Cyber Security and Privacy Research Institute

THE GEORGE WASHINGTON UNIVERSITY

Workshop to Develop a Building Code and Research Agenda for Power System Software Security: Final Report

Carl E. Landwehr
Cyber Security and Privacy Research Institute
George Washington University

Alfonso Valdes
Information Trust Institute (ITI)
University of Illinois

Report GW-CSPRI-2017-01

March 8, 2017

Support for this research was provided by the National Science Foundation (Grant CNS-145211), the Department of Energy (CREDC, DoE Award Number DE-OE0000780), and the IEEE Cybersecurity Initiative

Workshop to Develop a Building Code and Research Agenda for Power System Software Security: Final Report

Abstract

Both the attractiveness of power systems as targets of cyberattack and their vulnerability to remote attack via digital networks has been made clear by recent world events. While policy makers seek means to deter such attacks politically, surely the most effective way to reduce their attractiveness as targets is to reduce their vulnerability to such attacks. This can be done; these are engineered systems built to satisfy specifications. The results of the workshop presented here have the objective of reducing the vulnerability of future power systems to remote attacks that exploit vulnerabilities in the code – software or firmware – that controls their operation. The approach taken is to develop a consensus "building code" for building the software that controls these systems. Such a building code can provide a basis for customers to specify the security required of power system software components, for vendors to produce them, and for third parties to evaluate important aspects of their security properties. The availability and use of such a code can enable the marketplace to reward producers of systems with stronger security properties.

Workshop to Develop a Building Code and Research Agenda for Power System Software Security: Final Report

Table of Contents

1. Background

The idea of improving the security of fielded software in domains with critical security requirements through the development of a "building code" that might be used by customers, developers, and evaluators was first proposed in 2012 [1]. An initial workshop to establish such a code for software controlling medical devices was held in November 2013 with support of the IEEE Cybersecurity Initiative and the National Science Foundation's Secure and Trustworthy Cyberspace program. A draft code was published in 2014 [2, 3] and continues to provide a stimulus to those developing security standards in that domain. The present workshop was organized in a similar fashion to address development of a building code for software in the domain of power systems. The draft building code for power system software security is incorporated as Appendix A of this report, while Appendix B provides a research agenda motivated by the discussions held at the workshop. Appendices C and D provide the workshop preparation documents and the final agenda. Appendix E lists the participants.

2. Power System Context

This effort is motivated by the power system environment, including supply, demand, transmission, distribution, generation, smart grids, and microgrids, including residential use. The systems in this environment have requirements for both local and remote access and local and remote control. This access will be via networks that support digital communications. Some may be isolated, but some will be Internet-connected. To maintain the reliability and safety of these structures, cybersecurity is an issue of increasing concern in the power system environment as a whole.

In the realm of physical structures, building codes can incorporate a very broad range of requirements, from architectural and design requirements that apply to large public structures or neighborhoods to requirements on type and strength of materials to be used in construction. But a building code is not a design manual. It is a guideline that provides minimum expectations and recommended practices so that a building that conforms to the code should at least be safe and sound. While it cannot guarantee overall system security or reliability, a software building code for power systems will still improve the security posture of the software and systems being developed in this industry. Software must, as always, meet organizational and operational requirements, mitigate threats, and minimize flaws.

3. The Need for a Secure Software Development Process

It continues to be the case that most successful cyber intrusions exploit vulnerabilities that were accidentally introduced into the software at the implementation stage, *i.e.,* when programmers convert specifications to code. For this reason, this draft building code focuses most strongly on techniques for preventing the introduction of such implementation flaws or for finding and correcting them. However, the consensus of this workshop was that there is a fundamental need for a secure software development process to be put in place to organize the production of software for power systems. Participants proposed that two flows of requirements must be conducted in parallel as part of this process:

System requirements → device requirements → software
Security policy → security requirements → secure implementation

4. Security Policy's Central Role

In this context, security policy becomes part of the system requirements, and system security must be seen as not only preventing unintended things from happening but also ensuring that the system does perform its intended functions. Security policy in this light becomes the statement of what it means for the system to provide service that is dependable and secure in the sense of [4], in which a protection mechanism (e.g., a circuit breaker) is dependable to the extent that it operates at appropriate times and is secure to the extent that it doesn't operate at other times. In this lexicon, a system is considered reliable to the extent that it is both dependable and secure[1].

A system is secure only with respect to its stated security policy (and insecure only when and if those policy statements are violated). The specific security controls included in a system (e.g., authentication, access control, information flow control, cryptographic controls) are chosen in order to implement and enforce the policy.

Overall system design will determine whether software, hardware, or people operating the system are responsible for assuring that particular aspects of an overall security policy are correctly enforced. This document primarily addresses those aspects of security policies that are to be assured by software.

The building code can assist in the selection of proper controls to achieve the system's security policy as part of the software development process, just as codes for physical buildings assist the architect, developer, and builder in determining the safe width for stairways and fire exits. The essential first step in developing secure software is the security policy; the remainder of this building code is intended as a guideline to assist in the selection of controls and implementation of the controls necessary to enforce the policy.

5. Minimization of function

Among cybersecurity professionals, it is often said that complexity is the enemy of security [6,7,8]. Nevertheless, the economics of chip production and software production have led to the prevalence of computing hardware with broad capabilities and software that frequently includes many features and options bundled together. Features included a chip or application that the purchaser does not even know are present have often been exploited to penetrate a system.

Although not originally proposed as an element for the building code (and hence not included explicitly in Appendix A), the principle of disabling unneeded / unused functions was part of the workshop consensus. Different functions of a device might be disabled according to the application in which it is to be used; the building code would apply to the software developed for the device regardless of its application. Note that if the software implementing a disabled function is not removed, care must be taken to assure that it cannot be activated through the exploitation of flaws elsewhere in the system.

[1] Other technical communities define these terms differently, *e.g.* [5]

6. Challenges

Expanding the scope of the software building code from a focus on elimination of implementation errors to include system security policies and secure software development processes is a significant step. While a single organization may be able to implement and control secure software development procedures for software it develops internally, it is difficult to find a product today that doesn't incorporate software developed by others, including software with roots in the community of open source developers. Assuring that all of the software in a system was developed in accordance with a particular secure software development process will be a significant challenge for most companies. (The requirement for a software "bill of materials" in the draft code will at least allow the sources of software to be identified.)

