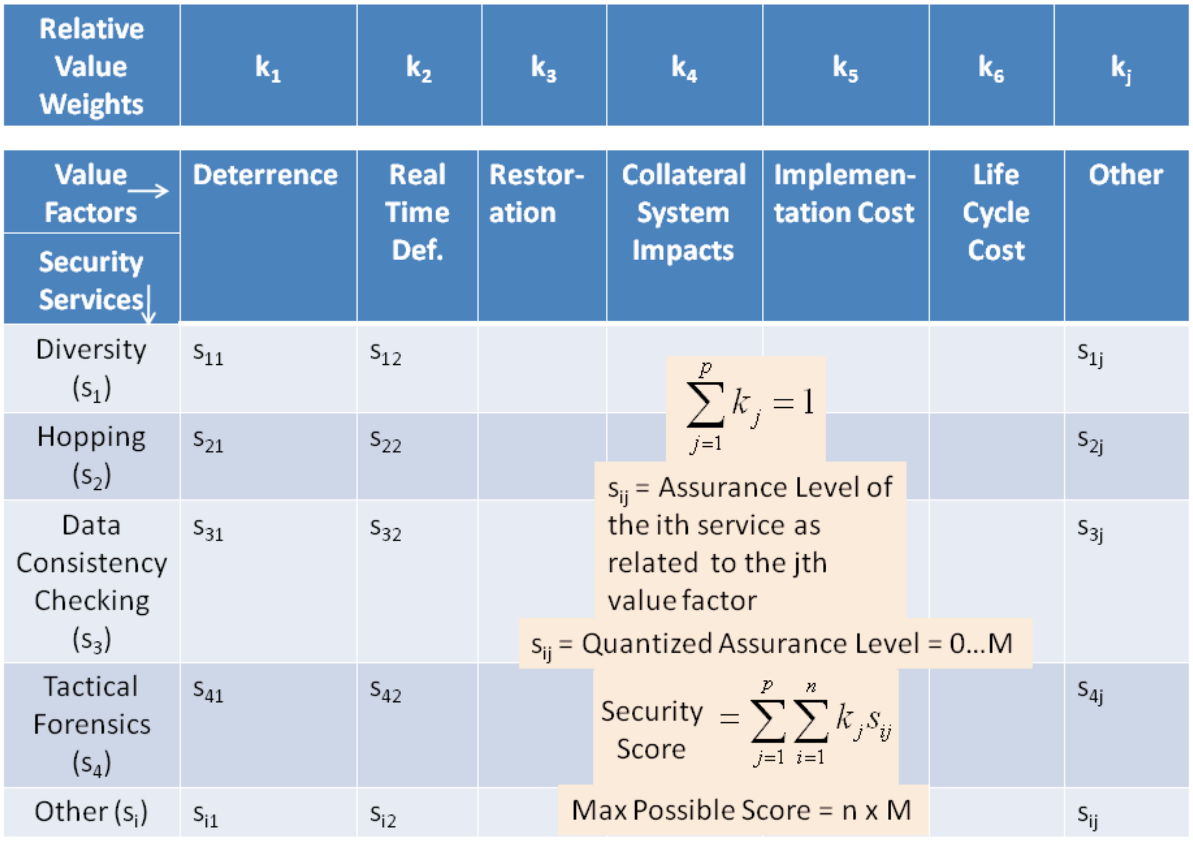
Max Possible Score = p x n x M

Figure 21. A possible representation of the scoring elements outlined in 4.4. Table composed of System-Aware design patterns and value factors. Each pattern-value factor pair is given an assurance level, *s*, based upon how the service level effects the given value factor.

To assess how design patterns affect value factors, a security assurance level, s , is assigned to every design pattern, i , based upon its contribution to a given value factor, j , yielding specific service assurance scores s_{ij} . Recognizing that System-Aware security architectures offer security values and unavoidable disvalues (e.g., increased cost), scores can be provided related to system impacts as well as security. Larger scores can be used to represent greater security value added or less collateral disvalue added, depending on the value factor being scored. It is noted that alternative scoring means could be utilized. For example, these scores could be represented as negative values. The assurance score is a discrete value selected from a range (0 to M inclusive) determined by a desired level of granularity. For example, an assurance level s could take on the value 0 or 1. This has the benefit of making a given solution easy to evaluate, but it would only be possible to compare alternative solutions based upon whether or not a given benefit or cost was offered by a given solution. However, if s could be a varying value, for example between 0 and 5, then it would be possible to compare the level of security offered by alternative solutions. This would require a more complete analysis to assign a given assurance score. For example, in the turbine control example outlined in section 3.2, there were two possible ways to provide diversity; one through controllers purchased from separate vendors and one through diverse components within a single vendor's controller. If s could only take on a value of 0 or 1, then both solutions would result in the same score, since the diversity affects the same value factors. However, if s could be any value from 0 to 5, the two solutions could score differently as the level of assurance provided by the differing diversity methods could be different.

Assuming a method for deriving individual scores, s_{ij}, several pieces of information can be derived from this scoring methodology. First, it is possible to derive a single security score for a given solution, $\sum\sum s_{ij}$, and compare it to a theoretical maximum score ($\sum\sum s_{ij} \leq i * j * M$). Second, it is possible to evaluate the strengths and shortfalls of a given security solution. For example, it is possible to evaluate whether a given solution is more effective in addressing real-time defense or restoration. Finally, recognizing that different combinations of value and disvalue scores require a multi-objective solution selection approach, it is possible to compare scores across alternative security solutions addressing a specific security need.

In addition to providing a method for scoring and evaluating multiple System-Aware architectural designs for securing a given system, the methodology can be augmented to also recognize that every system owner and operator may have a different perspective on the importance of different factors when securing their system. For example, some owners and operators may desire solutions that minimize the collateral impacts to the system, while others may seek solutions that maximize the security. To take these differences into account, system owners and operators can adjust scored System-Aware architectures to account for their individual assessments of which factors are most important by providing a set of relative value weights. A relative value weight, k , can be assigned to each value factor, j , such that $\sum k_j = 1$. Figure 22 provides a representation of a scored architecture filtered by a set of relative weights. As illustrated in the figure, it is still possible to derive a single security score for a given solution, $\sum \sum k_j s_{ij}$, and compare it to a theoretical maximum score ($\sum \sum k_j s_{ij} \leq i * M$). Furthermore, it is still possible to evaluate the strengths and shortfalls of a given security solution.



Relative Value Weights	k_1	k_2	k_3	k_4	k_5	k_6	k_j
Value Factors → / Security Services ↓	Deterrence	Real Time Def.	Restor-ation	Collateral System Impacts	Implemen-tation Cost	Life Cycle Cost	Other
Diversity (s_1)	s_{11}	s_{12}					s_{1j}
Hopping (s_2)	s_{21}	s_{22}					s_{2j}
Data Consistency Checking (s_3)	s_{31}	s_{32}					s_{3j}
Tactical Forensics (s_4)	s_{41}	s_{42}					s_{4j}
Other (s_i)	s_{i1}	s_{i2}					s_{ij}

$$\sum_{j=1}^{p} k_j = 1$$

s_{ij} = Assurance Level of the ith service as related to the jth value factor

s_{ij} = Quantized Assurance Level = 0…M

$$\text{Security Score} = \sum_{j=1}^{p} \sum_{i=1}^{n} k_j s_{ij}$$

Max Possible Score = n x M

Figure 22. An extension of the scoring representation shown in Figure 21. This extended table is composed of System-Aware design patterns and weighted value factors. Each pattern-value factor pair is given an assurance level, *s*, based upon how the pattern level effects the given value factor. Furthermore, each value factor is assigned a weight, *k*, based upon its importance to a given system owner and operator.

To illustrate how such a scoring methodology could be used to evaluate a given System-Aware security solution, the methodology is utilized to evaluate the example outlined in section 3.2. For this example, several assumptions were made. First, s_{ij} can be assigned discrete scores between 0 and 5 inclusive (5 representing the best score and 0 representing the worst score). Second, system owners are concerned about six value factors: deterrence, real-time defense, restoration, collateral system impacts, implementation cost, and life-cycle cost. Third, four design patterns are available for utilization: diversity (*Diverse Redundancy*), configuration hopping (*Physical Configuration Hopping*), data consistency checking, and tactical forensics. Finally, the scored system was filtered by a set of weighting factors, k_j, which, for this example, were selected by the authors, weighted to emphasize security—deterrence (0.30), real-time defense (0.20), and restoration (0.10)—over cost—collateral system impacts (0.20), implementation cost (0.05), and life cycle cost (0.15). Finally, given these assumptions, the maximum possible score is 20 ($\sum\sum k_j s_{ij} \leq i * M$).

In the absence of a needed methodology for deriving assurance scores, s_{ij}, the assurance scores were chosen by the authors based on the rationale provided in section 3.2. For example, tactical forensics was noted to be a security service that provided significant capabilities to aid in the proper restoration of a given system. Assuming, that the specific suggested tactical forensic solution is judged as providing a significant contribution towards restoration, the maximum score, 5 is assigned. Furthermore, tactical forensics did provide some deterrence by increasing the likelihood that an attacker would be caught. For the purpose of this example, this leads to a score of a 3 out 5 related to deterrence. On the other hand, tactical forensics does require that every failure go through additional testing before a given component can be replaced. This would results in increased life cycle cost and hence tactical forensics is assigned a low value of 2 out 5 for the life cycle cost. This same process was repeated to provide a complete table of assurance levels. Finally, the authors created a set of relative weights emphasizing the security of the system over the cost. Overall, utilizing the judgments of the authors for scoring and the authors' preference for security over cost, the security architecture scored an 11.5 out of a possible 20. Furthermore, the architecture was evaluated as being strongest in the area of restoration and weakest in the area of life cycle cost.

