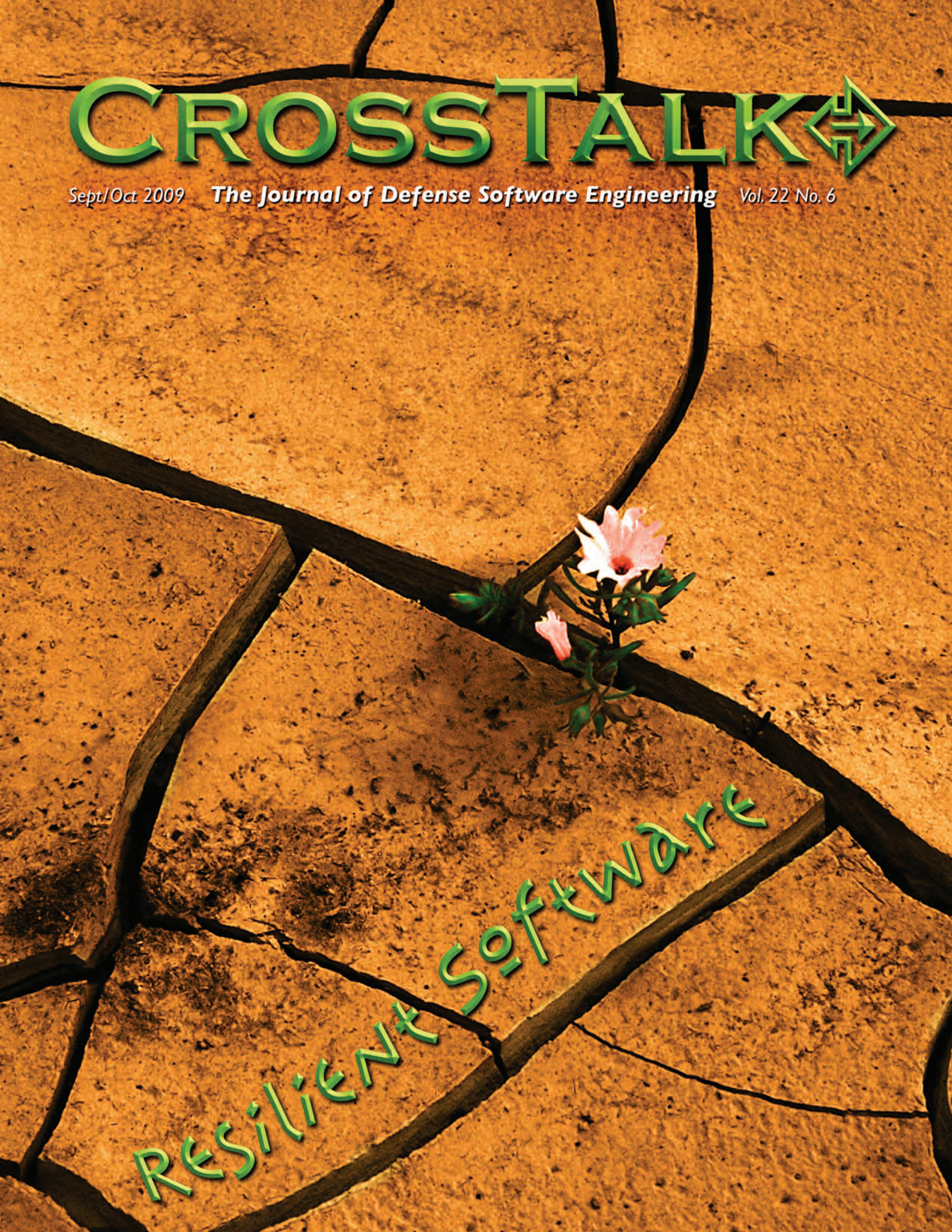

CrossTalk
Sept/Oct 2009 The Journal of Defense Software Engineering Vol. 22 No. 6
Resilient Software

Resilient Software

Software Defense Application: Each CROSSTALK article now has a sidebar that allows our authors to demonstrate the applicability of their article's central ideas, concepts, or processes for use in the DoD software community. Authors will use this section to share evidence of a demonstrative return on investment, process improvement, quality improvement, reductions to schedule, or other captured improvements.

ON THE COVER
Cover Design by Kent Bingham

Departments

CROSSTALK

OSD (AT&L) *Kristen Baldwin*
NAVAIR *Joan Johnson*
309 SMXG *Karl Rogers*
DHS *Joe Jarzombek*

MANAGING DIRECTOR *Brent Baxter*
PUBLISHER *Kasey Thompson*
MANAGING EDITOR *Drew Brown*
ASSOCIATE EDITOR *Chelene Fortier-Lozancich*
ARTICLE COORDINATOR *Marek Steed*
PHONE (801) 775-5555
E-MAIL stsc.customerservice@hill.af.mil
CROSSTALK ONLINE www.stsc.hill.af.mil/crosstalk

CROSSTALK, The Journal of Defense Software Engineering is co-sponsored by the Office of the Secretary of Defense (OSD) Acquisition, Technology and Logistics (AT&L); U.S. Navy (USN); U.S. Air Force (USAF); and the U.S. Department of Homeland Security (DHS). OSD (AT&L) co-sponsor: Software Engineering and System Assurance. USN co-sponsor: Naval Air Systems Command. USAF co-sponsor: Ogden-ALC 309 SMXG. DHS co-sponsor: National Cybersecurity Division in the National Protection and Programs Directorate.

The USAF Software Technology Support Center (STSC) is the publisher of CROSSTALK, providing both editorial oversight and technical review of the journal. CROSSTALK's mission is to encourage the engineering development of software to improve the reliability, sustainability, and responsiveness of our warfighting capability.

S T S C

Subscriptions: Send correspondence concerning subscriptions and changes of address to the following address. You may e-mail us or use the form on p. 14.

517 SMXS/MXDEA
6022 Fir AVE
BLDG 1238
Hill AFB, UT 84056-5820

Article Submissions: We welcome articles of interest to the defense software community. Articles must be approved by the CROSSTALK editorial board prior to publication. Please follow the Author Guidelines, available at <www.stsc.hill.af.mil/crosstalk/xtlkguid.pdf>. CROSSTALK does not pay for submissions. Published articles remain the property of the authors and may be submitted to other publications. Security agency releases, clearances, and public affairs office approvals are the sole responsibility of the author and their organizations.

Reprints: Permission to reprint or post articles must be requested from the author or the copyright holder and coordinated with CROSSTALK.

Trademarks and Endorsements: This Department of Defense (DoD) journal is an authorized publication for members of the DoD. Contents of CROSSTALK are not necessarily the official views of, or endorsed by, the U.S. government, the DoD, the co-sponsors, or the STSC. All product names referenced in this issue are trademarks of their companies.

CROSSTALK *Online Services:* See <www.stsc.hill.af.mil/crosstalk>, call (801) 777-0857 or e-mail <stsc.web master@hill.af.mil>.

Back Issues Available: Please phone or e-mail us to see if back issues are available free of charge.

Fortifying Our Cyber Defenses

Simply stated, absent secure and resilient software at the core of our cyber defenses, the nation's critical infrastructure is at risk. Everything we do as a nation—from national defense to re-energizing the economy—depends on secure information technology systems and networks.

Increasingly, however, these software controlled and enabled systems are vulnerable to exploitation by those that seek to do our nation harm, steal our intellectual capital, or simply collect our personal information. Making critical software assets secure and resilient is a necessary part of the nation's defense-in-depth approach to cybersecurity.

CrossTalk would like to thank the Department of Homeland Security for sponsoring this issue.

The DHS, and more specifically the Office of Cybersecurity and Communications, has the lead role in securing the civilian side of those critical networks and systems. A vital component of that effort is the National Cybersecurity Division's Software Assurance Program. The program works with its partners in the federal government, private sector, and international community to reduce software vulnerabilities, minimize exploitations, and develop secure and trustworthy software products. In short, it works to protect vital networks and systems by applying sound software supply-chain risk management.

With that in mind, two points merit emphasis. First, developing secure and resilient software alone is not enough. Increasingly, our critical cyber networks and systems are vulnerable to exploitation by a variety of actors. That means that unless the systems and networks controlled by the software in question are also protected, cybersecurity will remain an elusive goal. These factors are inexorably intertwined and must remain so in order to support mission requirements across enterprises and critical infrastructures. Sound cybersecurity practices must be overlapping, integrated, and supportive. In other words, they must be a "system-of-systems" that encompass all the people, activities, processes, and technologies that together promote and define a comprehensive national cybersecurity strategy.

Second, the DHS accomplishes its mission by working closely and collaboratively with the private sector. The government is best at developing policy objectives, identifying requirements, and facilitating the achievement of those objectives. The private sector specializes in finding ways to meet those objectives and requirements through technology innovation, experimentation, and innovative product development. Working separately, we will only get half of the job done. Working together, however, we can develop the necessary products to safeguard our critical systems.

So join us in our mission and be part of the software assurance solution. Visit our Web sites <https://buildsecurityin.us-cert.gov/swa/> and <http://www.us-cert.gov/>. Learn more about the Cross Sector Cybersecurity Working Group and the Software Assurance Forum. Better yet, become part of the public-private effort and learn how to participate in these important efforts. Together we can build a trusted and resilient information and communications infrastructure based on secure and resilient software.

I hope everyone appreciates the articles in this issue of CrossTalk that explore the multifaceted dimensions of software resiliency. I thank the authors for their important contributions. More importantly, the DHS continues to seek input and feedback on collaborative efforts to advance software assurance.

Michael A. Brown
Rear Admiral, USN
Deputy Assistant Secretary for Cybersecurity and Communications
U.S. Department of Homeland Security

Considering Software Protection for Embedded Systems

Dr. Yong C. Kim and Lt. Col. J. Todd McDonald, Ph.D[1]
The Air Force Institute of Technology

Software in modern embedded systems is often realized by using prefabricated reconfigurable computing devices such as Field Programmable Gate Arrays (FPGAs). Such devices support the use of portable hardware description languages and, as a result, have vulnerabilities consistent with normal software applications. In this article, we consider the nature of adversarial reverse-engineering attacks in this environment and measures of protection.

In our modern world, the meaning of a word can change quite often. Even the term *computer* previously referred to a human operator who crunches numbers while today we relate this term clearly to a machine. With the emergence of new reconfigurable computing technologies such as FPGAs, the definitions of software and hardware have become less clear. As Vahid suggests [1], we should stop calling circuits *hardware* and start broadening what we consider *software*.

In the traditional sense, software referred to the bits (1s and 0s) representing language statements that could be executed on hardware processors. Today, embedded systems utilizing FPGAs realize circuits merely by downloading a sequence of bits that instantiate gates, controllers, arithmetic logic units, crypto circuits, and even processors. Thus, a circuit implemented on embedded systems utilizing an FPGA is essentially software.

Considering the proliferation of embedded systems with reprogrammable hardware components in both commercial and military sectors, we can readily show the impact of malicious activity geared to reverse engineer, tamper, or copy critical technologies residing in those systems. In this article, we delineate protective transformations for such embedded logic and present a brief survey of reverse engineering attacks in this realm.

Characterizing Circuit Protection

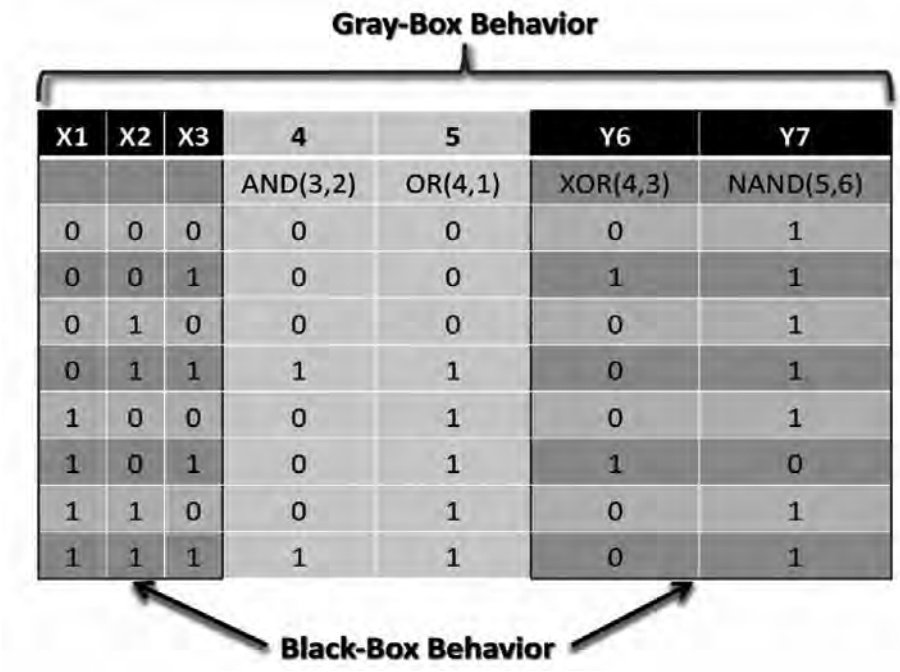
Both the DoD and the commercial sector have an interest in describing and measuring candidate protective measures, whether they derive from hardware anti-tamper realizations or software-based techniques. Adequately defining criteria for successful software *protection* in practice remains elusive mainly because full protection may not be possible, at least theoretically [2]. Collberg and Thomborson [3] describe three practical means of protecting software against copying, reverse-engineering, and malicious tampering; these include, respectively, watermarking, obfuscation, and tamper-proofing. In terms of analyzing protection mechanisms, they suggest measuring obfuscating transformations based on their obscurity (how much time is increased for understanding and reverse engineering), resilience (difficulty for reversing the transformation), stealth (the natural context of the transformation), and cost (overhead).

Though embedded systems may encompass a wide variety of custom processors and components, our discussion focuses on more fundamental logic programs represented as combinations of gate-level logic. In describing such circuits, we use two primary analysis paradigms: how they behave, and how they are constructed. We express the black-box behavior of a circuit by enumeration of all inputs, subsequent evaluation and propagation of signals on all intermediate gates, and the recording of the corresponding output. Figure 1 illustrates an input/output representation of a small combinational logic circuit with three inputs (X1, X2, X3), four intermediate gates (4, 5, 6, 7), and two distinguished intermediate gates (Y6, Y7) known as outputs.

We define a signal as a vertical reading of a column in the truth table (a fully enumerated input/output behavior, based on canonical ordering of inputs) and call the signature of a circuit the collection of its output signals. Given the full truth table of a circuit, we define its gray-box behavior as signals of all intermediate logic gates based on the enumeration of all possible inputs.

The white-box structure of a circuit may be represented by textual description languages (Bench, Verilog, VHDL, etc.), which are regular grammars that support expression of gates, electrical signals, components, and gate interconnections. Textual representations translate into graphical forms, which are referred to as

Figure 1: *Black-Box and Gray-Box Circuit Behavior*



the circuit topology. Figure 2 illustrates the circuit seen in Figure 1 in corresponding graphical representation and a Bench textual description [4]. We define a component within the circuit as a collection of lower-level elements (such as gates) that form a distinct sub-circuit.

The semantics (or black-box behavior) of a circuit consists of only the input and output signal pairs (the X and Y signals seen in Figure 1). Intuitively, one way to think of circuit protection is the act *hiding* all intermediate transitions that transform input to output. The collection of these transitions, in essence, represents the intellectual property of a circuit. Without knowledge of the original intermediate transitions, no human or automated process may derive other information about the original circuit such as topology, signal definitions, or component definitions. Many define the essence of circuit reverse engineering as the ability to correctly identify topology or components of the original circuit [4, 5].

To protect a circuit, replace the original circuit with a semantically equivalent version (one which does the same function) that hides the intellectual property of the original in some definable or measurable way. For the circuit in Figures 1 and 2, a replacement circuit would have identical signals for inputs and outputs (*X1, X2, X3, Y6, Y7*), but would have some other internal white-box construction (represented by gates 4 and 5 in Figures 1 and 2).

This formulation restates the essence of a virtual black box [2] because it defines full protection as a replacement circuit that does not leak any more information relative to an original circuit (other than its input/output characteristics). In more practical settings [3], the goal of using a replacement circuit becomes obscuring the original circuit in some way so that the cost of reverse engineering is maximized while operation characteristics of the circuit are not degraded beyond an acceptable level. Next, we delineate the permissible transformations on a circuit when obfuscation is in view.

Characterizing Circuit Transformations

We define an obfuscating transformation $O(\cdot)$ as an efficient, terminating program that takes circuit P as input and returns another circuit P': $O(P) = P'$. Of this assertion, all theoreticians and practitioners (that we are aware of) would agree. Beyond that, the majority of theoretical and practical models for obfuscation have

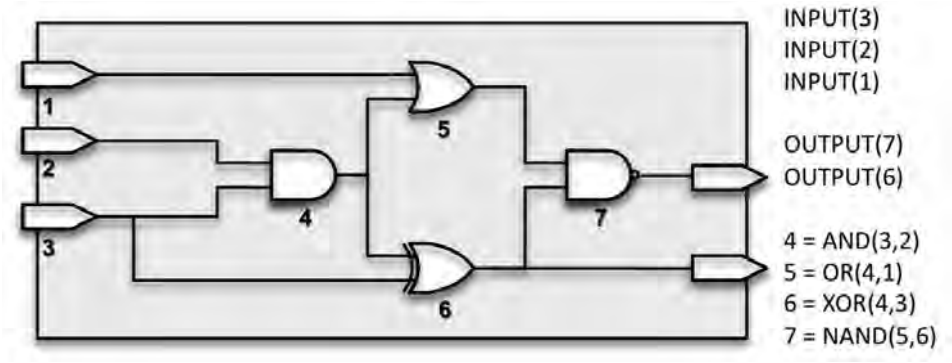


Figure 2: *White-Box Circuit Description*

at least two other requirements for the obfuscating program $O(\cdot)$: semantic equivalence and security.

We believe security may be provable in some circumstances if we are allowed to expand the semantic equivalence requirement (in other words, if an obfuscator can change the [white-box] structure of a circuit so that [black-box] input/output relationships of the original circuit *P* are also changed). We refer to black-box transformation with this meaning in mind. Likewise, the obfuscator may change (white-box) structure in such a way so that semantic equivalence with *P* is preserved. We refer to white-box transformation with this meaning in view.

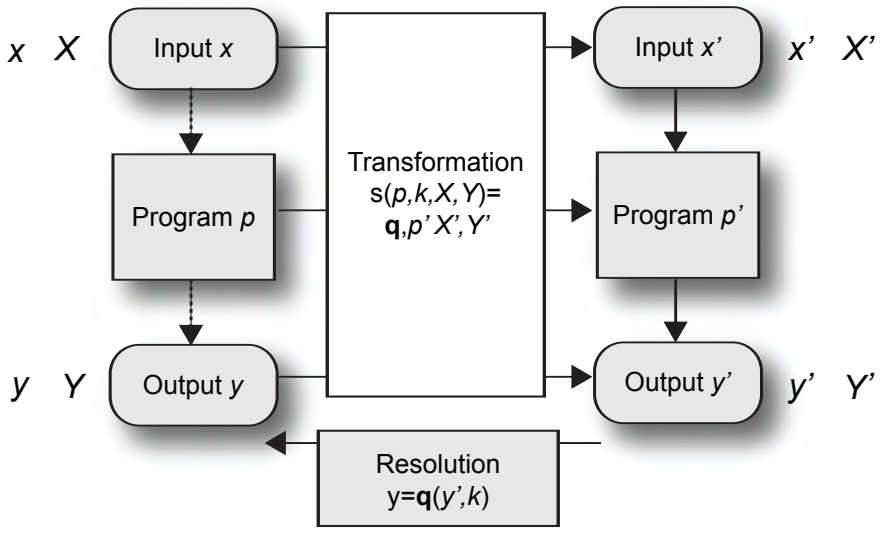
Black-Box Transformations

Sander and Tschudin [6] were one of the first to recognize the value of a black-box transformation as a means to hide functional intent. In discussing black-box changes to *P*, we assume there are certain programmatic environments where the output of the obfuscated circuit *P'* is conducive for off-line analysis and, therefore, open to the possibility of recovering the intended output of the original circuit *P*. In certain environments, this may not be possible. Black-box transformations, however, may be necessary to achieve stronger guarantees of security. In order to achieve a useful black-box transformation by some specific white-box changes to the structure of a circuit, an obfuscating operation must meet two requirements:

1. **Change in Black-Box Behavior.** The functional behavior changes for some majority of values in the domain x, $P(x) \neq P'(x)$. This leaves open the possibility that some transformations may produce equivalent values for certain values of x.
2. **Recovery of Black-Box Intent.** In order to recover the original functional output of *P*, some function $S(\cdot)$ must allow inversion: $V(x):P(x)=S(P'(x))$.

One way of hiding or masking input/output relationships is to do so through transformation that keeps the input/output values hidden in plain sight. We refer to such techniques as a black-box refinement of the original circuit *P* and present its algorithmic description in Figure 3. From the viewpoint of a circuit and its corresponding truth table, we can visualize at least five distinct operations that may be a part of a black-box refine-

Figure 3: *Black-Box Refinement*



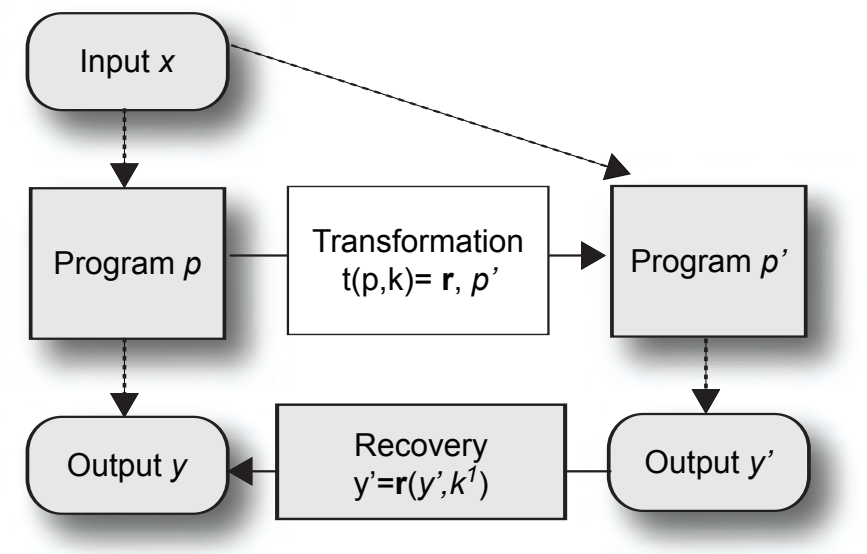


Figure 4: *Semantic Transformation*

ment. We envision that all five would be applied in a probabilistic manner based on configurable properties found in a (secret) key. If we let *X* represent the domain of the original *P* and confine it to a fixed number of bits, a black-box refinement may do any of the following:

1. Add input bits so that a new domain with a larger possible bit string *X'* is created.
2. Randomly permute the input bits themselves, resulting in a virtual reordering of the bits.
3. Introduce intermediate gates that would result in new truth table columns for *P'*.
4. Introduce a random number of output gates.
5. Randomly permute any of the output bits themselves.