In general, there are three ways to gain confidence that a piece of software will function as specified. First, one may have confidence in the people who built the software, for example, if they have produced similar software in the past and it has performed well. Seeking this kind of confidence might lead one to establish certification processes for individuals and for identifying what software was produced by certified individuals. Second, one may have confidence in the process or methodology used to build and test the software. This approach leads to the secure software development process requirement embraced by the workshop consensus, and might lead to certification of software development processes and identifying software that was produced in accordance with a particular process. A third way to assure that software will behave as specified is by examining the software itself, the output of the software development process. This third kind of assurance is the strongest, in the sense that it reasons about the actual code that will operate the system, but it is difficult (often impossible) to achieve simply by testing the code, because the state spaces involved are far too large for exhaustive testing. Techniques for mathematical verification of software can provide this kind of assurance. This approach might call for the certification of the tools and processes used in the verification. The size of software to which techniques have been successfully applied continues to grow, but remains a limiting factor.

A successful approach to the development of secure power system software may well involve all three of these kinds of assurance for the foreseeable future.

7. How might this report be used?

This report serves as an example of how a building code might be developed for software with security responsibilities in a particular domain. In itself, it records the consensus of a group of experienced industry, academic, and government laboratory individuals who are concerned with the security of future power systems. If it is to be used more widely, it needs to be circulated, read, considered, revised, amplified and perhaps eventually adopted by relevant organizations in the industry. It can also serve as a basis for industry and government standards groups considering how to proceed to help make the cybersecurity properties of future power systems an asset rather than a liability.

8. Acknowledgments

The authors thank all of the participants for their contributions to the workshop, which included considerable work in advance of the meeting itself. The willingness of all of the participants to travel to UIUC (including some from Europe and Australia), to share their views and to engage in spirited discussion made the workshop both productive and pleasurable. This report aims to capture the consensus of those present at the meeting. The authors are grateful to the group leaders and keynote speakers, who had the opportunity to review draft versions of the report, and whose comments have improved it. Responsibility for the final report, and any errors in it, remains with the authors.

Craig Preuss of the IEEE Power and Energy Society was particularly helpful in recruiting participants and assisting the organization of the workshop, although he was unable to participate in person. The IEEE Cybersecurity Initiative, and in particular Brian Kirk of the IEEE Computer Society, provided funds and organizational support that were essential to the conduct of the workshop. The U.S. Department of Energy, through its Cyber Resilient Energy Delivery Consortium (CREDC, DoE Award Number DE-OE0000780) activities at the University of Illinois at Urbana-Champaign (and in particular Amy Clay Moore) provided excellent facilities and logistics support. The National Science Foundation (NSF CNS-1452113) and the Cyber Security and Privacy Research Institute (CSPRI) at George Washington University provided additional support.

9. References

Participants also discussed and compiled a list of important interdisciplinary papers and resources in cybersecurity. Some participants felt that those individuals wishing to do interdisciplinary cybersecurity doctoral research should have read, or at least skimmed, most or all of these.

[1] Landwehr, C.E. **A Building Code for Building Code: Putting What We Know Works to Work.** In Proc. 29th Annual Computer Security Applications Conference (ACSAC), New Orleans, Dec 2013.

[2] Workshop to Develop a Building Code and Research Agenda For Medical Device Software Security: Final Report. Report GW-CSPRI-2015-01, January 8, 2015. Also available at: http://www.cspri.seas.gwu.edu/s/Landwehr-Building-Code-Final-Edit-Report-3-q0jj.pdf

[3] Building Code for Medical Device Software Security. (with Thomas Haigh). IEEE Computer Society, March, 2015. Also available at: http://cybersecurity.ieee.org/images/files/images/pdf/building-code-for-medica-device-software-security.pdf

[4] North American Electric Reliability Corporation (NERC). *Reliability Fundamentals of system Protection: Report to the Planning Committee*. NERC System Protection and Control Subcommittee. Dec. 2010. Available at: http://www.nerc.com/comm/PC/System%20Protection%20and%20Control%20Subcommittee%20SPCS%20DL/Protection%20System%20Reliability%20Fundamentals_Approved_20101208.pdf

[5] Avizienis, A., Laprie, J.-C., Randell, B. and Landwehr, C. **Basic Concepts and Taxonomy of Dependable and Secure Computing.** *IEEE Trans on Dependable and Secure Computing, Vol. 1*, No., 1 (Jan 2004), pp. 11-33.

[6] Zetter, Kim. "Three minutes with security expert Bruce Schneier. PC World, Sept. 28, 2001. Available at: https://www.schneier.com/news/archives/2001/09/three_minutes_with_s.html

[7] Geer, Dan. "Complexity is the enemy. *IEEE Security & Privacy Magazine*, August, 2008. P. 88. Available at: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4753682

[8] Goldfarb , Joshua. "Complexity is the enemy of security" *Security Week*, Feb. 11, 2015. Available at 2015: http://www.securityweek.com/complexity-enemy-security

APPENDICES

Appendix A

Draft Building Code for Power System Software Security

I. Purpose

This code is intended to provide a basis for reducing the risk that power system software is vulnerable to malicious attacks that might impede system operation or compromise the integrity or confidentiality of data used or generated by the system. The aim in specifying a model code is not to assure that future systems are invulnerable to any anticipated attack but to record a consensus among experts from industry, academia, and government laboratories that represents a baseline set of requirements for the security of software and firmware in power systems. To act in the same way as building codes for physical structures, such a code will need to evolve over time and hence will need to find an appropriate home in a body with a continuing existence and continuing participation by relevant groups. Procedures will need to be established for defining terms precisely, for proposing and adopting changes, for establishing conformance to the code, and so on. The workshop participants offer this baseline code in hope that it will eventually lead, either through the establishment of a more formal building code structure or through adoption in some other form by relevant bodies, to a safer and stronger cyber infrastructure for power systems generally.

II. Elements Recommended for Inclusion, by Category

In creating the categorization below, the aim is to be comprehensive. Consequently, there are some categories for which no proposed elements were identified or agreed upon by the participants. These empty categories are retained to highlight unmet needs.