4.4.5 Structured Arguments for Architectural Scoring

Section 4.4.4 outlined a possible scoring framework that could be utilized to evaluate System-Aware security architectures. Key to the framework was the ability to assign assurance scores to each design pattern based upon its contribution to a given value factor; however, no formal method for assurance score evaluation was provided. As part of introducing a formal method for assigning assurance scores for cyber security, one must recognize the need to rely on the judgments of experts in providing supporting rationale. This includes making judgments regarding the deterrence values for solutions, and the potential for adversaries to discover alternate attacks that could circumvent the defense. One possible approach for determining assurance scores could be derived from the use of Goal Structuring Notation (GSN) [Kelly, 2004] to communicate logically structured arguments in support of security claims in a clear and repeatable manner, utilizing rigorous evidence where it exists. Such an approach has been utilized by the UK Ministry of Defence [Menon, Hawkins, and McDermid, 2009; MoD, 2007] and the Food and Drug Administration (FDA) [FDA, 2009; FDA, 2010] for safety case evaluation. In the case of the FDA, safety cases (i.e., structured arguments) are starting to be used for the purpose of evaluating the safety of medical devices. A specific example is the case of approving new infusion pumps before they are certified for public use [FDA, 2010]. New infusion pumps may possess different technological characteristics—new implementations of software, hardware, and/or changes in material, design, and/or performance—that are both different from the existing approved infusion pumps and existing tools and methods of development, but are intended to provide similar functionality. In addition, in order to demonstrate the safety of these pumps, a large number of claims must be substantiated by a significant body of evidence and numerous arguments. This can result in a complex web of claims, arguments, sub-claims, and evidence that can potentially obscure relationships and makes the overall safety of the medical device difficult to assess. To address these issues, the FDA has called for manufacturers of medical devices to present safety cases that demonstrate that their devices meet certain claims about safety in a clear, traceable, structured manner. Safety cases force manufacturers to provide their reasoning at a level of granularity that clearly separates claims from arguments and supporting evidence. Furthermore, properly structured safety arguments make the underlying context and relationships between claims, arguments, and evidence explicit. The UK Ministry of Defense has also utilized safety cases for submarine propulsion system and air traffic management systems safety justifications.

Utilization of the GSN for evaluating System-Aware security architectures is based on the principle that the security architects should have a rational conceptual basis for their architectures (i.e., structured argument), and furthermore, these architectures should address a vast set of possible future attacks involving situations that have yet to occur, thus negating experimental validation. Where possible and worthwhile, solution architects should be required to gather or develop existing evidence to support their claims. One would anticipate that architects would rely upon support from expert testimony, analytical assessments, experimental data, and historical information. The set of claims, logical arguments, and evidence, could provide a basis for determining each of the assurance scores, s_{ij} .

It is important to note that in order to develop security claims, formulate rigorous arguments, and gather available evidence, a scoring team possessing a variety of skills would be required. For example, the skills required to determine scores related to the level of deterrence are different from those required for scoring restoration. To score deterrence, one would need to employ the experience and skills of a cyber security red team, as they would be most capable of providing the knowledge and perspective necessary to assert that a given solution would make attackers more hesitant to attempt an attack. However, a red team would not necessarily provide the skills and knowledge related to restoring a system. Rather, restoration assessment would likely be best served by a team of system architects and forensic analysts. Forensic analysts would likely know what tools exist or could be developed for obtaining information that would reveal a cyber attack in a timely manner. System architects could apply their knowledge of system architecture toward the evaluation of security solutions that could restore the system to a non-compromised state, and the amount of time that this would likely take.

It is recognized that the application of the GSN would not necessarily provide repeatability of results from one scoring team to another. Differences would arise regarding the claims, the corresponding arguments, and the supporting evidence. In addition, differences would arise regarding the translation of results into numeric scores. An approach for addressing these differences is to parallel what is occurring in the regulation of safety related systems. In the case of security scoring, the evaluation of a given system would rely on the owners and operators to supply an appropriate context, including an available risk analysis for the system being protected and scoring guidelines. Assuming that such an approach were accepted, one would expect the safety certification and security communities to share experiences and improve the methodologies surrounding the employment of GSNs for decision making.

4.5 Extending the Proposed Scoring Model

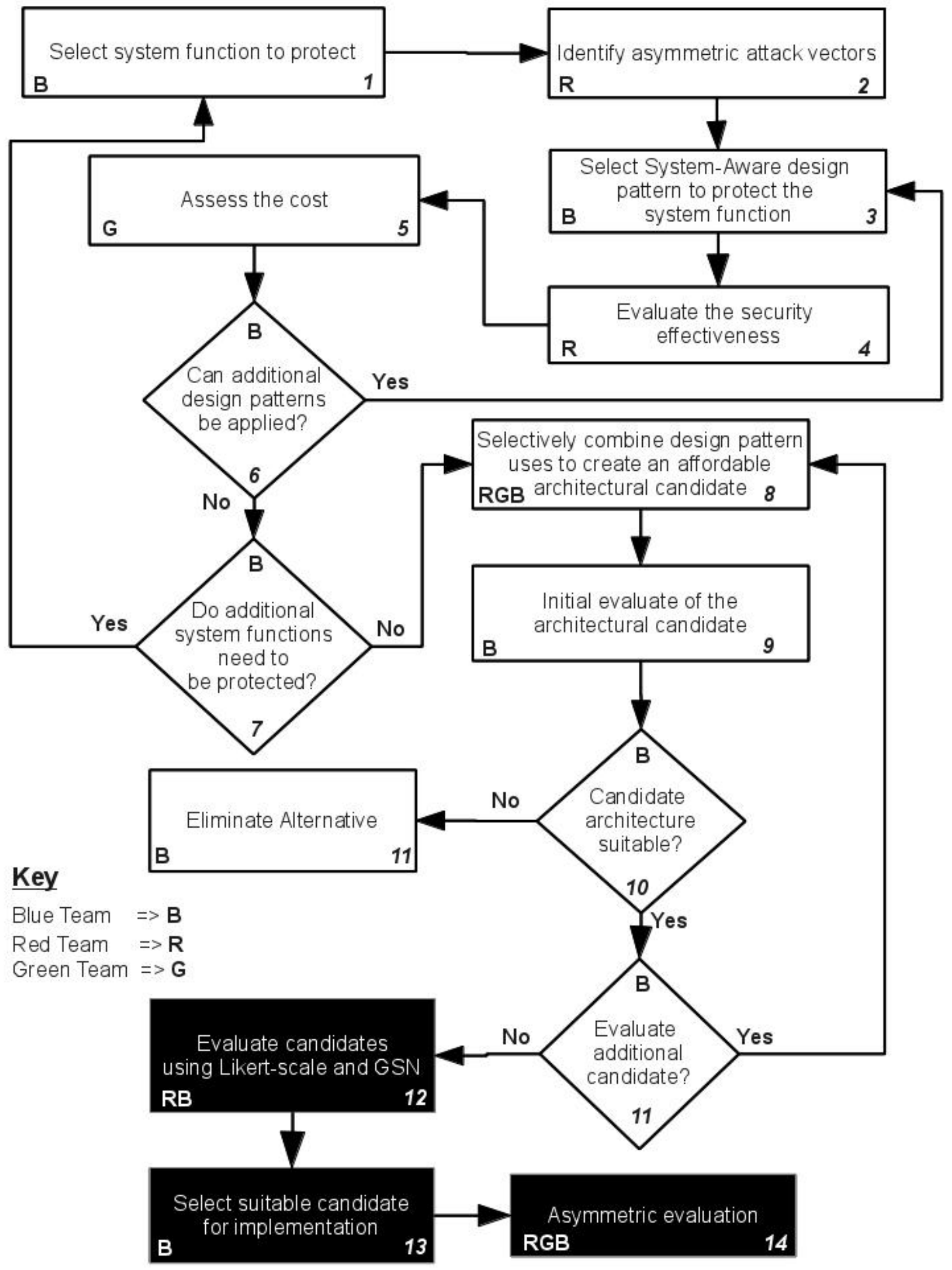
Section 4.4 introduced a framework for evaluating the possible effectiveness, from a cyber security viewpoint, of a given System-Aware architecture, and comparing the security afforded to alternative System-Aware architectural candidates. However, the proposed framework outlined in section 4.4 did not provide a method for designing and selecting architectural candidates to be evaluated. Supporting the design and selection of architectural candidate solutions is a non-trivial task, as the selection of an architecture is closely tied to how it is evaluated; i.e., it is desirable to present those candidate architectures that will evaluate well rather than those that will evaluate poorly. As a result, it is necessary that the scoring framework proposed in section 4.4 be extended.

4.5.1 Enhanced Architectural Scoring Framework

Figure 23 shows a flowchart detailing how the architectural scoring framework discussed in section 4.4 has been enhanced. As can be seen, the scoring method outlined in section 4.4 (black boxes with white text numbered 12, 13, & 14) has been extended to include a process to design and select architectural candidates for evaluation (white boxes with black text numbered 1 thorough 11). This process includes the selection of critical system functions to be protected using System-Aware security, identification of asymmetric attack vectors, selection of design patterns to protect those system functions from potential cyber attacks, the integration of those system functions and System-Aware design patterns into candidate architectures, a separate evaluation process to determine which of those candidate architectures should be evaluated using the rigorous scoring and GSN processes discussed in section 4.4, and finally an evaluation of how the architecture affects the asymmetry between potential attackers and the system being protected. Each of these steps is to be carried out by one of three teams:

- *System design (Blue) team* – Members of this team includes those that understand how the system was designed, implemented, and operates
- *Cyber attack (Red) team* – Members of this team are knowledgeable about the resources necessary to create and design exploits
- *Cost Analysis (Green) team* – Members of this team are responsible for determining the costs of designing, implementing, and integrating the selected System-Aware security services

The remainder of this section details how steps 1 through 11 of Figure 23 (i.e., selection of suitable architectural candidates to be evaluated using the more rigorous methodology discussed in section 4.4) is accomplished. Appendix A outlines a prototype implementation for the selection of suitable architectural candidates, as well as discusses its application to a use case.



Select system function to protect
B
1
Identify asymmetric attack vectors
R
2
Select System-Aware design pattern to protect the system function
B
3
Evaluate the security effectiveness
R
4
Assess the cost
G
5
B
Can additional design patterns be applied?
6
Yes
No
B
Do additional system functions need to be protected?
7
Yes
No
Selectively combine design pattern uses to create an affordable architectural candidate
RGB
8
Initial evaluate of the architectural candidate
B
9
B
Candidate architecture suitable?
10
No
Yes
Eliminate Alternative
B
11
Key
Blue Team => B
Red Team => R
Green Team => G
B
Evaluate additional candidate?
11
No
Yes
Evaluate candidates using Likert-scale and GSN
RB
12
Select suitable candidate for implementation
B
13
Asymmetric evaluation
RGB
14

Figure 23. Flowchart illustrating the enhanced architectural framework. The initial more rigorous method detailed in section 4.4 is shown as black boxes with white text (numbered 12, 13, and 14). This method has been enhanced by including a process to help design and select one or more architectural candidates from the potentially vast set of possible System-Aware architectures (shown as white boxes with black text numbered 1 through 11). Each step is also marked with the team(s) designated to perform that step.

