Cyber Security Policy and Research Institute

THE GEORGE WASHINGTON UNIVERSITY

Workshop to Develop a Building Code and Research Agenda For Medical Device Software Security

Carl Landwehr

January 8, 2015

Report GW-CSPRI-2015-1

Support for this research was provided through a grant from The National Science Foundation NSF CNS 14-52113, Creating a Building Code for Medical Device Software Security, NSF CNS 13-30491, Trustworthy Health and Wellness (THaW), and IEEE Cybersecurity Initiative, http://cybersecurity.ieee.org/

Abstract

An invitational workshop to develop both a draft “building code” that could be applied to improve the security of software operating medical devices and a research agenda to motivate research that could further develop such a building code was held in New Orleans, Louisiana from November 19-21, 2014 in New Orleans, Louisiana. This report documents the background for the workshop, how the workshop was organized and conducted, who participated, and includes as an appendix a draft of the building code developed during the workshop. It also provides guidance on possible uses for the report. The workshop participants were drawn from industry, including device developers and those involved in creating standards and evaluating device conformance, from the computer science, cybersecurity, and software engineering research communities, and from government. The draft code is intended to represent a consensus of the participants, as seen through the eyes of the chair and vice-chair of the workshop with the consultation of the five group leaders. The draft code provides a comprehensive structure with the most detail in the area of methods for limiting the vulnerabilities introduced in the implementation phase of the life cycle. The draft code is expected to be developed further by those involved in the industry. The research agenda incorporates items proposed for the code that the participants felt required further evidence of effectiveness (to be generated through research) before they could be included in the code.

Table of Contents

Workshop to Develop a Building Code and Research Agenda For Medical Device Software Security Final Report

1. Background

The aims of this workshop were (1) to establish an initial consensus among industry and academic participants on the components of a "building code" that would be appropriate to reduce significantly the vulnerability of medical devices to malicious attacks, and (2) to establish a research agenda for the creation of evidence that could justify the inclusion of additional elements in such a code. Support for the workshop was provided by the IEEE Computer Society's Cybersecurity Initiative[1] and the National Science Foundation's Secure and Trustworthy Cyberspace program, both directly[2] and through NSF's Trustworthy Health and Wellness (THaW) project[3]. This document incorporates a draft building code as Appendix A and the research agenda as Appendix B. Additional appendices provide a mapping of the code elements to the NIST Cybersecurity Framework [NIST14], the call for participation, agenda, and the list of participants.

Building codes for physical structures [IBC12] grow out of industry and professional society groups – suppliers, builders and architects – rather than from government, although adoption of codes by government provides the legal basis for enforcement. Building codes generally apply to designs, building processes, and the finished product. Code enforcement relies on inspections of structures during construction and of the finished product and also on certification of the skills of the participants in the design, construction, and inspection processes. Codes also account for different domains of use; code requirements for single-family dwellings differ from those for public buildings, for example. Although building codes arose largely from safety considerations (e.g. reducing the risk of widespread damage to cities from fires, hurricanes, or earthquakes), security from malicious attack has also motivated some aspects of building codes.

Following the ideas expressed in [L13], this workshop aimed to develop an analog to building codes focused on the security properties of software rather than the structure and characteristics of physical buildings. The objective of this code for software security is to increase assurance that software developed for the domain of medical devices[4] will be free of many of the security vulnerabilities that plague software generally. Since evidence to date suggests that a large fraction of exploitable security flaws are not design flaws but rather implementation flaws, an initial building code for medical device software security could focus on assuring that the final software that operates the device

[1] IEEE Cybersecurity Initiative, http://cybersecurity.ieee.org/
[2] NSF CNS 14-52113, Creating a Building Code for Medical Device Software Security
[3] NSF CNS 13-30491, Trustworthy Health and Wellness (THaW)
[4] We use "medical device" in this document in a general sense, with the intention to include mobile health apps as well as implants, monitors, and other healthcare-related devices. Any use of the workshop results in a legal context would of course need to be precise about the covered domain of devices.

is free of certain classes of implementation flaws, although it could address aspects of the development process as well. For example, the code might specify that modules written in a language that permits buffer overflows be subject to particular inspection or testing requirements, while modules written in type-safe languages might require a lesser degree of testing and inspection. The code can in this way help the industry establish a baseline for best practice for security and establish conventions for development as the use of standard materials and dimensions helps architects and builders.

2. Process

The draft building code and research agenda provided in this report are the product of a two-day invitational workshop convened in New Orleans, Louisiana, November 19-21, 2014. Forty people from a wide range of backgrounds including medical device development, standards, and regulation, cybersecurity research, programming languages, and software engineering participated in the workshop. Details on the process used to organize the workshop are provided as Appendix G.

After the workshop, the chair and vice-chair conferred to organize the agreed elements and bring them into a consistent structure. The draft building code resulting from this process was circulated to the workshop discussion group leaders and a portion of the steering committee and then revised in accordance with comments received. The resulting Building Code is provided as Appendix A to this report, and the research agenda is provided as Appendix B.

3. Discussion

The underlying motivation for creating this building code for the security of medical device software is to provide a basis that developers can use to rule out the most commonly exploited classes of software vulnerabilities. To accomplish this, the code elements must be effective and relatively easy to evaluate.

These considerations led the participants to focus on the elements in Category B: Elements intended to avoid / detect / remove specific types of vulnerabilities at the implementation stage. As noted in the draft code, memory safety errors in implementations are a major source of exploitable errors. Selecting a programming language that makes these errors impossible (element B.1) seems clearly desirable for any new effort. Developers who do not choose such a language can employ other options that can be inspected reasonably easily. These options include: subsetting the chosen language (B.2), automated memory safety error mitigation (B.3), and enforcing secure coding standards on the use of the chosen language (B.4), but none of these can provide the same level of assurance as simply using a language that eliminates the possibility of these errors.

Since there are classes of errors unrelated to memory safety, the participants identified several other elements for the draft building code, including: use of accredited cryptography, analysis of code correctness with respect to specifications and comprehensive test coverage. Specifying the critical properties of the software and applying automated analysis tools to assure that the implementation (source or binary) agrees with the specification (B.6) requires a higher level of technical sophistication on the part of the developer but can provide very high assurance of the absence of specified vulnerabilities, and this approach has been successfully applied to substantial bodies of code. Application of the Modified Condition Decision Coverage testing criterion (B.8), although resource intensive for the developer, has proven effective for life-critical avionics systems and it would seem an appropriate technique for life-critical medical devices as well.

The structure used to describe each of the elements in the draft code differs slightly from the template developed for sample elements prior to the workshop. The primary difference is the "references demonstrating effectiveness" section is omitted. One reason for the omission is that building codes for physical structures do not generally include such references; the presence of the item in the code is influenced by informed discussion among professionals and some process for generating consensus among experts. Secondly, the level of evidence cited in the submissions received – the number and types of references or other evidence – varied widely and not necessarily in proportion to the degree of consensus for including particular items in the code. So, while it continues to seem appropriate to the organizers to provide strong evidence of effectiveness of code elements before they are included, evidence of effectiveness has been omitted from the Appendix A element descriptions.