For each element of the code, the following 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?
e. References

Proposed elements with consensus support

A. Elements intended to avoid/detect/remove specific types of vulnerabilities at the implementation stage

1. Secure software development process with assurance against subversion along with evidence of conformance

a. Description: Vendors must develop security-critical software within the framework of an established methodology for secure software development. No specific methodology is required, but relevant examples include Microsoft's Secure Development Lifecycle (SDL) and the coding practices developed by SAFECODE. Evidence that the delivered software was developed within the chosen methodology must be available for review. Any third-party software incorporated into security-critical functions must be shown to provide equivalent assurance against accidental incorporation of vulnerabilities.

b. Vulnerabilities addressed: Methodologies of the required type aim to reduce or eliminate a wide range of software vulnerabilities including memory safety errors, integer overflows, SQL injection, etc.

c. Developer resources required: Developer must be able to select and implement a given methodology, develop software in accordance with it, and also develop the evidence to demonstrate conformance.

d. Evaluator resources required: Evaluator must be able to review the delivered software and the conformance evidence and assess compliance.

e. References: For information on Microsoft's Security Development Lifecycle, see https://www.microsoft.com/en-us/sdl/ Information on the industry-wide SAFECODE intiative, is available at https://www.safecode.org.

2. Static and dynamic code analysis (throughout development cycle)

a. Description: Apply static and dynamic code analysis techniques to expose (and remediate as appropriate) software vulnerabilities. For developers, it is likely to be most effective to apply these tools regularly to software as it is developed, so that errors are found, and can be fixed, as soon as possible. The tools can be applied after the software is developed (including to software provided by third parties) and can still provide valuable information about the presence (or absence) of classes of errors; however it is generally acknowledged that it is significantly more costly to remediate errors found later in the development process.

b. Vulnerabilities addressed: memory safety (buffer overflows, use-after-free errors, null pointer dereference errors, etc.)

c. Developer resources required: access to relevant program analysis tools and programmers trained to use them effectively.

d. Evaluator resources required: Access to the software and analysis tools in order to replicate (or

not) results supplied by the vendor.

e. References: See NIST Software Assurance Metric and Tool Evaluation (SAMATE) reports, available at: https://samate.nist.gov/index.php/SAMATE_Publications.html. For a list of source code security analyzers, see https://samate.nist.gov/index.php/Source_Code_Security_Analyzers.html

3. Use of memory-safe/type-safe languages

a. Description: memory-safe languages can eliminate or substantially reduce the likelihood of many classes of coding errors that have often led to exploitable vulnerabilities. These include buffer overflows, null pointer dereferences, use-after-free errors, and references to uninitialized memory. *Rust* and *Go* are relatively recent memory-safe languages; others include *Java*, *F#*, *C#*, *Python*, and *Haskell*. Developers who select other common languages (e.g., *C*, *C++*) that don't provide memory safety need to provide evidence that their implementations avoid these problems.

b. Vulnerabilities addressed: memory safety errors.

c. Developer resources required: Access to compilers and tools for memory safe languages and programmers trained in them.

d. Evaluator resources required: Ability to assure that the programming language was in fact used to create the software (e.g., source code and a compiler).

e. References: see results reported for probability of security errors in programming contest submissions reported in "Build It, Break It, Fix It: Contesting Secure Development."Andrew Ruef, Michael Hicks, James Parker, Dave Levin, Michelle L. Mazurek, and Piotr Mardziel. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, October 2016. Available at https://arxiv.org/abs/1606.01881

4. System and component fuzz-testing

a. Description: Conventional testing generally aims to compare the results of a software implementation against its specification by exercising the functions included in the design in both normal and limit cases, so the test inputs are often designed to check particular cases and are not random. Fuzz testing essentially submits random inputs to a software component or system to see if unexpected behavior can be elicited and possibly exploited to subvert the behavior of the component or system. Participants agreed that fuzz-testing at both the component and system level should be a requirement of the building code, since attackers are quite likely to use it to seek paths into the system.

b. Vulnerabilities addressed: Like other testing methodologies, fuzz-testing cannot guarantee the absence of vulnerabilities, but its use can reveal a broad range of vulnerabilities including memory safety problems, race conditions, and many others. If these vulnerabilities can be found and remediated prior to deployment, they will be unavailable for exploitation by attackers.

c. Developer resources required: requires personnel who understand fuzz testing, the intricate details of the interfaces implemented, and have the tools available to conduct it. Like any testing regime,

requires a specification of system behavior against which the tested behavior can be compared. Fuzz testing is random and cannot be exhaustive, and it provides more assurance as more tests are run. Consequently, an assurance regime that depends heavily on fuzz testing will demand significant computing resources.

d. Evaluator resources required: Requires the ability to review fuzz testing output and to judge its comprehensiveness.

e. References: The original paper on fuzz testing: "An Empirical Study of the Reliability of UNIX Utilities" , B.P. Miller, L. Fredriksen, B. So, *Communications of the ACM 33*, December 1990. Many tools are available for fuzz testing; some depart from the completely random model and incorporate coverage metrics or target boundary and limit cases. Microsoft has published guides on "how much" fuzzing is appropriate as well as on types of fuzzing to be applied.

5. Stress Testing

a. Description: the aim of stress testing is to explore the behavior of a component or system when it is operated with relatively limited resources – e.g., memory, CPU, or network communications bandwidth may be limited in relation for a high required demand for service. These conditions can occur in normal operation if there is high demand, but they may also be artificially induced by an attacker mounting, for example, a denial of service attack on the system. A properly designed system should show graceful degradation in the face of stress testing and should recover normal operation smoothly as the stress is removed. Participants agreed that stress testing at both the component and system level should be a requirement of the building code.

b. Vulnerabilities addressed: Like other testing methodologies, stress testing cannot assure flaws or design weaknesses are absent, it can only reveal only reveal those that the tests exercise. Stress testing may reveal a variety of implementation failures that occur when design parameters (e.g., maximum table sizes or queue lengths) are reached. Stress testing should also reveal failures in recovery mechanisms.

c. Developer resources required: Requires personnel who understand stress testing and have the tools available to conduct it. Requires a specification of expected system behavior under high-stress conditions and expected recovery modes when stress is removed.

d. Evaluator resources required: Requires the ability to review stress testing results and to judge the comprehensiveness of the tests.

e. References: Textbook on performance testing generally: Liu, H.H. *Software Performance and Scalability: A Quantitative Approach*. John Wiley & Sons., Inc. 2009.