Selection of System Functions for Protection

The first step in designing a System-Aware architecture is to identify which of the system functions will be protected using System-Aware Cyber Security. This includes, (1) identifying those system functions that could possibly benefit from System-Aware Cyber Security, and (2) determining the relative importance—from a cyber security viewpoint—of protecting those functions from a cyber attack. This importance is relative because it is based on a comparison of every system function to every other system function. Furthermore, as the importance is relative, it enables the ability for all system functions to be ranked. As discussed in Selecting Architectural Candidates in section 4.5.1, this ranking will be used to compose a subset of all identified system functions into a set of the most important functions to protect. It is emphasized that the relative importance assigned to protecting a system function is based on its contribution to achieving the mission objectives of the system and not on its susceptibility to attack.

Both the identification of system functions and the ranking of their security importance should be performed by the system design team (i.e., Blue Team), as it possess the knowledge necessary to determine functions are exposed to types cyber attacks System-Aware security is designed to address and how those functions are used to achieve the system's mission objectives.

Identification of Asymmetric Attack Vectors

After the system functions to be protected have been identified, each of those functions is assessed to determine possible asymmetric threats. An asymmetric threat is one that, with only a minimal analysis of the system function to be exploited, could be designed, developed, and maintained utilizing a small amount of resources, and has the potential to severely compromise or damage the system to be protected. In the absences of System-Aware security, such threats are potentially difficult to detect and deflect. This step is to be performed by a red team, as such an evaluation requires knowledge about the resources necessary to design and develop exploits.

Selection of System-Aware Design Patterns

Once all of the system functions that could benefit from System-Aware security have been identified, it is then necessary to select System-Aware design patterns to be considered for protecting those functions. A design pattern is selected if it provides one or more of the following

1. Makes it significantly more difficult for the compromised system function to cause damage to the system or compromise the system's ability to complete its designated mission objective
2. Improves the robustness of the system function by either
    a. Preventing a compromised system function from taking certain damaging actions
    b. Allowing for the restoration of a compromised system function into a non-compromised—and possibly less capable—state
3. Increases the likelihood of identifying when the system function has been compromised and/or information about the source of compromise (i.e., identifying attributes about the adversary)

Since determining the efficacy of applying a System-Aware design pattern to a given system function requires knowledge of how that function operates, as well as how it integrates into the overall system design, the selection of System-Aware design patterns is performed by the system design team (i.e., Blue Team). Note that this selection does not consider issues associated with attackers' developing and executing exploits. This is deferred for a subsequent stage in the analysis process.

Determining the Security Effectiveness

Given a set of system functions, each with a set of System-Aware design patterns, it is then necessary to determine the potential efficacy (i.e., the effective security) each System-Aware design pattern affords to the function it has been selected to protect. One such method was outlined in section 4.4; this method relied upon the use of structured arguments to organize a large body of evidence into well structured rigorous arguments about the effectiveness—from a cyber security viewpoint—of a proposed solution. However, such a process has the potential to require a significant effort to accomplish. In the scoring framework outlined in section 4.4, this level of effort was acceptable as it was assumed that this was being performed for the evaluation of only a small subset of the possible System-Aware architectural candidates. However, as there could potentially exists a vast number of possible architectural candidates, a more streamlined process is needed for creating a manageable set System-Aware architectural candidates to be evaluated using the method outlined in section 4.4. This screening process involves the identification and evaluation of all system functions that could benefit from System-Aware cyber security, as well as all possible System-Aware design patterns that could be utilized to protect those functions. This is accomplished by assigning an integer valued score to a single System-Aware design pattern protecting a single system function based upon two criteria

1. The System-Aware design pattern's ability to increase the complexity, cost, and time for a hypothetical adversary to develop an exploit for the system function being protected

2. The ability to decrease the probability that an attack against the system function being protected will be successful

This simplified analysis assumes that every combination of System-Aware design pattern and system function provides the system with its own independent effective security; i.e., there are no (dis)economies of scale exist. In addition, this method assumes that the security effectiveness of a given

System-Aware architecture is simply the sum of the security effectiveness scores for all of the included combinations of System-Aware design patterns and system functions.

Such an evaluation requires knowledge about the resources necessary to design and develop exploits; as such, this step is to be performed by a cyber attack (i.e., Red team). In addition, the cyber attack team should perform its evaluation without knowing the relative rank ordering of system functions performed by the system design team or the reasoning behind the selection of System-Aware design patterns.

Determining the Resources Necessary to Develop the System-Aware Architecture

As outlined in section 4.4, System-Aware architectures are evaluated in terms of their security effectiveness, collateral impacts on system performance, and costs to design, implement, and maintain. Thus, candidate System-Aware architectures cannot be designed or evaluated without a method for estimating their potential costs. For the enhanced scoring framework outlined in section 4.5.1, these costs are to be estimated by a special cost analysis team (i.e., Green Team) for all of the System-Aware design patterns proposed for all of the identified system functions. As was the case for security effectiveness, these costs represent the costs of designing, implementing, and maintaining a single design pattern for a single system function. This analysis assumes that no (dis)economies of scale exist, and the total cost of a System-Aware architecture is simply the sum of the cost for each of the selected combinations of System-Aware design pattern and system function. This analysis is to be performed by a specialized cost analysis team.

It is observed that the costs analysis team can also evaluate the costs for an adversary to analyze, design, implement, and maintain counter measures to the proposed System-Aware security measures. This information can be utilized to assess the amount of asymmetry, if any, between the costs to implement System-Aware security measures and the costs to overcome those measures. In turn, the relative levels of asymmetry can then be used to help evaluate the security effectiveness of the candidate architectures. It is noted that this asymmetry is dependent on the perceived resources of the adversary versus the system being defended; e.g., assume there is an adversary with a small amount of resources and a System-Aware architecture is being designed to protect a critical large system that was expensive to design and maintain. In this circumstance the adversary only has a small amount of resources to overcome any System-Aware defensive measures that are developed. However, the System-Aware architecture can consume a larger amount of resources, but this can still be relatively small compared to resources being protected. Thus, it is the costs relative to the resources available that determine the asymmetry.

For the purposes of designing System-Aware architectural candidates to be evaluated with the more rigorous methods outlined in section 4.4, collateral impacts on system performance is a rough estimate provided by the system design team and are used solely to eliminate System-Aware design patterns that would unacceptably degrade system performance. Collateral impacts are not used more extensively for the purposes of designing candidate architectures as only the system owner can decide whether the potential security benefits outweigh the loss in system performance, and due to the potential for mitigating solutions (e.g., increasing the system's processor speeds or the amount of available memory). During architectural evaluation, collateral impacts can be estimated using the more rigorous methods described in section 4.4.

Selecting Architectural Candidates

Once all of the system functions that could benefit from System-Aware Cyber Security have been identified, System-Aware design patterns have been selected and their potential security effectiveness have been evaluated, and their costs and impacts have been assessed, it is then necessary to select architectural candidates for evaluation. However, as noted in section 4.5, while selection and evaluation can be treated as separate activities, it can be beneficial to combine these steps. In the architectural scoring framework this is accomplished through a combination of automated decision tools and user interaction. Automation is utilized to generate a small set of candidate architectures. User input is utilized to provide constraints and to create candidate architectures by modifying the candidate architectures with the support of automation tools. The candidate architectures are created by utilizing the relative importance of system functions, as determined by the system design team, the potential security effectiveness afforded by applying System-Aware design patterns to system functions, and the costs of implementing the selected System-Aware design patterns. In addition, these criteria can be used to help users modify these architectures; e.g., users could replace a subset of System-Aware design patterns chosen by the automated system with more expensive System-Aware design patterns that offer a greater degree of effective security. Appendix A provides a discussion of a prototype architecture that illustrates one approach to using automation and user interaction to generate candidate architectures.

The result of this process is a set of suitable architectural candidates that can be evaluated using the more rigorous methods discussed in section 4.4. Alternatively, if the cost or time needed to perform the more rigorous analysis is deemed unacceptable, this process can be utilized to select a particular System-Aware architectural candidate to be implemented. Finally, all architectural candidates are evaluated to determine the resulting shift in asymmetry between developing exploits and protecting the system. This final step is performed regardless of whether or not the more rigorous GSN methodology is utilized.

Limitations of the Proposed Enhanced Architectural Scoring Framework

While the decision support system outlined in section 4.5 provides several enhancements to the System-Aware architectural evaluation process, it still offers room for improvement

1. The enhanced scoring framework assumes that no relationship exist among the System-Aware design patterns applied to protect the system function; i.e., no conflicts or redundancies are present

2. The security effectiveness and resource cost only apply to a single System-Aware design pattern applied to a single system function this allows them to be assessed independently and can be combined in a simple additive manner (i.e., linear) with the security effectiveness and resource cost of other design patterns to obtains the security effectiveness and resource cost of the final architecture; i.e., no economies—or diseconomies—of scale exist

3. System-Aware architectures that have the same costs and security effectiveness are equally desirable to a decision maker regardless of the number and particular System-Aware design patterns implemented and system functions protected; i.e., an architecture with a single highly secured system function based on the integration a number of design patterns, can be as valuable as an architecture with several system functions protected, but with fewer design patterns utilized to protect each function.

Chapter 5
Conclusion and Future Work

5.1 Conclusion

This dissertation presents System-Aware Cyber Security, a new systems engineering approach for addressing the threat of cyber attacks that have breached the perimeter—particularly supply chain and insider attacks. It also outlines a security analysis framework for designing, selecting, and comparing System-Aware security architectures. It describes several example security services, discusses how these services can be converted into reusable design patterns for implementation across multiple domains, and illustrates how each of these services can be utilized to enhance the security of a system. Through the use of three examples, it is shown how System-Aware Cyber Security can enhance pre-attack, trans-attack, and post-attack security by deterring attackers, preventing damage, isolating compromised components and enabling restoration to a non-compromised state. In addition, a methodology for selecting system functions for protection, selecting System-Aware design patterns to protect those functions, and generating candidate architectures from those selections is suggested. This included a description of a possible prototype implementation and its usage in designing a particular System-Aware architecture. It also involved the use of a structured argument methodology utilizing GSN as a means for rigorously evaluating those architectural candidates.

