A Building Code for Building Code

Putting What We Know Works to Work

Carl E. Landwehr
George Washington University
1923 Kenbar Ct.
McLean, VA 22101

ABSTRACT

Systems of programs control more and more of our critical infrastructures. Forty years of system development and research have taught us many lessons in how to build software that is reliable, relatively free of vulnerabilities, and can enforce security policies. Those years of experience seem not to have taught us how to get these lessons put into practice, particularly with respect to security, except in a few specialized places. This essay suggests an approach to capturing what we know in a way that can make a difference in systems on which we all rely.

Categories and Subject Descriptors

1998 CR classification: D.2.0 SOFTWARE ENGINEERING General (Protection Mechanisms, Standards) K.4 COMPUTERS AND SOCIETY K.4.1 Public Policy Issues (Regulation)

General Terms

Security, Standardization, Management

Keywords

Security policy, critical infrastructure software, building code

1. INTRODUCTION

In *The Mythical Man-Month* Fred Brooks writes, under the heading "The Joys of the Craft":

> "... The programmer, like the poet, works only slightly removed from pure thought-stuff. He builds his castles in the air from air, creating by exertion of the imagination. Few media of creation are so flexible, so easy to polish and rework, so readily capable of realizing grand conceptual structures. ...
>
> "Yet the program construct, unlike the poet's words, is real in the sense that it moves and works, producing visible outputs separate from the construct itself. It prints results, draws pictures, produces sounds, moves arms. The magic of myth and legend has come true in our time. One types the correct incantation on a keyboard, and a display screen comes to life, showing things that never were nor could be.
>
> "Programming then is fun because it gratifies creative longings built deep within us and delights sensibilities we have in common with all." [1, p.7]

Though I won't claim any software I ever wrote rose to the level of poetry, I quote these lines in part because they capture what first drew me into computing and computer science.

These lines also remind us that although the execution of a program by a computer can have very concrete effects, the program itself is a relatively abstract creation – "only slightly removed from pure thought-stuff." But unlike the poet, whose language communicates directly to readers, the programmer's creation is visible to most people only through its physical effects.

Metaphor is a figure of speech in which two otherwise unrelated objects are asserted to be the same on some point of comparison (without using "like" or "as", which would convert metaphor to simile). To think about ethereal things, or things they don't fully understand, people often resort to metaphors – things in the real world that they do understand and that they can use to talk about and think about those things made of "pure thought stuff".

This kind of thinking can be wonderfully helpful. A good metaphor can provoke insights about the problem domain that might be difficult or impossible to achieve through direct analysis. But there is a risk that in embracing the metaphor, we will lose sight of the places where metaphor and reality depart. Based on the metaphor, we may believe things about the program that are not necessarily true.

The balance of this essay proposes the adoption of the metaphor of a building code as a framework to capture what we know about how to build software that can weather attacks and as a vehicle to put that knowledge into practice where it counts. But first, it considers briefly the merits of some metaphors currently in wide use for software and computing systems.

2. METAPHORS IN USE TODAY

If we think of a metaphor as a sort of mapping from a domain we know something about – a source domain – to another domain we are less certain of – the target domain, the metaphor may help us understand the target domain if (1) the relationship captures an essential aspect of the target, (2) it hides irrelevant details of the target, and (3) reasoning in the source domain yields results in the target domain that remain valid.

The well-known story of the six blind men examining the elephant exemplifies metaphors that fail the second and third of these tests. Each of the examiners creates his own metaphor for the beast based on the particular part of the animal he is exposed to: the one at the tail thinks the elephant is a rope, the one at the trunk thinks

ACSAC '13, December 09 - 13 2013, New Orleans, LA, USA

http://dx.doi.org/10.1145/2523649.2530278

it is a snake, the one at the tusk thinks the elephant is a spear and so on.

Metaphors have been used to explain computer and information security problems to people for a long time. We assess several commonly used ones below.

2.1 Trojan Horse

Perhaps the oldest metaphor in computer security is the Trojan horse. The story originates in Homer's Iliad, in which the Greeks appear to admit defeat and abandon the field, leaving behind what seems to be a trophy to the Trojans: a large wooden replica of a horse. The Trojans move the horse inside the city walls. But the Greeks have concealed a few men inside it who then escape the following night and open the city's gates, allowing the Greek army to invade and slaughter the inhabitants.