6. Fault-injection testing

a. Description: fault injection testing aims to evaluate component and system behavior when faults occur. This testing approach therefore focuses on exercising fault- and error-handling code within the system that may be rarely invoked in operation. Faults may be injected at compile time by modifying

the source code or at run time by modifying system data or protocol messages flowing over a network. Specifications must address the expected response to induced failures so that test results can be evaluated. Participants agreed that this type of testing should be applied to high-fidelity representations of operational power systems but should definitely not be conducted on live operational systems.

b. Vulnerabilities addressed: vulnerabilities likely to be revealed through fault-injection testing are those found in error-handling and recovery routines.

c. Developer resources required: Requires personnel conversant with fault injection testing and tools to assist in conducting tests and evaluating results.

d. Evaluator resources required: Requires the ability to evaluate fault-injection test results and to assess their comprehensiveness.

e. References: M.-C. Hsueh, T.K.Tsai, R. Iyer. "Fault Injection Techniques and Tools," *IEEE Computer*, April 1997, p. 75 ff.

B. Elements intended to assure proper use of cryptography

1. Accredited cryptographic algorithms and implementations

a. Description: Cryptographic algorithms that resist serious cryptanalysis are notoriously difficult to invent and to program correctly. While different environments make different demands on cryptography (for example, differing amounts of energy and time to devote to cryptographic operations and different time horizons for protecting keys), developers should seek algorithms that have received some external, open certification rather than attempt to develop their own. If for some reason suitable algorithms are not available and invention is required (this should be a last resort), developers should take care to get expert review prior to adopting and implementing their own crypto- algorithms. Weaknesses in cryptography often come in the implementation of the algorithm, key management, and surrounding protocols. Externally developed and certified implementations should be sought; custom implementations of cryptographic components require careful vetting by experts. In power system environments, cryptography may more often be called upon to assure the integrity of commands from operators and data from sensors rather than to protect their secrecy. Proper selection and implementations of algorithms for these requirements, proper use of cryptographic software packages, and proper management of keys will be essential to assuring that the requirements are met in practice.

b. Vulnerabilities addressed: addresses weaknesses in cryptographic algorithms, implementations, and use.

c. Developer resources required: requires the ability to understand the cryptographic requirements of the system, select appropriate algorithms and implementations, and to use the selected packages correctly.

d. Evaluator resources required: Requires the ability to review and evaluate the system requirements

and the developers design, selections, and implementations.

e. References: "Use Cryptography Correctly," in IEEE Cybersecurity Initiative: Avoiding the Top Ten Software Security Design Flaws., p. 19. https://www.computer.org/cms/CYBSI/docs/Top-10-Flaws.pdf

2. Secure random numbers

a. *Description*: Generating random numbers for use in initializing pseudorandom number generators and cryptographic algorithms, using them correctly, and avoiding reusing them are challenging problems. Mistakes can nullify even well-designed and implemented cryptographic mechanisms. As advised in other work, developers should adopt established approaches that experts have vetted rather than attempting novel solutions. Even established approaches for random number generation need to be subjected to appropriate testing to assure their effectiveness.

b. *Vulnerabilities addressed*: susceptibility to cryptanalytic attacks on integrity and confidentiality that exploit poor selection of keys and other numbers intended to be random.

c. *Developer resources required*: Requires access to vetted procedures for random number generation; these may be platform-dependent. Requires testing the procedures and documenting the results.

d. *Evaluator resources required*: Ability to review and evaluate developer's design and implementation of random number generation and use, as well as reviewing test results.

e. *References*: "Use Cryptography Correctly," in IEEE Cybersecurity Initiative: Avoiding the Top Ten Software Security Design Flaws., p. 19. https://www.computer.org/cms/CYBSI/docs/Top-10-Flaws.pdf

C. Elements intended to assure software/firmware provenance and integrity, but not to remove code flaws

1. Software Bill of Materials

a. Description: Originally posed as "layered, traceable assurance and verification," the participants felt that it was too difficult to formulate as a checkable building code element, but agreed that a bill of materials, specifying what software (including version or release number) is included in a system and the source of all of the software components in the system is both critical and checkable.

b. Vulnerabilities addressed: This element does not prevent vulnerabilities but permits identifying whether vulnerabilities discovered in software components are included in the system and hence may require patching/remediation. In this way it can be a critical tool for system defenders, but also for attackers, if they have access to it.

c. Developer resources required: Ability to determine and specify where each line of code in the delivered product originated

d. Evaluator resources required: Ability to map provided bill of materials against delivered software

components.

e. References: "H.R.5793 - 113th Congress (2013-2014): Cyber Supply Chain Management and Transparency Act of 2014 - Congress.gov - Library of Congress". Available at https://www.congress.gov/bill/113th-congress/house-bill/5793

2. Digitally signed software and firmware with update validation

a. Description: Both firmware and software that implement critical functions should be digitally signed, and the private signing keys must be carefully managed. The developer must either identify and distinguish critical vs. non-critical functions, or else the signatures must apply to all software and firmware. Files containing critical system configuration data will also benefit from these controls.

b. Vulnerabilities addressed: This element does not prevent or eliminate vulnerabilities in software or firmware but aids in addressing software provenance (see Software Bill of Materials as well) and accountability in case of failures or attacks. Reduces vulnerability to spoofed updates or rollbacks.

c. Developer resources required: infrastructure to generate, distribute, update and protect signing keys; ability to integrate signing and validation functions in delivered system.

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

e. References: Arbaugh, William A., David J. Farber, and Jonathan M. Smith. "A secure and reliable bootstrap architecture." *Proc., 1997 IEEE Symp. on Security and Privacy.* IEEE, 1997.

Nilsson, Dennis K., Lei Sun, and Tatsuo Nakajima. "A framework for self-verification of firmware updates over the air in vehicle ECUs." *Proc. 2008 IEEE Globecom Workshops.* IEEE, 2008.

Cui, Ang, Michael Costello, and Salvatore J. Stolfo. "When Firmware Modifications Attack: A Case Study of Embedded Exploitation." *Proc. 20th Network and Distributed Systems Symp. (NDSS) 2013*, Internet Society, San Diego, CA, Feb. 2013.