5.2 Future Work

Future work to further develop the System-Aware Cyber Security methodology includes,

1. Exploring a broader set of applications for System-Aware Cyber Security. This broader set of applications can help to develop additional design patterns, as well as to refine both those design patterns that have been presented and the architectural scoring framework.

2. Expanding the set of System-Aware design patterns based upon the expanded set of applications. A larger pool of design patterns will help extend System-Aware security to a broader range of fields.

3. Extending System-Aware security to protect against additional attack scenarios. The greater the range of threats that can potentially be addressed with System-Aware security the greater its potential cyber security value.

4. The development of policies, techniques, and technologies to ensure the proper implementation and integration of System-Aware services. While design patterns have proven to be a helpful tool in the development of software [Gamma et al., 1995], they are certainly not the only one. Likewise, while design patterns play an important role in development and implementation of System-Aware security, there still remains room for the creation of additional tools and technologies to further reduce the resources needed for the creation, development, and implementation of System-Aware security services. For example, in the design pattern *Physical Configuration Hopping* it is noted that the rate of hopping must be set in relation to the time it would take an attack to cause damage. Thus, possessing tools that would allow system designers and owners to more easily and quickly analyze their systems to determine this value could prove extremely effective.

5. Methods for minimizing the costs and collateral impacts of integrating System-Aware services. For example, in the design pattern *Physical Configuration Hopping* it is noted that one of the possible collateral impacts is bumpless control. Thus, it would be beneficial to have methods readily available to aid in the design and development of such bumpless control mechanisms.

6. Processes for integrating System-Aware Cyber Security into the system design process. Just as traditional cyber security should not be seen as an after thought in the design and development of a system, nor should System-Aware cyber security. This is particular true of System-Aware security as it makes use of application specific knowledge.

7. Enhancing the suggested System-Aware Cyber Security scoring framework to generate architectural candidates more aligned with stake holder values. The better a stake holder's values are known the easier it is to develop suitable architectural candidates.

8. Relax the assumptions in the suggested System-Aware scoring framework to provide a more accurate assessment of architectural candidates. Relaxing these assumptions can potentially improve the selection of suitable architectural candidates and reduce the number of candidates that need to be analyzed using the more rigorous GSN method.

9. Extend the suggested System-Aware cyber security scoring framework to further facilitate user exploration of the architectural design space. The easier it is for stake holder's to explore the space of possible architectural candidates the easier it will be to find suitable candidates.

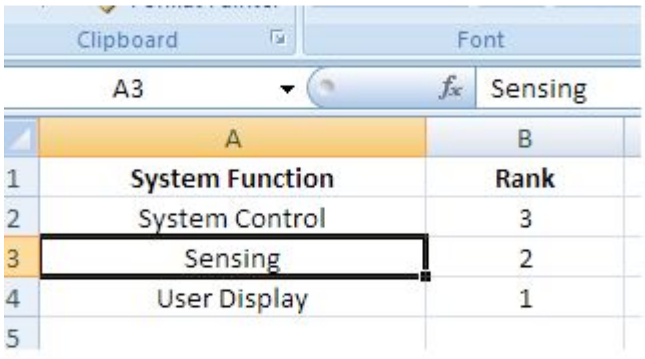
10. Further exploration of the utilization of structured arguments using GSN as a method for deriving specific assurance scores for suggested architectures. It may be possible to incorporate aspects of the GSN earlier in the selection and evaluation of architectural candidates. In addition, it may be possible to mitigate the costs of utilizing the GSN.

Appendix A

To help in the creation of the enhanced architectural scoring framework described in section 4.5, a prototype decision support system was constructed using Microsoft Excel. There were two principle goals in developing this prototype system. First, was to develop algorithms for automatically generating architectural candidates. Second, was the creation of tools to help users explore the set of all candidate architectures by making adjustments to the automatically generated architectures. This also included the creation of tools and visualization aids to help users compare and contrast the candidate architectures. Finally, while these make up the main contribution of the prototype decision support system, it is noted that the prototype is fully functional; i.e., it supports all of the steps laid out in section 4.5.1.

Selection of System Functions for Protection

Figure 24 shows how the prototype system supports the selection of system functions for protection for a simple example system. As can be seen, three system functions have been identified and assigned a relative ranking according to their importance to protect by the Blue Team. For the system shown in Figure 24, the System Control function has been designated the most important system function to protect, and the User Display the least; i.e., a higher ranked function is more important then a lower ranked function. It is noted that while only three system functions are shown here, the prototype system can support any number of functions.



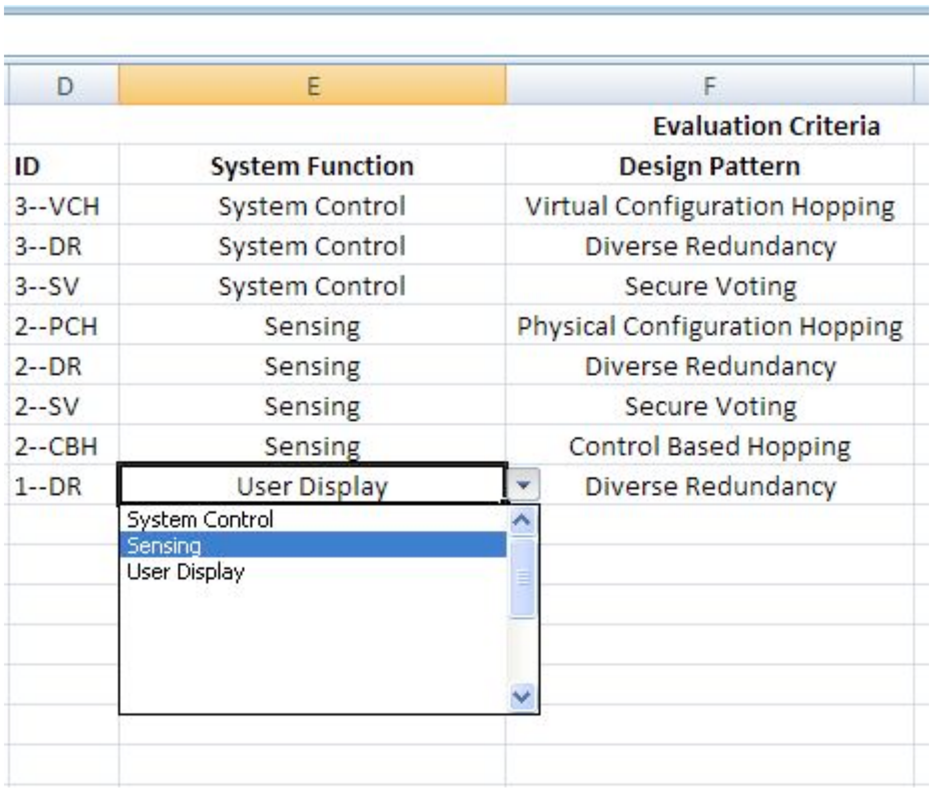
	A	B
1	System Function	Rank
2	System Control	3
3	Sensing	2
4	User Display	1
5		

Figure 24. Represents the prototype decision support system's support for selecting system functions to protect and assigning them a relative rank ordering. In this case three system functions have been identified by a Blue Team for protection and ranked. A higher rank signifies a system function that is more important to protect; e.g., in this instance the most important function to protect is the one responsible for System Control.

Selection of System-Aware Design Patterns

After the system functions that would benefit from System-Aware security solutions have been selected, System-Aware design patterns are chosen by the Blue Team to protect these functions. Figure 25 shows this for the System Functions identified in Figure 24. As can be seen, the user first selects one of the identified system functions from a drop down list. After selecting a function for security consideration, a System-Aware design pattern is chosen for that system function. Finally, each combination of system function and System-Aware design pattern is assigned a unique identifier.

The prototype system offers two methods for creating this identifier. The method illustrated in Figure 25 uses the previously designated relative rank combined with a short acronym of the System-Aware design pattern. This method is intended to produce an identifier that would allow the user to intuitively recognize the relative importance of the system function as well as the method used to protect it. However, this scheme requires all available System-Aware design patterns to be assigned a unique short hand identifier before the process is started. Depending on the number of design patterns available, this may not be practical. To address this potential pitfall, a second method is available. This method recognizes that a System-Aware design pattern can only be applied to a system function once; i.e., each combination of system function and design pattern should be unique. A unique identifier is generated by hashing these unique combinations. This has the benefit of generating unique identifiers without requiring additional information from the user; however, unlike the former method, the identifiers will not have intuitively derived meaning.



D	E	F
		Evaluation Criteria
ID	System Function	Design Pattern
3--VCH	System Control	Virtual Configuration Hopping
3--DR	System Control	Diverse Redundancy
3--SV	System Control	Secure Voting
2--PCH	Sensing	Physical Configuration Hopping
2--DR	Sensing	Diverse Redundancy
2--SV	Sensing	Secure Voting
2--CBH	Sensing	Control Based Hopping
1--DR	User Display	Diverse Redundancy

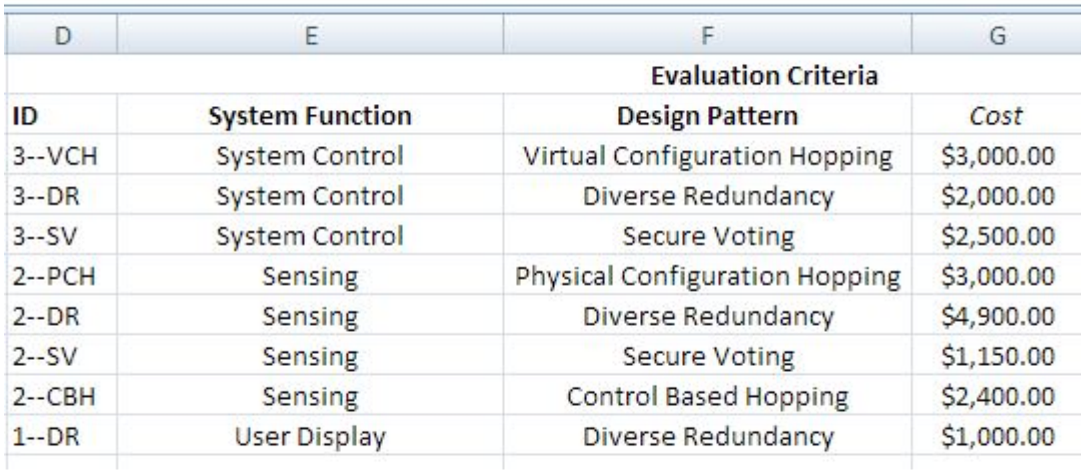
Figure 25. Illustrates how the prototype system is used to support the selection of System-Aware design patterns. For this instance the available system functions are assumed to be those presented in Figure 24. As can be seen, only those functions previously selected for protection can be selected in this step. Also note that every combination of system function and System-Aware design pattern is assigned a unique identifier. For this instance, the identifier is composed a combination of the relative rank of the system function and the first letter of every word of the System-Aware design pattern.