4. How might this report be used?

This report can be used in several ways. There are many standards groups in the medical device domain. As security joins safety, interoperability, and usability concerns as a primary concern of these groups, Appendix A offers a concrete approach to addressing security concerns, particularly in the Section B items that aim to reduce vulnerabilities introduced in the implementation process. Any use of this material will certainly require elaboration, context setting, and many other details to be fleshed out, but an initial skeleton is now available.

The FDA recently issued its nonbinding guidance on managing cybersecurity in medical devices [FDA]. The draft code can be used by developers to help them respond to parts of this guidance.

Other industries such as automotive control, power grid management, and aviation, where security vulnerabilities are an increasing concern might wish to develop building codes relevant to their respective domains. The workshop structure and the draft code may help them organize similar efforts.

5. Conclusion

The work reported here must be viewed as the beginning, not the end, of an effort to create a foundation for building medical devices that are free of the kinds of vulnerabilities that are most commonly exploited. For this work to have real effect, it must be carried forward by those with responsibilities for building and evaluating medical devices and for creating the framework of standards surrounding their development and use. One step forward in relating this work to existing standards and frameworks is the mapping of the categories of code elements used at the workshop to those of the NIST Cybersecurity Framework [NIST14], provided in Appendix C,

6. Acknowledgments

Many people collaborated to create this workshop and produce the results. The chair and vice-chair particularly want to thank the IEEE Cybersecurity Initiative (CSI) and the National Science Foundation (NSF CNS 1329737 (THaW) and NSF CNS 1452113) for providing the funds to support participant travel and expenses. Kathleen Clark-Fisher, Program Director for the IEEE CSI, also provided logistical support during the meeting, and Theresa McNeill of IEEE was instrumental in making the meeting arrangements. Katelyn Anders of the Cyber Security and Policy Research Institute (CSPRI) at George Washington University assisted with participant travel reimbursement. We are grateful to the members of the workshop Steering Committee for their encouragement and advice, and particularly Pat Baird and Ken Hoyme who provided helpful contacts and assisted in recruiting participants as well. The leaders of the group discussions – Brian Fitzgerald, Michelle Jump, Michael McNeil, Roshan Thomas, and Chuck Weinstock -- put forth additional effort during the workshop and assisted in editing the final report. Steve Lipner also provided extensive comments an the final report drafts. Most of all, we thank the workshop participants for their contributions both in advance of the workshop to the website used to develop and coordinate the draft building code and for their active collaboration during the meeting.

7. References

[FDA] U.S. F.D.A., Center for Devices and Radiological Health. Content of Premarket Submissions for Cybersecurity Management. Oct. 2, 2014. Available at: http://www.fda.gov/downloads/MedicalDevices/DeviceRegulationandGuidance/GuidanceDocuments/UCM356190.pdf

[L13] Landwehr, C. E. "A Building Code for Building Code: Putting What We Know Works to Work," Proc. 29th Annual Computer Security Applications Conference (ACSAC), New Orleans LA, ACM, NY, pp.139-147. Available at: http://www.landwehr.org/2013-12-cl-acsac-essay-bc.pdf

[NIST14] National Institute of Standards and Technology. Framework for Improving Critical Infrastructure Cybersecurity, version 1.0, Feb. 12, 2014. http://www.nist.gov/cyberframework/upload/cybersecurity-framework-021214.pdf

Appendices

Appendix A

Draft Building Code for Medical Device Software Security

I. Purpose

This code is intended to provide a basis for reducing the risk that software operating medical devices is vulnerable to malicious attacks. Such attacks might impede or alter the device's function, leak sensitive data, or otherwise impede the device's ability to achieve its medical purpose. In some lexicons, a code begins as a standard and only becomes a code once it is adopted legally and has the force of law behind it. On the other hand, the International Building Code [IBC] was developed as a model code, one that can be tailored for different environments and adopted legally by different jurisdictions. What is developed here is intended as the beginnings of a model code for software security in the same sense.

The aim in specifying such a model code is not to assure that future medical devices can resist every imaginable attack, but rather to establish a consensus among experts in medical devices, cybersecurity, and computer science on a reasonable model code for the industry to apply. Metaphorically, the aim is to specify the needed properties of the bricks used to construct the building, not to specify the architecture of the structure. The reason for focusing on the "bricks" at this time is that it continues to be the case that the majority of vulnerabilities currently exploited in cyberattacks of various sorts are errors in implementation rather than design. By focusing particularly on properties desired of the code as implemented, we hope to be able to constrain implementations so that these "bricks" are sound in that they do not exhibit many of the kinds of vulnerabilities currently exploited.

Proper architecture and design are, of course, critical for the safety, usability, maintainability, and effectiveness of these systems, and responsible developers need to apply sound methodologies to every phase of construction, from requirements through final testing and delivery. There are many other activities underway that aim to establish standards and guidelines in these areas, but few focus on the areas emphasized in this document.

II. Scope and Applicability

This section of the code should specify the domain of devices/systems to which the code is intended to apply. This draft code is intended to apply to software that operates or executes within the context of a broad range of medical devices (see Definitions, below). It was not developed with other software domains (e.g., software controlling automobiles, software in large scale IT systems) in mind, although if software suffers from similar vulnerabilities, those domains might well benefit from a similar set of constraints. The present code focuses primarily on assuring that the software is protected from malicious inputs or errors in cryptographic functions; it does not address many common security functions (e.g., authentication, authorization, auditing), although security logging and whitelisting mechanisms are included. A mature building code may well allow or require variations according to the particular type of device covered and its capabilities and operating environment, but such differences, particularly in terms of analysis of the software developed for the device, would need to be justified. These considerations were beyond the scope of a two-day workshop and are not incorporated here.

III. Governance

This section of the code should specify the organization responsible for establishing and maintaining the code and related procedures. The code elements presented here represent the consensus (seen through the eyes of the chair and vice-chair) of those present at the workshop, those who contributed ideas via the workshop website but who could not participate directly, and those who have drafted and reviewed this report. It is available to any standards group or other body interested in taking up these ideas. There are no plans at present for this group itself to maintain and update the code.

IV. Definitions

This section should define terms used in the code that may otherwise be unclear. There was not time at the workshop to define all the needed terms. The term "medical device" is used loosely in this document and merits a brief discussion. The participants recognized that a "medical device" might be an implant, a wearable device (sometimes called a "health management device"), a bedside device in a hospital, a large-scale diagnostic device such as an MRI system, or a Medical Device Data System (MDDS) that may transfer, store, and display data generated by other medical devices. These cover a huge range of technologies, complexities, environments, and risks. The practical adoption of a code such as the one proposed here should include a more precise specification of the characteristics of the devices to which it is intended to apply. It may be that the code should be tailored to different devices and to environments where the device is used. These considerations could not be addressed in this preliminary version.