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

1. Specification of system information flows with effective enforcement

a. Description: While the confidentiality of information in power systems is a concern, the integrity and flow of information, particularly control information sent to and received from cyberphysical systems, is usually the most critical concern. The developer must specify the flow of critical information through software and hardware components and make use of software and hardware mechanisms, including mandatory access controls (MAC), rings of protection, privilege mechanisms, capability mechanisms, one-way flow devices, etc., as available and appropriate. This broad requirement concerns both system security policy and system architecture.

b. Vulnerabilities addressed: Enforcement of information flow constraints does not necessarily

eliminate implementation errors that could be exploited by maliciously crafted inputs, but it can limit the effects to the domains "downstream" from the exploitable flaw.

c. *Developer resources required*: Ability to understand and architect information flows within the system and to employ available mechanisms to enforce them.

d. *Evaluator resources required*: Ability to understand and assess both system function and developer's information flow specification and implementation.

e. *Reference*s: See French ANSSI http://www.ssi.gouv.fr/uploads/2014/01/Managing_Cybe_for_ICS_EN.pdf , esp. Appendix B, GP02, and US DHS https://ics-cert.us-cert.gov/sites/default/files/documents/Seven%20Steps%20to%20Effectively%20Defend%20Industrial%20Control%20Systems_S508C.pdf, item 4, for examples of related guidance.

2. Input Validation: All input accepted by control software must be well-defined (via a grammar or equivalent means), as syntactically simple as possible (regular or context-free syntax preferred), and fully validated before use.

a. Description: Demonstrating the effectiveness of input validation, i.e., demonstrating that invalid inputs can be identified and are in fact rejected, was agreed to belong in the code. Simplification of inputs, which can reduce the difficulty of validation, was considered desirable but did not gain consensus as a code requirement.

b. Vulnerabilities addressed: Exploitation of input-handling code by maliciously crafted input. Accepting invalid inputs can lead to unpredictable system behavior. Input validation can protect against buffer overflows and related memory safety errors.

c. Developer resources required: Requires that for each possible system input, the range of acceptable inputs be unambiguously specified and that the implementation assure inputs are validated as specified.

d. Evaluator resources required: Ability to review both the input specification and the code responsible for validating inputs.

e. References: "Security Applications of Formal Language Theory", Len Sassaman, Meredith L. Patterson, Sergey Bratus, Michael E. Locasto, *IEEE Systems Journal*, *Volume 7*, Issue 3, Sept. 2013, see also http://langsec.org/papers/langsec-tr.pdf)

3. Appropriate component separation / isolation

a. Description: Providing isolation between components so that malfunction or penetration of one component cannot affect those isolated from it is a fundamental software and security engineering technique. In power systems, it is appropriately used to separate non-critical functions from critical ones, which implies that the critical functions have been explicitly identified. Mechanisms for achieving the separation can include hardware support for isolating machine domains (e.g., privilege modes, rings, segmentation, capabilities). Software sandboxing mechanisms can also be effective but

may require additional evidence to assure their strength.

b. Vulnerabilities addressed: this element does not remove specific classes of vulnerabilities but prevents or raises the difficulty for an attacker who exploits a vulnerability in one component to leverage that exploitation in other components.

c. Developer resources required: ability to distinguish more critical from less critical functions/components; ability to organize security architecture to exploit underlying security isolation mechanisms (processes, sandboxes, virtual machines), ability to map the design to the underlying separation mechanisms correctly.

d. Evaluator resources required: Access to relevant design and implementation documents from developer and ability to interpret and evaluate them correctly.

e. References: C. Greamo and A. Ghosh, "Sandboxing and Virtualization: Modern Tools for Combating Malware," in *IEEE Security & Privacy, vol. 9*, no. 2, pp. 79-82, March-April 2011.

4. Authentication and access control (human – device and device – device)

a. Description: Authentication of human operators to machines is critical to providing accountability for operator-initiated actions and a basis for implementing role based (or other) access controls. As automation and attack sophistication increase, it will become more important for the machine to authenticate itself to the operator as well (i.e., so that the operator can be sure she is communicating with the intended machine and that its configuration is accurately portrayed). A complicating factor may be the need for emergency access by human operators. In addition, devices will require mutual authentication, for similar reasons. Some current standards for substation operation already impose authentication requirements. The consensus was that the code should require two-factor authentication of operators, but at the same time should provide for audited “break-glass” emergency access for critical functions. Device-device authentication was seen as important, but requiring further research prior to imposing a building code requirement.

b. Vulnerabilities addressed: This element does not generally detect or remove vulnerabilities in software or hardware, but it provides accountability for actions taken and provides the basis for authorizing system access. Authenticated communications can enable detection of traffic inserted by unauthorized third parties.

c. Developer resources required: Ability to design and incorporate appropriate authentication mechanisms, including two-factor authentication.

d. Evaluator resources required: Ability to evaluate authentication mechanism design and implementation.

e. References: IEEE Std. 1686-2007 for Intelligent Electronic Devices. IEEE Std. 1815 (DNP3) also describes a machine-to-machine authentication process.

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 software installation, user authentication, and attempted intrusion. The audit trail must not be overwritten by a flood of events; and there shall be a provision for offline storage. It was noted that certain kinds of power fluctuations might themselves be indicators of security-relevant events, but such fluctuations are expected to be captured by the power monitoring systems and hence did not require inclusion in this element.

b. Vulnerabilities addressed: This element does not prevent or detect vulnerabilities, it aims to provide a record that would permit reconstruction and understanding of adverse activities after the fact and may assist in restoring the system to a valid state. If software monitoring a log can detect a malfunction or attack based on the logged actions, it may be able to initiate recovery actions or inhibit further damage.

c. Developer resources required: Requires identification of security related event types (for example, authentications, privilege level changes, and software updates) including intrusion attempts, and implementation of tamper-resistant, append-only security event logs.

d. Evaluator resources required: Requires manual review of identified security related event types and of design and implementation of logging mechanisms and security event generation mechanisms.

e. References: IEC 61850, IEEE 1815 (DNP3), IEEE 1686 already call for related functions.

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

No elements proposed specific to this category, but see "back-out" functionality element.