Determining the Resources Necessary to Develop the System-Aware Architecture

As discussed in section 4.5.1, every combination of system function and System-Aware design pattern should be evaluated independently to determine its costs and collateral impacts on the system. For the prototype architecture, it is assumed that all resources can be represented by a single monetary cost.

This is more restrictive then outlined section 4.5.1, where there can be multiple costs represented in different units (e.g., money, power, and space). The decision to reduce this to a single monetary cost was made to both support a more robust set of algorithms for automatically generating architectural candidates, and to make it easier for a user to compare and contrast competing architectural candidates.

Figure 26 builds upon Figure 25 to illustrate how this is done in the prototype system. As can be seen, every combination of system function and System-Aware design pattern is assigned a single monetary cost. For the prototype system, these costs are considered to be independent.



D	E	F	G
		Evaluation Criteria	
ID	System Function	Design Pattern	*Cost*
3--VCH	System Control	Virtual Configuration Hopping	$3,000.00
3--DR	System Control	Diverse Redundancy	$2,000.00
3--SV	System Control	Secure Voting	$2,500.00
2--PCH	Sensing	Physical Configuration Hopping	$3,000.00
2--DR	Sensing	Diverse Redundancy	$4,900.00
2--SV	Sensing	Secure Voting	$1,150.00
2--CBH	Sensing	Control Based Hopping	$2,400.00
1--DR	User Display	Diverse Redundancy	$1,000.00

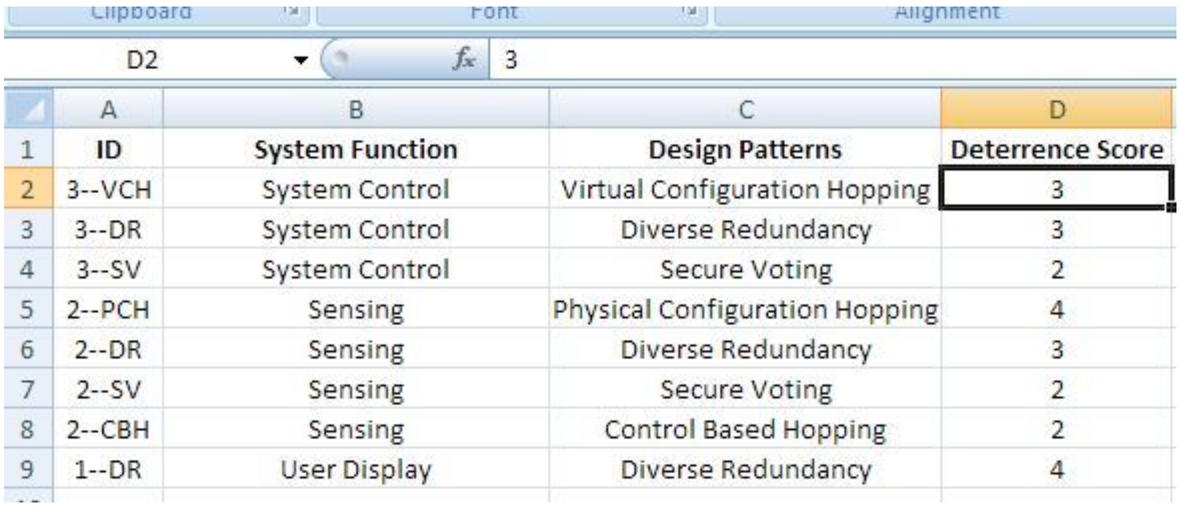
Figure 26. Illustrates how the prototype system supports the assessment of the necessary resources needed to implement a given System-Aware architecture. In this instance every combination of System-Aware design pattern and system function generated in Figure 25 is assigned a monetary cost.

Determining the Security Effectiveness

Figure 27 illustrates how the prototype system supports the evaluation of the security potentially afforded by each combination of system function and System-Aware design pattern to be evaluated. In the prototype evaluation system the ID, System Function, and Design Patterns, fields are automatically populated based on the inputs received in the previous steps The Deterrence Score is used to represent the security effectiveness potentially afforded by each combination of system function and design pattern. As noted in section 4.5.1, the security effectiveness score can be any integer value; however, for the prototype system, this score is limited to 1, 2, 3, or 4, with higher scores relating to more value. This limitation is not imposed by the evaluation system, but rather stems from the criteria used to determine the effective security score

- Score = 4, Complexity, cost, and time to develop exploits are high and the probability of a successful exploit is low
- Score = 3, Complexity, cost, time to develop exploits are high and the probability of a success exploit is high
- Score = 2, Complexity, cost, time to develop exploits are low and the probability of a success exploit is low
- Score = 1, Complexity, cost, time to develop exploits are low and the probability of a success exploit is high

The particular system in Figure 27 is built using those combinations illustrated in Figure 25.



	A	B	C	D
1	ID	System Function	Design Patterns	Deterrence Score
2	3--VCH	System Control	Virtual Configuration Hopping	3
3	3--DR	System Control	Diverse Redundancy	3
4	3--SV	System Control	Secure Voting	2
5	2--PCH	Sensing	Physical Configuration Hopping	4
6	2--DR	Sensing	Diverse Redundancy	3
7	2--SV	Sensing	Secure Voting	2
8	2--CBH	Sensing	Control Based Hopping	2
9	1--DR	User Display	Diverse Redundancy	4

Figure 27. Illustrates how the prototype supports the assigning of security effectiveness scores to all combinations of system function and design pattern. This instance assigns scores for those combinations identified in Figure 25.

Selecting Architectural Candidates

As discussed in section 4.5.1, the selection of architectural candidates is performed through a combination of automation tools and user input. The user provides constraints that are used to guide the automated execution of algorithms in the creation of a subset of candidate System-Aware architectures. In addition, the user is then able to modify one or more of these candidate architectures to generate additional candidate architectures. This process can result in one of two outcomes

1. A set of candidate architectures to be evaluated using a more rigorous criteria
2. Select the architecture that will be implemented

The prototype system can support both outcomes; however, the prototype only supports the creation of candidate architectures and does not provide any additional support to perform a more rigorous evaluation.

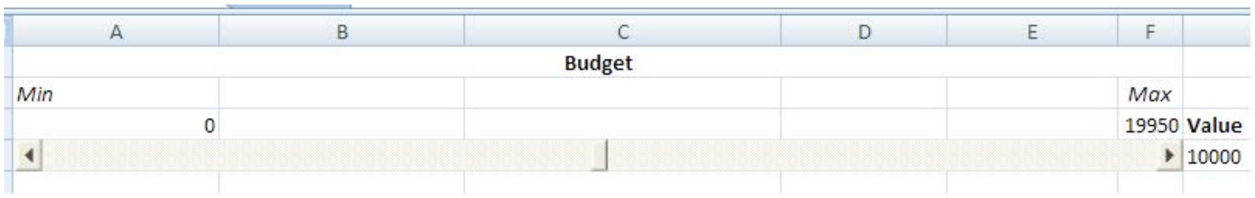
User Constraints

The prototype decision support system accepts user constraints on the total amount of resources that are available for implementing System-Aware design patterns to protect system functions. Since the prototype system assumes that only one monetary resource exist, this step is presumed to be the user allocating a budget. Figure 28 illustrates the options available to the user (it assumes the resources shown in Figure 26).

- User is allowed to set a value between 0 and the sum total of the resource costs of selecting every combination of system function and design pattern (this is 19,950 in Figure 28)

- User can move a slider between the values of 0 and the sum total of the resource costs of selecting every combination of system function and design pattern (this is 19,950 in Figure 28)

The maximum budget value is automatically computed based on the inputs supplied. Furthermore, the budget values are always integer—the maximum value is always rounded up to ensure that is can cover the difference. This is done to support the automatic generation of candidate architectures.



A	B	C	D	E	F	
		Budget				
Min					*Max*	
0					19950	**Value**
						10000

Figure 28. Illustration of the budget slider used in the prototype system. This slider is currently set to a budget of $10,000, has a maximum possible value of $19,950, and a minimum value of $0.

Automated Support to Generate Candidate Architectures

After a maximum budget has been set and System-Aware design patterns have been selected, their security effectiveness evaluated, and their cost determined for each of the system functions identified as possibly benefiting from System-Aware security, an automated decision support system is then utilized to generate two exemplar System-Aware architectures

1. *Blue Perspective* – The prototype decision support system will select a system function to protect based upon the importance placed on protecting that system function by the system design team. First, the prototype system will select the highest ranked (i.e., most important) system function. Next, System-Aware design patterns will be selected to maximize the protection of the chosen system function. Design patterns will be selected until either the costs meet the available budget or all available patterns have been selected. In the former case, design patterns will be selected to maximize the total security effectiveness offered (the sum of the selected combinations of system functions and design patterns) within the user defined budget. This process will be repeated until either all of the identified system functions have been protected by all available System-Aware design patterns or the total cost meets the maximum user defined budget. As discussed in section 4.5, it is assumed that the costs and security effectiveness scores for each combination are independent of all other combinations; thus, the total cost and security effectiveness of the candidate architectures can be computed through a simple summation of the individual values.

2. *Red Perspective* – The prototype decision support system will select system functions and corresponding System-Aware design patterns in order to maximize the security effectiveness of the final System-Aware architecture. To do so the prototype decision support system makes the same assumptions as the *Blue Perspective:* that the individually assigned scores for security effectiveness and the costs can be computed from a simple summation in order to derive the values for the candidate architecture. In the event that two or more System-Aware design patterns consume the same amount of resources and afford the same security effectiveness for different system functions, and only one of those functions can be protected, the importance

placed upon protecting the system function by the system design team will be used to determine which of the functions is protected.