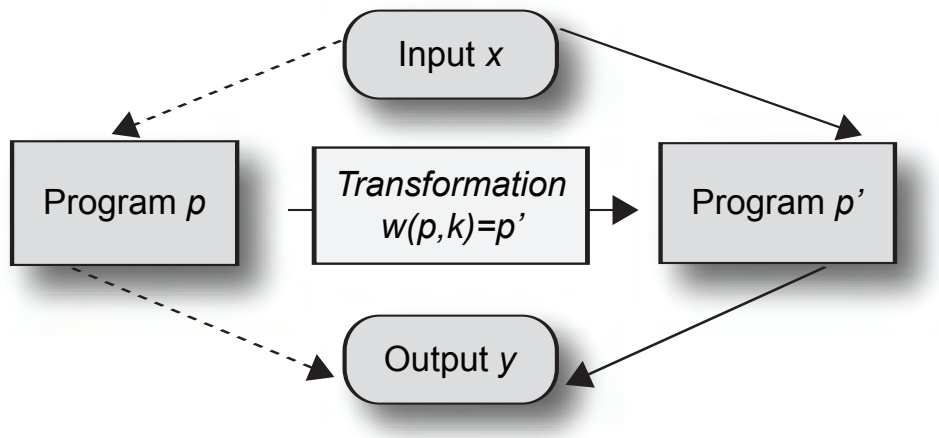
Changing the full input/output relationships of a circuit may truly hide the original black-box intent of a circuit. By composing a circuit with a semantically strong data encryption algorithm, the resulting program exhibits input/output relationships with desirable cryptographic properties. Figure 4 depicts this black-box change, known as *semantic transformation*.

White-Box Transformations

We define a structural white-box change to a circuit as a change to the topology of the underlying directed acyclic graph, which represents the circuit. Topological changes may involve textual renaming of signals or gates, changing the Boolean function type of particular gates, reordering input or output signals, introducing additional inputs, introducing additional outputs, concatenating the serial composition of the entire circuit with another circuit, merging the parallel composition of the circuit with another circuit, or replacing one or more gates within the circuit with a functionally equivalent set of gates.

Figure 5 shows the traditional meaning of obfuscation as understood in both theoretical and practical study: A transformation $w(P, k) = P'$ takes as input a circuit *P* with some (possibly) probabilistic information embodied in key *k*. We consider any random choices made by an obfuscation process to be part of this key. The output of $w(\cdot)$ is a circuit *P'* that remains functionally equivalent to the original circuit *P* and represents a different version of the original. Current obfuscation research centers on the transformation algorithm and defining the security that is achieved by its use.

Figure 5: *White-Box Transformation*



Reverse-Engineering Attacks

In the world of real circuit analysis, the typical goal of a reverse engineer is to recover the input/output of the circuit in question by some method less than full exponential enumeration. As we have already alluded to with black-box refinement or semantic transformation, such transformations would (at a minimum) prevent this form of reverse engineering while simultaneously introducing the need for output recovery in order to maintain functional utility. There are a number of different ways to discover and alter the functionality of a circuit. The term *tampering* refers to broad categories of circuit exploitation, including subversion, modification, and reverse engineering. Reverse engineers typically target reproduction of a circuit's functionality, usually for capital gain or malicious intent. Specific attacks can be roughly categorized as brute force, white-box/gray-box, side-channel, and fault injection.

Brute Force Attacks

Brute force attacks are based on black-box circuit behavior and are performed either while the circuit is in its natural environment or standalone in a simulator. Such attacks can be categorized as either *general* or *passive*.

- **General black-box attacks.** Traditionally, black-box attacks are the first and simplest means to reverse engineer a circuit. Adversaries glean black-box behavior by enumerating all possible input combinations and recording corresponding outputs. Using a large truth table, data analysis algorithms—or in some cases visual inspection—the adversary may re-create the underlying Boolean equations that define the circuit's logic; this type of attack works well on circuits with well-defined inputs and outputs.

  There exists potentially $2n$ input combinations to fully characterize any combinational circuit and potentially $2n + m$ or more input combinations for sequential circuits with m sequen-

tial elements. For a typical circuit with 100 or more inputs, a conventional black-box attack is not practical due to an enormously large search space. For example, a simple 64-bit adder with a carry-in pin has a total of 129 input pins and 65 output pins. If the reverse engineer, with no prior knowledge of the circuit applies the inputs, it would take 2^{129} attempts or 2^{99} seconds, roughly 2×10^{22} years, using a state-of-the-art 1 gigahertz automatic test equipment costing well over $1 million.

- **Passive attacks.** In passive attacks, adversaries examine circuits in their native environment (i.e., while they are being used in an actual circuit). Input and output pins are monitored, using either an oscilloscope or logic analyzer, and data is recorded giving a good picture of the chip's functionality. Typically, adversaries use passive attacks to provide focus for later black-box attacks that require a smaller distribution of input values.

White-Box/Gray-Box Attacks

In physical realizations, white-box attacks focus on the structure of a circuit. An adversary attempts to gain access to the internal nodes of a circuit without having to go through input/output evaluation, allowing a better functional understanding. Even though adversaries may risk destroying delicate circuit internals, these techniques are the only way to get direct access to the underlying white-box structure of a circuit in the real world. In order to extract white-box descriptions, adversaries focus attention on silicon characteristics using specific technologies such as ion beams and optical equipment:

- **Focused Ion Beam.** The focused ion beam is a semiconductor fabrication device similar to the scanning electron microscope (SEM), but it uses gallium ions instead of electrons. Unlike the SEM, it has a destructive effect as the gallium ions are implanted into the sample surface. This method allows an adversary to set specific intermediate nodes to specific values (0 or 1), including modifying existing connections to bypass normal input signal propagation. Likewise, an adversary does not have to rely on the actual output of the circuit in order to examine intermediate propagation values.
- **Optical Equipment.** Optical attacks rely on the interaction of photons with silicon devices and take two forms: optical probe and optical attack. Optical probing focuses on circuit examination by looking at transistor states. Adversaries essentially use pictures to observe signals that are propagated by means of applied input values.

Side-Channel Attacks

We observe that even circuits that may be provably secure according to a theoretical model—based on static white-box and dynamic black-box behavior—may still leak critical information relative to the circuit's function (based on real-world implementation issues). Rather than use brute force (to glean black-box behavior) or physically probe the internals of a circuit (to glean white-box and gray-box behavior), side-channel attacks use secondary information to create a picture of circuit functionality. Side channels are areas of a circuit that leak unintended information. They include power consumption and timing analysis:

- **Power Consumption.** Power consumption attacks mainly focus on breaking cryptographic schemes. The concept is that through an examination of the power used by a circuit, the underlying encryption algorithm can be found. This approach gives an attacker insight into the data values that are being manipulated on a chip. It is possible to then correlate this collected data to known functions in order to see exactly what is happening.
- **Timing Analysis.** With brute force attacks, synchronous circuits add additional complexity in the reverse-engineering process due to the timing constraints that are introduced. Timing attacks focus on taking the circuit outside of normal parameters by modifying the speed of the clock, either speeding it up or slowing it down. Because timing is linked directly to real-world physical implementations of various circuit technologies, our existing obfuscation framework requires additional information regarding structural characteristics of the circuit implementation.

Fault Injection

Fault injection is a generic term describing the injection of faults into digital systems using a variety of attacks: raising voltage higher or lower than system tolerances, inducing voltage spikes, or introducing clock glitches. An adversary may use any of these methods to cause the system to malfunction with intentions of revealing information useful in further attacks. The adversary performs fault injection dynamically at circuit run-time combined with power analysis techniques. Encryption algorithms, such as the Advanced Encryption Standard (AES), provide strength against brute-force key discovery from black-box behavioral analysis. However, an adversary may use fault injections with realized AES circuits in order to reduce encryption strength via key-space reduction. This exploit requires internal circuit access and reduces the goal of the adversary from using brute-force methods to interrupt the successful encryption/decryption process itself.

Software Defense Application

Considering the proliferation of embedded systems with reprogrammable hardware components in both commercial and military sectors, we can readily show the impact of malicious activity geared to reverse engineer, tamper, or copy critical technologies resident in those systems. Both the DoD and industry have interest in understanding how to describe and measure candidate protective measures, whether they derive from hardware anti-tamper realizations or software-based techniques. This article deals specifically with the characteristics of protection, possible transformations, and the delineation of malicious attacks.

Conclusion

Given the current trend of reprogrammable embedded devices within the DoD and industry, attention needs to be refocused on the benefits or measurability of software protection applied to this domain. Modern reconfigurable embedded systems now require us to consider circuits as software and the tamper methods applicable to physical circuits as new threats to a broadened definition of software. This article has presented a brief overview of the characteristics, transformations, and attacks possible in the realm of software implemented as circuits on an embedded system. Ultimately, we must turn our attention to the protection of critical technology resident in such an embedded system, mindful of the possible threats and techniques at our disposal.◆

References

1. Vahid, Frank. "It's Time to Stop Calling Circuits 'Hardware'." *IEEE Computer Magazine* 40.9 (Sept. 2007): 106-108.
2. Barak, Boaz, et al. "On the (Im)possibility of Obfuscating Programs." *Electronic Colloquium on Computational Complexity*. 15 Aug. 2001 <http://eccc.hpi-web.de/eccc-reports/2001/

TR01-057/Paper.pdf>.
3. Collberg, Christian S., and Clark Thomborson. "Watermarking, Tamper-Proofing, and Obfuscation—Tools for Software Protection." *IEEE Transactions on Software Engineering* 28.8 (Aug. 2002): 735-746.
4. Hansen, Mark C., Hakan Yalcin, and John P. Hayes. "Unveiling the ISCAS-85 Benchmarks: A Case Study in Reverse Engineering." *IEEE Design & Test of Computers* 16.3 (1999): 72-80.
5. Nohl, Karsten, et al. *Reverse-Engineering a Cryptographic RFID Tag.* Proc. of the USENIX Security Symposium. San Jose, CA. 31 July 2008 <www.cs.virginia.edu/~evans/pubs/usenix08/usenix08.pdf>.
6. Sander, Tomas, and Christian F. Tschudin. "On Software Protection via Function Hiding." *Lecture Notes in Computer Science* 1525 (1998): 111-123.

Note

1. The views expressed in this article are those of the authors and do not reflect the official policy or position of the U.S. Air Force, DoD, or U.S. government.

About the Authors

Yong C. Kim, Ph.D., is an assistant professor in the department of electrical and computer engineering at the Air Force Institute of Technology (AFIT). He received his doctorate in electrical engineering from the University of Wisconsin-Madison (UW-M), his master's degree in computer engineering from the UW-M, and his bachelor's degree in computer engineering from the University of Washington.

Dept. of Electrical and Computer Engineering
AFIT
2950 Hobson Way
Wright-Patterson AFB, OH 45433-7765
Phone: (937) 255-3636 ext. 4620
Fax: (937) 656-7061
E-mail: yong.kim@afit.edu

Lt. Col. J. Todd McDonald, Ph.D., is a Lieutenant Colonel in the USAF and an assistant professor in the department of electrical and computer engineering at the AFIT. He received his doctorate in computer science from Florida State University, his master's degree in computer engineering from the AFIT, and his bachelor's degree in computer science from the USAF Academy.

Dept. of Electrical and Computer Engineering
AFIT
2950 Hobson Way
Wright-Patterson AFB, OH 45433-7765
Phone: (937) 255-3636 ext. 4639
Fax: (937) 656-7061
E-mail: jmcdonal@afit.edu

Resilient Mixed-Criticality Systems

Dr. Lui Sha
University of Illinois at Urbana-Champaign

Most complex cyber-physical systems (CPSs) are mixed-criticality systems that have to be resilient against software design faults, hardware failures, and physical hazards under software control. This article reviews useful design principles and architecture patterns for the development of such systems.

Complex CPSs are typically mixed-criticality systems. They have to be resilient against not only faults and failures in a cyber subsystem but also hazards in a physical subsystem (plant, or device) under software control. Consider the following examples:

Controlling Human Errors and Hazards

After a major surgery, the patient is allowed to operate an infusion pump (patient-controlled analgesia [PCA]) with potentially lethal painkillers such as morphine sulfate. When pain is severe, the patient can push a button to get more pain-relieving medication. This is an example of a safety-critical device controlled by an error-prone operator (the patient). Nevertheless, the PCA system as a whole needs to be certifiably safe in spite of mistakes made by the patient. To solve this problem, medical instruments (sensors) are used to monitor the vital signs. An important element to the sensors is this: A safe dosage, with respect to the patient's conditions, will be delivered for a fixed duration only if all vital signs are within thresholds. Otherwise, an alarm sounds and the infusion stops.

In this example, the cyber subsystem is the computing hardware and software, the *plant* is the patient, the infusion pump is the safety-critical *actuator device*, and the vital signs are the states of the device (or plant) being monitored by sensors. Finally, a CPS is said to be certifiably safe if we can verify that the plant can remain in a safe state (the pain-killer concentration in patient's blood is below a dangerous threshold) with respect to a given set of internal system faults and external safety hazards including a patient's incorrect commands, power failure, and the loss of vital signs signals due to sensor and/or connection failures.

Dangers of Implicit Assumptions and the Need for Both Worst and Average Case Analysis

The safety certification procedure includes the assumptions and specifications of the operational environment, the set of devices and their configurations, the software, and the faults and hazards model. Note that a common cause of system failures in the field is that environmental assumptions embedded in software are implicit.

For example, the Ariane 5 rocket (also known as Flight 501) reused Ariane 4's software, which had correctly assumed that the rocket's (Ariane 4's) horizontal velocity could not overflow a 16-bit variable. Unfortunately, this was not true for Ariane 5 and led to its explosion during its maiden flight [1]. Making assumptions explicit and preferably machine-checkable is an important aspect of building resilient systems. In the development of a system of systems, the new integrated system typically contains many reused subsystems with implicit assumptions embedded in the software.

In addition to safety, the manufacturer of a resilient system must demonstrate the effectiveness of the system under nominal operational conditions. Note that safety is a worst-case analysis, while effectiveness is an average-case analysis. For example, if all components work normally, the PCA pump should deliver the painkiller according to the prescription. Furthermore, it should never overdose the patient even if the patient pushes the deliver button too many times, sensors fail and/or disconnect, and/or there is power failure.

Impractical Correctness

In a typical flight-control system, the autopilot is classified at DO-178B, Level A—the highest safety-critical level—while the flight guidance system (FGS), because of its complexity, is only certified to Level C [2]. Nevertheless, the Level C guidance system issues commands to steer the Level A autopilot. This is an example of safely using a component whose correctness is impractical to verify under current technologies. The overall flight control has to be certified to Level A again.

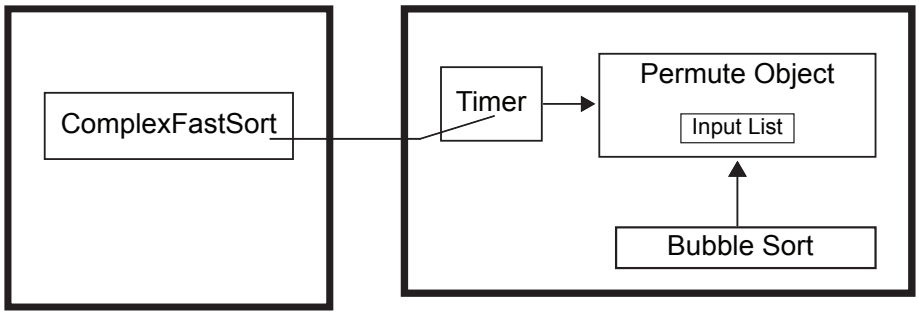
To solve this problem, the control authority of the FGS is first constrained so that the dynamics of the airplane cannot be changed abruptly. This gives the pilot enough time to detect the problem(s) and to take control in time. In addition, a Level A monitor can be used to 1) monitor stability margin in the control of the plane, and to 2) monitor if the plane closely follows the flight path. If any one of the thresholds is violated, an alarm will be sounded and the control is transferred to the pilot.

The following section reviews useful architecture patterns and tools to build resilient systems against software faults and hazards in the physical plants under software control. We (that is, organizations using these methods) begin with software design and/or implementation faults.

Architecture Patterns for Resiliency

Logical complexity is a major driver of software defects. I begin with a simple example to illustrate the idea of "using simplicity to control complexity" to build resilient applications [3]. Consider the problem of sorting. In sorting, the *safety* property is to sort items correctly. The *effectiveness* property is to sort them fast. Suppose that we could formally verify a Bubble Sort program but were unable to verify a ComplexFastSort program. Can we *safely* use the unverified ComplexFastSort for *effectiveness*? Yes, we can.

Figure 1: *Always Correct Sorting System*



As illustrated in Figure 1 (on the previous page), to guard against all possible faults of ComplexFastSort, these two programs are put into two virtual machines. In addition, a verified object called *permute* is developed that will: 1) allow ComplexFastSort to perform all the list operations that are useful in sorting related operations, but not to modify, add, or delete any list item; and 2) check in linear time that the output of ComplexFastSort is indeed sorted. Finally, a timer is set based on the promised speed of ComplexFastSort that is supposed to be faster than the Bubble Sort. If ComplexFastSort does finish in time and the answer checked is correct, then the result is given as output; if it does not finish in time or does so with an incorrect answer, then Bubble Sort sorts the data items. Note that if ComplexFastSort works with time complexity *n log(n)*, the system has the same time complexity *n log(n)*. That is, the lion's share of the effectiveness offered by unverified ComplexFastSort is captured with a small amount overhead when ComplexFastSort works. In addition, we guarantee the safety (items sorted correctly) with respect to the given set of specified safety hazards and faults, namely arbitrary application software errors. So far, there is no protection against virtual machine and/or hardware failures. Such limitations must be noted explicitly. If the application requires the tolerance of hardware failures, then fault-tolerant hardware must be used.

The moral of this story is that we can safely exploit the features and performance of complex components, even if it may have residual defects, as long as we can guarantee the critical properties by simple software and an appropriate architecture pattern. This is the idea of using simplicity to control complexity, which is the guiding principle of building resilient mixed-criticality systems. We want an architecture that can safely utilize the features and performance of lower-criticality components that are impractical to fully verify.

Checking the correctness of an output before using it—such as in the sorting example—belongs to a fault-tolerant approach known as a recovery block [4]. However, in CPS applications, it is often not possible to determine if every command from a complex controller is correct (meeting the specifications).

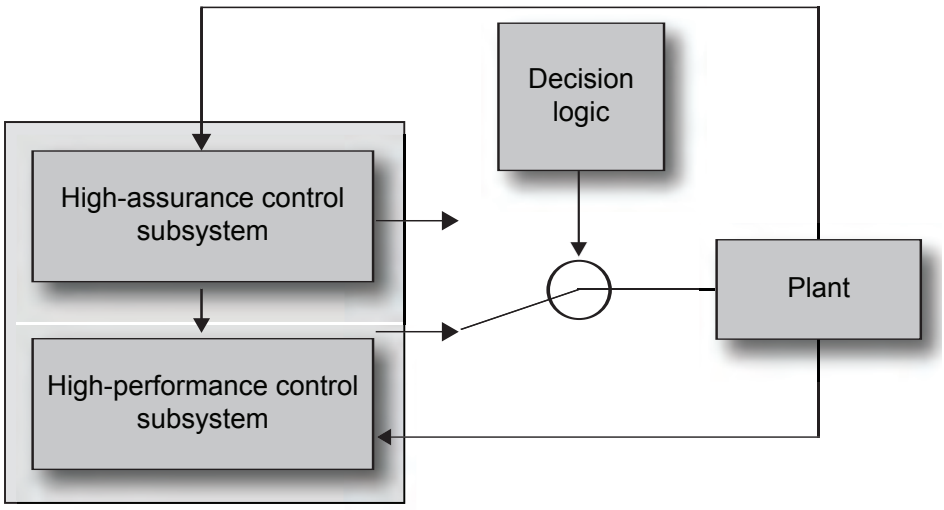
Simplex Architectures

A Simplex Architecture [3] is an architecture pattern for resilient control systems. As illustrated in Figure 2, a Simplex Architecture consists of 1) a safety core with a simple and verifiable high assurance controller and decision logic, and 2) a complex high performance system that cannot be fully verified. There are many failure modes of software. To protect the safety core, it should be run in a different real-time virtual machine.

To guard against real-time operating systems failures and/or security attacks that may breach the firewalls, we can put the safety core into a field programmable gate array (FPGA), a programmable hardware device. For protection purposes, FPGA devices should not be allowed to be reprogrammed during runtime. To ensure the correctness of the FPGA programming, we first perform model checking on the safety core design and then directly generate very high-level hardware description code to program the FPGA.

As shown in Figure 3, this hardware and software co-design approach is known as System-Level Simplex Architecture [5], which was developed for the design of a prototype pacemaker for patients with heart diseases. Pacemakers are also mixed-criticality systems. The safety core is a simple timer for rest rate pacing. If heartbeat is detected, the timer is reset. Otherwise, it sends out a pulse. This simple safety core can be done directly in hardware but it must be safely interfaced with microprocessor-based adaptive pacing, which will pace the heart faster if built-in sensor and motion detection software detects that a patient is exercising. Additional effectiveness features may include the detection and storage of the most important abnormal heartbeats. The rest rate pacing must work even if the microprocessor and its software fail. This is an example of discrete control of a *plant* (in this case, a human heart). The following will illustrate how a Simplex Architecture handles incorrect control commands from the complex high-performance controller for continuous dynamics.