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

1. Inherent "back-out" functionality // trustworthy recovery

a. Description: provide mechanisms that support restoration to secure functional state after a successful attack has been detected. Providing this capability can affect the system design broadly.

b. Vulnerabilities addressed: This element does not prevent or eliminate vulnerabilities but aims to restore system function after a vulnerability has been exploited.

c. Developer resources required: Requires the developer anticipate potentially successful attack modes and provide recovery mechanisms (e.g. backups inaccessible to attackers) that can be invoked when system degradation is detected.

d. Evaluator resources required: Ability to assess adequacy of developer's design and recovery mechanisms.

e. References: Gallagher, P. *A Guide to Understanding Trusted Recovery in Trusted Systems*.. NCSC-TG-022. U.S. National Computer Security Center, Dec., 1991. Available at: https://fas.org/irp/nsa/rainbow/tg022.htm

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

No elements proposed specific to this category. However, it is related to software/firmware update validation under the previous element “Digitally signed software and firmware with update validation”

Appendix B

Research Agenda for Power Systems Software Security

Input Simplification: some participants proposed that inputs should be required to be simplified to improve the assurance that inputs can be mechanically validated. For example, "Protocols with complex message formats such as DNP3, IEC 61850, etc. must be restricted by their recognizer modules to subsets actually used by specific devices and valid for these devices. Non-conforming inputs should be rejected." The consensus was not to include this requirement in the initial code because protocol implementations may be licensed from third parties and hence difficult to modify. Research may be warranted into techniques for simplifying input language complexity and for wrapping existing implementations so that (potential) flaws in third party implementations cannot be exploited.

Verified OS and hardware. Some participants proposed to require the use of verified operating system and hardware platforms for critical devices. Some low power/low function devices do not include operating systems, so if such a requirement were included in a future code, the scope of its application would need to be made clear. Verfication would enable assuring that system initialization leads to a secure initial state. Critical properties desired of a binary (or source) program would need to be 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. The participants recognized there have been substantial advances in tools that can be applied to carry out formal verification of software and that some substantial software systems, including the seL4 kernel, have been verified. The technology is seen as cost-effective and is in use by chip vendors to verify hardware designs. The relative simplicity of some power system components would seem to bring them within reach of the technology. However, on balance, the participants felt that there was more research to be done before this element could be placed into the code.

Automated conformance checking. This proposed element is meant to cover mechanisms to check whether a software program conforms to the building code. Tools (some more automated, like SAT solvers, and some requiring more manual assistance, like theorem provers) are, and have been for some time, available for this purpose. This element is closely related to the proposed "Verified OS and hardware" element, except that the proposed conformance is to the building code rather than to a functional specification, and a similar discussion applies.

Formal requirements specification. At least three senses of formal requirements specification were discussed during the meeting. For those pursuing formal verification of programs, a formal (in the sense of mathematical logic) specification of the desired properties of the program is required. The difficulty of creating such a specification is an impediment to the development of verified OS and hardware, just discussed, and suggested that incorporating a building code element for a formal specification is premature at this time. The participants also discussed formal security policy models, in the sense of the Trusted Computer System Evaluation Criteria, and endorsed the idea that without such a model, particularly one addressing mandatory integrity requirements, it is essentially

impossible to specify when a security violation has occurred. On the other hand, "formal" used in the sense of having a form, a structure, leads to a different interpretation of "formal requirements specification". It was noted that IEEE Standard 1686 for Intelligent Electronic Devices provides a framework for cybersecurity requirements for such devices. The consensus placed this proposed element on the research agenda.

Active defense and automated response. This proposed element aims to automate the current activities of attack detection and response. As such, it aims to reduce vulnerabilities only as mitigations to observed attacks. Some current activities such as the DARPA Cyber Grand Challenge have incentivized this approach, but the participants felt that the technology is not mature enough to include in a building code at this time.

Assurance cases with 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, their development for use in the security domain is less mature. 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 specifically for power system devices.

Appendix C

Call for Participation, including Workshop Proposal

Call for Contributions and Participation
IEEE Invitational Workshop to Create a Building-Code for building code
for Power System Software Security: (BC)^2 Power
November 16-18, 2016
University of Illinois at Urbana-Champaign

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 cyber components of electric grids 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 Cybersecurity Initiative**, **IEEE Smart Grid**, and **IEEE Power and Energy Society**, with participation from **UIUC's Information Trust Institute**; additional support is being sought from the NSF Secure and Trustworthy Cyberspace program.

The workshop proposal describing the scope, objectives, and the building code metaphor is included as an appendix to this call for contributions and participation.

What might a building code for power system software/firmware 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 power system device software/firmware 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.

Considerations for Including a Particular Requirement in a Building Code for Power Systems Software Security

1 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 Requirements affecting only a narrow scope of vulnerabilities may not be appropriate to incorporate.
4

For an example of such a building code, targeted at medical device software security, see:

https://www.computer.org/cms/CYBSI/docs/BCMDSS.pdf.

Participants Sought

To succeed, the workshop needs participation from (1) industry personnel familiar with the architecture and tools used to build power system devices and the software that controls them, (2) people familiar with the history of power system 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 power system software in particular, and (4) experts in relevant aspects of software engineering, including requirements, design, and (especially) implementation, test, and validation/verification methods.

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. 16) and ending in the afternoon two days later (Nov 18). 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 power system 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 will be placed on IEEE Cybersecurity's website and will be published by the IEEE as well, similar the **Building Code for Medical Device Software Security document**.

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 (B) 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 power system software development or in assuring software security. Submit this information to:

http://goo.gl/forms/ysWg3JwJZyEd0Is13 no later than September 14, 2016.

Travel Support

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

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.

2

Appendix to the Call for Participation (Workshop Proposal)

Workshop Proposal

A Building Code for Building Code for

Power System Software Security: $(BC)^2$ Power

Introduction

Based on a recent essay by Landwehr [L13], the cybersecurity community has undertaken a number of initiatives exploring the metaphor of structural building codes as a guiding framework for building secure codes in critical systems. Under the auspices of the IEEE Cybersecurity Initiative [IEEE] and the National Science Foundation, a workshop was held in November 2014 to describe such a framework for medical devices [MDSSA, MDSSB]. The purpose of this document is to propose a workshop to explore the building code metaphor in the domain of electric power systems.

Modern infrastructure systems, such as those in electric power grids, are rapidly evolving into cyber-physical systems (CPS) in which distributed cyber assets for monitoring, communication, and control interface with a physical process for safe and efficient operation. In the power sector, assets include embedded systems in devices located at substations, poletops, or on customer premises (for example, smart meters) as well as more conventional server and workstation platforms hosting data historians and human-machine interfaces (HMI). This proliferation of assets exposes a growing and poorly understood attack surface, with potential attacks ranging from theft of service to massive, protracted outages. Since all other critical infrastructures and indeed the smooth function of modern society depend on electric power, the effect of a long-term, wide-area outage could be destabilizing on a historically unprecedented scale.