For both of these cases, it is recognized that the maximization of the security effectiveness subject to a budget constraint is an instance of the knapsack problem [Martello and Toth, 1990]. As a result, the prototype decision support system has been designed to try and take advantage of this fact. First, it is known that the knapsack problem can be solved in pseudo polynomial if all of the weights (i.e., resource costs associated with each combination of system function and design pattern) are non-negative integers. As discussed earlier, the weights are monetary cost estimates. This means that all of the weights are nonnegative. In addition, the maximum budget is an integer value. Finally, the prototype system will round up all of the individual weights to integer values before trying to maximize the security effectiveness. This last step ensures that all of the weights (i.e., costs) are integers. This last step is deemed acceptable as the costs of each of the proposed solutions (i.e., combinations of system functions and design patterns) are considered to be large enough that the change can be safely ignored. Of course, it is still possible that the time required to determine the optimal solution is unacceptable. As a result the prototype systems allows user to use a heuristic algorithm in place of the optimal solution. This algorithm is not guaranteed to find the optimal solution, but it will run in polynomial time.

These candidate architectures are meant to represent two edge cases, thus providing a starting point for user the exploration of the available design

- *Blue Perspective* – Protects the system by protecting system functions according to the system design teams evaluations
- *Red Perspective* – Protects the system by maximizing the security effectiveness of the final architecture based upon the evaluations of the cyber attack assessment team

Tools to Support User Exploration

After the prototype decision support system has generated the two candidate architectures, the user is then able to adjust those architectures to generate additional candidates. The prototype decision support system offers several tools to support the user in this given task

- Overview and summary statistics describing the candidate architecture generated using the *Blue Perspective* approach. As seen in Figure 29, this includes a list detailing which combinations of system functions and design patterns were selected and summary information about this architecture
    - Deterrence Score: The sum of the deterrence scores of the selected combinations as well as the deterrence score of summing all possible combinations
    - Cost: The sum of the costs of the selected combinations as well as the costs of selecting all combinations
    - Higher Ranked Blue: Represents the number of important system functions that were included in the candidate architecture. A system function is important if its relative ranking is greater than or equal to the maximum ranked function divided by half
    - Lower Ranked Blue: Represents the number of less critical system functions that were included in the candidate architecture. A system function is less critical if its relative ranking is less than half the maximum ranked function
    - Bigger Det Red: Represents the number of design patterns included in the candidate architecture that contributed a significant amount of effective security. A significant amount of effective security is a security effectiveness score greater than or equal to

half the maximum possible score. For the prototype decision support system this is a score of 4 or 3

  - Smaller Det Red: Represents the number of design patterns included in the candidate architecture that contributed a small amount to the effective security. A small amount of effective security is a security effectiveness score less than half the maximum possible score. For the prototype decision support system this is a score of 2 or 1

- Overview and summary statistics describing the candidate architecture generated using the *Red Perspective* approach. As seen in, this includes a list detailing which combinations of system functions and design patterns were selected and summary information about this architecture. This information is exactly same as that discussed earlier for the *Blue Perspective*

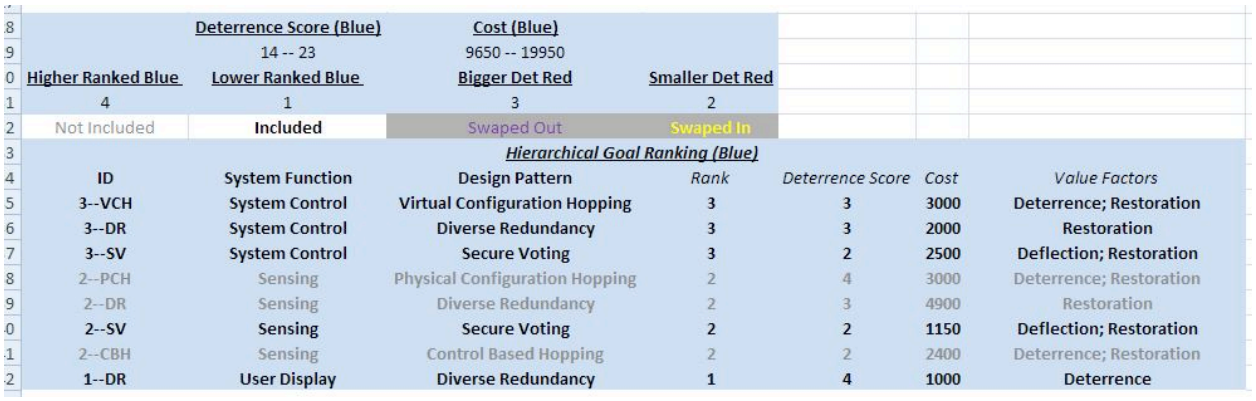
- Two quad charts representing the summary statistics of the candidate architectures. The first chart, seen in Figure 31, shows all of the combinations including in the candidate architectures (*Blue Perspective* and *Red Perspective*). This information includes the relative importance of the system function being protected and its contribution to the effective security of the candidate architecture. This also includes the amount the combination contributes to the overall costs of architecture (this is represented by the size of the glyph). The second chart, seen in Figure 32, shows the same information as the first; however, the combinations plotted are those NOT included in the candidate architecture

- As discussed in section 4.5.1, the user should be able to create additional architectural candidates by adjusting those generated through more automated means. In the prototype architecture this is done by allowing the user to select a candidate architecture and add or remove combinations of system functions and design patterns. This interface is shown in Figure

33. In addition, the prototype decision support system highlights how these changes affect the selected architecture. These include highlighting which combinations have been added and removed, as well as highlighting any positive or negative changes in the summary statistics. An example of this can be seen in Figure 34

- When the user creates a new architectural candidate, the prototype decision support system allows the user to save that candidate so it can be compared to all other candidate architectures created. The prototype system provides tools to allow for the user to compare candidates in terms of their security effectiveness, costs, and selected combinations of system functions and design patterns in a pair wise manner

- The prototype decision support allows the user to keep a history of all candidate architectures created, the steps (i.e., adjustments to the automatically generated architectures) that were taken to create those candidates, and a history of the reasoning behind those changes

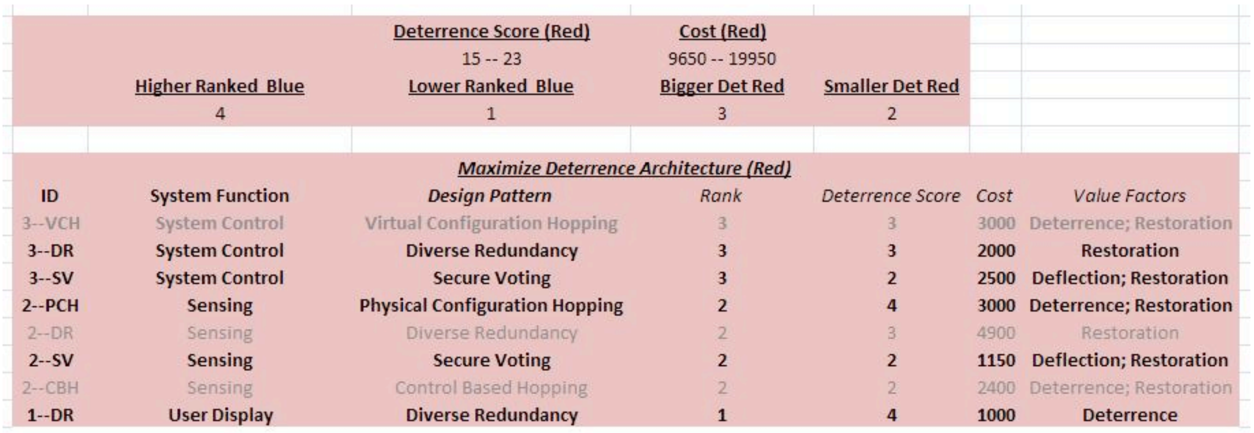


	Deterrence Score (Blue)	Cost (Blue)	
	14 -- 23	9650 -- 19950	
Higher Ranked Blue	Lower Ranked Blue	Bigger Det Red	Smaller Det Red
4	1	3	2
Not Included	Included	Swaped Out	Swaped In

Hierarchical Goal Ranking (Blue)

ID	System Function	Design Pattern	Rank	Deterrence Score	Cost	Value Factors
3--VCH	System Control	Virtual Configuration Hopping	3	3	3000	Deterrence; Restoration
3--DR	System Control	Diverse Redundancy	3	3	2000	Restoration
3--SV	System Control	Secure Voting	3	2	2500	Deflection; Restoration
2--PCH	Sensing	Physical Configuration Hopping	2	4	3000	Deterrence; Restoration
2--DR	Sensing	Diverse Redundancy	2	3	4900	Restoration
2--SV	Sensing	Secure Voting	2	2	1150	Deflection; Restoration
2--CBH	Sensing	Control Based Hopping	2	2	2400	Deterrence; Restoration
1--DR	User Display	Diverse Redundancy	1	4	1000	Deterrence

Figure 29. The architecture candidate created automatically using the *Blue Perspective*. The top displays summary information, including deterrence score, total costs, and rough break down of the importance of the system functions protected and the effectiveness of the design patterns chosen to protect those functions. The bottom shows which combination of system functions and design patterns were selected. This instance was generated using the information shown in Figure 26 and Figure 27.



	Deterrence Score (Red)	Cost (Red)	
	15 -- 23	9650 -- 19950	
Higher Ranked Blue	Lower Ranked Blue	Bigger Det Red	Smaller Det Red
4	1	3	2

Maximize Deterrence Architecture (Red)

ID	System Function	*Design Pattern*	*Rank*	*Deterrence Score*	*Cost*	*Value Factors*
3--VCH	System Control	Virtual Configuration Hopping	3	3	3000	Deterrence; Restoration
3--DR	**System Control**	**Diverse Redundancy**	**3**	**3**	**2000**	**Restoration**
3--SV	**System Control**	**Secure Voting**	**3**	**2**	**2500**	**Deflection; Restoration**
2--PCH	**Sensing**	**Physical Configuration Hopping**	**2**	**4**	**3000**	**Deterrence; Restoration**
2--DR	Sensing	Diverse Redundancy	2	3	4900	Restoration
2--SV	**Sensing**	**Secure Voting**	**2**	**2**	**1150**	**Deflection; Restoration**
2--CBH	Sensing	Control Based Hopping	2	2	2400	Deterrence; Restoration
1--DR	**User Display**	**Diverse Redundancy**	**1**	**4**	**1000**	**Deterrence**

Figure 30. The architecture candidate created automatically using the *Red Perspective*. The top displays summary information, including deterrence score, total costs, and rough break down of the importance of the system functions protected and the effectiveness of the design patterns chosen to protect those functions. The bottom shows which combination of system functions and design patterns were selected. This instance was generated using the information shown in Figure 26 and Figure 27.