V. Procedures

If a code such as this one were adopted by a consortium of developers, a standards body, or a regulatory agency, it would be necessary to specify a number of procedures in relation to the code. For example:

- How will a software component be evaluated to determine if it meets the code? What about a system comprising many components?
- Who will decide if the submitted component satisfies the code? Companies producing or using software might self-certify that their software satisfies the code, but such assertions should be subject to impartial review by an outside group and should be documented consistently.
- How will the code be modified over time? The group administering the code should be able to update it through a voting or consensus process as conditions warrant.
- How will legacy devices be handled?

- How will incorporation of software produced by third parties be handled?

Again, specifying these procedures is beyond the scope of this initial effort but would be needed in practice.

VI. Elements Recommended for Inclusion, by Category

In creating the categorization below, the chair and vice-chair attempted to be comprehensive, although they recognized that the proposed elements fail to address some of the categories. These empty categories are retained to highlight potential unmet needs. It is worth noting that the objective of reducing implementation errors is generally captured in Category B, and that almost half the elements originally proposed for consideration were in that category.

For each element of the code, four subsections are provided:

a. Description: What is the meaning and purpose of this element?
b. Vulnerabilities addressed: What kinds of vulnerabilities will be reduced or eliminated if this element is implemented properly?
c. Developer resources required: What resources will the individual or organization developing the software/device require in order to satisfy this element?
d. Evaluator resources required: What is required for a third party to assess whether the device satisfies this element?

A. Elements intended to remove / avoid flaws at the design stage

1. **Secure Random Numbers**
    a. Description: Generating random numbers for use in initializing pseudorandom number generators and cryptographic algorithms, using them correctly, and avoiding re-using them are challenging problems. Mistakes can nullify cryptographic mechanisms no matter how well designed. As advised in [IEEE14], developers should adopt established approaches that have been vetted by experts in the field rather than attempting novel solutions.
    b. Vulnerabilities addressed: use of non-random seeds can render cryptographic mechanisms ineffective and permit attackers to spoof, tamper, or disclose sensitive data
    c. Developer resources required: access to vetted procedures for random number generation; may be platform-dependent
    d. Evaluator resources required: manual review of design and code or attested use of vetted sources of randomness.

2. **Full Recognition of Inputs Before Processing**
    a. Description: Designers should consider the grammar of the language of inputs and use the most restrictive grammar consistent with required component functions. They should then be sure that inputs are checked for conformance to that grammar before processing those inputs. A component that accepts an input without checking its validity presents a path that an attacker can probe.
    b. Vulnerabilities addressed: Exploitation of input-handling code by maliciously crafted inputs

   c. Developer resources required: Specification of input language, program source code, and software framework for generating recognizer for input language.
   d. Evaluator resources required: Requires audit of software and of its data language definitions for adherence to the design principle. Audit must identify the code that checks and handles inputs immediately upon receipt, and evaluate whether the checking code is (a) complete as a recognizer for a given definition of valid and expected data (b) isolated from other functionality. Automation might assist the review with appropriate constraints on specification and implementation languages and procedures.

3. **Least Operating System Privilege**
   a. Description: The least-privilege principle calls for the operating system to grant programs/processes only those privileges required for them to carry out their specified functions [SS75]. Programs requiring root or administrator privileges should use fine-grained operating system level privileges when available. Additionally, for those systems that allow enabling/disabling of privileges (e.g., effective UIDs, effective and maximum privilege/capability sets, etc.), privileges should be enabled only for those system calls needing them. Privileges should be removed when no longer needed. Programs should be designed so that the number of privileges needed and the amount of time those privileges are needed is minimized.
   b. Vulnerabilities addressed: exploitation of over-privileged processes.
   c. Developer resources required: Designers need to keep the principle in mind as they organize the system components. Implementers need to abide by the constrained design and avoid granting privileges in the implementation not called for in the design. One approach is to list the privileges available, list the entities that have that privilege, and provide a rationale as to why the entity requires the privilege. Testing the system as an ordinary user rather than as a system administrator is also helpful.
   d. Evaluator resources required: Automated static analysis can help reveal whether privileges are enabled only where specified. Manual analysis is required to determine if the design adheres to the principle.

B. Elements intended to avoid / detect / remove specific types of vulnerabilities at the implementation stage. This section opens with several items related to programming language selection, use, and analysis (B.1 – B.6) and then proceeds to other topics (B.7-B.9).

Several building code elements recommended for adoption aim to reduce vulnerabilities by controlling the selection and use of programming language. Languages such as C and C++ are widely used because they provide programmers with flexibility, efficiency, and compatibility with large bodies of legacy software. They also permit programmers to easily make mistakes that are hard to find and provide attackers with exploitation opportunities. In a way, basing a program on this kind of language is like constructing a building from flammable materials: the structure may be inexpensive and provide shelter for a while, but it may be easy for an arsonist to sabotage it. Of particular interest is the loosely defined property of "memory safety" which is defined informally [SPWS13] as the prevention of memory access errors of the following kinds:

- buffer overflow
- null pointer dereference
- use after free
- use of uninitialized memory
- illegal free (of an already-freed pointer, or a non-malloced pointer)

Elements B.1 – B.4 represent alternative and, roughly speaking, successively weaker approaches from a security standpoint to dealing with memory safety vulnerabilities. B.5 provides an approach to assuring thread safety. B.6 covers the use of a wide range of tools from static analyzers to SAT solvers and theorem provers that can provide very strong assurance of specified properties. These tools are increasingly capable but still limited in the size of code base they can process. The use of approved cryptography (B.7) involves both algorithm and implementation so may be considered a design requirement. B.8, like B.6, provides relatively strong assurance but is unlikely to be feasible for large code bases.

1. Use of Memory-Safe Languages

a. Description: Some programming languages are designed to prevent memory safety errors listed above. Selecting such a language effectively rules out these large classes of vulnerabilities.
b. Vulnerabilities addressed: memory safety vulnerabilities including buffer overflow, null pointer dereference, use after free, use of uninitialized memory, and illegal free.
c. Developer resources required: requires programmers trained in the selected language, compilers and run-time libraries for the language.
d. Evaluator resources required: ability to recompile the source code using a compiler for the memory-safe language (in order to confirm that the object modules have been produced as claimed).

2. Language Subsetting

a. Description: To reduce the possibility that known exploitable language constructs will occur in programs, the developer restricts implementers to a use only a subset of language features or constructs, avoiding those known to be risky or ambiguous. Use of a restricted subset of a language may also improve performance of static analysis tools on the software. Subsets of several languages, including C (MISRA C), Ada (SPARK Ada), are available.
b. Vulnerabilities addressed: memory access and other implementation errors.
c. Developer resources required: programmers trained in use of the subset, code scanners to enforce subset constraints.

   d. Evaluator resources required: access to source code and scanning tool to confirm programs abide by subset constraints.