Workshop Objectives

The proposed workshop seeks to define the building code framework for software systems in electric power, from HMI to embedded systems firmware in field devices. The workshop will enumerate elements of such a code, and identify tools, processes, and methodologies that support these elements. This in turn will define adoption strategies and identify outstanding gaps to be addressed by the research and academic communities. The eventual goal is an adaptable structure to guide code implementation, addressing the following points as well as other aspects identified by workshop participants:

- Approaches to avoid and remove security flaws at design and implementation
- Choice of development environment, language, and libraries
- Tools for static and dynamic code analysis
- Evaluation of the finished product
- Certification
- Resilience, defined as safe operation or “soft landing” in case of attack or adverse event
- Built-in features to support attack detection, attribution, and forensics

Intended Audience

We invite participation from diverse stakeholders with an interest in secure software in the power sector.

This includes the academic and research community, but it is essential to enlist participation from power system equipment vendors (responsible for HMI software and device firmware), asset owners, security consultants, and policy specialists. This diversity is essential to ensure that the workshop product reflects the interests of all involved, and to maximize the probability that the framework will be adopted.

Document Organization

Subsequent sections of this document provide initial background discussion addressing workshop objectives, although we anticipate that the workshop will significantly change this understanding. We first sketch out the building code metaphor in the power system context, and enumerate a candidate list of elements of the building code. Next, we position our effort in relation to initiatives from NIST and the Security Development Lifecycle [SDL06, MSSDL], as well as the earlier work by members of the team in the medical device domain. We conclude with an overview of the particular challenges that arise applying this approach to the power system domain.

Building Code as Metaphor

The workshop aims to develop an analog to structural building codes focused on security properties of software. The objective of the code is to increase assurance that developed software will be free of many software vulnerabilities typically present in software developed according to current practices. Evidence suggests that exploitable security flaws arise from implementation rather than design flaws. The underlying motivation for creating this building code for the security of power system software and firmware is to provide a basis that developers can use to rule out the most commonly exploited classes of software vulnerabilities, as well as to build in mechanisms for recovery and attribution. To accomplish this, the code elements must be effective and relatively easy to adopt and evaluate.

Elements of a Building Code for Code

We enumerate candidate elements of (BC)2 Power by analogy to elements of structural building codes, given in the table below.

Analogs Between Structural Building Codes and Building Codes for Code

(adapted and extended from [L13])

Structural Building Code	**$(BC)^2$ Power**
Structural integrity against hazards such as wind, lightning, rain, earthquake	Structural integrity of isolation and other mechanisms to resist tampering and other attacks
Fire safety: prevent, detect, isolate, safe exit	“Fireproof” materials in the form of coding standards, mechanisms to detect, isolate (domain separation), and recover
Physical safety of occupants: door and stairway	Security mechanisms that are easy to use and

dimensions, handrails	understand
Water and energy	Efficient and secure information flows
Security certified materials, components, and subassemblies	Secure development environments, languages, libraries, and modules
Inspection	Code review, static analysis
Stress test: Bring pipes to pressure, foundation bolts pull-out test	Dynamic analysis, fuzz testing. Design embedded systems sufficiently robust to enable scanning and fuzz testing
Certification	Certification, signed software

Complementary Initiatives

Security Development Lifecycle

The Security Development Lifecycle [SDL06, MSSDL] outlines a sequence of practices to be followed from the security requirements phase through design, implementation, verification, release, and (incident) response. Analysis and, to the degree possible, reduction of the attack surface is a theme that recurs in various phases. SDL calls for threat modeling, which may guide adoption of elements in our building code, but may not be part of the building code itself.

The building code elements enumerated in this document in areas such as secure development environments, static and dynamic analysis, and fuzz testing map directly to counterparts in SDL, particularly in the implementation and verification phases.

NIST

The NIST framework [NIST14] provides a risk-based approach to managing cybersecurity risk, but is more oriented to the enterprise user rather than the developer. It outlines a tiered approach whereby an organization examines components of cybersecurity risk and undertakes measures to address it. The framework specifies functions to identify, protect, detect, respond, and recover. We may envision that building codes in the sense we propose support some of these functions. For example, an implementation of the NIST framework may legitimately claim that the identified risk is lower for software that is certified as having been built according to codes such as we propose than for software claimed to be functionally equivalent but without the certification. The objectives of the NIST framework and the building codes initiative are different in that the NIST framework addresses business process and risk management, while the building codes approach identifies practices, tools, and procedures to implement secure software.

Medical Device Software Security

Some of the organizers of the workshop proposed here have applied he building code concept in a workshop addressing software security in medical devices [MDSSA, MDSSB]. The workshop report includes an appendix that identifies code elements, grouped as follows:

- Elements intended to remove/avoid flaws at design
- Elements intended to avoid/detect/remove vulnerabilities at implementation
- Elements intended to assure software/firmware provenance and integrity
- Elements intended to impede attacker analysis or exploitation
- Elements to enable detection and attack attribution

We note that the latter three categories above do not remove or avoid vulnerabilities, but promote resiliency in the presence of vulnerabilities that persist in the developed software.

The workshop identified element categories for safe degradation, restoration, and maintenance without loss of integrity. No specific elements were proposed in these categories.

Challenges and opportunities in the power sector

The following are some of the specific challenges to security in the power grid domain. The list is not intended to be exhaustive.

Long component life

Power systems are characterized by a mixed ecosystem of legacy and modern components. The useful life of a component with respect to its power function is typically much longer than the firmware refresh cycle. Legacy components cannot support modern security measures, and today's newly installed device will be "legacy" from the cyber standpoint through most of its useful life in the field. The building code must recognize this, and anticipate requirements for secure firmware upgrades and interoperability in mixed legacy environments.

Computational, communication, and power constraints

Power system devices must react to adverse conditions that arise suddenly (a tree falling across a power line) so as to maintain system safety, minimize outage, and protect difficult-to-replace equipment. Legacy devices achieve these objectives through thermal or electro-mechanical means with no or minimal programmable logic or inter-device communication. Modern protection schemes require rapid intra-device and distributed communication to respond intelligently to adverse conditions. It must be the case that the building code does not impede these requirements, especially considering the fact that field devices may be limited in computational and communication resources.