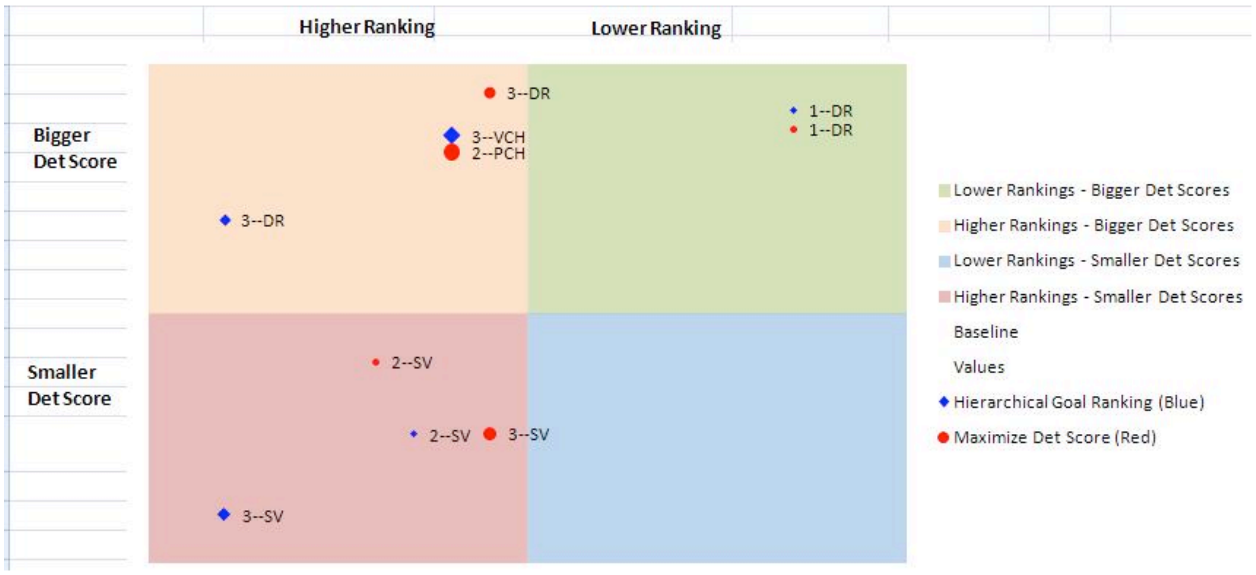


Figure 31. Quad chart displaying all of the selected combinations of system functions and design patterns in the candidate architectures generated by the prototype decision support system for the *Blue Perspective* and the *Red Perspective*. Those combinations selected as part of the *Blue Perspective* are represented by blue diamonds. Those combinations selected as part of the *Red Perspective* are represented by red circles. The size of the glyph (diamond or circle) represents that combinations relative contribution to the overall cost (bigger glyph represents a larger contribution). This instance represents the candidate architecture shown in Figure 29.

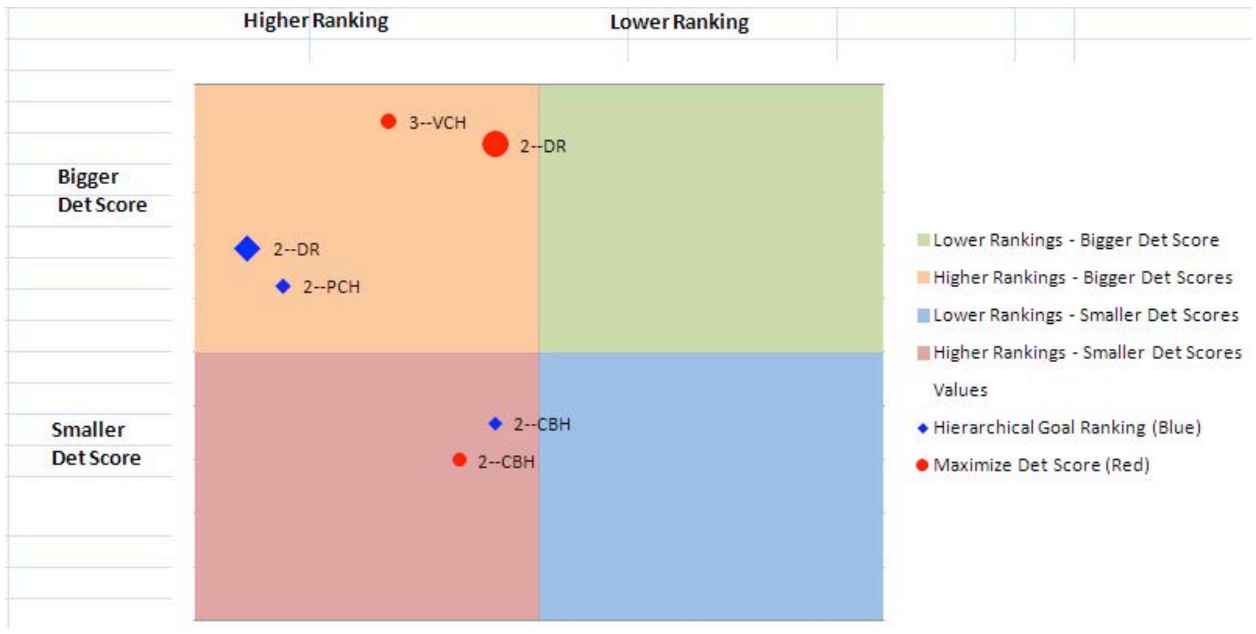


Figure 32. Quad chart displaying all of the combinations of system functions and design patterns not included in the candidate architectures generated by the prototype decision support system for the *Blue Perspective* and the *Red Perspective*. Those combinations not selected as part of the *Blue Perspective* are represented by blue diamonds. Those combinations not selected as part of the *Red Perspective* are represented by red circles. The size of the glyph (diamond or circle) represents that combinations costs compared to the costs of the other combinations not included in that perspective. This instance represents the candidate architecture shown in Figure 30.

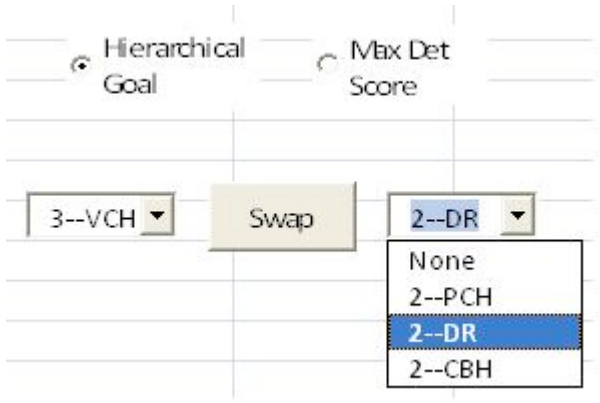
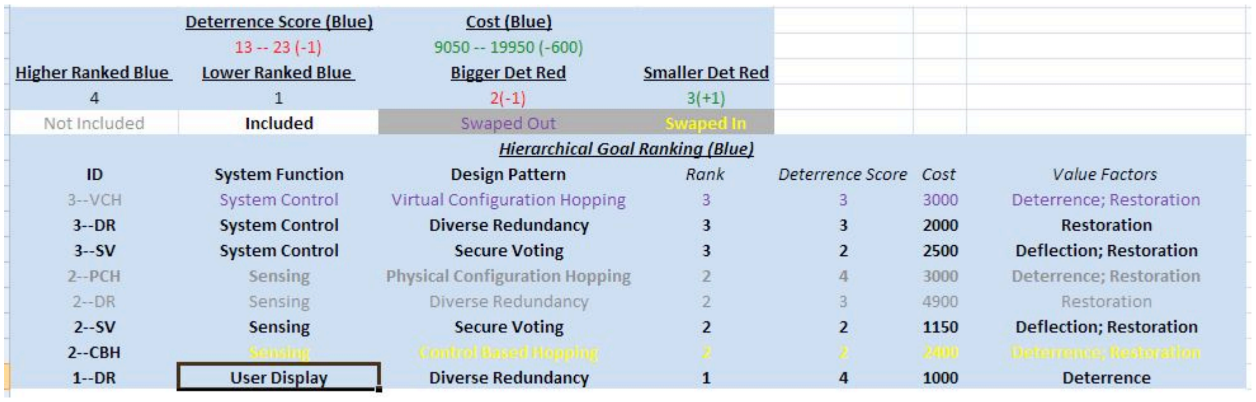


Figure 33. Set of tools the user can use to adjust the candidate architectures generated by the prototype decision support system (*Blue Perspective* and the *Red Perspective*). This includes the ability to select an architecture to modify—Hierarchical Goal (*Blue Perspective*) or Max Det Score (*Red Perspective*)—a selected combination of system function and design pattern to remove (left of the Swap button), and a combination not included in the candidate architecture to add (right of the swap button). This instance represents the information show in Figure 29.



	Deterrence Score (Blue)	Cost (Blue)	
	13 -- 23 (-1)	9050 -- 19950 (-600)	
Higher Ranked Blue	Lower Ranked Blue	Bigger Det Red	Smaller Det Red
4	1	2(-1)	3(+1)
Not Included	Included	Swaped Out	Swaped In

Hierarchical Goal Ranking (Blue)

ID	System Function	Design Pattern	*Rank*	*Deterrence Score*	*Cost*	*Value Factors*
3--VCH	System Control	Virtual Configuration Hopping	3	3	3000	Deterrence; Restoration
3--DR	System Control	Diverse Redundancy	3	3	2000	Restoration
3--SV	System Control	Secure Voting	3	2	2500	Deflection; Restoration
2--PCH	Sensing	Physical Configuration Hopping	2	4	3000	Deterrence; Restoration
2--DR	Sensing	Diverse Redundancy	2	3	4900	Restoration
2--SV	Sensing	Secure Voting	2	2	1150	Deflection; Restoration
2--CBH	Sensing	Control Based Hopping	2	2	2400	Deterrence; Restoration
1--DR	User Display	Diverse Redundancy	1	4	1000	Deterrence

Figure 34. Illustrates how the prototype decision support system supports the user as they adjust the candidate architectures into new architectures. In this instance the combination of system function and design pattern with ID 3—VCH was removed and the combination with ID 2—CBH was added. The changes to the effectiveness and costs of the architectures are show in summary statistics. Furthermore, these changes have been highlighted to indicate (potentially) positive (green) or negative (red) changes.