3. **Automated Memory Safety Error Mitigation and Compiler-Enforced Buffer Overflow Elimination**
   a. Description: For software written in non-memory-safe languages (e.g., C/C++), use compiler transforms that enforce memory safety (e.g., SAFECode, WIT, Baggy Bounds Checking, SoftBound). Develop policy on what to do when a run-time error is detected (e.g., reset device and produce an audit event)
   b. Vulnerabilities addressed: memory safety errors
   c. Developer resources required: access to software checking tools, source code
   d. Evaluator resources required: Ability to re-run tools used by developer on the source/binary; requires confirming that an appropriate compiler has compiled all of the software with the instrumentation enabled. Alternatively, binary scanning tools may be used to confirm the code has been transformed.

4. **Use of Secure Coding Standards**
   a. Description: To reduce the possibility of exploitable vulnerabilities in languages susceptible to memory access errors, but without restricting programmers to a subset language, adherence to standard usages of the language structures is required. Using the standard can reduce the possibility of memory safety and other exploitable errors substantially. Secure coding standards are available for C, C++, and Java.
   b. Vulnerabilities addressed: memory safety and other implementation errors.
   c. Developer resources required: programmers trained in use of the coding standard, software to check programs produced for conformance to the standard.
   d. Evaluator resources required: source code, automated checker for conformance to standards. If conformance cannot be mechanically checked, manual auditing may be required.

5. **Automated Thread Safety Analysis**
   a. Description: Multi-threaded code is annotated by the developer to declare desired thread safety properties. Tool (compiler option) assures that the policies are enforced.
   b. Vulnerabilities addressed: race conditions and deadlocks
   c. Developer resources required: Programmers capable of developing correct annotations, access to compiler capable of processing them.
   d. Evaluator resources required: Ability to determine that appropriate annotations have been made (manual) and that all the software was processed with the appropriate compiler and options (or re-compile it); ability to review the specified safety policies.

6. **Automated Analysis of Programs (Source/Binary) for Critical Properties**
   a. Description: Critical properties desired of a binary (or source) program are specified precisely. The subject program is then analyzed against a model embodying the semantics of the (hardware/software) execution environment to verify that the desired properties are present.
   b. Vulnerabilities addressed: the approach can address any vulnerability that can be specified.
   c. Developer resources required: Requires developers capable of specifying the properties desired of the implementation in the language accepted by the verification tools involved, or access to experts with this ability and associated automated tools.
   d. Evaluator resources required: Ability to generate and review output of verification tools applied to the programs analyzed

7. **Accredited Cryptographic Algorithms and Implementation**
   a. Description: Cryptographic algorithms that resist serious analysis are notoriously difficult to invent and program. Weaknesses also surface in key management and surrounding protocols. Developers should seek algorithms and tools that have received some external, open certification (e.g., from NIST or other organizations) rather than attempting to develop their own. If this is not feasible for some reason solutions should be expertly reviewed.
   b. Vulnerabilities addressed: weaknesses in cryptographic algorithms and implementations
   c. Developer resources required: Access to expertly vetted cryptographic algorithms and implementations
   d. Evaluator resources required: Ability to audit software for use of vetted cryptography or to automatically verify implemented cryptography against vetted specification

8. **Modified Condition Decision Coverage**
   a. Description: This is a criterion for test coverage that has been successfully applied to life-critical avionics software for many years and is part of standards for automotive, rail, and process control systems. It requires a specification of system behavior and that testing against that specification achieves the following coverage:
      - Each entry and exit point is invoked at least once
      - Each decision has taken each possible outcome at least once
      - Each condition in a decision takes on every possible outcome at least once
      - Each condition shown to independently affect the outcome of the decision.

      This test coverage criterion subsumes statement and branch coverage, requires k+1 tests for k conditions, and ensures t-way combination coverage of at least (1+t)/2t. The required testing may not be feasible for large code bases; it is appropriate for life-critical medical software.

   b. Vulnerabilities addressed: this is a general tool for assuring implemented software performs as designed. It is not targeted at detecting specific vulnerabilities but has proven effective for assuring safety in many life-critical systems.
   c. Developer resources required: Requires system specification at the level of detail that it can be used to validate test results. The coverage criterion demands extensive testing of the software.
   d. Evaluator resources required: Resources to review system specification and test results (some automation should be possible to see that test and specification match).

9. **Operational Use Case Identification and Removal of Unused Functions**
   a. Description: Use cases for the device are specified and software components required by each use case are identified. Software not required by any use case is considered for removal / disablement to eliminate / reduce the possibility of attacks exploiting unneeded software. This element is probably most effective at a relatively high level of abstraction to be sure that unused libraries, collections of functions, and applications are eliminated rather than at a detailed, line-by-line code level.
   b. Vulnerabilities addressed: software vulnerabilities located in unused components
   c. Developer resources required: Requires identification of complete set of use cases (sometimes difficult in practice) and ability to track use case back to software required for it.
   d. Evaluator resources required: Manual review of software components present against use cases specified.

C. Elements intended to assure software / firmware provenance and integrity, but not to remove flaws in code. The mechanisms for C.1 and C.2 are essentially the same; they are listed as separate elements to emphasize that not only the initial distribution / hardware integrity needs to be assured but also the integrity of subsequent updates that are virtually certain to be required.

1. **Digitally Signed Firmware and Provenance (supply chain)**
   a. Description: Developer/integrator affixes digital signature to software/firmware installed in device. In case of subsequent device malfunction or compromise, the digital signature of the software present at the time of failure can be recomputed and compared with the signature of the distributed version to detect tampering.
   b. Vulnerabilities addressed: this element addresses software provenance, helping establish accountability for fielded software. It does not aim to eliminate vulnerabilities in the software/firmware.
   c. Developer resources required: Developer (or third party) needs a private signing key, needs to protect that key, and needs to make the corresponding public key available for checking.

   d. Evaluator resources required: Evaluator needs to assure integrity of signing mechanisms and operational mechanisms for signature verification.

2. **Software/firmware Update Validation**
   a. Description: The aim is to enable valid updates to operational software while minimizing the possibility that the update mechanisms can be subverted to install fraudulent updates. The method is to require an encrypted checksum (digital signature) on the updated software and to validate this checksum using the developer's public key.
   b. Vulnerabilities addressed: installation of fraudulent software updates, loss of accountability to system producer.
   c. Developer resources required: Developer (or third party) needs a private signing key, needs to protect that key, and needs to compute and needs to make the corresponding public key available.
   d. Evaluator resources required: Evaluator needs to assure integrity of signing mechanisms and operational mechanisms for signature verification.