Figure 2: *Simplex Architecture*



Operational Constraints

In the operation of a plant with continuous dynamics, there is a set of state constraints called operational constraints that represent the safety, device physical limitations, environmental, and other operational requirements. Consider the example of controlling an inverted pendulum mounted on a cart that runs on a track (see Figure 4). The controller must actively move the cart left or right to keep the rod balanced in the upright position. The safety constraint is that it cannot fall down. That is, the angle of the rod must be always less than 90 degrees from the upright position. Device constraints include the length of the track and the limited torque of the motor that runs the cart. Intuitively—to keep the pendulum balanced near the center of the track—the angle should be kept so it doesn't deviate too far from the upright position, and the cart doesn't veer too far from the center of the track. In addition, we need to keep the angular velocity and cart velocity limited. Otherwise, the cart may hit the end of the track, or the rod may fall down with too large of an angle and too large of an angular velocity, such that the inverted pendulum mounted on the cart becomes impossible to keep from falling down.

In control theory, this intuition is represented by the notion of a stability envelope within all of the operational constraints. This envelope represents a subset of the plant states in control of the pendulum. They are: angle, angular velocity, track position, and track velocity, within which the controller can keep the rod upright without violating any of the operational constraints. This envelope is a function of the plant model, the controller design, and the operational constraints. It can be computed by following the steps outlined next.

The operational constraints are represented as a normalized polytope in the N-dimensional state space of the system under control (as shown in Figure 5). Each line on the boundary represents a constraint. The states inside the polytope are called *admissible states* because they obey the operational constraints. We must ensure that the system states are always admissible. This means that we must be able to: 1) take the control away from a faulty control subsystem and give it to the high assurance control subsystem before the system state becomes inadmissible; 2) ensure that the system is controllable by the high assurance control subsystem after the switch; and 3) keep the future trajectory of the system state, after the switch, within the set of admissible states. Note that we cannot use the boundary of the polytope as the switching rule just as we cannot stop an out-of-control car, inches from a wall, from colliding with it. Physical systems, just like a moving car, have inertia.

The Recovery Region

A subset of the admissible states that satisfies these three conditions is called a recovery region. The recovery region is represented by a Lyapunov function[1] within the admissible states. Geometrically, a Lyapunov function defines an N-dimensional ellipsoid in the N-dimensional system state space. The boundary of this ellipsoid corresponds to the system's stability envelope. A two-dimensional example is illustrated in Figure 5. A Lyapunov function has the important property that as long as the system state is inside the ellipsoid associated with a controller, the system states under that controller will stay within the ellipsoid and converge to the set point. The largest ellipsoid inside a polytope can be found by using the Linear Matrix Inequality method [6]. The inner ellipsoid is the recovery region used for operation. The shortest distance between the outer and inner ellipsoids is the stability margin. The stability margin allows us to compensate for approximation errors in the plant model, measurement errors in sensing, actuation errors during operation, and disturbance to the plant (e.g., such as wind gusts against an airplane in a storm).

Controllers

During runtime, the plant is normally under the control of a high-performance-control subsystem. The high-assurance controller is a simple and well-understood classical controller. It is executing in parallel to the high-performance controller (HPC). A typical design is to run the two controllers at the same rate, for example, at 100 hertz.

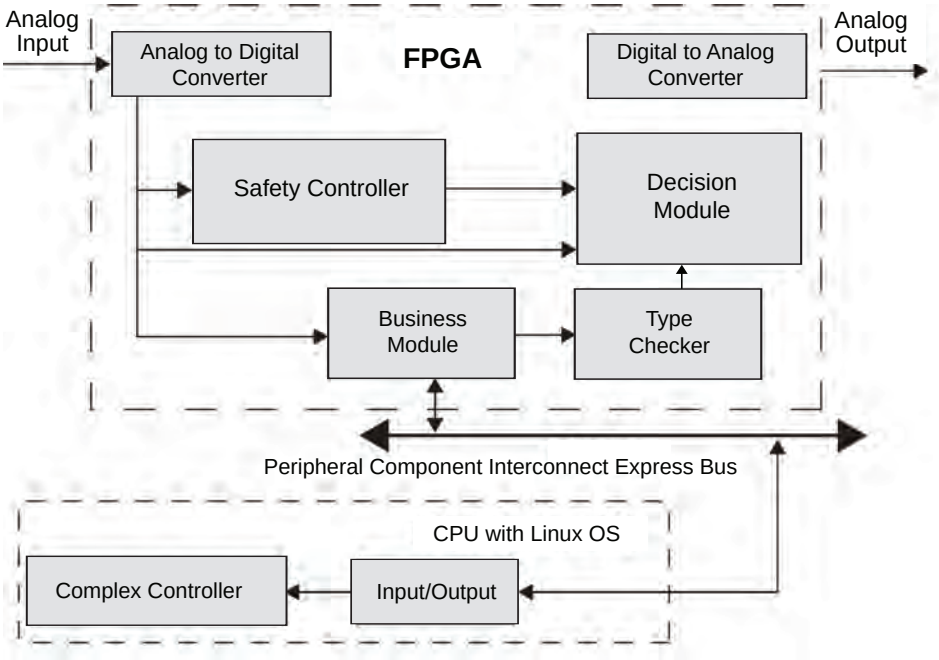


Figure 3: *System-Level Simplex Architecture*

Schedulability analysis is performed to ensure that both can finish before the end of the period. As a further precaution, the safety controller is run first. Should the HPC not finish by its deadline, it will be terminated and the command from the safety controller is used. Otherwise, for each command from the HPC, the decision logic estimates the next state if the command was used. If the next state is within the recovery region, the command will be executed. Otherwise, the high-assurance controller takes over and the HPC is terminated (a small number of restarts are typically permitted). In addition, the operator may terminate the HPC for other reasons such as certain features not being suitable for the current operation.

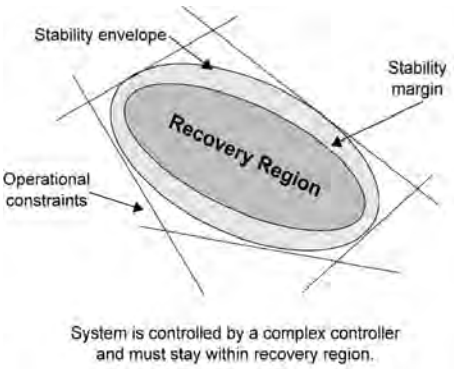
Switching from one controller to the other (hybrid control) may introduce transient errors in the control. A common example is the transient *jump* of a car's velocity when the transmission shifts gears. In the stability analysis, such transient errors must add to the fault and hazard model. That is, in the design of stability margin, it is assumed that when the plant state is at the boundary of the inner-recovery region, the HPC fails and the system switches to the safety controller and the control error, due to switching, reaches its maximal value. As well, the plant model approximation error is maximal, and the actuation errors and external environment disturbances such as wind gusts against the plane are also maximal. In a storm, for example, wind gusts reach a maximal value according to a storm model. The safety controller and the stability margin are designed to accommodate the worst-case scenarios with respect to the fault and hazard model. As a result, a Simplex Architecture tolerates concurrent software faults and disturbance to the physical plant.

A Real-World Example

A noteworthy example of using simplicity to control complexity is the flight control system of the Boeing 777 [7]. It uses triple-triple redundancy for hardware reliability. At the software application level, it uses two controllers. The sophisticated control software, specifically developed for the Boeing 777, is the normal controller because it has many new effectiveness fea-

Figure 4: *An Inverted Pendulum*

Figure 5: *Recovery Region*



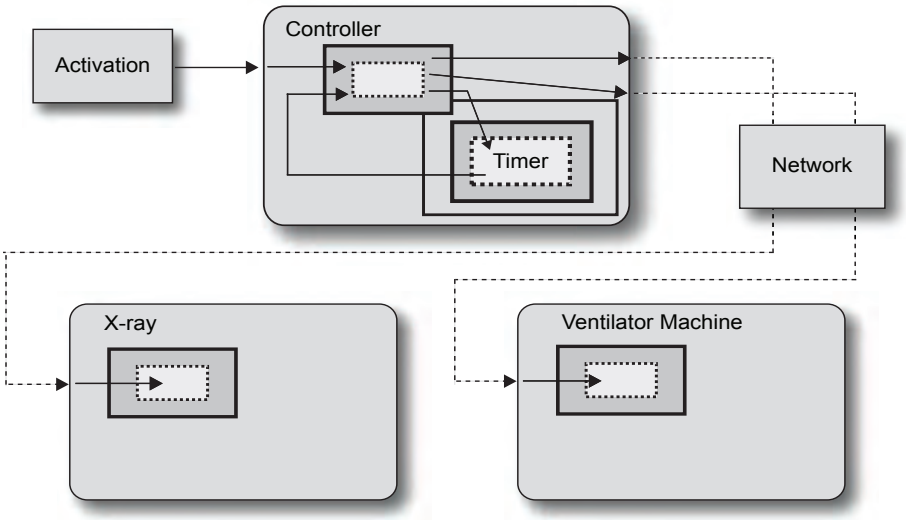


Figure 6: *Configuration 1 of Ventilator Machine With X-ray Machine and Controller*

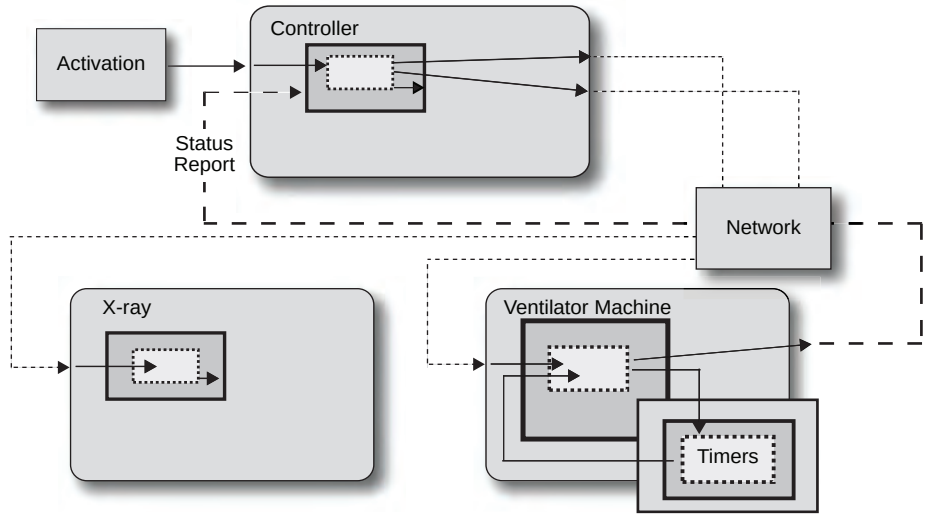


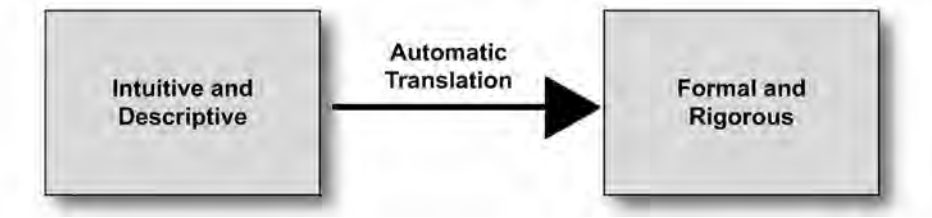
Figure 7: *Configuration 2 of Ventilator Machine With X-ray Machine and Controller*

tures such as highly effective automatic stabilization against wind gusts, advanced fuel-savings, and reduced wear-and-tear to mechanical actuators.

As a result, the 777 controller software is much larger and complex than the 747 controller software. The secondary controller is based on the control laws developed for the Boeing 747, which have been used for decades. It is a mature technology: simple, reliable, and well understood. It is a simple component since it has low residual complexity[2]. It should be noted that the 777 control software was certified to meet safety-critical requirements.

The use of the simpler 747 controller-based software as backup is a precautionary measure for added reliability. This is a best practice because it is uncertain if the process-oriented DO-178B can remain effective when used with increasingly complex software that cannot be exhaustively tested. And while formal model-checking technologies can be scaled up to practice systems and are effective in detecting many types of software defects in a design, it is not formal proof of software meeting all of the specifications—nor is it a verification of the software implementation that flies an airplane.

Figure 8: *AADL to Real-Time Maude Translation*



Formalized Architecture Patterns

Since architecture patterns often need to be adapted for new application requirements, we need to not only verify a collection of commonly used architectural patterns, but also provide computer-aided verification for the adaptation of architectural patterns. Furthermore, model-based approaches are the most common way of capturing architectural designs and architectural patterns; it is important to provide formal verification support for architectural patterns expressed in software modeling languages. The following example illustrates the use of:

- Complexity control architectures and design rules for a medical system.
- A formalized SAE International Architecture Analysis and Design Language (AADL) [8] subset to specify these architectures. AADL is a standard architecture analysis and description language.
- AADL models automatically transformed into algebraic expressions in Real-Time Maude [9] for formal analysis. Real-Time Maude is a model-checking language that supports the checking of hard real-time constraints in addition to temporal logic expressions.

Prevention Through Automation

One example comes from the Anesthesia Patient Safety Foundation:

> A 32-year-old woman was having a laparoscopic cholecystectomy (surgical removal of the gall bladder) performed under general anesthesia. During that procedure and at the surgeon's request, a plain film X-ray was shot during a cholangiogram. The anesthesiologist stopped the ventilator for the X-ray. The X-ray technician was unable to remove the film because of its position beneath the table. The anesthesiologist attempted to help the technician, but found it difficult because the gears on the table had jammed. Finally, the X-ray was removed, and the surgical procedure recommenced. At some point, the anesthesiologist glanced at the EKG and noticed severe bradycardia. He realized he had never restarted the ventilator. This patient ultimately died. [10]

This accident could have been prevented by automation. However, there are two candidate configurations:

- **Configuration 1.** As illustrated in Figure 6, the X-ray machine and ventila-

tor are networked together with a control station. The control station could command the ventilation to pause, the X-ray machine to take a picture, and then command the ventilator to resume. In addition, two watchdog timers are added to the control station. The first one limits the maximum duration of each pause. The second one ensures that pauses are separated by the minimum duration. Both of them are configuration time constants set by medical personnel. However, such a design is unacceptable because if either the network or the control station fail—after commanding the ventilator to pause—the ventilator will be stuck at the pause state. This is known as *dependency inversion*. The safety-critical component ventilator depends on less critical components, such as the network and operator console. Potential dependency inversion is easily detected by automated analysis of AADL design.

- **Configuration 2.** As illustrated in Figure 7, a better design is to put these two timers inside the ventilator. From an architecture perspective, this design minimizes the safety dependency tree into a single node: the ventilator. Under this design, as long as the ventilator is verifiably safe, the overall system is safe in spite of the faults and failures in the network, the command station, and the X-ray machine. From a safety perspective, we can now safely integrate the ventilator into different networks with different but interoperable consoles and X-ray machines without recertifying the safety of the system. This is because the network, X-ray machine, and console are not part of the safety dependency tree.

From the perspective of the Simplex Architecture in Configuration 2, the ventilator is required to be verifiably safe. Once this is done, it can safely collaborate with non-safety critical devices such as the network and a command station. The command station and network should be industrial grade, not certifiably safe, because certifying the operating system and the network is prohibitively expensive. Furthermore, if they were certified, any change in the operating system and/or network would trigger recertification. As well, any non-safety critical device or network information flows connected with this certified network would trigger recertification. Minimizing the use of certifiably safe components—especially the infrastructure components such as the operating system and/or the network—is critical to the economics of medical device networks.

Under the Simplex Architecture, non-safety critical devices can be added, modified, and replaced without jeopardizing safety invariance—provided that architecture design rules are followed. This is done by ensuring that the safety invariants are satisfied by the set of safety-critical components. In this example, the safety invariants of the ventilator are the limit on the maximal duration of each pause and the limit on the minimal duration of separation between pauses. These invariants are specified by means of a configuration time constant set by medical personnel and enforced by the two timers at runtime. Assuming that medical personnel set the constants correctly and the timers embedded in the ventilator design work, the ventilator is safe for all possible inputs from the command station because the timeouts are not a function of inputs from the commands.

The ventilator pause is instantiated from the command station and the command goes through the network. Thus, we say that the architecture *employs* the network, X-ray, and command station, but the safety does NOT depend on them. The idea of *employ but not depend* is a key principle of the Simplex Architecture, which minimizes the use of safety-critical components while maximizing the safe utilization of non-critical components. When critical components employ but do not depend on less critical components, the system safety dependency tree is defined as *well-formed*. Otherwise, it is defined as a (safety) *dependency inversion*.

Checking to see if a candidate configuration is well-formed is done by first developing a model of the composition in AADL with a behavior specification. The AADL model is then translated into Real-Time Maude (as illustrated in Figure 8) using its rewriting logic semantics. The fault model is a specification of possible incorrect state transitions. Using the Real-Time Maude models of faulty transitions in unverified components and systems, we are able to verify (by model-checking) that the AADL model of the ventilator operation satisfies the two safety invariants on maximum pause time and on minimum time between pauses and is, therefore, verifiably safe for such invariants. Furthermore, the effectiveness of the system (liveness property)—wherein the X-ray will be taken during the pause of the ventilator in the absence of faults—was also verified.

Conclusion

The convergence of sensing, control, communication, and coordination in CPS—such as with modern airplanes, power grids, transportation systems, and medical device networks—poses an enormous challenge because of its complexity. Work in all of the areas mentioned in this article is certainly relevant and useful. However, to address the hard challenges of CPS system design, the focus is on a synergistic combination of specific technologies to support the model-based design of highly reliable CPS systems. These combined technologies include: architectural patterns, fault-tolerant techniques, model-based software engineering, object-based formal specification, and the verification of real-time systems.◆

Software Defense Application

Many defense systems are mixed criticality systems with a high level of complexity. The reduced complexity architecture patterns provide an effective approach to address this challenge.

Acknowledgements

The works described in this article are sponsored in part by the Office of Naval Research, the National Science Foundation, Lockheed Martin, Rockwell Collins, and the SEI. Many have contributed to this work, and I thank all of our collaborators in developing these ideas and case studies, including Artur Boronat, Darren Cofer, Peter Feiler, Steve Miller, Peter Ölveczky, Joe Hendrix, Minyoug Nam, and Xiaokang Qiu.

References

1. "Ariane 5 Flight 501." *Wikipedia* <http://en.wikipedia.org/wiki/Ariane_5_Flight_501>.
2. Tribble, Alan C., and Steven P. Miller *Software Safety Analysis of a Flight Guidance System* <http://shemesh.larc.nasa.gov/fm/papers/Tribble-SW-Safety-FGS-DASC.pdf>.
3. Sha, Lui. "Using Simplicity to Control Complexity." *IEEE Software*. July/Aug. 2001 <https://agora.cs.illinois.edu/download/attachments/10581/IEEESoftware.pdf>.
4. Tyrrell, A.M. *Recovery Blocks and Algorithm-Based Fault Tolerance*. Proc. of the 22nd EU-ROMICRO Conference. Prague, Czech Republic: 2-5 Sept. 1996.
5. Bak, Stanley, et al. *The System-Level Simplex Architecture for Improved Real-Time Embedded System Safety*. Proc. of the 15th IEEE Real-Time and Embedded Technology and Applications Symposium. San Francisco: 13-16 Apr. 2009.
6. Boyd, S., et al. "Linear Matrix Inequality in Systems and Control Theory." *Studies in Applied Mathematics*. 1994.
7. Yeh, Y.C. *Dependability of the 777 Primary Flight Control System*. Proc. of the Dependable Computing for Critical

Applications Conference. Los Alamitos, CA: 1995.

8. SEI. "Model-Based Engineering with SAE AADL." 2009 <www.sei.cmu.edu/products/courses/p52.html>.
9. Ölveczky, Peter C., and Jose Meseguer. "Semantics and Pragmatics of Real-Time Maude." *Higher-Order and Symbolic Computation* 20(1-2): 161-196 (June 2007).
10. Lofsky, Ann S. "Turn Your Alarms On!" *ASPF Newsletter* 19.4:43. Winter 2004-2005 <www.apsf.org/assets/documents/winter2004.pdf>.

Notes

1. The Lyapunov function is a sufficient but not necessary condition to improve the stability of an equilibrium in an autonomous system. For details, see <http://mathworld.wolfram.com/LyapunovFunction.html>.
2. The logical complexity of a software system can be measured by the number of states that we need to check. A program could have high logical complexity initially. However, if it has been formally verified and can be used as is, then its residual logical complexity is zero.

About the Author

Lui Raymond Sha, Ph.D., is the Donald B. Gillies Chair and professor of computer science at the University of Illinois at Urbana-Champaign. He is active in dependable real-time computing systems research. Sha served on the National Academy of Science's committee on certifiably dependable software, served on the Office of Secretary of Defense's avionics advisory team, and is fellow for the Association for Computing Machinery and the IEEE. He has been a member of the National Science Foundation-sponsored CPS research initiative's planning group since its inception. He has also served on the technical staff of the SEI for 13 years.

University of Illinois
201 North Goodwin AVE
Urbana, IL 61801
Phone: (217) 244-1887
Fax: (217) 244-8363
E-mail: lrs@illinois.edu

Software Survivability: Where Safety and Security Converge

Karen Mercedes Goertzel
Booz Allen Hamilton

As safety-critical software moves from closed environments to open and commodity technologies, security threats will inevitably increase. Organizations dependent on mission-critical systems and networks are recognizing that the traditional "protect-detect-react" (PDR) strategy for countering intrusions and attacks is ineffective. A new information assurance and cybersecurity strategy is needed that augments PDR with the ability of systems and networks to "fight through" attacks. This article examines techniques that both security- and safety-critical software developers can leverage to increase their software's survivability.

Software security and software safety share the need to assure that software will remain dependable under *extraordinary conditions*. Extraordinary conditions—those which software was not intended to gracefully tolerate—will either cause it to behave unpredictably or fail outright. What distinguishes software *safety* from software *security* is what constitutes an extraordinary condition for that software, and what is at stake if it fails as a result.