Security perimeter: Substation, poletop, customer premise

Intelligence in smart grids is increasingly distributed and migrating to the grid edge (both physically and logically). Substations now include a variety of devices such as transformers, bus bars, and intelligent relays and breakers. More and more intelligence is moving to the field, in the form of devices such as poletop reclosers. This trend extends all the way to the customer premise in the form of smart meters.

While one may argue for physical security of substation equipment within a fenced perimeter, we must assume that there is no meaningful physical security perimeter beyond rudimentary tamper detection on field and customer-premise equipment.

3rd party connections

Among other goals, smart grid is intended to enable the integration of renewable resources as well as new energy markets. Renewable resources may interface to utility grids at large scales (wind farms) or distribution scale (microgrids, customer-premise solar). Smart grid also enables third-party markets such as aggregation and home energy management. In all these cases, utilities must ensure the integrity of physical and financial transactions across interfaces with systems over which they have no administrative control.

References

[IEEE] **http://cybersecurity.ieee.org/**

[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**

[MDSSA] *Workshop to Develop a Building Code and Research Agenda For Medical Device Software Security:* Final Report. Report GW-CSPRI-2015-01, January 8, 2015:

http://www.landwehr.org/2015-01-landwehr-gw-cspri.pdf

[MDSSB] Landwehr, C.E., and Haigh, T. "Building Code for Medical Device Software Security", IEEE Computer Society, March 2015. **http://cybersecurity.ieee.org/images/files/images/pdf/building-code-for-medica-device-software-security.pdf**

[MSSDL] Microsoft Security Development Lifecycle,

http://www.microsoft.com/en-us/sdl/default.aspx

[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

[SDL06] Michael Howard and Steve Lipner. "The Security Development Lifecycle: SDL: A Process for Developing Demonstrably More Secure Software (Developer Best Practices)"

Appendix D

IEEE Invitational Workshop to Create a Building-Code for <building code> for Power System Software Security: $(BC)^2$ Power

November 16-18, 2016

University of Illinois at Urbana-Champaign

Final Agenda

Time	Event		Location
Wed. 11/16, 5:30 pm	Arrival	All workshop participants	Hampton Inn
6:00 pm	Registration / Pre-dinner reception	All workshop participants	Bahl Meeting Room 3002 Electrical and Computer Engineering Building 306 N. Wright St. Urbana, IL 61801-2918
7:00 pm	Dinner	All workshop participants	
8:00 pm	Talk 1: History of software security and approaches to marketplace	Speaker: Andrew West, SCADA Consultant	
8:45 pm	Talk 2: History of software security and approaches to regulation	Speaker: Roger Schell, Aesec	same as above
9:30 pm	Adjourn		
Thurs. 11/17			
7:00am	Continental Breakfast available	All participants	

8:00 am	Introduction of the participants and outline of the workshop	Bill Sanders, David Nicol, Carl Landwehr, Al Valdes	
8:45 am	Proposed building code element talks	Category A items	
10:00 am	Morning Break		
10:30 am	Talks continue	Category B, C, D, etc. items	
12:00 noon	Lunch		
1:00 pm	Breakout groups convene	Please check your name badge for your group assignment	Breakout rooms: Group 1 (Pink): RM 3034 Group 2 (Red): RM 3036 Group 3 (Yellow): RM 4036 Group 4(Green): RM 5034 Group 5(Orange): RM 5086
3:00 pm	Afternoon Break		
3:15 pm	Breakouts continue		
4:00	Breakouts conclude		
4:10 pm	Plenary presentations / discussion	Breakout group leads report on conclusions, new proposals	
5:10 pm	Discussion period if needed		
5:30 pm	Adjourn		

	Group leads / steering	Coordination meeting	
6:30 pm	Reception		
7:30 pm	Dinner		
Fri. 11/18			
7:30 am	Continental Breakfast available		
8:30 am	Plenary	Report on consensus from group leads meeting, charge for morning breakouts	
9:00 am	Breakouts re-convene		Breakout rooms: Group 1 (Pink): RM 3034 Group 2 (Red): RM 3036 Group 3 (Yellow): RM 4036 Group 4(Green): RM 5034 Group 5(Orange): RM 5086
10:00 am	Morning break		
10:30 am	Breakouts re-convene		
11:20	Breakouts conclude		
11:30 am	Closing plenary		
12:30 pm	Box lunches and departure		

Appendix E – List of Participants

IEEE Invitational Workshop to Create a Building-Code for <building code> for Power System Software Security: $(BC)^2$ Power

November 16-18, 2016

University of Illinois at Urbana-Champaign

Note: Organizational affiliations are shown for information only. The workshop results and report do not necessarily represent the views of these organizations.

Kaibin Bao, Karlsruhe Institute of Technology (KASTEL)
Ian Bryant, Trustworthy Software Foundation
Chris Chelmecki, Basler Electric, *Discussion Group Leader*
Art Conklin, University of Houston
Adam Crain, Automatak
Dennis Gammel, Schweitzer
Andrew Ginter, Waterfall-Security
Mark Heckman, University of San Diego
Marijn Heule, University of Texas -- Austin
Carl Landwehr, George Washington University (CSPRI)
Chad Lloyd, Schneider Electric
Dario Lobozzo, Radiflow
Johan Malmström, ABB
Scott Mix, North American Electric Reliability Corporation (NERC)
Ken Modeste, UL, *Discussion Group Leader*
Tommy Morris, University of Alabama -- Huntsville
David Nicol, University of Illinois at Urbana-Champaign
Rajesh Nighot, Nebulian
Michael Pyle, Schneider Electric
Edwards Reed, AESec, Inc.
Craig Rieger, Idaho National Laboratory
Benjamin Salazar, Lawrence Livermore National Laboratory
Chet Sandberg, Consulting Engineer
William Sanders, University of Illinois at Urbana-Champaign
Roger Schell, AESec, Inc., *Keynote address*
Steven Templeton, University of California -- Davis
Eric Thibodeau, Gentec
Mike Thiems, Basler Electric
Alfonso Valdes, University of Illinois at Urbana-Champaign
Zhenyuan Wang, ABB, *Discussion Group Leader*
Sam Weber, New York University
Jin Wei, University of Akron
Andrew West, SUBNET Solutions, Inc., *Keynote address and Discussion Group Leader*
Chuck Weinstock, Software Engineering Institute (CMU), *Discussion Group Leader*
Reid Wightman, RevICS
Carol Woody, Software Engineering Institute (CMU)
Tim Yardley, University of Illinois at Urbana-Champaign