3. **Whitelisting**
   a. Description: The aim is to avoid execution of untrustworthy, possibly malicious, applications. Prior to execution of application software, the software is checked against a list of authorized applications (the whitelist). Entering new applications in the whitelist is a privileged operation, not under operator control.
   b. Vulnerabilities addressed: execution of unvetted application software. The mechanism does nothing to remove vulnerabilities from applications; however, by assuring that the application to be executed is included on the whitelist it provides a significant barrier to the execution of malware even after a successful penetration [in this sense it could also be placed in category D below].
   c. Developer resources required: whitelisting mechanism and attendant software to permit privileged users to update the whitelist needs to be incorporated in system design and implementation.
   d. Evaluator resources required: Manual review of whitelisting mechanism specification and implementation (if the mechanisms are specified formally, automated assistance is possible).

D. Elements intended to impede attacker analysis or exploitation but not necessarily remove flaws

1. **Non-executable Data Pages**
   a. Description: Storage is divided into code segments that may be read or executed but not written and data segments that may be read or written but not executed. Temporary storage (stacks, heaps, global variable) is assigned to data segments and so cannot be used by attackers to execute instructions.

b. Vulnerabilities addressed: many attacks depend on inserting new code into the system stack or other writeable storage locations (data pages). Preventing the execution of instructions located on writeable pages thwarts these kinds of attacks, though they may still achieve a denial of service when the attempted execution fails. It does not prevent attackers from using ROP (return oriented programming) attacks in which the attack relies on executing instructions already present on executable pages.
c. Developer resources required: Developer needs to organize code to take advantage of this structure and hardware mechanisms supporting it. Note that JIT (just-in-time) compilation and other mechanisms designed to develop and install code during operation will pose problems.
d. Evaluator resources required: Evaluator needs to review the use of mechanisms for assigning code and data to storage segments.

2. **Anti-tamper for Hardcoded Secrets / Keys / Data Within Medical Device Software**
    a. Description: Employ appropriate software/hardware protections against malicious modification of medical device secrets by the possessor of the device. Solutions relying solely on software ("white box cryptography") and solutions that exploit widely available hardware (Trusted Platform Modules (TPMs) with supporting software) are available.
    b. Vulnerabilities addressed: Unauthorized access to, or deliberate modification of, application generated and/or managed data by malicious possessor of device.
    c. Developer resources required: Access to appropriate software/hardware packages and expertise to apply them correctly.
    d. Evaluator resources required: Manual review of application of the selected mechanisms; potentially red-team testing to evaluate overall effectiveness.

E. Elements intended to enable detection / attribution of attack

1. **Security Event Logging**
    a. Description: Provide a tamper-resistant audit trail for security-related events, such as installation of software, authentication of a user, detection of attacks that are mitigated by security mechanisms, etc.)
    b. Vulnerabilities addressed: this element addresses accountability by providing an after-the-fact trail for forensic analysis.
    c. Developer resources required: Requires identification of security related event types and implementation of append-only log of security related events (e.g., authentications, privilege level changes, software updates), strongly resistant to tampering.
    d. Evaluator resources required: Manual review of identified security related event types and of design and implementation of logging mechanisms and security event generation mechanisms. [Note that operator resources to review the logs are also required; tools for log analysis may be needed.]

F. Elements intended to assist in safe degradation of function in face of attack

[None proposed]

G. Elements intended to assist in restoration of function after attack

[None proposed]

H. Elements intended to support maintenance of operational software without loss of integrity

[See Software Update Validation, Element C.2]

I. Elements intended to support privacy requirements

X. Characteristics desired of the building code itself (e.g. use of standard names, maintenance of the building code over time, scope)

- Use within the code itself of standard names for types of attacks / attack patterns and types of vulnerabilities

References

[IBC12] International Building Code. http://publicecodes.cyberregs.com/icod/ibc/

[IEEE14] IEEE Center for Secure Design. Avoiding the Top Ten Security Flaws. http://cybersecurity.ieee.org/images/files/images/pdf/CybersecurityInitiative-online.pdf

[SS75] Saltzer, J.H. and M. D. Schroeder. The protection of information in computer systems. Proc. IEEE., 63, 9 (Sept. 1975) 1278-1308.

[SPWS13] Szekeres, L., M. Payer, Tao Wei; D. Song. SoK: eternal war in memory. Proc. 2013 IEEE Symp. Security & Privacy, 48-62. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?reload=true&arnumber=6547101

Appendix B
Research Agenda for Medical Device Software Security

Several of the elements proposed for the building code were found not to be ready for inclusion in the code for a variety of reasons. Some were considered to be potentially valuable but not ready for practical use; others were specified a bit too abstractly for practical evaluation without substantial human effort and interpretation. Others were felt to require additional research to show their effectiveness. These elements are listed and described below. The order of the items is not significant.

1. **Assurance case research**
   A. Using OMG structured assurance case metamodel (SACM) based tooling.
   SACM provides a general method of stating and analyzing security claims and for exchanging these claims among vendors and providers. This sort of technique could have great value as medical devices become more complex and more highly connected. However, there is not yet a convincing body of evidence that the current SACM approach will prove effective for medical devices. Research must be performed to apply SACM and related tools to medical devices and then to measure the effectiveness of the tools and techniques. Questions to answer include:
   - Which sorts of assurance properties for medical devices can be established more easily using the tools and techniques, and what sort of cost reduction can be achieved?
   - How much do the tools and techniques reduce ambiguity and confusion when exchanging security claims among multiple parties?
   - How much safer and more secure are devices when developers and analysts use the tools and techniques?
   - How much faster can faults be fixed and the device be recertified by using the tools and techniques?

   It may be useful to apply this kind of analysis to the building code itself to assess whether the application of the code yields the desired security properties.
   B. Using eliminative arguments
   Analysts who use this technique try to increase the confidence in a security assertion by posing counter-examples and then presenting evidence that eliminates as many counter-examples as possible. When a counter-example cannot be eliminated completely, the evidence can provide bounds on the potential impact of the counter-example. While assurance cases have been used successfully in the safety domain, they have not been used as much in the security domain. Also, the strength of any eliminative argument depends on the completeness of the set of posited counter-examples. No work has been done to identify security-related counter-examples for medical devices.

2. **Minimization of computational power exposed to inputs**
   While it seems intuitive that adopting a formal, language theoretic approach to analyzing and limiting the computational power associated with device inputs, there

is no strong evidence confirming this intuition. Moreover, there are not yet techniques and tools that developers can use to limit inputs or that evaluators can use to assess size of the gap between what computational power the inputs must provide and the power they actually provide.

3. **Protection of critical state data**
   An attacker who gains access to critical state data can wreak havoc in many ways, such as collecting or modifying data, changing program execution, or even seizing complete control of the device. Although workshop participants suggested several forms of protection that might be applicable (encryption, code obfuscation, oblivious computing), there was no consensus that any of these techniques would actually work in the context of medical devices.

4. **Risky module identification**
   Software engineering research has produced techniques to identify error-prone software modules based on problem reports and software development records. It is possible these or similar techniques will be useful in assuring classes of security vulnerabilities are absent in medical device software, but the evidence remains to be generated.