Extraordinary conditions that threaten software safety are termed *hazards*, reflecting the perception that such conditions are accidental. By contrast, extraordinary conditions that threaten software security are termed *attacks* or *exploits*, indicating their intentionality. The objective of most attacks on software is to sabotage or subvert the software's operation by exploiting one or more weaknesses in the software's execution environment (e.g., failure of the application firewall that blocks malicious input from entering the system), design (e.g., accepting input from unrecognized entities), implementation (e.g., accepting input in a fixed-length buffer without first validating that input's length), operation (e.g., a failure of user interface software, thereby exposing the system's command line), or development process (e.g., poor configuration control, peer review, and testing practices that allow a disgruntled programmer to surreptitiously embed malicious logic).

Intentional Threats to Safety-Critical Systems

Software failures that result from safety hazards can have dire, even fatal, consequences due to the extremely strong linkage between the software and the physical system that it is supposed to control. Whether the software constitutes the single small, closely contained embedded program that controls an automobile's anti-lock braking system, or several dozen modules dispersed throughout a distributed supervisory control and data acquisition (SCADA) system controlling an entire region's wastewater treatment, the functions performed by the physical components are what determine whether the system (including its software) is safety-critical. If a system failure results in damage to the physical environment in which people live, physical maiming, damage to health, or death of one or more humans, the system is safety-critical. The failure of software in such a system can have catastrophic results.

Safety hazards tend to be straightforward and accidental. By contrast, security threats are intentional: the result of human creativity and perspicacity absent from safety hazards (although a hazard may introduce a vulnerability that an attacker can intentionally exploit). Because they are guided by human intelligence, security threats are usually less predictable, more complex, more numerous, and more persistent than safety hazards. The same system may be repeatedly targeted by a variety of simultaneous and sequential attacks, some aimed at the interface level, others at the application components, and still others at the execution environment level—all orchestrated to accumulate and intensify until they collectively produce the critical failure(s), enabling the attacker to achieve his objective.

Google "Ariane 5 Flight 501," "Therac-25 accidents," or "Toyota Prius software bug" to read about some dramatic instances of safety-critical systems that failed as a result of design flaws or implementation errors in their software. These were unintentional flaws and errors, caused by developer inadvertence, negligence, or misapprehension, but their impact was dramatic. How much more disastrous might they have been had their cause been intentional exploitation or implanted malicious logic?

Now Google "trans-Siberian gas pipeline" + "software bug." What you'll get are reports of the 1982 technology coup. The CIA, having learned that Soviet spies planned to secretly acquire a gas pipeline controller developed in Canada, planted a Trojan horse (logic bomb) in the controller's software. Once installed on the trans-Siberian pipeline, the controller ran a test of the pipeline's pressure gauges during which the logic bomb reset those gauges to double gas pressure in the pipeline. The resulting explosion was, up to that time, the largest non-nuclear explosion ever photographed from space [1].

In the 25-plus years since that incident, attacks on safety-critical systems involving the embedding of malicious code or direct penetrations have proliferated, several of which have been perpetrated by the systems' own disgruntled developers or administrators. Such attacks are proliferating due in part to opportunity: More safety-critical systems are built from or hosted on commodity software, the vulnerabilities of which are widely publicized and well understood by attackers, then exposed on semi- or fully open networks (including the Internet). The increasing software intensiveness of safety-critical systems means more of their critical functions are performed by software than by hardware, and that software is necessarily larger and more complex, making its vulnerabilities harder to predict and detect.

As with safety hazards, the impact of software failures resulting from attacks and exploits depends on the nature of the targeted system. A threat to a safety-critical system can have the same dire consequences as a hazard. Even in non-safety-critical systems, the consequences of failure can be catastrophic: Insider sabotage of an intelligence database application may enable an attacker to steal the names of undercover operatives in an adversarial country and sell it to that country's

counterintelligence service, which then has them captured and executed. The subversion of software in a military logistics system that calculates the number of biochemical suits may result in a shortage of protection for forward-deployed forces during a chemical weapons attack.

Embedded, Not Isolated

Many safety-critical systems are embedded. Until recently, that meant they were small, relatively simple, and isolated from direct interaction with humans (they even lacked means for such interaction). Today's embedded systems are different. They both benefit and become vulnerable from the increased power of the processors on which they are hosted. These are processors that enable the use of commodity operating systems, such as Microsoft CE, which share security problems with non-embedded operating systems sharing the same kernel code[1].

The less proprietary and more connected embedded systems become, the less specialized expertise attackers need to target them. Systems from temperature controls to medical devices to on-board automobile computers and sensors are now accessible via wireless Radio Frequency Identification (RFID), cellular, and satellite links that use standard communications protocols. Implanted medical devices are increasingly accessible via RFID [2]. A DoD telemedicine application enables surgeons in U.S. military hospitals to issue commands, via a satellite uplink, to a software-controlled robot in Iraq, thereby performing laser surgery on wounded soldiers in theater [3, 4].

But where there is a wireless network, one can almost guarantee there will be an attacker attempting to locate, intercept, and tamper with the signals transmitted between the systems at either end of the wireless link. Consider telematic systems such as GM's OnStar, Ford's remote emergency satellite cellular unit and vehicle emergency messaging system, Volvo's On Call, BMW's Assist, and Mercedes-Benz's Tele Aid and COMAND. They all use cellular or satellite connections to allow their call center representatives to perform remote diagnostics on the onboard computers of subscribers' vehicles. Privacy concerns about certain data collected by these telematic services are well documented, but a recent addition to OnStar is even more worrying. Owners of 1.7 million OnStar-equipped 2009 GM vehicles can allow their engines to be "remotely switched off through the OnStar mobile communications system" [5] at the behest of the police. The goal is to stop stolen GM vehicles in their tracks during high-speed police car chases, thereby reducing the number of fatal accidents associated with such chases. The implications of OnStar's transition from a passive monitoring and diagnostics system to an active controller of a safety-critical embedded system (the engine) have been noted:

> [Some] automotive communication networks have access to crucial components of the vehicle, like brakes, airbags, and the engine control. Cars that are equipped with driving aid systems allow deep interventions in the driving behavior of the vehicle .... Malicious attackers are not to be underestimated. [6]

The next logical step—remote updates via telematic links to embedded software and firmware—would create an ideal conduit for insertion of malicious logic into embedded computers or causing denial-of-service by injecting "garbage bits" into telematic data streams [7].

Security of Safety-Critical Infrastructure

Along with embedded systems, another type of safety-critical system never originally intended to support publicly discoverable/accessible wireless network connections is the industrial control system. Both the SCADA and distributed control systems (DCSs), along with air traffic control systems, are safety-critical hybrids of information systems, command and control systems, and physical process control systems. They support the same open networking protocols, remote accessibility, and even Internet connectivity typically found in information and command and control systems. Like those systems, safety-critical control systems are being built from commodity and open components and hosted on mobile devices running commodity and open operating systems.

A sobering example of where such advances can lead occurred in the Maroochy wastewater treatment facility in Queensland, Australia [8, 9]. The DCS that controlled the facility included remote administration software that ran on Microsoft Windows and provided remote wireless network access to the facility's physical control functions (including opening and shutting valves). Vitek Boden, a former contractor who helped install the system, later submitted a job application that was rejected. The vengeful engineer applied his expert knowledge: Over the next four months, on more than 40 separate occasions, he parked his car near the water treatment plant and, with a laptop that had a wireless radio transmitter, used a stolen copy of the DCS's remote administration software to identify himself to the DCS as "Pumping Station 4," then issued commands that suppressed the DCS's alarms and changed its settings to place excessive back-pressure on the valves.

By the time the plant's operators finally figured out that the series of inexplicable failures in the plant were caused by sabotage of its DCS and notified police, Boden was in the midst of his 46th incursion into the system. In the end, he managed to release between 264,000 and 1.18 million gallons of raw sewage (including human waste): The Maroochy River tributaries turned black, marine life was poisoned, and the air reeked.

Not only does the Maroochy incident vividly illustrate the danger of the insider threat, it shows how vulnerable remote-controlled safety-critical systems can be[2]. As the *Washington Post* observed:

> … like thousands of utilities around the world, Maroochy Shire allowed technicians operating remotely to manipulate its digital controls. Boden learned how to use those controls as an insider, but the software he used conforms to international standards, and the manuals are available on the Web. Nearly identical systems run oil and gas utilities and many manufacturing plants. [7]

Secure Development of Safety-Critical Software

Software engineering for safety-critical systems is impressively scientific and disciplined. It is driven by heightened quality and fault-tolerance imperatives and has careful, thorough hazard analyses, fault-tolerant designs, and rigorous testing. As well, *safe* subsets of programming languages are used and formal specification, modeling, and verification is utilized. As a result, most safety-critical systems can tolerate and continue operating dependably in the presence of the unintentional faults and failures associated with safety hazards.

But safety-critical software must be equally intolerant of failures caused by intentional threats and keep operating dependably even under attack. This means eliminating weaknesses, bugs, flaws, errors, etc., that don't necessarily lead to failures, but which can be exploited by attackers.

Security for safety-critical systems—and indeed for all software-based systems—must be achieved at the functional, data, and environmental levels. At the functional level, the software must be able to withstand threats to its own integrity and availability; these include threats of denial-of-service, intentional corruption or tampering with the software's executables and/or control files, and embedding/insertion of malicious logic. At the data level, inputs received and outputs produced by the software may be tampered with or intentionally corrupted. If the system stores, manipulates, or transmits information, that information is also subject to the same threats, plus the threat of inappropriate disclosure. The software's execution environment is subject to threats to its availability and integrity, along with a further threat of hijacking or *theft* of computing resources (memory, disk space, computing power) by illicit processes that make those resources unavailable to valid processes.

Software security focuses on specifying software's internal workings to remain dependable in the presence of potentially hostile external interactions. Moreover, the software must not contain design weaknesses or implementation errors that, if intentionally or accidentally escalated, could lead to a failure (i.e., any incorrect or unpredictable behavior) that could leave the software exposed and vulnerable to direct attack. Such failures may result from a hostile input to the software itself, or from a fault triggered by an attack on the software's environment.

Secure = Survivable

To date, the established paradigm for system security has combined proactive PDR strategies (which includes recovery). Protection is often achieved through defense-in-depth layering of security mechanisms, controls, and procedures at the functional, data, and environmental levels of the system. Detection of threats (or more accurately, their manifestation as intrusions and attacks) is achieved by a combination of intrusion detection, event logging, and usage auditing and monitoring. Reaction to intrusions and attacks focuses on minimizing the extent, intensity, and duration of the incidents' impact and the likelihood of their recurrence. Reaction often comes at the expense of dependability because it requires rejecting certain types of inputs (some of which may in fact be valid), terminating some user sessions, shutting down some or all functions, or disconnecting the system from the network (to disengage it from the suspected attack source).

In the DoD, practitioners of information assurance, computer network defense, and cybersecurity have begun to admit that this PDR paradigm is essentially flawed. Attackers have become too skilled, too expert, too flexible, and too ingenious for countermeasures that rely on the ability to recognize the threat to keep up. Information and cyber warfare fought on current terms is not just being lost, it is unwinnable.

The DoD and numerous other organizations now recognize the need for a *paradigm shift* to enable their systems to survive high-intensity intrusions and attacks. Survivability (also referred to as resilience), which has always been required for safety-critical systems, must become the norm for mission- and security-critical software as well.

Designing for survivability means including redundancy and rapid recovery features at the system level (e.g., automated backups and hot-sparing with automatic swap-over of high-consequence components and modularized designs that enable those components to be decoupled and replicated on *hot spare* platforms). It means implementing significantly more error and exception-handling functionality than program functionality: error and exception handling that is purpose-built, not generic, to minimize the possibility of faults escalating into failures. If possible, rather than failing, the software should be able to keep running at a degraded level of operation (i.e., reduced performance, termination of lower-priority functions, rejection of new inputs/connections). If it must fail, its exception handler should prevent the failure from placing the software into an insecure state, dumping core memory, or exposing the content of its caches, temporary files, and other transient data stores. For safety-critical software—in which there is no threshold of tolerance for the delays typically involved in post-failure recovery and restoration—survivability measures must prevent failures, full stop. This is true whether the failure was accidental or intentionally induced.

Engineering for Survivability

Survivability has become the subject of research, as demonstrated by the Survivable Systems Engineering program at Carnegie Mellon University's Computer Emergency Response Team Coordination Center (see <www.cert.org/sse>), the Willow Survivability Architecture developed by University of Virginia's Dependability Research Group (see <http://dependability.cs.virg

Coming Events

November 1-5
SIGAda 2009
Tampa Bay, FL
www.sigada.org/conf/sigada2009

November 2-4
The 13th IASTED International Conference on Software Engineering and Applications
Cambridge, MA
www.iasted.org/conferences/home-669.html

November 2-6
Software Assurance Forum
Washington, D.C.
https://buildsecurityin.us-cert.gov/daisy/bsi/events/1102-BSI.html

November 9-11
International Conference on Software Quality
Chicago, IL
www.espresso-labs.com/icsq2009

November 9-13
16th ACM Conference on Computer and Communications Security
Chicago, IL
www.sigsac.org/ccs/CCS2009

November 9-13
Agile Development Practices Conference 2009
Orlando, FL
www.sqe.com/agiledevpractices

December 13-16
Winter Simulation Conference 2009
Austin, TX
http://wintersim.org

April 26-29, 2010
22nd Annual Systems and Software Technology Conference
www.sstc-online.org

Coming Events: **Please submit coming events that are of interest to our readers at least 90 days before registration. E-mail announcements to: <marek.steed.ctr@hill.af.mil>.**

inia.edu/research/willow>), and Mithril, developed by the National Center for Supercomputing Applications (see <http://security.ncsa.uiuc.edu/research/mithril/>). These efforts combine techniques for engineering high-confidence and safety-critical software with software security assurance principles and practices, and generate methodologies and tools for producing software that can remain survivable in the face of intentional threats as well as accidental hazards.

Emerging survivability techniques, as well as more established high-confidence and safety engineering techniques, methods, and tools can benefit developers of software-based systems with strong security imperatives, including systems that are larger, more complex, more interactive, and more extensively networked than most safety-critical systems, high-confidence embedded systems, cryptosystems, and so forth.

Software Practices that Aid Security and Safety

Just as developers of security-critical software can benefit from safety engineering practices, safety-critical software development needs to undergo its own *paradigm shift* to account for *intentional hazards*. Researchers in both the safety and security communities are adopting and adapting software assurance principles, practices, and tools from the other community to aid them in producing software that is safe and secure. Among these efforts, three significant trends stand out:

Simplification of Formal Methods

Tool-supported modeling and proofs of security properties in large, complex software systems is made possible by *semi-formal* methods, such as: 1) Praxis High Integrity Systems' Correctness-by-Construction, which is a structured development methodology into which formalisms have been selectively incorporated; and 2) tools that automate formal activities so they can be performed by non-experts. Examples include Correctness-by-Construction's supporting tools, Munich University of Technology's Autofocus and Quest, Jean-Raymond Abrial's B-Method, and (to some extent) the Object Management Group's Model-Driven Architecture.

Hybrid Assurance Cases

Hybrid assurance case standards, templates, and processes (including both safety and security arguments and evidence) are emerging. Examples include the SafSec standard developed by Praxis High Integrity Systems for the United Kingdom's Ministry of Defense and ISO/IEC 15026, System and Software Engineering–System and Software Assurance. Also noteworthy are the safety and security extensions defined for the integrated CMM® and CMMI® by the Federal Aviation Administration and the DoD [10]. Their objective: extend processes defined by and validated under those CMMs to include safety and security engineering practices.

Biological Models and Computer Immunology

Biological models and computer immunology are being applied to software resilience/survivability to achieve diversity and evolution/adaptation through: 1) creation of different instantiations of software programs whereby the computational results are identical but the architectures, source code, and/or binary images diverge and thus are not all equally susceptible to the same threats and 2) use of pseudo-genetic algorithms to gradually evolve executables over time within the acceptable bounds of the software's functional specifications, thus enabling them to continue operating correctly despite the transformation. Specific techniques include: dynamic software composition, N-version programming, and code filtering. Other biological metaphors have resulted in *software rejuvenation*, phylogenetic trees for predicting vulnerabilities, and techniques for nature-based modeling of software systems.

Conclusion

Survivability as an adjunct to the PDR model of information assurance and cybersecurity is expected to be embraced more fully by DoD and by other communities that operate mission-critical, safety-critical, and life-critical systems. To the extent that software safety engineering minimizes or eliminates implementation errors and environment faults, it contributes to the security of that software. It cannot, however, achieve security on its own because it does not consider design weaknesses that can be exploited as vulnerabilities or exploitable errors and faults that are not expected to result in failures. Adding software security principles and practices to software safety engineering can bridge the gap between producing software that remains dependable in the presence of unintentional hazards and software that remains dependable in the

® CMM and CMMI are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

presence of both hazards and intentional threats.◆

Resources

Information on secure software development practices, methodologies, and tools is proliferating in print and on the Internet. Three resources of particular value (both for their own content and for the extensive lists of references they contain) are:

1. The DHS' "Build Security In" Web site at <https://buildsecurityin.us-cert.gov/>.
2. The Information Assurance Technology Analysis Center and the Data and Analysis Center for Software's (DACS) "Software Security Assurance: A State-of-the-Art Report," is available online at <http://iac.dtic.mil/iatac/download/security. pdf>.
3. The DHS (sponsor) and DACS (publisher) document, "Enhancing the Development Life Cycle to Produce Secure Software," is available online at <https://www.thedacs.com/techs/enhanced_life_cycles>.

Software Defense Application

DoD developers of weapons systems, avionic systems, surgical robots, and other safety-critical systems should find this article helpful in clarifying the security threats such systems face as they are increasingly networked and, thus, exposed not only to safety hazards but to intentional attacks and exploits by nation states and cyberterrorists. DoD developers of classified information systems, security controls, cryptosystems, and other security-critical software-based systems should benefit from the discussion of safety engineering techniques that can increase the survivability of those systems. Finally, survivability (as a concept) provides a shared point reference from which developers of safety-critical and security-critical defense systems can establish an ongoing dialogue to share the contributions each of their communities can make, in terms of engineering methods, techniques, and tools, to advancing the state-of-the-art of software survivability engineering.

References

1. Quinn-Judge, Paul. "Cracks in the System." *Time*. 9 Jan. 2002 <www.time.com/time/magazine/article/0,9171,260664,00.html>.
2. Becker, T.J. "Improving Medical Devices: Georgia Tech Research Center Expands Testing Capabilities to Help Reduce Potential Interference." *Georgia Tech Research News*. 25 July 2006 <www.gtresearchnews.gatech.edu/newsrelease/eas-center.htm>.
3. Ackerman, Robert K. "Telemedicine Reaches Far and Wide." *SIGNAL*. Mar. 2005 <www.afcea.org/signal/articles/templates/SIGNAL_Article_Template.asp?articleid=693&zoneid=128>.
4. Murakami, H., et. al. Tohoku University Sendai. "Telemedicine Using Mobile Satellite Communication." *IEEE Transaction on Biomedical Engineering* 41:5 (1994): 488-497.
5. Woodyard, Chris. "Device Can Remotely Halt Auto Chases." *USA Today*. 9 Oct. 2007 <www.usatoday.com/money/autos/2007-10-09-onstar-stop-pursuits_N.htm>.
6. Farwell, Jennifer. "Hijacked, Corrupted and Crashed: Does the New Generation of Computerized Cars Pose a Security Threat?" *PC Today* 3.10. Oct. 2005 <www.pctoday.com/editorial/article.asp?article=articles%2F2005%2Ft0310%2F05t10%2F05t10.asp>.
7. Gellman, Barton. "Cyber-Attacks By Al Qaeda Feared." *The Washington Post*. 27 June 2002 <www.washingtonpost.com/wp-dyn/content/article/2006/06/12/AR2006061200711. html>.
8. Slay, Jill, and Michael Miller. "Lessons Learned from the Maroochy Water Breach." *Critical Infrastructure Protection* 253 (2007): 73-82.
9. Graham-Rowe, Duncan. "Power Play." *The New Scientist* 2447: 15 May 2004.
10. Goertzel, Karen Mercedes. "Computer Immunology." *DoD IA Newsletter* 7:(2). Fall 2004 <http://iac.dtic.mil/iatac/download/Vol7_No2.pdf>.

Notes

1. Granted, additional processor power also makes computing resources available for security countermeasures, such as input validation and sophisticated exception handling.
2. In the 1990s, the Nuclear Regulatory Commission prohibited remote control of industrial control systems at nuclear power plants. See Scott Berinato's article for CIO entitled: "Cybersecurity – The Truth About Cyberterrorism" <www.cio.com/article/30933/CYBERSECURITY_The_Truth_About_Cyberterrorism>. Serious efforts to improve SCADA and DCS security increased after Sept. 11, notably in the Departments of Homeland Security and Energy and their counterparts in other countries. Most of these efforts have focused on system-level and cybersecurity threats; few are attempting to address software vulnerabilities or malicious code embedded during software's development.

About the Author

Karen Mercedes Goertzel, Certified Information Systems Security Professional, leads Booz Allen Hamilton's Security Research Service. She is a subject matter expert in software assurance, cybersecurity, and information assurance. She was lead author of "Software Security Assurance: A State-of-the-Art Report" and "The Insider Threat to Information Systems," published by the Defense Technical Information Center. She has advised the Naval Sea Systems Command and the DHS Software Assurance Program; for the latter, she was lead author of "Enhancing the Development Life Cycle to Produce Secure Software." Goertzel was also chief technologist of the Defense Information Systems Agency's Application Security Program, for which she co-authored a number of secure application developer guides. She also tracks emerging technologies, trends, and research in information assurance, cybersecurity, software assurance, information quality, and privacy. Before joining Booz Allen Hamilton, Goertzel was a requirements analyst and architect of high-assurance trusted systems and cross-domain solutions for defense and civilian establishments.

Booz Allen Hamilton
8283 Greensboro DR
H5061
McLean, VA 22102
Phone: (703) 902-6981
Fax: (703) 902-3537
E-mail: goertzel_karen@bah.com

Investing in Software Resiliency

Dr. C. Warren Axelrod
U.S. Cyber Consequences Unit

Software is inherently error-prone and such errors can lead to failure of those systems of which the software is part. On the other hand, with software being only one of many components of a system, there are many choices in regard to attaining a particular level of system resiliency, not all of which are software-related. It is important to consider software resiliency in relation to the resiliency of the entire system, including the human and operational components. The goal of this article is to help those who develop, implement, and operate computer networks and systems in determining the factors to include when investing in software resiliency.