Use Case

During the course of developing the prototype decision support system, the authors were fortunate enough to have the opportunity to be able to use the prototype to aid in the design of a System-Aware architecture for an unmanned aerial vehicle (UAV). This afforded the opportunity to learn more about the benefits and limitations of both the prototype system and the enhanced scoring framework approach generally. However, before discussing the lessons learned several important limitations should be noted. First, the entire process was condensed into a short 5 hour window. This means that the process had to be simplified in order to fit into the allotted time. This led to the aggregation of system functions so as to create fewer categories for ranking. Second, due to the short window and small number of participants, separate teams could not be created for each of the steps as outlined in section 4.5. Instead all of the steps were performed by having all participants serve as the members for each team.

Despite the limitations of the meeting's setting, the proposed decision support was able to provide the group with a useful way of designing and selecting a System-Aware architecture. This included the ability to aid in the identification of important system functions as well in the selection—and creation—of System-Aware design patterns to protect those system functions. In addition, several additional areas for improvement were suggested. First, the prototype system currently assumes that all system functions are prioritized only according to the possible consequences that could result from a cyber attack; however, initial usage suggest that the prioritization of a system's functions is, in fact, a combination of multiple factors. For example, during the discussion all system functions were found to classified into three broad categories, (1) functions related to the system's operations (e.g., navigation), (2) functions used to carry out the system's specific mission objectives (e.g., radar), and (3) functions related to the system operators ability to control the system (e.g., the ability to set way points). Some members considered those functions related to the performance of the system as most critical: believing that these functions could be used to cause catastrophic damage to the entire system and compromise the mission. Other members disagreed with this assessment. They believed that such attacks could be easily identified and deflected by other means—such as operator intervention—and, furthermore, may degrade the system's functionality but not prevent it from accomplishing its mission. Instead, these members advocated that those system functions that were directly related to the mission and could be compromised in ways that were difficult to detect were most important to protect, as the consequences of such an attack might not result in catastrophic losses, but could result in mission failure. In addition, as these attacks would be difficult to detect, they could persist for multiple missions. For example, a cyber attack against a UAV's engine could result in the aircraft crashing—a catastrophic loss. However, the loss of an engine might be detected by the operator who could possibly take action to ensure the UAV's mission was completed before it crash landed. Alternatively, if the cyber attack compromised the UAV's radar system and resulted in it reporting misleading information, the UAV would be in no danger of loss, but its mission of detection and scouting might be compromised. In addition, as the radar is reporting realistic, and false information, such tampering may not produce any obvious signs and be difficult to detect; thus, persisting for an extended time frame. Finally, all members noted that those system functions that would be built from more stable components should be ranked higher; i.e., some system functions may be built using components that will be in service for years, while others may be built from components that will be upgraded frequently. Those system functions that will be upgraded frequently should receive a lower priority as the constant upgrading would provide a certain degree of protection by possibly deterring an adversary.

Currently the prototype system only provides the cyber attack assessment (red) team with the ability to assign an integer value of one to four to the security afforded by each system function design pattern solution. During the discussion, it was suggested that a larger range of values be available to allow the red team to more accurately assess the security offered by a given solution.

References

D. Albright, P. Brannan, and C. Walrond, Did stuxnet take out 1,000 centrifuges, Institute of Science and International Security, 2010.

A. Avizienis and J. P. J. Kelly, Fault tolerance by design diversity - concepts and experiments, Computer, Vol. 17 (1984), 67-80.

G. Babineau, R. A. Jones, and B. M. Horowitz, A system-aware cyber security method for shipboard control systems with a method described to evaluate cyber security solutions, IEEE International Conference on Technologies for Homeland Security, 2012, Accepted.

R. J. Bolton and D. J. Hand, Statistical fraud detection: a review, Statistical Science, Vol. 17, No. 3 (2002), 235-249.

J. L. Bayuk and B. M. Horowitz, An architectural systems engineering methodology for addressing cyber security, Systems Engineering 14 (2011), 294-304.

J. L. Bayuk, B. M. Horowitz, and R. A. Jones, Security via related disciplines, Procedia Computer Science 8 (2012), 338-344.

J. Cai, V. Yegneswaran, C. Alfeld, and P. Barford, Honeygames: a game theoretic approach to defending network monitors, University of Wisconsin-Madison, Technical Report 1577, 2006.

M. R. Clarkson, S. Chong, and A. C. Myers, Civitas: toward a secure voting system, IEEE Symposium on Security and Privacy, 2008, pp. 354-368.

US Department of Defense, Department of Defense Strategy for Operating in Cyber Space, Washington, DC, 2011.

Defense Science Board, High performance microchip supply, Department of Defense, Washington, DC, 2005.

DoD, The Under Secretary of Defense for Acquisition Technology and Logistics and The Assistant Secretary of Defense for Networks and Information Integration DoD Chief Information Officer, Report on trusted defense systems, Department of Defense, Washington, DC, 2009.

N. Falliere, L. O. Murchu, and E. Chien, W32.Stuxnet Dossier, Symantec, 2011.

FDA, Guidance for industry: evidence-based review system for scientific evaluation of health claims, U.S. Department of Health and Human Services, Silver Spring, MD, 2009.

FDA, Guidance for industry and FDA staff: total product life cycle: infusion pump – premarket notification [510(k)] submissions, U.S. Department of Health and Human Services, Silver Spring, MD, 2010.

I. N. Fovino, M. Masera, and A. D. Cian, Integrating cyber attacks within fault trees, Reliability Engineering & System Safety 94 (2009), 1394-1402.

A. Fujioka and T. Okamoto, A practical secret voting scheme for large scale elections, Advances in Cryptology – AUSCRYPT '92, 1992, pp. 244-251.

E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides, Design patterns: elements of reusable object-oriented software, Reading, Mass.: Addison-Wesley, 1995.

M. Howard, Writing secure code, 2nd edition, Microsoft Press, Seattle, WA, 2002.

Institute of Internal Auditors, International standards for the professional practice of internal auditing (standards), Issued 2008, Revised 2012.

International Maritime Organization, Resolution A.821(19) Performance Standards for Gyro-Compasses for High-Speed Craft, ISO 16328, London, United Kingdom: United Nations International Maritime Organization, 1995.

ISO/IEC (International Organization for Standardization/International Electrotechnical Commission), Risk management - risk assessment techniques, ISO/IEC 31010, Geneva, Switzerland, 2009.

R. A. Jones and B. M. Horowitz, System-Aware cyber security, itng, 2011 Eighth International Conference on Information Technology: New Generations, 2011, pp. 914-917.

R. A. Jones, T. V. Nguyen, and B. M. Horowitz, System-Aware security for nuclear power systems, 2011 IEEE International Conference on Technologies for Homeland Security, 2011.

R. A. Jones and B. M. Horowitz, System-Aware Cyber Security Architecture, journal: Systems Engineering, Vol. 15 (2), 2012, pp. 224-240

B. Horowitz and K. Pierce, Application of dynamic system models and state estimation technology to the cyber security of physical systems, journal: Systems Engineering, 2012.

T. P. Kelly and R. A. Weaver. The goal structuring notation – a safety argument notation, Proceedings of the Dependable Systems and Networks Workshop on Assurance Cases, 2004.

K. C. Knowlton, A combination hardware-software debugging system, *Computers, IEEE Transactions on*, vol.C-17, no.1, pp.84-86, 1968.

K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage, Experimental security analysis of a modern automobile, IEEE Symposium on Security and Privacy in Oakland, 2010, pp. 447-462.

Y. Kou, C. Lu, S., Sirwongwattana, Y. Huang, Survey of fraud detection techniques, Networking, Sensing and Control, 2004 IEEE International Conference on , vol.2, pp. 749- 754, 2004.

L. Lamport, R. Shostak, and M. Pease, The byzantine generals problem, ACM Transactions on Programming Languages and Systems, Vol. 4, No. 3, pages 382-401, July 1982.

M. Lennon, DDoS attacks exceed 100 Gbps, attack surface continues to expand. Retrieved October 7, 2011, from securityweek.com: http://www.securityweek.com/ddos-attacks-exceed-100-gbps-attack-surface-continues-expand

W. Lin and H. Chin, A new approach for distribution feeder reconfiguration for loss reduction and service restoration, *Power Delivery, IEEE Transactions on* , vol.13, no.3, pp.870-875, 1998.

S. Martello and P. Toth, Knapsack problems, Algorithms and Computer Implementations, Wiley-Interscience Series in Discrete Mathematics and Optimization, John Wiley and Sons Ltd., Chichester, 1990.

C. Menon, R. Hawkins, and J. McDermid, Defence standard 00-56 issue 4: towards evidence-based safety standards, Seventeenth Safety-Critical Systems Symposium, 2009, pp. 223-243.

MoD, Defence standard 00-56 issue 4: safety management requirements for defence systems, UK Ministry of Defence, 2007.

M. G. Noblett, M. M. Pollitt, and L. A. Presely, Recovering and examining computer forensic evidence, Forensics Science Communications 2 (2000).

J. Reynolds, J. Just, E. Lawson, L. Clough, R. Maglich, and K. Levitt, The design and implementation of an intrusion tolerant system, *Dependable Systems and Networks 2002, DSN 2002, Proceedings, International Conference on*, 2002, pp. 285-290.M. Schumacher, Security patterns: integrating security and systems engineering, John Wiley & Sons, Virginia, 2006.

B. Stackpole and P. Hanrion, Software deployment, updating, and patching, CRC Press, Boca Raton, FL, 2007.

G. Stoneburner, A. Goguen, and A. Feringa, Risk management guide for information technology systems, NIST (National Institute of Standards and Technology), Gaithersburg, MD, Special Publication 800-03, 2002.

E. A. Strunk and J. C. Knight. The essential synthesis of problem frames and assurance cases. Expert Systems, Vol. 25 (1), 2008, pp. 9–27.

US Comptroller General, Report to Chairman, Committee on Government Operations, House of Representatives of the United States, Washington, DC, 1981.L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, A break in the clouds towards a cloud definition, SIGCOMM Computer Communication Review 39 (2009), 137-150.

F. Webber, P. P. Pal, M. Atighetchi, C. Jones, and P. Rubel, Applications that participate in their own defense (APOD), BBN Technologies, Cambridge, MA, 2003.

W. A. Wulf and A. K. Jones, Reflections on cyber security, Science Magazine, vol. 326, 2009, pp. 943-944.