5. **Runtime detection of code tampering via anti-tamper / anti- corruption mitigation techniques**
   Digital signatures can provide assurance that code that is loaded has not been altered but after the signature has been checked and the code is executing, it may be subject to attack. Some hardware/software mechanisms have been devised to continue to check software while it is running; application of such mechanisms to medical device software is a topic for study.

6. **Trusted computing base for medical devices**
   The notion of a Trusted Computing Base (TCB), comprising the hardware and software in system that is responsible for enforcing the security policy, is a well-established. The challenge is to build a system of significant size in which the TCB is kept small. In many systems today, the TCB is the entire system – an exploitable vulnerability anywhere in the system can lead to a compromise of the security policy. It may be possible to develop a TCB for platforms used in medical devices that would support the kinds of security policies these devices require, but little if any research has been performed on this topic as yet. Different device classes may require different TCBs. In principle, it should be possible for vendors and evaluators to agree on a common TCB that could be tailored for different device classes.

7. **Notations that expose cyber mitigations (like insulation diagrams)**
   In designing physical buildings, different types of diagrams are generated. Some show the physical dimensions and composition of walls and foundations, for example. Others show plumbing, wiring, and heating/ventilation functions. Similarly a logical circuit diagram may illustrate the logical paths for signals and data in a computational component without showing the physical routing and the insulation

along paths. To carry out a proper failure analysis for a circuit board, both the logical and physical diagrams are needed. Are there any similarly useful diagrams in the context of software that might reveal, for example, what effects breaking through one security barrier might be? Are there realistic diagrams that can convey the true "depth" of a set of defenses?

8. **Compiler-based integer overflow protection.**
   Techniques such as "As-if Infinite Range" integer models (specifically for C and C++ languages) have been developed and prototyped but have not been incorporated in production compilers as yet. Some some safe integer computation libraries are available for use with existing compilers.

Appendix C

Mapping Between BCforMDSS Categories and Code Elements to NIST Cybersecurity Framework

NIST Framework for Improving Critical Infrastructure Cybersecurity

<http://www.nist.gov/cyberframework/upload/cybersecurity-framework-021214.pdf>

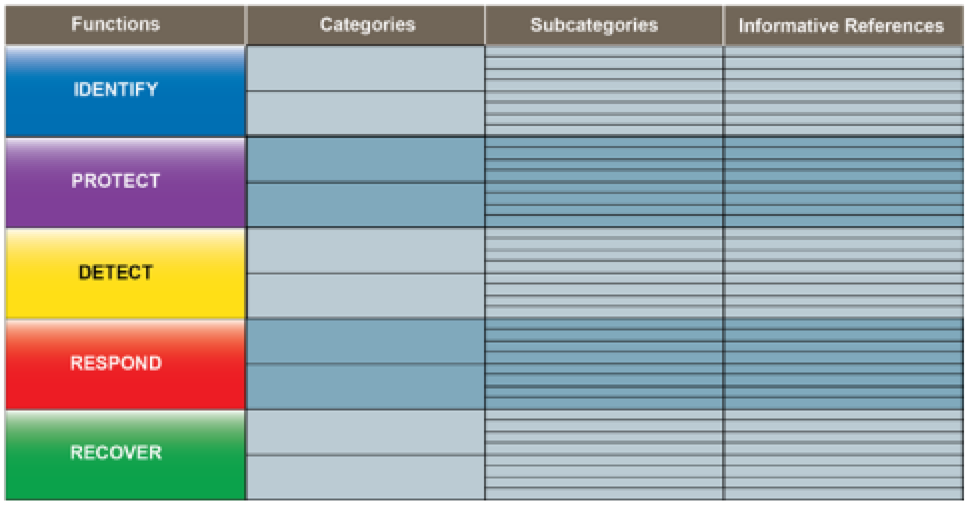

Figure 1: Framework Core Structure

On Feb 12, 2014, National Institute of Standards and Technology (NIST), released a report entitled Framework for Improving Critical Infrastructure Cybersecurity in response to Executive Order 13636 (EO), "Improving Critical Infrastructure Cybersecurity," issued by President Obama a year earlier. E.O. 13636 called for the development of a voluntary Cybersecurity Framework to provide "a prioritized, flexible, repeatable, performance-based, and cost-effective approach" to manage cybersecurity risks. While this report was not prepared specifically for the medical device industry, the FDA hosted a NIST presentation of it at their Cybersecurity Public Workshop, "Collaborative Approaches for Medical Device and Healthcare Cybersecurity," October 21-22, 2014. In addition, the FDA's guidance document on Content of Premarket Submissions for Managment of Cybersecurity in Medical Devices, issued October 2, 2014, is built around the same key principles of Identify, Protect, Detect, Respond, and Recover.

The Building Code Workshop focused on the Protect category but generated some elements that fall in other categories as well. Provided below are mappings from the NIST categories to the proposed Building Code elements and vice versa.

NIST Category Definitions

• **Identify** – Develop the organizational understanding to manage cybersecurity risk to systems, assets, data, and capabilities. The activities in the Identify Function are foundational for effective use of the Framework. Understanding the business context, the resources that support critical functions and the related cybersecurity risks enables an

organization to focus and prioritize its efforts, consistent with its risk management strategy and business needs.
Examples of outcome Categories within this Function include: Asset Management; Business Environment; Governance; Risk Assessment; and Risk Management Strategy.
• **Protect** – Develop and implement the appropriate safeguards to ensure delivery of critical infrastructure services. The Protect Function supports the ability to limit or contain the impact of a potential cybersecurity event.
Examples of outcome Categories within this Function include: Access Control; Awareness and Training; Data Security; Information Protection Processes and Procedures; Maintenance; and Protective Technology.
• **Detect** – Develop and implement the appropriate activities to identify the occurrence of a cybersecurity event. The Detect Function enables timely discovery of cybersecurity events.
Examples of outcome Categories within this Function include: Anomalies and Events; Security Continuous Monitoring; and Detection Processes.
• **Respond** – Develop and implement the appropriate activities to take action regarding a detected cybersecurity event. The Respond Function supports the ability to contain the impact of a potential cybersecurity event.
Examples of outcome Categories within this Function include: Response Planning; Communications; Analysis; Mitigation; and Improvements.
• **Recover** – Develop and implement the appropriate activities to maintain plans for resilience and to restore any capabilities or services that were impaired due to a cybersecurity event. The Recover Function supports timely recovery to normal operations to reduce the impact from a cybersecurity event.
Examples of outcome Categories within this Function include: Recovery Planning; Improvements; and Communications.