What is software resiliency? How can it be achieved? What does it cost? What are its benefits and how can they be measured? Is an investment in software resiliency worthwhile? These questions might appear simple, but none of them have an easy answer.

The very concept of software resiliency is frequently ambiguous. For example, are we talking about the inherent or intrinsic resiliency of the software itself, the resiliency that the software imparts upon other components of the system, or both? There is a significant difference in requirements based upon how one defines software resiliency. There is even a question as to what software to include (e.g., end-user applications, system software, embedded firmware) and what to exclude. In this article, I will define software resiliency, examine how it fits into the overall resiliency agenda, and show how one might determine an appropriate level of investment in it.

Background

According to [1], 57 percent of about 1,200 responding organizations experience one or more application failures per month, resulting in user inconvenience or business disruptions. Interestingly, the survey shows that larger organizations tend to have more failures on average, which is thought to be due to the greater complexity in larger environments.

Application failures resulted in decreasing order from software component failure, failure or reduced performance of networks, and physical component failures through power outages and brownouts. Major reasons for application failures include inadequate configuration or change management, system sizing or capacity planning problems, IT staff errors, patch management issues, and security breaches. Another finding was that (for the most part) expenditures on resiliency are not made early enough in the application development life cycle. By delaying consideration of resiliency until late in the cycle, the costs are much higher and there are often insufficient funds to do the job.

For the purposes of this article, software includes any programs that are developed through a regular development life cycle, such as the software development life cycle. This applies whether the end product is a set of *soft* computer program code, firmware (which is program code etched into hardware), or even programmed hardware[1].

> *"The very concept of software resiliency is frequently ambiguous ... are we talking about the inherent or intrinsic resiliency of the software itself, the resiliency that the software imparts upon other components, or both?"*

Resiliency

In [2], the authors define a resilient system as one that can *take a hit* to a critical component and recover and come back for more in a known, bounded, and generally acceptable period of time.

This definition raises as many questions as it answers. Taking a hit can result from accidental activities or intentional attacks: It is when unauthorized damaging activities cause the system to fail noticeably and invoke some form of recovery-and-repair process. A hit can be as simple as a PC freezing and having to reboot it to a complex event that may take a long time and many resources to examine forensically and respond appropriately.

From a more general perspective (particularly when it comes to economic evaluations), one is interested in both how resistant the system is to events that threaten to cause it to fail, and how quickly the system can be brought back to an acceptable level of functioning.

In order to evaluate software resiliency sufficiently, one must always include the environment in which the software operates. The resilience of systems containing a particular piece of software will vary considerably within a particular context.

User View of Availability

There are a number of situations in which a system can be considered not to be available. *Unavailable time* is defined in [3] as the time during which any of the following takes place:

- The system fails to operate.
- The system fails to operate in accordance with formal specifications.
- The system operates inconsistently or erratically.
- The system is in the process of being maintained or repaired.
- A hardware or software component of the system is inoperative, which renders the entire system useless for user purposes.
- The system is not operated because there is a potential danger from operation of the system to employers or employees.
- There is a defect in software supplied by the manufacturer.

This is a more realistic view of availability since there are frequently arguments between the user population and the support technologists as to the real status and usability of a system. Therefore, it pays to be as specific as possible.

Software Resiliency

We now consider resiliency as it specifically pertains to software, as I have defined. First we look at those factors which

reduce resiliency. We then look at specific design and development attributes that affect software resiliency.

Factors Working Against Software Resiliency

The introduction of [4] provides a number of factors and trends that impact software trustworthiness. Many of these factors also affect software resiliency. What follows are some of the broader issues from [4] as well as some additional factors to consider:

Complexity

The size and complexity of software systems is increasing, thus the ways in which a system can fail also increases. It is fair to assume that the increase in failure possibilities does not bear a linear or additive relationship to system complexity. For example, combining two or more systems leads to a greater level of complexity than the combination of the complexities of the individual systems. Thus, if System A has a complexity of 5 and System B a complexity of 7, the combination of Systems A and B will be significantly greater than 12—perhaps in the 20 range.

This complexity attribute makes it increasingly difficult to incorporate resiliency routines that will respond effectively to failures in the individual systems and in their combined system. The cost of achieving an equivalent level of resiliency due to the complexity factor should be added to that of the individual systems.

Interdependency and Interconnectivity

Interdependency or interconnectivity via ever-larger networks adds to complexity in that as systems become increasingly interconnected and interdependent, achieving resiliency becomes a greater task. Another aspect of interconnectivity is the growth in infrastructures that contain systems belonging to different organizations. Thus, the resiliency of an entity's systems is increasingly dependent on the resiliency of systems over which the entity has no control. This means that a failure of another party's systems can have a ripple effect on your systems.

In order to protect against this situation, an entity must develop routines that preserve the integrity and operational continuity of its systems even if the systems of business partners, service providers, and customers were to fail.

Net-Centricity

This is somewhat similar to the interdependency case, except that it focuses on systems that include the Internet or other public/private network as part of its design. For example, service-oriented architecture and software as a service fall into this category, as do a whole range of so-called Web 2.0 applications and services. Again, the issue is whether the systems and networks not under the direct control of the customer organization can be trusted, and what evidence is available to verify such trust. In such situations, there is a need to ensure that software components can be trusted to interact securely without supervision [4]. It should also be noted that security assurance has to cover resiliency and integrity as well as confidentiality.

Globalization

With the growth in increasingly extended software development supply chains, the concern is that the focus will be more on functionality and low cost rather than resistance to attack and resiliency. The challenge is to spread the knowledge as to how to design and build more secure and resilient systems to the far reaches of the development universe and to enforce standards. It is essential to introduce mechanisms that reward such aspects as security, resiliency, and integrity rather than only functionality and speed to market.

"The time that it takes to recover depends mostly on the degree of preparation made through business continuity and disaster recovery plans."

Open Source Software

To some extent, open source products are the software equivalent of the interconnectivity and net-centricity aspects of networking in that there is not necessarily a specific group to go to in order to ensure trustworthiness and resiliency and resolve any failures. It is true that there are *communities* that are responsible for the evolving and fine-tuning of the software (and some of the open source software that is supported by commercial enterprises). However, as shown in a recent study by application security firm Fortify, these groups may not be responsive [5]. Another challenge raised in [4] is the funding of evaluations of such software. There has been some movement in regard to the latter, such as the Software Assurance Initiative being conducted for the banking and finance sector by the Financial Services Technology Consortium in collaboration with the Financial Services Roundtable[2].

Hybridization

Hybridization relates to the increasing trend of combining into single systems software of different origins, and subject to different development methodologies, time and cost constraints, and so on. Thus commercial and government off-the-shelf software, custom and proprietary software, and open-source software may be combined in various ways in the ultimate realization of a particular system. One could argue that such a system is, as a result, only as resilient or secure as its weakest component. This aspect of context is key when attempting to evaluate the combined resiliency or security of a complex system.

Rapid Change

The common belief that change is the only certainty is particularly true in the software arena, where new versions of existing software and frequent releases of new software make for a very dynamic and highly complex environment. Such rapid change creates innumerable problems with software security and resiliency. There is often not the time to test one version of a software product before a new one appears, making the tests on the original software obsolete. A frequently held criticism of Common Criteria testing is that, by the time the results are available, there is a good chance that the tested software has already been replaced.

The danger here is that the new software may contain new vulnerabilities that may not have existed in prior versions. Thus, determining that an obsolete piece of software is sufficiently resilient is not particularly indicative of the state of the newest version and, therefore, is not very useful.

Reuse in Different Contexts

As organizations are being driven by economic and speed-to-market considerations, there is a tendency to increase the use of off-the-shelf and open-source software. While such systems may have been designed to operate in a specific environment, they are being increasingly used in situations for which they were not designed. As a result, they may not meet

the security and resiliency requirements of the new environments.

While one might be cynical in interpreting the standard software use agreement (that protects the software vendor against virtually any liability if the software doesn't do as intended), there is a valid argument about it not working when used inappropriately. This is particularly true of lightweight software applied to critical large operations uses.

Specific Design and Development Issues

There are many situations in which systems fail because they do not even incorporate necessary resiliency routines, or the ones that are inserted do not perform as needed or have not been tested thoroughly enough.

Poor Design

Regarding the absence of resiliency routines, I recall a development manager expressing amazement at the general lack of understanding—both by the presenters of a new product and the audience—of the need to design-in the ability to restart a program from a prior status. The developers were relatively new to the profession.

Inadequate Testing

In another situation, I was about to implement a leading-edge digital telephone turret on a newly built trading floor. The only other installation to date was experiencing intermittent crashes. After weeks of research, the turret vendor determined that the reason for failure was an untested error routine. Apparently, in the pristine and carefully engineered test version at the vendor's testing laboratory, the system did not invoke this particular routine because of the high quality of the installation. Out in the *real world*, the less well-engineered cable runs began generating errors that forced the software into the untested error routines leading to the consequential crashes.

Inappropriate Use

Another resiliency issue arises when the software is used incorrectly or is inappropriate for a particular purpose. Software for the PC is generally not as reliable and does not have the same fail-safe design as software intended to be used in a demanding production environment—yet such unreliable software regularly becomes incorporated into critical production or financial systems. These systems are not held to the same standards for testing and documentation as are major production systems and, as a result, can be the *Achilles heel* of the overall system.

Ineffective Change Management

In order to maintain a high level of application security, integrity, and resiliency, it is necessary to carefully control the software change process. There are many instances where programming errors can result in major failures.

Table 1: *Protection, Costs, and Benefits for Different Types of Events*

Type of Event	Protection	Costs	Benefits
Component failure	• Hardening • Fault tolerance • Redundant components	• Additional components • Software overhead • Hardware overhead • Increased complexity • Maintenance and support	• Increased availability • Reduced downtime
System failure	• Fail-over • Redundant systems		

Site down	• Off-site backup
	  • Hot
	  • Warm
	  • Cold
	  • White-wall
		• Facilities
		• Systems
		• Networks
		• Staffing
		• Utilities
			Ability to restore operation when primary facility inoperable with minimal downtime

Regional disaster	• Out-of-region backup
	  • Hot
	  • Warm
	  • Cold
	  • White-wall		

National or global catastrophe	• Out-of-country facility • Catastrophe contingency planning and backup	As for regional disaster	Ability to recover from a disastrous event affecting large regions of the country or the world.
All off-site backup	• Backup in the cloud	As for some on-site and all off-site	• Ability to purchase amount of resources for backup as needed • Largely independent of location

As an example, on January 15, 1990, AT&T's long-distance network failed and was down for nine hours. The failure occurred when a system-wide software upgrade was installed on 4ESS digital circuit switches. It was reported that the failure began when a switch in New York City suffered a minor hardware glitch, which caused it to go offline [6].

While scheduled changes can clearly cause problems, unscheduled or emergency changes represent an even greater danger to the integrity and continuing operation of software.

Fault Tolerance and Failure Recovery

Anderson points out that "... failure recovery is often the most important aspect of security engineering, yet it is one of the most neglected" [7].

Fault tolerance is the ability of the software to resist damage or destruction from errors. Thus, if there is an error condition, the software has the capability of recognizing the error and correcting it according to some pre-specified set of rules. The tolerance level is only as good as the rules. Therefore, the software, on recognizing an error, will correct it with the most likely correct condition. There is, of course, a possibility that the correction is not appropriate, in which case either the integrity of the system is called into question or a subsequent test will reveal that the attempted correction was inappropriate.

In other cases, if the fault is thought by the system to be a component failure, the fault tolerance results in automatic switching to a backup component or software routine. The system continues processing in backup mode while the faulty component is being fixed. This latter situation is failure recovery within the primary system.

Fail-Over to Other Systems

Fail-over can also be to an on-site or off-site backup system. While fail-over within a system usually assumes operational continuity, fail-over to backup systems can be hot, warm, or cold.

If hot, the backup system is running in parallel with the primary system and automatically detects a failure in the primary system and switches to the backup, which may be on-site or off-site. If off-site, it can be in-region, out-of-region, or *in the cloud*. There are often technology restrictions on the allowable distance between sites for hot backup. One common limitation comes from the technical feasibility of maintaining data current at two or more sites via disk shadowing or similar technologies.

Recovery and Restoration

The ability to resist *attacks*—and to recover quickly to an acceptable level of performance after failure due to successful exploits, unintended damaging actions, or accidents—is crucial for the systems running in most organizations.

The time that it takes to recover depends mostly on the degree of preparation made through business continuity and disaster recovery plans. There are escalating levels of backup and recovery, each costing more but enabling improving recovery from increasingly destructive events. These levels are shown in Table 1. The table also shows the various forms of protection that can be instituted and their respective costs and benefits.

In the commercial world, unavailability costs might include loss of productivity for internal users and business partners, loss of business in the form of failure to attract new customers or retain existing customers, and so forth. In the government sector, lack of availability might result in military compromise or a reduction in safety. While difficult, it is necessary to come up with cost estimates related to unavailability of critical systems. These costs will typically not be easy to estimate. They will also typically not be linear, but more in the form of exponentially increasing costs.

In terms of recovery costs, these are usually minimal when recovery involves a *hot backup system* or *facility* where switching or fail-over to the backup system—whether on-site or at another facility—is virtually instantaneous and there is no loss of data or processing availability. Such a transition is effectively transparent to end-users and business partners. Of course, a hot backup is considerably more expensive to design, implement, and maintain than other forms of backup.

Warm backup is where the backup system or facility is up and running and on standby and can be brought into operation within a short time, typically minutes. The recovery time usually consists of a process for bringing the backup system up and synchronizing it to the point in processing at which the primary system failed. The activation of such a process usually takes from minutes to hours to accomplish and the time when the switchover takes place (i.e., whether the system is in use or idle at the time of failure) can have a significant impact on end-users and business partners.

It is interesting to note that hot backup is not always better than warm backup from operational and availability perspectives. I recall a situation in which two sister organizations had taken different approaches to achieving high availability for critical financial systems. The larger, wealthier organization had both hot backup and warm backup, and a process whereby the warm backup system was activated to hot status if the primary system failed over to the hot backup. The smaller, less affluent organization ran two separate systems in parallel and, in the event of a failure of the primary system, physically switched to the backup system. This resulted in having to reenter the few missed transactions that were lost in the switchover. It turned out that the highly automated larger systems were considerably more expensive and far less reliable than the simpler manually operated systems. This was because, in the highly automated case, an error occurred in common memory resources, which brought down all three systems for an extended period. The lesson learned was that one has to be aware of single points of failure and how they might impact the recovery process.

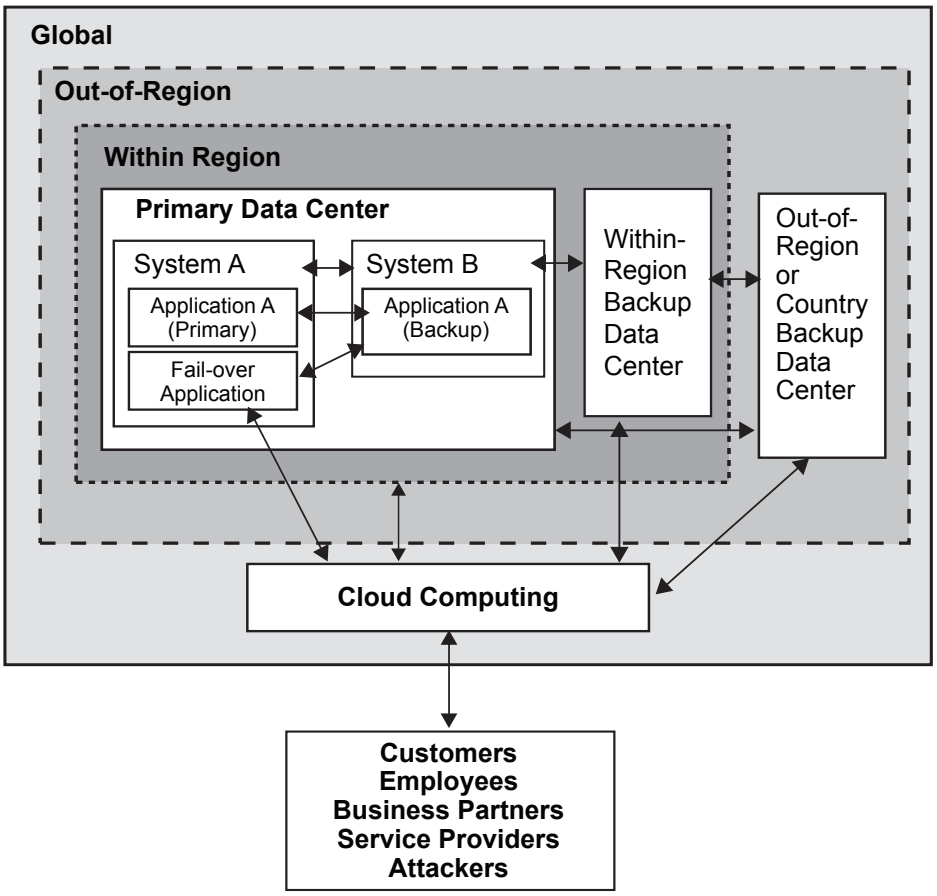


Figure 1: *Various Degrees of Backup at Application, System, and Facility Levels*

With regard to backup sites, there are a number of lower-cost options. One option is to have a *cold site*, which will generally have power, cabling, communications, and some systems installed. Either all necessary equipment will be on a site but not necessarily powered up or an arrangement with a vendor will be in place for the rapid shipment of standard equipment, such as PCs. The timeframe for activating a cold site can range from hours to days of elapsed time, depending on factors such as the time it takes for required staff to travel to the backup site, the time to deliver additional equipment and software, and the time to initiate and synchronize systems. I recall a personal experience where a Florida company declared a disaster due to a major storm soon after the Sept. 11 terrorist attacks, but was delayed several days in effecting its backup plan because the backup facility was in Chicago and there were no planes flying. The need to use land-based transportation to move people and resources (such as physical data media) significantly extended the recovery time.

A *white-wall facility*[3] is an extreme form of a cold site. It is essentially an empty space that the organization has previously obtained for its use in a disaster. It generally offers little more than the bare walls, with a minimum of power, heating, and cooling utilities, and perhaps some minimal telecommunications, installed. Such a site must be built out on demand. This generally means that the organization must order, receive, and install necessary

equipment, software, and other resources at the time of an incident.

Organizations are well advised to at least have previously negotiated arrangements and agreements with vendors and service providers for the rapid delivery and installation of required resources. Experience has shown that vendors and service providers are usually very responsive in the face of an emergency, giving the affected organization priority service in order to minimize downtime. Of course, it is also in the self-interest of vendors to do what they can to enable the customer organization to survive a disaster. While setting up a white-wall facility can take weeks, it is a big step ahead of having no facility at all[4].

If no backup system or facility has been provided, the choice (depending on the nature of the failure) is to fix and recover the primary system or facility, or to build a new system from available resources. Here, a good practice is to take snapshots of the system and data at predetermined intervals and go through a restart process. Depending on the level of recovery and reconstruction necessary, this type of recovery can be an extremely expensive endeavor in and of itself and the consequential financial losses due to lost productivity, damaged reputation, fleeing customers, and the like can be enormous.

Another option that should be mentioned is a one-way or two-way agreement with another party. One can subscribe to commercial disaster recovery services and pay a monthly fee to have the right to use their facilities and additional fees if a disaster is declared. Such facilities can be very effective as they are often staffed and run continuously. One issue to be aware of is that when disaster recovery facilities are shared among a number of customers within a given region, the amount of backup services might not be available to the degree expected if a disaster were to be regional in scope. The level of services provided can range from hot backup to white-wall, with corresponding charges.

Another option is to institute a reciprocal arrangement with another nearby company, often in the same or similar business. However, they can be difficult to implement since there is no guarantee that the reciprocating partner will be able to provide the facilities when needed. I recall a situation in which my company needed to invoke such an arrangement at 6 a.m. one day. However, my staff could not get into the other company's facility since the persons familiar with the arrangement were in transit and the building guards had not been informed about the arrangement and would not let the operators into the building. Ironically, when the other party needed to invoke the arrangement a few months after the unsuccessful attempt by my company to use their facilities, they needed to invoke the mutual backup arrangement. In contrast, however, my company was able to provide the resources on demand. As seen by this example, such arrangements may be very low cost, but they are also unreliable and difficult to enforce.

The use of *cloud computing* is similar to the disaster recovery services model except that cloud computing services might not require a monthly fee if the arrangement is only to pay for cloud services actually used.

Figure 1 (see previous page) illustrates the various backup relationships previously discussed. It also shows that other parties—such as customers, service providers, and business partners—need to be included. In particular, it is highly advisable to test connectivity and operability between backup facilities and third parties. More recently, there have been calls for backup-to-backup testing between organizations and third parties.

Table 2 shows, for various failure scenarios and types of backup, the relative costs of setting up and operating the backup capabilities, how much (on a relative basis) it might cost to recover if an incident occurs, as well as what the combined costs might be.

Please note that these are very rough ordinal assessments that do not allow for essential characteristics of systems (such as their criticality to the business and their technical complexity) nor do they account for the frequency and magnitude of events. The assessments are provided as guidance as to what one might find in a typical business or government situation.