Map from NIST to BCMDSS

NIST Framework Categories	Items Proposed for BCMDSS
Identify	
Protect	A.1-3. B.1-9, C.1-3, D.1-2
Detect	C.1-3, E.1
Respond	
Recover	

Map from BCMDSS to NIST

BCMDSS Categories	NIST Framework Categories	Items Proposed for BCMDSS
A. Items intended to remove / avoid flaws at the design stage	Protect	A.1, A.2, A.3
B. Items intended to avoid / detect / remove specific types of vulnerabilities at the implementation stage	Detect and Protect	B.1-B.9
C. Items intended to assure software / firmware provenance but not necessarily remove flaws in code	Detect and Protect	C.1, C.2, C.3
D. Items intended to impede attacker analysis or exploitation but not necessarily remove flaws	Protect	D.1, D.2
E. Items intended to enable detection / attribution of attack	Detect	E.1
F. Items intended to assist in safe degradation of function in face of attack	Respond	
G. Items intended to assist in restoration of function after attack	Recover	
H. Items intended to support maintenance of operational software without loss of integrity	Protect	C.2
I. Items intended to support privacy requirements	Protect	
X. Characteristics desired of the building code itself (e.g. std use of names, maintenance of the building code over time, scope)		

Appendix D

Call For Participation

Call for Contributions and Participation
NSF/IEEE-CS Invitational Workshop to
Create A Building Code for Medical Device Software Security
November 19-21 2014
Hyatt French Quarter, New Orleans

1. Purpose

The aim of this workshop is (1) to establish an initial consensus among industry and academic participants on the appropriate components of a "building code" that would be appropriate to reduce significantly the vulnerability of medical devices to malicious attacks, and (2) to establish a research agenda for the creation of evidence that could justify the inclusion of additional elements in such a code. The workshop will be held under the auspices of the IEEE Computer Society's Cybersecurity Initiative, with participation from NSF's Trustworthy Health and Wellness (THaW) project; additional support is being sought from the NSF Secure and Trustworthy Cyberspace program.

2. What might a building code for medical device software security look like?

Building codes applied to physical structures generally grow out of industry and professional society groups – suppliers, builders and architects – rather than from government, although adoption of codes by government provides a legal basis for enforcement. Building codes generally apply to designs, building processes, and the finished product. Code enforcement relies on inspections of structures during construction and of the finished product and also on certification of the skills of the participants in the design, construction, and inspection processes. Codes also take account of different domains of use of structures: code requirements for single-family dwellings differ from those for public buildings, for example.

Following the ideas expressed in [1] we aim to develop an analog to these processes that will improve assurance that software developed for the domain of medical devices will be free of many of the security vulnerabilities that plague software generally. Evidence to date is that a large fraction of exploitable security flaws are not design flaws but rather implementation flaws. An initial building code for medical device software security could focus on assuring that the final software that operates the device is free of these kinds of flaws, although it could address aspects of the development process as well. For example, the code might specify that modules written in a language that permits buffer overflows be subject to particular inspection or testing requirements, while modules written in type-safe languages might require a lesser degree of testing but a stronger inspection of components that translate the source language to executable form.

3. Considerations for including a particular requirement in a building code for medical device security
1. Effectiveness. The first criterion should be the ability of the required item to reduce the vulnerability of software to exploitation. Specific evidence should be available to support claims of effectiveness.
2. Ease of evaluation. Requirements that are effective but require unusual expertise, time, or other resources to evaluate are not appropriate for inclusion in the code.
3. Scope. Requirements affecting only a narrow scope of vulnerabilities may not be appropriate to incorporate.

4. Participants sought
To succeed, the workshop needs participation from (1) industry personnel familiar with the architecture and tools used to build medical devices and the software that controls them, (2) people familiar with the history of medical device regulation in general and people familiar with the history of computer security regulation, (3) researchers and practitioners familiar with cybersecurity issues generally and with security issues in medical device software in particular, and (4) experts in relevant aspects of software engineering, including requirements, design, and (especially) implementation, test, and validation/verification methods.

5. Workshop organization and products
The meeting will be organized as two-day event with approximately 40 invited participants, starting in the evening of the first day (Nov. 19) and ending in the afternoon two days later (Nov 21). The meeting will open with a dinner session accompanied by an invited talk or panel on the history and current state of official guidance on the security of medical device software. The next day will open with a general talk on the concept of a building code for security-critical software that will address the types of requirements a building code might include and the possible basis for deciding whether a particular element should be included in the code. Following this introduction, a series of short talks proposing possible elements of the code will be presented, based on submissions received in advance of the meeting.

In afternoon breakout sessions, the participants will be asked to discuss the proposed elements and to assess the strength of the evidence for including each proposed item in the code. When the group consensus is that stronger evidence is needed, research topics that might help establish that evidentiary basis will be identified. At the end of the afternoon, groups will report on their progress in a brief plenary session.

The breakout sessions will reconvene on the final morning of the workshop to consider the results of the plenary session and any evening discussions. The meeting will close with a two-hour plenary session in which consensus on an initial building code and research agenda will be sought.

Following the meeting, the chair and vice-chair, in consultation with the workshop participants and Steering Committee will develop a report on the workshop documenting the initial draft building code and research agenda. The report be placed on a George Washington University website and will be published by the IEEE as well.

6. Where to send your contribution / request for invitation
If you are interested in participating, please send a note of not more than 600 words explaining (A) which of the four groups listed above you would represent and (2) at least one requirement you think would be appropriate to discuss at the workshop as a candidate for an initial building code, as well as evidence supporting the effectiveness of that requirement. If you are interested in the workshop but don't have a specific element to propose, please include a description of your role in medical device software development or in assuring software security. E-mail this information to: **BCforMDSS@gmail.com**

7. Travel Support
Support for those requiring reimbursement of travel, lodging, and meal expenses is expected to be available from the workshop sponsors.

8. Reference
1. Landwehr, C. E. "A Building Code for Building Code: Putting What We Know Works to Work," Proc. 29th Annual Computer Security Applications Conference (ACSAC), New Orleans LA., ACM, NY, pp.139-147. Available at: http://www.landwehr.org/2013-12-cl-acsac-essay-bc.pdf

Cyber Security Policy and Research Institute
THE GEORGE WASHINGTON UNIVERSITY

NSF

IEEE CYBER SECURITY

Appendix E

Workshop to Develop a Building Code for Medical Device Software Security

Workshop Agenda Nov. 19-21, 2014

Time	*Event*	*Participants*	*Location*
Wed. 11/19			
6:00 PM	Arrival	All workshop participants	Hyatt Regency French Quarter, New Orleans
6:30 PM	Registration / Pre-dinner reception	All workshop participants	
7:00 PM	Dinner	All workshop participants	
8:30 PM	Talk 1: History of medical device software and approaches to regulation	Speaker: Anura Fernando, UL	
9:15 PM	Talk 2: History of software security and approaches to regulation	Speaker: Steve Lipner, Microsoft	Atrium
10:00 PM	Adjourn		
Thurs. 11/20			
7:00am	Breakfast available	All participants	Holmes Foyer
8:00 AM	Introduction of the participants and outline of the workshop	Carl Landwehr	Holmes BC
8:45 AM	Proposed building code element talks	Category B items	
	Automated Memory Safety Error Mitigation	John Criswell	10 min.
	Automated Analysis of Binaries	Marijn Heule / Warren Hunt	10 min.
	Language Subsetting	Paul Anderson	10 min.
	Operational Use Case Documentation to Eliminate Unused Functions	Scott Erven	10 min.
	Modified Condition Decision Coverage	Rick Kuhn	10 min.
	Several language and analysis-related items	Robert Seacord	25 min.
10:00 am	Morning Break		
10:30 am	Talks continue	Category A, C, D, etc. items	
	Full recognition of inputs / minimization of computation power exposed to inputs	Sergey Bratus	15 min.
	Assurance Cases using OMB Structured Assurance Case Metamodel based tooling	Robert Martin	10 min.
	Security Assurance Cases Using Eliminative Arguments	Chuck Weinstock	10 min.
	Design stage threat modeling / attack surface reduction	Scott Erven	10 min.
	Non executable data pages	Jeremy Epstein	10 min.
	Protection of critical state data	Sol Greenspan	10 min.
	3 anti-tamper, obfuscation related items	Jonathan Carter	15 min.
12:00 noon	Lunch		Holmes Foyer