The Economics of Resiliency

It is clear that there is a need to balance

Table 2: *Costs of Backup, Response, and Recovery by Scope of Event and Type of Backup*

Scope of Failure or Event	Type of Backup Put in Place (if any)	Cost of Setting Up Backup	Ongoing Costs of Maintenance and Support	Typical Time to Respond and Recover	Cost of Incident Response and Recovery	Probable Frequency of Event per Period*	Incident Cost per Period (Magnitude x Frequency)
Component	Hot fail-over	High	High	Seconds/Minutes	Low	Moderate	Low
Component	Warm fail-over	Moderate	Moderate	Hours	Moderate	Moderate	Low
Component	No fail-over	Low	Low	Days	Very high	Moderate	Extremely high
System	Hot backup	High	High	Seconds/Minutes	Low	Moderate/High	Moderate
System	Warm backup	Moderate	Moderate	Hours	Moderate	Moderate/High	High
System	No backup	Low	Low	Days	Very high	Moderate/High	Extremely high
Site (Facility)	Hot site	Very high	Very high	Seconds/Minutes	Low	Moderate	Moderate
Site (Facility)	Warm site	High	High	Hours	Moderate	Moderate	High
Site (Facility)	Cold site	Moderate	Moderate	Days	Very high	Moderate	Very high
Site (Facility)	White wall	Low	Low	Weeks/months	Extremely high	Moderate	Extremely high
Regional	Hot site	Extremely high	Extremely high	Seconds/Minutes	Low	Low/Moderate	Low
Regional	Warm site	High	High	Hours	Moderate	Low/Moderate	Moderate
Regional	Cold site	Moderate	Moderate	Days	Very high	Low/Moderate	Very high
Regional	White wall	Low	Low	Weeks/months	Extremely high	Low/Moderate	Extremely high
National/Global	Hot site	Extremely high	Extremely high	Seconds/Minutes	Low	Low	Low
National/Global	Warm site	High	High	Hours	Moderate	Low	Moderate
National/Global	Cold site	Moderate	Moderate	Days	Very high	Low	Very high
National/Global	White wall	Low	Low	Weeks/months	Extremely high	Low	Extremely high

* Note that the frequency of events, other than those outside the control of the organization, can be influenced by those responsible for designing and setting up systems, facilities, and infrastructures. The levels shown are for frequency are based on experience, but may not be applicable to a particular case.

the cost and effectiveness of backup and recovery capabilities against the expectations of damaging and destructive events. The mentioned scenarios and costs relate to recovery from successful attacks or damaging events. These costs may be reduced if the expectation of failure or compromise is lowered through preventative measures, deterrence, or avoidance.

There is a trade-off between protective measures and investments in survivability. The determination of the optimum level of backup is based on the expectations of damaging events, the impact of these events, and the ability to recover quickly and return to acceptable operation.

It should also be noted that the different levels of backup are not independent. Hence, if one has a hot backup system installed in a within-region backup facility, it may not be cost-effective to have an on-site backup system. Conversely, if one installs a highly resilient primary system with various degrees of internal redundancy, it is less likely that a backup system will be required and thus a warm off-site backup system may be adequate.

This suggests that a number of combinations need to be evaluated, depending on the resiliency of the primary systems, the criticality of the application, and the options as to backup systems and facilities. Thus, it is up to the analyst to determine which options and which combinations make the most sense for a particular environment and then to cost out the preferred options.

Summary

The topic of software resiliency is not addressed at a level appropriate to its impact on organizations. This article has examined the factors that affect software resiliency and the contexts in which applications might run, particularly in regard to the wide choice of backup options.

Further work is needed—particularly with respect to running some numbers for a variety of cases and reviewing the results. It may be that the realistic options are much more limited than expected. Also, the growing availability of cloud computing may completely change the results of disaster backup analyses in favor of backup in the cloud. At the same time, cloud computing introduces its own issues in regard to resiliency and recovery.◆

Software Defense Application

As software and its implementation become increasingly complex and dependent on diverse infrastructures, it has become essential for those designing and developing computer applications to be aware of, and allow for, the evermore challenging environments into which software is installed. This article provides those in the DoD responsible for software design and development, infrastructure support, data center operations, disaster recovery planning, and incident response with the necessary guidance, concepts, techniques, and methodologies to provide the overall level of resiliency required for specific systems. As cyber attacks grow in their capabilities and effectiveness, those developing and deploying DoD systems must enhance their understanding of the impact of failures from attacks, inadvertent actions, and natural events on the availability of computer systems and networks. They need to take steps so that systems can rapidly and accurately be recovered from failures and outages, whatever their cause.

References

1. The Register and Freeform Dynamics, Ltd. "Risk and Resilience: The Application Availability Gamble." *The Register*. July 2008 <http://white papers.theregister.co.uk/paper/view/485/c-documents-and-settings-regmar06-002-desktop-pdf-risk-resilience.pdf>.
2. Marcus, Evan, and Hal Stern. *Blueprints for High Availability: Designing Resilient Distributed Systems*. New York: John Wiley & Sons, 2000.
3. Brandon, D.H., and Sidney Segelstein. *Data Processing Contracts: Structure, Contents, and Negotiation*. New York: John Wiley & Sons, 1976.
4. Architecture-Driven Modernization Object Management Group. "A White Paper of Software Assurance." 28 Nov. 2005 <http://adm.omg.org/SoftwareAssurance.pdf>.
5. Fortify's Security Research Group, and Suto, Larry. "Open Source Security Study." July 2008 <www.fortify.com/landing/oss/oss_report.jsp>.
6. Gershenfeld, Neil. "Everything, the Universe, and Life." *IBM Systems Journal* 39.3/4 (2000): 932-934.
7. Anderson, Ross J. *Security Engineering*. 2nd ed. New York: John Wiley & Sons, 2008: 192.

Notes

1. Such a technology is described in the article, "Soft Hardware for a Flexible Chip," which is available at <http://cordis.europa.eu/ictresults/index.cfm?section=news&tpl=article&id=90572>.
2. Information regarding the Software Assurance Initiative and other projects of the Financial Services Technology Consortium is available at <www.fstc.org/projects/index.php?new=1>.
3. A white-wall facility is a term that I heard while developing disaster recovery plans for a major financial institution. The term does not appear to be in the literature and a search for its particular use in the context of disaster recovery did not produce any results.
4. It is necessary to begin looking for a building and then negotiating a lease or purchase, which can take weeks or months.

About the Author

C. Warren Axelrod, Ph.D., is the research director for financial services for the U.S. Cyber Consequences Unit and is executive adviser to the Financial Services Technology Consortium. Previously, he was the chief privacy officer and business information security officer for the U.S. Trust Division of Bank of America. He is also a founder of the Financial Services Information Sharing and Analysis Center. He received the Information Security Executive Luminary Leadership Award in 2007 and *Computerworld*'s Premier 100 Leadership and Best in Class awards in 2003. He has published three books, two of which are on computer management, and numerous articles on a variety of information technology and information security topics. Axelrod holds a doctorate in managerial economics from Cornell University, as well as a master's degree in economics and statistics, and a bachelor's degree in electrical engineering from Glasgow University. He is a Certified Information Systems Security Professional and Certified Information Security Manager.

U.S. Cyber Consequences Unit
P.O. Box 234030
Great Neck, NY 11023
Phone: (917) 670-1720
E-mail: warren.axelrod@usccu.us

Making Security Measurable and Manageable

Robert A. Martin
The MITRE Corporation

The security, integrity, and resiliency of information systems is a critical issue for most organizations. Finding better ways to address the topic is the objective of many in industry, academia, and government. One popular approach is the use of standard knowledge representations, enumerations, exchange formats and languages, and a sharing of standard approaches to key compliance and conformance mandates. By standardizing and segregating the interactions among their operational, development, and sustainment tools and processes, organizations gain great freedom in selecting technologies, solutions, and vendors. These "Making Security Measurable" (MSM) initiatives provide the foundation for answering today's increased demands for accountability, efficiency, resiliency, and interoperability without artificially constraining an organization's solution options.

Since 1999, The MITRE Corporation and others have developed a number of information security standards that are increasingly being adopted by vendors and form the basis for security management and measurement activities across wide groups of industry and government. This article explores how these standards are facilitating the use of automation to assess, manage, and improve the security posture of enterprise security information infrastructures while also fostering resiliency and effective security process coordination across the adopting organizations.

The basic premise of the MSM effort is that for any enterprise to measure and manage the security of their cyber assets, they are going to have to employ automation. For an enterprise of any reasonable size, the automation will have to come from multiple sources. To make the finding and reporting issues consistent and composable across different tools, there has to be a set of standard definitions of the things that are being examined, reported, and managed by those different tools. That standardization is what comprises the core of the MSM efforts.

Information security measurement and management—as currently practiced—is complex, expensive, and fraught with unique activities and tailored approaches. Solving the variety of challenges currently facing enterprises with regards to incident and threat management, patching, application security, and compliance management requires fundamental changes in the way vendor technologies are adopted and integrated. These changes include the way enterprises organize and train to utilize these capabilities. Likewise, to support organizational discipline and accountability objectives while enabling innovation and flexibility, the security industry needs to move to a vendor-neutral security management and measurement strategy. The strategy must be neutral to the specific solution providers while also being flexible enough to work with several different solutions simultaneously. Finally, the new approach should enable the elimination of duplicative and manual activities as well as improve both the resiliency and organizational ability to leverage outside resources and collaborate with other organizations facing the same threats and risks.

These objectives can be met by bringing architecturally driven standardization to the scoping and organization of the information security activities that our enterprises practice. By acknowledging the *natural* groupings of activities or domains that all information security organizations address—independent of the tools and techniques they use—a framework can be established within which organizations can organize their work independent of their current technology choices and flexible enough to adapt to future offerings. Likewise, by examining these domain groupings and the types of practices of coordination and cooperation that persist across and between them, it is possible to improve the interoperability and independence of these groups by standardizing common concepts in the information that flows across and between them. These shared concepts are sometimes referred to as *boundary objects* and are a phenomenon known to those who study inter-community communications[1], but have not been leveraged explicitly for information security standardization.

Using Architecture and Systems Engineering Principles

By leveraging the practices of systems engineering [1], an organization can recast current cybersecurity solutions into a launching point for standard functional decomposition-based security architectures. These architectures will provide a flexible, logical, and expandable approach to building and operating cybersecurity solutions for the enterprise—one that improves resiliency and is more supportive of security measurement, management, and sharing goals.

In this article, I will examine the collection of cybersecurity-related activities that most enterprises practice including: inventorying assets; analysis of system configurations; analysis of systems for vulnerabilities; analysis of threats; study of intrusions; reporting and responding to incidents; change management; systems development assessment; integration and sustainment activities; and certification and accreditation of systems being deployed into the enterprise[2].

I will also examine the different types of information that have been identified to support these activities. Finally, I will identify the key activities and information that need to be sharable and unambiguous in and amongst the different functions of today's cybersecurity environment. Identifying and collecting these functional components as standard reusable concepts illustrates one of the major benefits that architecture brings to the study of security in the enterprise information technology landscape.

Architecting Security

We can lay the foundation for architecting measurable security by looking at security measurement and management as an architecture issue and using a systems engineering approach to functionally decompose it, identifying the basic functions and activities that need to be done, and then getting the appropriate technology to support the functions and activities.

Through the development and adoption of standard enumerations, the establishment of languages and interface standards for conveying information amongst tools and organizations, and by the shar-

ing guidance and measurement goals with others by encoding them into these standard languages and concepts, organizations around the world can dramatically change the options available to address the enterprise's cyber environment security.

Both the U.S. government and commercial enterprises are already starting to deploy new approaches to security measurement and management that leverage interoperability standards and enable enterprise-wide security measurement and policy compliance efforts. These security architecture-driven measurement and management standards [2] are already providing ways for these organizations to create test rules about their minimum secure configurations, mandatory patches, and/or unacceptable coding practices that can be assessed, reported, and any subsequent remediation steps planned, executed, and confirmed using commercial tools. At the same time, these standards also provide a basis for repeatable, trainable processes and sharing along with enabling automation-based testing methods for deployment validation and regression testing throughout the operational lifetime of the systems.

Maybe more importantly, the establishment of architectural methods within the cybersecurity community will help open the doors to more resilient, faster, and better-coordinated approaches to dealing with the next set of security problems. There is little doubt that each of the current solutions being implemented to fight today's threats will be attacked in-turn by advances in how systems and enterprises are attacked. But with a more consistent basis for considering these new threats and methods, solutions can be leveraged faster and applied in more predictable timeframes and with more understanding for the risks that remain.

Building Blocks for Architecting Measurable Security

I believe there are four basic building blocks for architecting measurable security:

- Standardized enumerations of the common concepts that need to be shared.
- Languages for encoding high-fidelity[3] information about how to find the common concepts and communicating that information from one human to another human, from a human to a tool, from one tool to another tool, and from a tool to a human.
- Sharing the information through content repositories[4] in languages for use in broad communities or individual organizations in a way that minimizes loss of meaning when content is being exchanged between tools, people, or both.
- Uniformity of adoption achieved through branding and vetting programs to encourage the tools, interactions, and content remain standardized and conformant.

The following sections discuss these building blocks in more detail.

Enumerations

Enumerations catalog the fundamental entities and concepts in information assurance, cybersecurity, and software assurance that need to be shared across the different disciplines and functions of these practices. The June 2007 National Academies report on the state of cybersecurity and cybersecurity research, "Towards a Safer and More Secure Cyberspace" [3], highlighted that metrics and measurements particularly rely on enumerations. As an example, the report cited the Common Vulnerabilities and Exposures (CVE) [4] list—run by MITRE under funding from the National Cyber-Security Division of the Department of Homeland Security—as an enumeration that enables all kinds of measurement by providing unique identifiers for publicly known vulnerabilities in software. There are a number of enumerations in the information assurance, cybersecurity, and software assurance space. Some examples are shown in Table 1.

Languages

Standardized languages and formats allow uniform encoding of the enumerated concepts and other high-fidelity information for communication from human to human, human to tool, tool to tool, and tool to human. For example, a configuration benchmark document written in the XML Configuration Checklist Data Format (XCCDF) and Open Vulnerability and Assessment Language (OVAL) languages [5, 6] would be readable by a human and it would be consumable by an assessment tool, in that the tool would be able to directly import the tests and checks that are expressed in the document. As with the enumerations, there are a number of information assurance, cybersecurity, software

Table 1: *Enumerations*

Name	Topic
CVE	Standard identifiers for publicly known vulnerabilities.
Common Weakness Enumeration (CWE)	Standard identifiers for the software weakness types in architecture, design, or implementation that lead to vulnerabilities.
Common Attack Pattern Enumeration and Classification (CAPEC)	Standard identifiers for attacks.
Common Configuration Enumeration (CCE)	Standard identifiers for configuration issues.
Common Platform Enumeration (CPE)	Standard identifiers for platforms, operating systems, and application packages.
The SANS Institute Top 20 Security Risks	Consensus list of the most critical vulnerabilites that require immediate remediation.
Open Web Application Security Project's Top 10	List of the 10 most critical Web application security flaws.
Web Application Security Consortium's Threat Classification	List of Web security attack classes.
CWE/SANS Top 25 Most Dangerous Programming Errors	Consensus list of the most dangerous types of programming errors that require immediate attention.

assurance measurement, and management-oriented languages and formats. Some examples are shown in Table 2.

Repositories

Repositories allow common, standardized content to be used and shared, whether across broad communities or within individual organizations. The sharing of content has been done for some time but doing so in standard machine-consumable languages and formats using standard enumerated concepts is fairly recent. Most of the listed repositories are in the midst of converting their content into machine-consumable form. Examples are shown in Table 3.

These are all examples of very public repositories with a variety of types of content that will be recast into standardized machine-consumable form using some of the languages identified in Table 2 and the enumerations in Table 1. However, there are also closed repositories where, for instance, a company may write a tailored set of policies about what they want to do to comply with the Sarbanes-Oxley Act or something similar. They don't necessarily want to share this with the world, but they do want to be standard across all of the different elements of their company and they want their policies available for their auditors and possibly their partners.

Uniformity of Adoption

Uniform adoption of standards by the community is best achieved through branding/vetting programs that can help the tools, interactions, and content remain conformant with the accepted standards.

MITRE's CVE project employs a highly successful CVE Compatibility Program that has vetted numerous information security products and services to ensure they are *CVE Compatible*; that is, they can interoperate with other compatible products that each have correctly mapped their capabilities concept of a particular vulnerability to the correct CVE Identifier for that vulnerability. Similarly, OVAL employs an OVAL Compatibility Program and CWE has begun a CWE Compatibility Program. The National Institute of Standards and Technology (NIST) has also initiated a Security Automation Validation Program (SCAP) for those vendors that currently provide (or intend to provide) SCAP-validated tools.

All of these programs—and others that may be developed in the future—will help ensure consistency within the security community regarding the use and implementation of the standards. They also assure users that the tools, services, and information from those organizations adopting the standards are doing so correctly and that there is a high confidence that they will work correctly when the tools and services are used together.

Table 2: *Languages*

Name	Topic
XCCDF	An XML specification language for writing security checklists, benchmarks, and related documents.
OVAL	An XML state expression language for writing assessment tests about the current state of an asset and expressing the results.
Common Vulnerability Scoring System (CVSS)	A method for conveying vulnerability-related risk and risk measurements.
Common Result Format (CRF)	A standardized IT asset assesment result format that facilitates the exchange and aggregation of assessment results.
Semantics of Business Vocabulary and Business Rules (SBVR)	A vocabulary and rules for documenting the semantics of an area of a business' vocabulary, facts, and processes.
Common Event Expression (CEE)	A language and syntax for describing computer events, how the events are logged, and how they are exchanged.
Malware Attribute Enumeration and Characterization (MAEC)	A language for decribing malware in terms of its attack patterns, detritus, and actions.
Common Announcement Interchange Format (CAIF)	An XML-based format for storing and exchanging security announcements.

How the Architectural Building Blocks Come Together

The building blocks of architecting for measurable security are already in use in the enterprise security areas of configuration compliance assessment, vulnerability assessment, system assessment, and threat assessment.

Configuration Guidance, IT Change Management, and Centralized Reporting

An Office of Management and Budget (OMB) memorandum [7] references the content in NIST's National Vulnerability Database (NVD). This guidance is also referred to as part of the Federal Desktop Core Configuration (FDCC) [8] and is intended to bring consistency in the specific secure system software configuration of Microsoft Windows XP and Vista in use by the federal government. The part of the memo that is directed at Vista directly points to a set of content that uses the XCCDF and OVAL languages along with the CPE and CCE enumerations [9, 10]. This is a fairly public example of benchmark documents in a repository using standard languages and enumerations.

Figure 1 shows how an organization can utilize a tool-consumable benchmark document from a knowledge repository for configuration guidance. The benchmark provides the checking logic for a commercial tool that is used by the organization to conduct their configuration guidance analysis for assessing the configuration compliance of the organization's computer systems. OMB's Vista Guidance from the NVD is an example of this.

As shown in Figure 1, the results of the benchmark examination are also provided in standard language and enumeration terms as it is fed to the enterprise's IT change management and central reporting processes. Figure 1 also shows how security measurement and management activities can be abstracted through a systems

engineering analysis view to establish the security activities of configuration guidance analysis, enterprise IT change management, and centralized reporting as functional areas that can be managed.

Vulnerability alerts (e.g., those referenced in the NVD) are another case in point. Sometimes these are standardized already, depending which source they come from. Figure 2 (see next page) shows how an organization can utilize a tool-consumable vulnerability assessment document from a knowledge repository: It will provide the checking logic for a commercial tool that is used by the organization to conduct their vulnerability analysis for assessing the vulnerability remediation compliance status of the organization's computer systems. One example is errata from Red Hat, Inc., which are regularly posted with CVEs, OVAL definitions, and CVSS scores. As shown in Figure 2, the results of the vulnerability assessments are fed to the enterprise's IT change management and central reporting processes.

Figure 2 also shows how vulnerability assessment and analysis can be abstracted through a systems engineering analysis view as a functional area that can be managed.

System Assessment

System assessments and certifications are not currently standardized. This is an area where standardization is being pursued through the development of efforts like CWE and CAPEC to address the developed components of a system along with the vulnerability and configuration assessment illustrated in Figures 1 and 2.

Figure 3 (see next page) shows how an organization could utilize a tool-consumable body of certification requirements from a knowledge repository for system certification guidance in order to capture the criteria for assessing the status of an organization's computer systems. One example is the Enterprise Mission Assurance Support Service effort being developed within the DoD. As shown in Figure 3, the results of the certification and accreditation examination is fed to the enterprise's IT change management and central reporting processes.

Figure 3 also shows how certification activities can be abstracted through a systems engineering analysis view as a functional area that can be managed.

Threat Assessment

Threat alerts and assessment is another area that has not yet been fully standardized. Imagine how an organization could utilize tool-consumable information from a knowledge source (about new and existing threats) that provided an efficient way of comparing threat information such as targeted platforms, vulnerabilities, or weakness against the enterprise's information about their assets and their status. One example is the commercial threat reports that several security service providers offer. Imagine that results of analyzing new threat information can be fed to the enterprise's IT change management and central reporting processes. In this vision, threat analysis would be abstracted to a vendor and tool-neutral activity through a systems engineering analysis view.

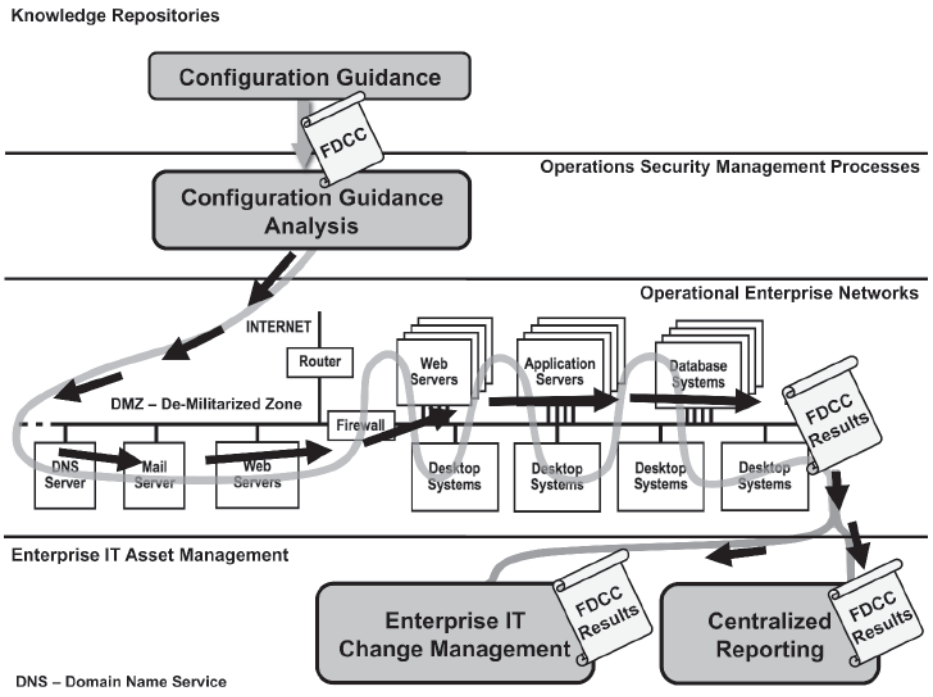
This same process of abstraction can be used to identify and define the other security measurement and management activities that an organization conducts.