1:00 pm	Breakout groups convene	maximum 8 per group	Holmes A, B, C (separated) plus Dauphine A&B
3:00 pm	Afternoon Break		
3:15 pm	Breakouts continue		
4:00	Breakouts conclude; reconfigure space for plenary		
4:10 pm	Plenary presentations / discussion	Breakout group leads report on conclusions, new proposals	Holmes BC
5:10 pm	Discussion period if needed		
5:30 pm	Adjourn		
	Group leads / steering	Coordination meeting	Dauphine A
6:30 pm	Reception		Batch
7:00 pm	Dinner		Redfish Grill
Fri. 11/21			
7:00 am	Breakfast available		Orleans Foyer
9:00 AM	Plenary discussion		Holmes BC
9:30 am	Breakouts re-convene		As before
10:00 am	Morning break, check out		
10:30 am	Breakouts re-convene		
11:20	Breakouts conclude; reconfigure for plenary		
11:30 am	Closing plenary		Holmes BC
12:30 pm	Box lunches and departure		

Appendix F

List of Participants

Homa Alemzadeh, *University of Illinois, Urbana-Champaign*
Paul Anderson, *Grammatech*
Sergey Bratus, *Dartmouth College*
Tom Brennan, *Proactive Risk, OWasp*
Ian Bryant, *UK Trustworthy Software Initiative, University of Warwick*
Blaine Burnham, *University of Sourthern California*
Jonathan Carter, *Arxan Technologies, OWasp*
John Criswell, *University of Rochester*
Jeremy Epstein*, U.S. National Science Foundation*
Scott Erven, *Protiviti*
Dale Fay, *Univ. of Michigan Radiology*
Anura Fernando, *UL*
Brian Fitzgerald, *U.S Food and Drug Administration,* ***Discussion Group Leader***
Ken Fuchs, *Center for Medical Interoperability*
Christopher Gates, *Illuminati Engineering*
Sol Greenspan, *U.S. National Science Foundation*
Tom Haigh, *Adventium (ret.),* ***Vice-Chair***
Marijn Heule, *University of Texas - Austin*
Warren Hunt, *University of Texas - Austin*
James Jacobson, *Siemens Healthcare Diagnostics*
Michelle Jump, *Stryker Corp.,* ***Discussion Group Leader***
Chandu Ketcar, *Cigital*
D. Richard Kuhn, *U.S. National Institute of Standards and Technology*
Carl Landwehr, *George Washington University,* ***Chair***
Steve Lipner, *Microsoft*
Dan Lyon, *Cigital*
Robert Martin, *MITRE*
James McDonald, *Kestrel Institute*
Michael McNeil, *Philips,* ***Discussion Group Leader***
Steve Myers, *Indiana University*
Nathanael Paul, *University of South Florida*
Eric Petersen, *Welch Allyn*
Stephanie Preston, *Battelle*
Robert Seacord, *Carnegie Mellon University - SEI/CERT*
Shahid Shah, *Netspective*
Tucker Taft, *Adacore*
Roshan Thomas, *MITRE,* ***Discussion Group Leader***
Eugene Vasserman, *Kansas State University*
Sam Weber, *Carnegie Mellon University - SEI/CERT*
Chuck Weinstock, *Carnegie Mellon University - SEI,* ***Discussion Group Leader***

Appendix G

Process

In advance of the workshop, participants were invited to propose elements for inclusion in the building code. A public website was created to host materials generated by the organizers and submitted by the participants. It was also possible for interested individuals who were unable to attend the workshop to submit proposed elements. A collective e-mail address was also established for the workshop participants shortly before the meeting. By the time of the meeting, 35 specific elements had been placed on the website as proposed code elements.

The workshop itself opened with two general talks, one from an expert in the history of medical device history and regulation and one from an expert in the history of computer security technology and regulation, as a way to help participants coming from different fields establish a common basis for discussion.

The goal of the workshop was to enable presentation and discussion of each submitted element by the 40 participants and lead to a consensus set of elements for the building code. The original plan was to allow the proposer of each element to present it to the entire group and then break into smaller groups (maximum of 8 in a group) to allow parallel discussions of all proposed elements and let each group establish a consensus on which elements belonged in the code and which required further research to justify their inclusion.

With 35 proposed elements and a workshop length of less than 48 hours, it was infeasible to present all the elements in plenary session and retain the time needed for discussion. To focus the presentations on the items that seemed most likely to be contentious, the chair and vice-chair identified four elements that seemed very likely to be included in the code and seven elements that seemed relatively unlikely to be included or lacked individuals to present them. The 24 remaining elements were grouped for presentation using the set of categories found in Appendix A, Section VI. Following the presentations, the participants divided into five parallel discussion groups to review the proposed elements (including those not presented) in more detail to see if a consensus existed for including each specific element. To assure full coverage, each group was assigned a different starting point in the full list of 35 proposed elements and specifically tasked to first discuss the seven elements in their portion of the list and then broaden the discussion to other elements. In fact, most groups devoted most of their time to their assigned seven elements and then discussed other elements that were of particular interest to group members.

Following report-outs from each of the group leaders at the end of the day, the group leaders, workshop vice-chair, and workshop chair reviewed the list again and established a consensus list of 18 elements to be included in the code. In some cases, it was agreed that related elements should be consolidated into a single element, or that a very specific element might be generalized.

In the final morning of the workshop, each discussion group reviewed the entire consensus list, which had been winnowed to 18 elements, and established a consensus "top ten" list from it. These consensus lists were combined to see whether there were particular elements that might be dropped from the code. Since every element save one appeared in at least one of the consensus "top ten" lists, the entire set of 18 is retained in this report. Nine other proposed elements were placed on the research agenda as having potential value but requiring further research to validate this potential. These results are captured in Appendix A and Appendix B.

The chair drafted an initial version of this report in the week following the workshop. The vice-chair provided comments and suggestions that resulted in a second draft circulated to the Group Leaders and a few members of the Steering Committee. The comments from those individuals have been incorporated in this final version of the report.