Figure 4 (on page 31) contains our current cut at abstracting and decomposing the overall security management and measurement activities of an enterprise (as described so far in this article), along with the other enterprise security management processes of an inventory asset activity,

Name	Topic
DoD Computer Emergency Response Team (CERT)	Information Assurance Vulnerability Alerts (IAVAs) and Defense Information Systems Agency's (DISA) Security Technical Implementation Guides (STIGS)
The Center for Internet Security (CIS)	CIS Security Configuration Benchmarks
National Security Agency (NSA)	NSA Security Guides
National Vulnerability Database (NVD)	US-CERT advisories, US-CERT Vuln Notes, CVE and CCE Vulnerabilities, checklists, OVAL definitions, and U.S. Information Security Automation Program (ISAP) and Security Content Automation Protocol (SCAP) content.
Red Hat Repository	OVAL Patch Definitions for Red Hat Errata security advisories
OVAL Repository	OVAL Vulnerability, compliance, inventory, and patch definitions.

Table 3: *Repositories*

Figure 1: *Assessment of Configuration Compliance Using Standards Vulnerability Assessment*



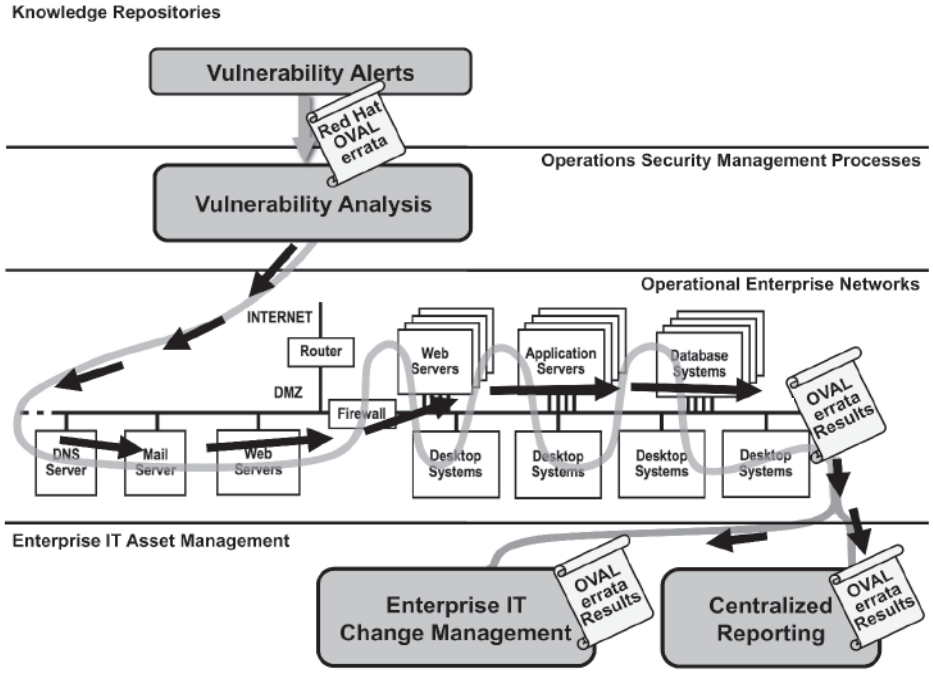


Figure 2: *Assessment of Vulnerability Remediation Status Using Standards*

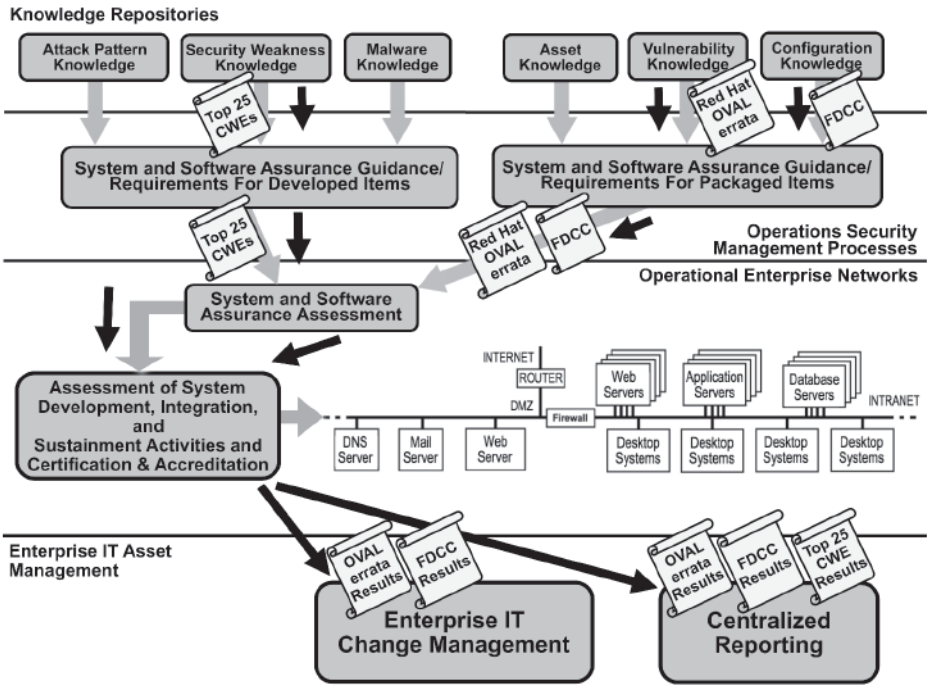
studying incidents, assessment of systems development, integration, and sustainment activities.

Furthermore, Figure 4 illustrates how the different security measurement and management activities are tied together through standards-based data interfaces that utilize the standard enumerations and standard languages discussed earlier. By utilizing these abstracted activities and enforcing the use of the standards-based interactions between them, an organization can bring commercially available technologies and tools to bear on their security problems while still keeping control of the processes and activities[5].

Standard repositories of governance and guidance can help drive the business value of these standard measurement and management activities. As shown in the OMB guidance example, the information about how systems should be configured is captured by OVAL, XCCDF, CCE, and CPE.

The configuration guidance analysis, enterprise IT change management, and centralized reporting activities depicted in Figures 1 through 3 are several of the security measurement and management activities abstracted by taking a systems engineering analysis view of some of the different security activities of an organization.

Figure 3: *System Certification and Accreditation Using Standards*



Reusable and Shared Repositories

Similarly, as shown on the left side of Figure 4, these same standards can be used to capture how an organization has configured and set up a new system when it has been approved for use in an enterprise. By using these standards, this information can go right into operational network management so that an organization can make sure the new system continues to be configured in the way that it was approved. Standard guidance can also be included about what weaknesses from CWE [11] should be reviewed in an organization's or supplier's development activities. In addition, the common attack patterns from CAPEC [12] can be used to define and document the types of penetration testing and attack scenarios a development team thought about defending against when they were doing their development and penetration testing.

For asset inventory, standards-based information utilizing CPE and OVAL will let an organization know exactly what assets they have in a manner that is tool-independent and usable in the other standard activities (such as configuration analysis). Similarly, if an organization knows exactly how their assets are configured, it is much easier to perform vulnerability analysis based on CVE, CWE, OVAL, and CVSS. Likewise, if an organization knows what they have, how it is configured, and what it is vulnerable to, that will change the context and framework of how the threat analysis is done.

As mentioned earlier, vulnerability alerts are sometimes standardized already, depending which source they come from. Red Hat errata, for example, are regularly posted with CVEs, OVAL definitions, and CVSS scores. In this area particularly, the standards have already been adopted by industry.

Since threat alerts are not as of yet standardized, this is an area where standardization could happen, and efforts like MAEC are aimed at enabling that. Similarly, there are a lot of different ideas in incident reporting regarding what should be standardized and to what extent those areas should be standardized.

There are many aspects of usage that are still evolving, including the correct approach to managing changes, updates, or new content for shared repositories. The question of whether the repositories should be enabled as services, as static col-

lections, or both is also open. Similarly, as new insights are made with respect to vulnerabilities, weaknesses, threats, and attacks, there certainly will be changes needed in how the different aspects of these types of information are woven together and used. By bringing the various aspects of cybersecurity, information assurance, and software assurance into a consistent security architecture framework, there will be many new opportunities and much faster responses to new threats and new information. A compelling use of the enumerations, languages, and repositories can be found in the new "Consensus Audit Guidelines" [13], offered by the Center for Strategic and International Studies to advance key recommendations from the report on Cybersecurity for the current 44th Presidency [14]. The guidelines incorporate many of the items described in this article as an approach to clearly and concisely communicate what needs to be done and what needs to be audited.

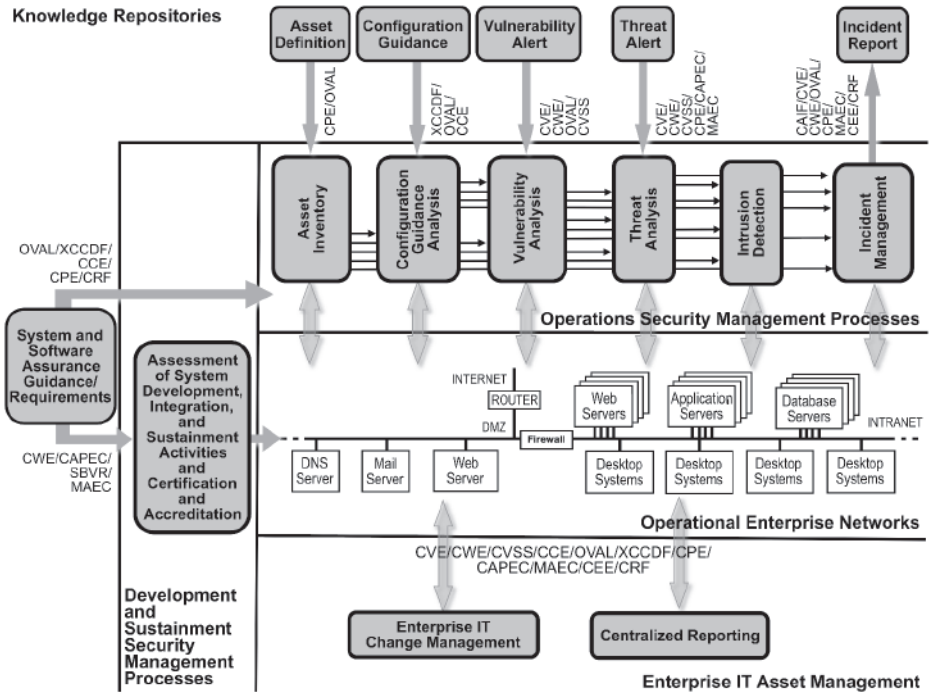


Figure 4: *Decomposition and the Repositories Feeding Standard Measurement and Management Activities*

Conclusion

Measurable security and automation can be achieved by having government and public efforts:

- Address information security during the creation, adoption, operation, and sustainment—in a holistic manner.
- Use common, standardized concepts.
- Communicate this information in standardized languages.
- Share the information in standardized ways.
- Adopt tools that adhere to the standards.

Much has already been done to transform the way security measurement and management is conducted, but there is still plenty of work that needs to be addressed. The use of architecture and systems engineering principles has been shown to be effective and enabling. Ongoing efforts to address and evolve all of the activities in this arena will greatly benefit from the continued application of this methodology. Like most architecture efforts today, the true value of architecture is not apparent or appreciated until its enabling properties start to manifest themselves. This article has outlined the changes in security practices and technologies and has shown specific and measurable changes that are directly related to the use of architectural methods on *security of information technologies* in government and private industry. This article also showed the benefits in sharing that standardized information.

By creating and evolving these types of standards and new approaches to security measurement and management, each of us will need to step away from the traditional focus on local and enterprise issues. We must realize that much more powerful and productive solutions to these issues can be fostered through an emphasis on community-wide examinations of each of the technical areas where a multitude of concerns and needs are balanced and considered. The increased insights, resiliency, and ability to leverage the collective knowledge and first-hand experience of what vulnerabilities and attacks affect us are valuable benefits to trading off local versus community-wide concerns.

To further the goal of making security measurable and encouraging the participation and adoption of the different aspects of this work, MITRE has established a public MSM Web site <http://makingsecuritymeasurable.mitre.org> that informally collects all of the efforts listed in this article, as well as others that are known about, which together are helping or will help in making security more measurable.◆

References

1. Chestnut, Harold. *Systems Engineering Tools*. New York: John Wiley & Sons, 1965.
2. Martin, Robert A. "Transformational Vulnerability Management Through Standards." CROSSTALK May 2005 <www.stsc.hill.af.mil/crosstalk/2005/05/0505Martin.html>.
3. Goodman, Seymour E., and Herbert S. Lin. *Toward a Safer and More Secure Cyberspace*. Washington, D.C.: National Academies Press, 2007.
4. "Common Vulnerabilities and Exposures." The MITRE Corporation. 25 July 2009 <http://cve.mitre.org>.
5. Ziring, Neal, and Stephen D. Quinn. "The Specification for the Extensible Configuration Checklist Description Format Vers. 1.1.4." National Institute of Standards and Technology. Jan. 2008 <http://csrc.nist.gov/publications/nistir/ir7275r3/NISTIR-7275r3.pdf>.
6. "The Open Vulnerability and Assessment Language." The MITRE Corporation. 25 July 2009 <http://oval.mitre.org>.
7. Evans, Karen S. "Ensuring New Acquisitions Include Common Security Configurations." Memorandum for Chief Information Officers and Chief Acquisition Officers. 1 June 2007 <www.whitehouse.gov/omb/memoranda/fy2007/m07-18.pdf>.
8. "The Federal Desktop Core Configuration – FDCC." NIST <http://nvd.nist.gov/fdcc>.
9. "Common Platform Enumeration." The MITRE Corporation. 25 July 2009 <http://cpe.mitre.org>.
10. "Common Configuration Enumeration." The MITRE Corporation. 25 July 2009 <http://cce.mitre.org>.
11. "Common Weakness Enumeration." The MITRE Corporation. 25 July 2009 <http://cwe.mitre.org>.

Software Defense Application

The security, integrity, and resiliency of cyber systems is critical within the DoD and is essential for its mission and support capabilities. This article describes and defines how the use of standard knowledge representations, enumerations, exchange formats and languages, and a sharing of standard approaches is helping transform key compliance and conformance mandates for the DoD, such as the Information Assurance Vulnerability Management process, the Security Technical Implementation Guidelines, and systems development. By adopting standards and segregating the interactions amongst their operational, development, and sustainment tools and processes, the DoD is and will gain greater freedom in selecting technologies, solutions, and vendors while also obtaining deeper insights into the current operational security and integrity of mission systems. These MSM initiatives answer today's increased process demands without artificially constraining the solution options of the DoD.

12. "Common Attack Pattern Enumeration and Classification." The MITRE Corporation. 25 July 2009 <http://capec.mitre.org>.
13. "Twenty Critical Controls for Effective Cyber Defense: Consensus Audit Guidelines." SANS Institute. 9 May 2009 <www.sans.org/cag/print.php>.
14. "Commission on Cybersecurity for the 44th Presidency." Center for Strategic and International Studies Corporation. 2009 <www.csis.org/tech/cyber>.

Notes

1. To learn more about inter-community communications, see "Sorting Things Out: Classification and Its Consequences" by Geoffrey C. Bowker and Susan Leigh Star, MIT Press, 1999.
2. This is an integrated list that includes activities tied to the operation of systems in the enterprise as well as those they create, deploy, and update.
3. High fidelity refers to the level of detail of the information encoded in a language that is sufficient to convey the understanding and knowledge of the one encoding the information to the one who decodes the information. If a person writes a test for how to check a configuration setting in a language, then that language needs to be able to convey the specifics of the test so that another person or a tool reading the check as written in the language understands enough about the check to actually perform the test that was intended by the original author. If a language cannot retain the fidelity of the information to support this, then it is not of sufficient fidelity.
4. Content repositories are currently envisioned to be collections of tests to verify settings, patches, and installed software on systems to comply with organizational policies regarding their information technology systems and processes. Repositories are typically meant to be understandable by humans but are used by tools to automate checking for compliance with the tests in the repository. Many different organizations are hosting public and private repositories already and this is anticipated to continue and expand as the need to share grows.
5. The unwanted alternative is ending up with activities that are defined by the scope of the tools being used and that are coupled together by proprietary mechanisms.

About the Author

Robert A. Martin is a principal engineer in MITRE's Information and Computing Technologies Division. For the past nine years, his efforts have been focused on the interplay of enterprise risk management, cybersecurity standardization, critical infrastructure protection, and the use of software-based technologies and services. Martin is a member of the Association for Computing Machinery, Armed Forces Communications and Electronics Association, IEEE, and the IEEE Computer Society. He has bachelor's and master's degrees in electrical engineering from Rensselaer Polytechnic Institute, and an MBA from Babson College.

The MITRE Corporation
202 Burlington RD
Bedford, MA 01730-1420
Phone: (781) 271-3001
Fax: (781) 271-8500
E-mail: ramartin@mitre.org

Meeting the Challenge of Assuring Resiliency Under Stress

Don O'Neill
Independent Consultant

An emerging issue, especially critical to the DoD and DHS, is that of managing network security, assuring the continuity of operations for critical defense missions and the resiliency of the private sector's critical infrastructure. Making systems of systems resilient requires accountability and transparency. This article provides a framework for assuring resiliency under stress expressed in terms of the management, process, and engineering indictors useful in asserting resiliency assurance claims, validating assurance arguments, and verifying assurance evidence.

Next generation software engineering faces many challenges [1], and the impacts of these challenges are being encountered every day by acquisition agents, software developers, and operating commands alike:

1. Acquisition agents need to deliver more with less ... fast.
2. Software developers need to shorten software development life cycles in producing trustworthy software systems composed of existing components.
3. Both acquisition agents and software developers need to exhibit better user domain awareness.
4. Operating commands need to field and sustain resilient systems of systems composed of legacy systems.

The industry has been grappling with many of these issues for years [2, 3]. Persistent acquisition challenges and chronic software development cost and schedule overruns frequently obscure the needs of the user. Despite this past neglect and unfinished business, the challenge of assuring resiliency under stress in systems of systems has emerged as an imperative that needs attention now.

In managing the investment needed to meet these objectives, capability portfolio investments are organized by management, process, and engineering. To receive results, utilize the objective (shown in Table 1) from top to bottom. In this way, user domain awareness, shortened life cycles, systems from parts, and systems of systems from systems provide a natural spiral of incremental activities where current work in progress builds on preceding work accomplished.

Resiliency Defined

The attribute of resiliency is an emerging property of large complex software-intensive systems. Accordingly, the base definition of resiliency is:

> ... the ability to anticipate, avoid, withstand, minimize, and recover from the effects of adversity, whether natural or manmade, under all circumstances of use. [4]

The base definition of resiliency is not limited as to scale, does not preclude the possibility for avoiding the condition or situation that brings impact or shock, does not limit the focus to a means like risk management, and does not limit the focus to enumerated outcomes like cost effective or timely restoration. However, in applying the base definition to a particular situation, it is permissible and even required to constructively instantiate it for targeted scale, impact expected, means employed, and outcome anticipated [5, 6].

Claiming Resiliency Assurance

The purpose of assurance assertion management is to reason about the emergent properties of large complex software-intensive systems in order to steer acquisi-

Table 1: *Practical Next-Generation Software Engineering (NGSE)*

Objective	Management Action	Process	NGSE Technology
Objective 1: *Drive user domain awareness towards more harmonious cooperation among people and machines.* **Strategic Measures:** 1. User satisfaction. 2. Trustworthiness.	**Integrate needs of systems, software, and user:** • Synthesize mission needs in terms of systems, software, and user. • Apply team innovation management.	**User domain awareness maturity:** • Assessment of user domain awareness.	• Simulation. • Virtual user experience.
Objective 2: *Simplify and produce systems and software using a shortened development life cycle.* **Strategic Measures:** 1. Speed. 2. Trustworthiness.	**Eliminate bottlenecks:** • Automation of labor-intensive activities.	**Accelerate delivery:** • Wiki-based requirements. • Incremental development. • Agile approaches.	• Formality in requirements expression. • Smart compilers. • Correctness by construction.
Objective 3: *Compose and field trustworthy applications and systems from parts.* **Strategic Measures:** 1. Frequency of release. 2. Trustworthiness.	**Rapid Release:** • Aspect-based commitment management. • Fact-based aspect and attribute assurance. • Real-time risk management.	**Supplier Assurance:** • Process maturity. • Global supply chain management. • Configuration management.	• Attribute-based architecture. • Smart middleware. • Interoperability. • Intrusion detection, protection, and tolerance.
Objective 4: *Compose and operate resilient systems of systems from systems.* **Strategic Measures:** 1. Control. 2. Resilience.	**Control:** • Exercise control.	**Awareness:** • Intelligent middlemen. • Information sharing. • Situation awareness.	• Coordinated recovery time objectives. • Distributed supervisory control. • Operation sensing and monitoring.

Overall Goal: *Drive systems and software engineering to do more with less ... fast.*

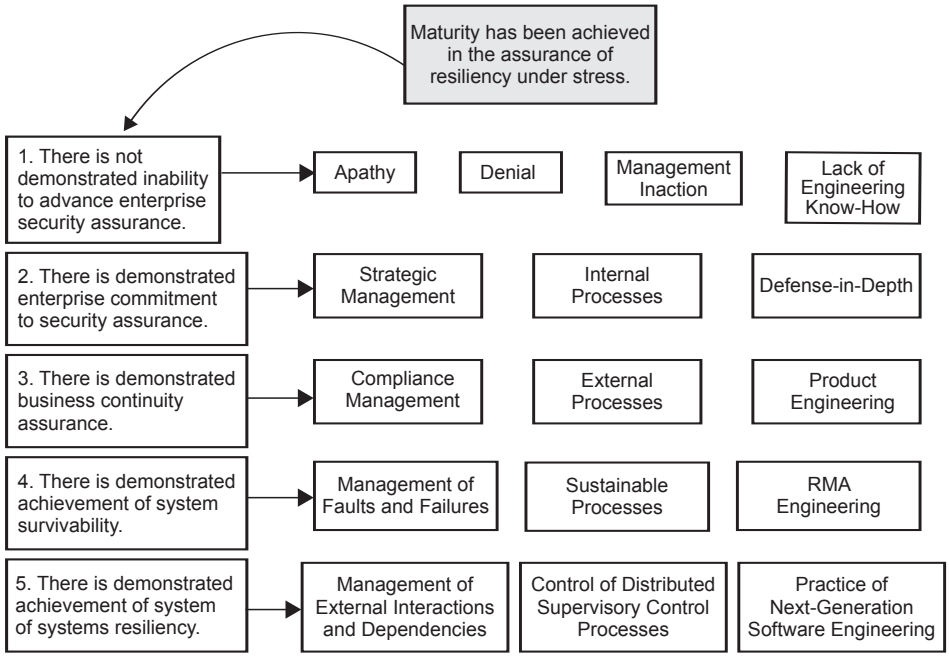


Figure 1: *Claim-Argument-Evidence Chain for Assessing Resiliency Assurance*

tion, development, and operational commitment towards their assurance and to guide users in setting the appropriate level of confidence in these systems and systems of systems [7].

An assurance assertion is a statement designed to inspire confidence. These emergent product properties transcend the rigorous and precise methods of assessing essential compliance beyond those used in process conformance [8, 9] and product testing. Some attribute and aspect examples of emergent properties associated with software products, systems, and system of systems include safety, security, resiliency, privacy, and trustworthiness [10].

The assurance claim for assuring resilience under stress in an enterprise is organized around five arguments expressed as questions:

1. Is there no demonstrated inability to advance enterprise security assurance?
2. Is there demonstrated enterprise commitment to security assurance through strategic management, internal processes, and defense-in-depth?
3. Is there demonstrated business continuity assurance through compliance management, external processes, and product engineering?
4. Is there demonstrated achievement of system survivability through the management of faults and failures, sustainability processes, and Reliability, Maintainability, and Availability (RMA) engineering?
5. Is there demonstrated achievement of system of systems resiliency through the management of external interactions and dependencies, the control of distributed supervisory control processes, and the practice of next generation software engineering?

The assurance claim for resilience assurance, the five arguments demonstrating resiliency assurance, and the types of evidence expected for each argument are shown in the Claim-Argument-Evidence Chain (Figure 1).

Assurance assertions themselves are subject to validation and verification, and it is here that managing the risk associated with assuring resiliency is focused. The claim-argument segment of the assurance assertion chain is validated when the correspondence between a claim and its arguments is shown to be clear and convincing with respect to completeness and correctness.

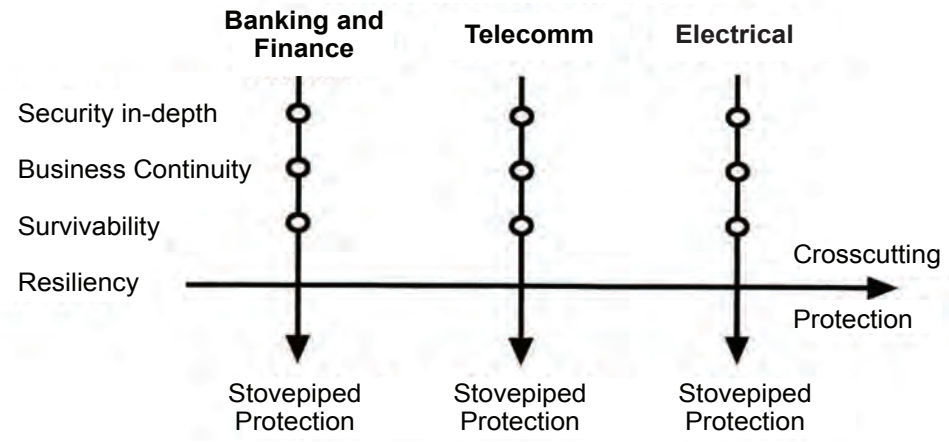
The argument-evidence segment of the assurance assertion chain is verified according to the degree of correspondence between the evidence and the argument. Four levels of confidence for appraising evidence are identified as follows:

1. The evidence in support of the argument is insufficient.
2. The preponderance of the evidence supports the argument (e.g., through assessment, interview, testimony, and inspection).
3. The evidence in support of the argument is clear and convincing (e.g., measurement and static analysis).
4. The evidence in support of the argument is beyond a shadow of a doubt (e.g., demonstration and dynamic analysis).

Achieving Resiliency

Assuring resiliency under stress is achieved through a framework of management, process, and engineering capabilities and indicators organized around managed review, defined process capability, and a designed engineering solution. Achieving system of systems resiliency brings with it an architectural challenge associated with the need to counter the effects of crosscutting and cascading triggers. Borrowing an example from the critical infrastructure and a dependency from the industrial base, stovepiped vertical protection, and crosscutting horizontal protection through resiliency are illustrated in Figure 2.

Figure 2: *Vertical Protection and Horizontal Resilience*



Crosscutting effects stem from dependent relationships. Some dependent relationships are planned and intended interactions between industry sectors—such as financial transactions embedded in telecommunications, electrical, transportation, and medical operations—where cross sector impacts are surprisingly pervasive [11]. Other dependent relationships are indirect and stem from outsourced commoditized services that bring with them opportunities for common single-point failures among industry sectors—

such as the Internet and global positioning systems [5].

Building on security in depth [12, 13, 14], business continuity [15], and system survivability [16], a defined engineering challenge of adopting system of systems resilience must be addressed [17]. The recovery time objectives among systems must be coordinated, interoperability of information sharing and platform operations must be assured, distributed supervisory control protocols must be in place, operation sensing and monitoring must be embedded, and digital situation awareness must be achieved. These capabilities are designed to counter crosscutting effects and cannot be expected to evolve in a loosely coupled environment. They must be holistically specified, architected, designed, implemented, and tested if they are to operate with resilience under stress [18]. A management, process, and engineering framework is necessary to advance the assurance of software security, business continuity, system survivability, and system of system resiliency capabilities (see Table 2).

Conclusion

This article has sought to point the way towards accountability and transparency in assuring the resiliency of systems of systems. Each operating command and critical infrastructure sector must insist on accountability from each system manager for its security in-depth, business continuity, and survivability. In addition, system managers must adopt transparency to the resiliency assurance claims, arguments, and evidence as the preferred means to achieve and demonstrate coordinated recovery time objectives, interoperability, operation sensing and monitoring, digital situation awareness, and distributed supervisory control.◆

Software Defense Application

The critical infrastructure is the industrial base on which the competitiveness and security of the nation are dependent. The defense industrial base finds itself in the mesh of the critical infrastructure. Diverse cybersecurity threats to the defense industrial base are posed by various factors parsed into type of risk, actor, attack, target, and countermeasure. For example, the type of actor includes disgruntled employee, hacker, criminal, terrorist, organized crime, and nation state. Faced with a complex array of threats, the critical infrastructure protection (CIP) model is insufficient to ensure the continuity of operations for critical missions. In addition to CIP, a critical infrastructure resiliency model is needed to anticipate, avoid, and mitigate cascading and propagating effects within systems of systems. "Meeting the Challenge of Assuring Resiliency Under Stress" provides a definition and framework of assurance claims useful in assuring resiliency maturity throughout industry, government, and defense.

The challenge associated with assuring the resiliency of systems of systems —based on a broad definition for resiliency—calls for a framework for assuring resiliency under stress expressed in terms of management, process, and engineering indictors useful in asserting resiliency assurance claims, validating assurance arguments, and verifying assurance evidence. The targeted users for assuring resiliency under stress include selected sectors within the critical infrastructure and defense industrial base and certain operating commands within the defense establishment. These are characterized by their increasing dependence on the acquisition, development, fielding, and sustainment of large-scale, complex systems of systems.

References

1. O'Neill, Don. "Preparing the Ground for Next Generation Software Engineering." Annual Technology Report. IEEE Reliability Society, 2008: 148-151.
2. "Software 2015: A National Software Strategy to Ensure U.S. Security and Competitiveness." Center for National Software Studies. 25 Apr. 2005 <www.cnsoftware.org/nss2report/NSS2FinalReport04-29-05PDF.pdf>.
3. "Future Directions in Software Engineering." *Software Tech News* 10.3. Oct. 2007 <www.softwaretechnews.com/pdf/stn10_3.pdf>.
4. O'Neill, Don. "Calculating Security Return on Investment." *Build Security In*. DHS National Cybersecurity Division. 6 Feb 2007 <https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/business/677-BSI.html>.
5. "Critical Thinking: Moving From Infrastructure Protection to Infrastructure Resilience." George Mason University School of Law: Critical Infrastructure Protection Program. Discussion Paper Series. Feb. 2007 <http://cip.gmu.edu/archive/CIPP_Resilience_Series_Monograph.pdf>.

Table 2: *Framework for Assuring Resiliency Under Stress Goals and Indicators*

Framework for Assuring Resiliency Under Stress•	Focused Management Review	Defined Process Capability	Designed Engineering Solution
Demonstrate commitment to security assurance through strategic management, internal processes, and defense-in-depth.	• Competitiveness vs. security assessment and tradeoff. • Security return on investment. • Incident management.	• Chief Security Officer (CSO) leadership program. • Security assurance operations. • Configuration management.	• Encryption. • Identity management. • Access control. • Authorization management. • Accountability management.
Demonstrate business continuity assurance through compliance management, external processes, and product engineering.	• Regulatory compliance. • Aspect oversight and assesment.	• Global sourcing. • Risk management. • Crisis management.	• Open source. • COTS software. • Security assurance evaluation tools.
Demonstrate the achievement of system survivability through the management of faults and failures, sustainability processes, and RMA engineering.	• Incident management. • Cyber forensics. • Management of defects, faults, and failures.	• Resistance. • Recognition. • Recovery. • Reconstitution.	• Reliabilty engineering. • Availability engineering. • Maintainability engineering.
Demonstrate the achievement of system of systems resiliency through the management of external interactions and dependencies, the control of distributed supervisory processes, and the practice of next generation software engineering.	• Coordinated recovery time objectives. • Interoperability of data and information exchange.	• Operation sensing and monitoring. • Digital situation awareness. • Distributed supervisory control. • Information and data recovery.	• Next-generation software engineering.

• The overall goal is to drive the business case and enterprise commitment towards the assurance of software security, business continuity, system survivability, and system of systems resiliency.

6. Miller, Robert A., and Irving Lachow. "Strategic Fragility: Infrastructure Protection and National Security in the Information Age." *Defense Horizons* 59. Jan. 2008 <www.ndu.edu/ctnsp/defense_horizons/DH59.pdf>.
7. Goodenough, John, Howard Lipson, and Chuck Weinstock. "Arguing Security – Creating Security Assurance Cases." *Build Security In*. DHS National Cybersecurity Division. 4 Jan. 2007 <https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/assurance/643-BSI.html>.
8. CMMI Product Team. "CMMI for Development, Version 1.2." SEI, Carnegie Mellon University. Technical Report CMU/SEI-2006-TR-008. Aug. 2006 <www.sei.cmu.edu/pub/documents/06.reports/pdf/06tr008.pdf>.
9. Caralli, Richard A., et al. "Introducing the CERT Resiliency Engineering Framework: Improving the Security and Sustainability Processes." SEI, Carnegie Mellon University. Technical Report CMU/SEI-2007-TR-009. May 2007 <www.sei.cmu.edu/pub/documents/07.reports/07tr009.pdf>.
10. Jackson, Daniel, Martyn Thomas, and Lynette I. Millett. *Software for Dependable Systems: Sufficient Evidence?* Washington, D.C.: National Academies Press, 2007.
11. Borg, Scott. "Recommendations for NDU Cyber Risk and Response Conference." U.S. Cyber Consequences Unit, National Defense University. Jan. 2009 <www.ndu.edu/CTNSP/cyberworkshop/1030%20BORG.pdf>.
12. Chess, Brian, Gary McGraw, and Sammy Migues. *The Building Security In Maturity Model*. 2009 <www.bsimm.com>.
13. "Fundamental Practices for Secure Software Development: A Guide to the Most Effective Secure Development Practices in Use Today." *SAFE Code*. 8 Oct. 2008 <www.safecode.org/publications/SAFECode_Dev_Practices1008.pdf>.
14. Collins, Rosann W., et al. "The CERT Function Extraction Experiment: Quantifying FX Impact on Software Comprehension and Verification." SEI, Carnegie Mellon University. Technical Note CMU/SEI-2005-TN-047. Dec. 2005 <www.sei.cmu.edu/pub/documents/05.reports/pdf/05tn047.pdf>.
15. O'Neill, Don. *Inside Track to Offshore Outsourcing Using the Trusted Pipe™: What Global Enterprises Look For in Offshore Outsourcing*. Proc. of the Making the Business Case for Software Assurance Workshop, Carnegie Mellon University, Pittsburgh. 26 Sept. 2008 <www.sei.cmu.edu/community/BCW_Proceedings.pdf>.
16. Ellison, R.J., et al. "Survivable Network System: An Emerging Discipline." SEI, Carnegie Mellon University. Technical Report CMU/SEI-97-TR-013. Nov. 1997, Rev. May 1999 <www.cert.org/research/97tr013.pdf>.
17. O'Neill, Don. "Maturity Framework for Assuring Resiliency Under Stress." *Build Security In*. DHS National Cybersecurity Division. 11 July 2008 <https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/business/1016-BSI.html>.
18. Northrop, Linda. *Architecting High Quality Software: The Role of Software Architecture in System Development and Evolution*. Proc. of the 2nd Annual Team Software Process Symposium, Orlando, FL. Keynote Presentation. 19 Sept. 2007 <www.sei.cmu.edu/ tsp/symposium/2007/Day%203%20830AM%20SEI%20keynote.pdf>.

About the Author

Don O'Neill is a seasoned software engineering manager and technologist. As an independent consultant, O'Neill conducts defined programs for managing strategic software spanning competitiveness, security, and process improvement. Following his 27-year career with IBM's Federal Systems Division, O'Neill completed a three-year residency at the SEI under IBM's Technical Academic Career Program, and currently serves as an SEI visiting scientist. He served on the executive board of the IEEE Software Engineering Technical Committee and as a distinguished visitor of the IEEE. He is a founding member of the Software Process Improvement Network and served as the president of the Center for National Software Studies.

9305 Kobe WY
Montgomery Village, MD 20886
Phone: (301) 990-0377
E-mail: oneilldon@aol.com

Web Sites

Embedded.com

www.embedded.com

If this issue's "Considering Software Protection for Embedded Systems" spurred your interest, you'll want to explore this companion to *Embedded Systems Design* magazine and the Embedded Systems Conferences. Embedded.com—a resource for technical information and news on embedded design—is the global online authority for embedded designers and technical managers who are responsible for defining systems, selecting the critical hardware and software components, building the systems, and integrating the hardware and firmware designs. The Web site provides practical design techniques, new product updates, how-to technical features, as well as weekly columns and polls.

When Robots Invaded the Senate

www.nsf.gov/news/news_summ.jsp?cntn_id=115211&org=NSF&from=news

At the heart of "Robots" are resilient mixed-criticality systems called cyber-physical systems (CPSs), an emerging technological field that incorporates computing power to improve virtually every facet of modern life—and the government, industry, and mainstream media are taking notice. Experts believe that CPS technologies will increasingly affect our well-being, security, and competitiveness in a variety of areas including aerospace, automobiles, civil infrastructure, energy, finance, healthcare, and manufacturing. In this article, the National Science Foundation discusses the basics of these systems and details a recent luncheon briefing and open house on CPSs for members of the U.S. Senate.

Survivable Systems Engineering

www.cert.org/sse

After reading Karen Mercedes Goertzel's article on software survivability, you may want to learn more about survivable systems engineering. This Carnegie Mellon University Computer Emergency Response Team-sponsored Web site explores the current state of systems to identify problems and proposes engineering solutions. The work focuses on the development life cycles for both new development and COTS-based systems. It includes analysis of how susceptible these systems are to sophisticated attacks and provides suggestions for improving the design of systems based on this analysis.

The U.S. Cyber Consequences Unit

www.usccu.us

The U.S. Cyber Consequences Unit (US-CCU) is an independent, non-profit research institute providing assessments of the strategic and economic consequences of possible cyber-attacks and cyber-assisted physical attacks. At this Web site, learn how the US-CCU investigates the likelihood of such attacks and examines the cost-effectiveness of possible countermeasures. The US-CCU's primary concern is the sort of larger scale attacks that could be mounted by criminal organizations, terrorist groups, rogue corporations, and nation states.

What a Great Ride It Was!

A Farewell From a Longtime CROSSTALK Staffer

Before I go, I would like everyone to know: my name is Nicole. I started at CROSSTALK in March 2001 as an article coordinator. Just days later, my first stop was a conference in New Orleans.

"Sounds great! But wait, what am I doing again?"

I was lucky: When I started here, current BACKTALK regular Dave Cook worked in our organization. He was tasked with introducing me to people and showing me the ropes. Dave was a lifesaver. I left New Orleans armed with 500 names, an open view of my new co-workers and organization, and a new insight into what software engineering was all about.

I learned quickly that the CROSSTALK staff was a tight-knit family that liked having fun. There was the time we emulated a video of a bunch of cubicle-dwellers hooking their office chairs together and "rowing" around the office. We all grabbed our chairs, linked them together, and started rowing down the hallway. Despite our airtight design, excellent teamwork, and perfect rowing skills, we ran right into our boss, Tony Henderson. Silently looking at all of us and shaking his head (as he always did), he just walked on by. Laughing hysterically, we went back to our desks.

Although a lot of long hours and hard work goes on around here, this certainly wasn't the last time we saw the head shake. Both Tony and our current boss, Brent Baxter, have that move down pat when the CROSSTALK staff is around.

There was the time after a fire safety briefing when we decided to practice the fireman's carry. There have been the "off-topic" CROSSTALK production meeting conversations: Drew Brown (our current managing editor) threatening to send my ancient cell phone to the Smithsonian, a child's science project consisting of Jell-O and a Quaker Oats canister, or Chelene Fortier (the associate editor) asking, for the 4,326th time, for a "dedicated color printer." There was Chelene and I holding "cute boy" counting contests at the Systems and Software Technology Conference (SSTC). The tally? Nine years, two cute boys (and you both know who you are!). And there was the staff member who we liked so much that we volunteered him to run for Utah governor. We made posters and buttons and, for a brief morning, our building became his campaign headquarters (of sorts). And there are the old favorites that caused beet-red faces and silent screams of laughter: flatulence machines, keyboard letter-switching, and a pair of strategically placed red balloons (Hi, Bruce!).

I often say, "Aww, good times." But maybe not for the Software Technology Support Center staff (bless their hearts) who sit near us. During my time here, they have all developed nervous ticks and invested in good pairs of earphones. And our poor publisher (Kasey Thompson) and Brent: They come to our area daily, but are no match for our conversational skills. I've heard "I forgot what I came over here for" more times than I can count.

There have also been good times in pursuit of great CROSSTALKs.

There was our November 2002 issue cover <www.stsc.hill.af.mil/crosstalk/2002/11/>. It was August, 95 degrees, and we were bundled up as if it was late autumn. Of course, I got to be the one to climb the tree and sit on the branch ... but what a view!

Every year, I got to play a major role in the SSTC. Meeting attendees, talking with prospective authors, and learning about cutting-edge issues from our presenters was an amazing experience. What each SSTC lacked in "cute" boys, it made up for in CROSSTALK issue topics and new authors.

And there's the process of working with all the authors: getting to really know them, watching their article go from excellent to exceptional, and seeing the reward and excitement when they publish for the first time. The thing that astonishes me most is that all of these people are striving for the same thing: to make better, faster, more cost-effective software that will benefit the government and industry alike. I've learned a lot over the years and truly respect what everyone in the field is doing.

To our CROSSTALK authors: What can I say? CROSSTALK wouldn't be here if it wasn't for you. I am so thankful you all continue to write for our journal.

To our CROSSTALK Editorial Board: Thank you for reviewing all of those articles! Your hard work makes CROSSTALK the high-quality publication it is today.

To our CROSSTALK sponsors: I can't thank you enough for going to bat for our journal every year, standing up and recognizing that CROSSTALK is a highly valuable and extremely beneficial resource for the software engineering field.

To all of my fabulous co-workers, near and far: You have been with me through the good, the bad, the ugly, and the sweet (of course). You have put up with my silly sense of humor, singing, and G-rated swears.

Again, my name is Nicole. I can't tell you how many times people have called me by my now defunct last name, Kentta ("Dear Kentta," or "Thanks Kentta"), despite big fonts, gargantuan signature blocks, or my voice mail message that starts off with "You've reached the voice mail of NICOLE Kentta .... ." I still get a chuckle thinking about it.

I also have to thank all the people with the names more confusing than mine. I've worked with two Kents (Bingham, the man of many amazing CROSSTALK covers, and Poorman, the guy we complain to when the AC is not working), a Ken (a former managing editor), a Kase (another former managing editor), and Kasey. And, of course, I'll never forget the good times during the "all-women" days of CROSSTALK: Tracy, Beth, Pam, Chelene, and Janna.

So, as this chapter in my life ends, the next chapter begins across the country at Shaw AFB in South Carolina (the Air Force is relocating my husband). I promise to bring all of my experience and knowledge with me to my next adventure. And if you're ever in the South, don't hesitate to look me up!

I'm now getting on the next ride with new hope and a new last name: French. Or, as Drew likes to call me, Nicole Freedom.

I will miss you all!

—Nicole French
CROSSTALK Article and Publishing Coordinator
March 2001-July 2009
whritwuz@juno.com

Introducing Marek

We'd like to welcome our new Article Coordinator, Marek Steed. Be nice to her. And remember: her name is MAREK. <marek.steed.CTR@hill.af.mil>

CrossTalk / 517 SMXS/MXDEA
6022 Fir AVE
BLDG 1238
Hill AFB, UT 84056-5820

PRSRT STD
U.S. POSTAGE PAID
Albuquerque, NM
Permit 737

CrossTalk thanks the above organizations for providing their support.