In the computer security context, the earliest use I have found of this metaphor is in the Anderson Report in 1972, where the identification of this kind of attack is attributed to Dan Edwards of the NSA [2, p.62]. In the computing context, the Trojan horse is a program that provides a function appealing enough that a user (or administrator) is willing to install it even though its internal details are not known. Once activated in the victim's computing context, the Trojan horse program takes advantage of the privileges of that context to perform whatever functions its author built into it, possibly including downloading additional malicious software, for example.

This metaphor seems to work pretty well. The story is widely understood, the metaphor captures an essential aspect of the target domain – installing a dangerous component inside a security perimeter, and reasoning about what the Trojans might have done to avoid disaster carries over reasonably well to the computing domain.

2.2 Worm

The original use of "worm" in a computing context apparently comes from the novel *Shockwave Rider*, published by John Brunner in 1975 [3]. As used in that story, the worm is a (virtual) tapeworm and thus a parasite. In 1982, John Shoch and Jon Hupp implemented a worm at Xerox PARC to take advantage of unused computing cycles on a distributed set of machines [4].

The term is now defined in Internet RFC 1135 [5] as follows:

> A "worm" is a program that can run independently, will consume the resources of its host from within in order to maintain itself, and can propagate a complete working version of itself on to other machines.

Again, the notion of biological worms, including tapeworms, is widely understood. Biological parasitic worms may require alternate hosts to propagate, and computational worms may also reflect that aspect. So it seems the metaphor does capture essential aspects of the target domain. It definitely hides many inessential details, and reasoning about biological parasitic worms seems to carry over reasonably well in the computational domain: worms consume host resources, can propagate to other systems, and can be difficult to eradicate.

2.3 Virus

The precise origins of the virus metaphor for a particular kind of software (today malware) are a but murky. David Gerrold's science fiction novel, *When HARLIE Was One* [6], published in 1972, is said to include "one of the first fictional representations of a computer virus"[7]. The earliest use of the term in the technical literature is a paper by Fred Cohen in 1984 [8]. Again drawing on RFC 1135:

> A "virus" is a piece of code that inserts itself into a host, including operating systems, to propagate. It cannot run independently. It requires that its host program be run to activate it.

Viruses are a widely understood biological phenomenon, and as the definition above indicates, the computational version displays the ability to infect and modify the behavior of the host system but depends on mechanisms in the host for replication, as the biological version does. As biological viruses sometimes mutate to form strains that resist prior treatments, computational viruses have developed (albeit with human assistance) means of resisting computational countermeasures. A new strain of virus, biological or computational, may require new detection mechanisms and new cures. So this metaphor seems apt.

2.4 Firewall

Physical firewalls are designed to prevent, or at least delay, the propagation of a fire between parts of a building. The International Building Code includes the following definition:

> FIRE WALL: A fire-resistance-rated wall having protected openings, which restricts the spread of fire and extends continuously from the foundation to or through the roof, with sufficient structural stability under fire conditions to allow collapse of the construction on either side without collapse of the wall.

The Anderson report [2] actually used "firewall" as a description for the barriers an operating system should provide between different user domains in a time-sharing system in 1972, but the term gained its modern meaning with the advent of internet packet filters in the late 1980s and early 1990s. By the time Bellovin and Cheswick's classic book [9] appeared in 1994, it was in wide use.

Unfortunately this metaphor has some serious problems. As noted above, conventional firewalls are there to stop pretty much anything, particularly fire, from penetrating them. Internet firewalls aim to stop only the traffic they can detect as evil and to let everything else pass through – so their fundamental purpose is to provide communication, not to stop it. Indeed, firewalls barely slow down a capable attacker, and this has been true for a long time. So this seems to be a case where the metaphor, though widely used, has fooled many people into thinking this component provides a much greater degree of protection than it can achieve in fact. A propped-open firedoor, perhaps manned by a sleepy attendant, might be a better visualization of the operation of these components.

2.5 Public Health

Cybersecurity is frequently described using the terms of public health. This metaphor fits well with the virus and worm metaphors. For example,users and system administrators are admonished to observe proper "hygiene.". Systems hosting malware are "infected." Large clusters of machines should be "immunized" so they will display "herd immunity," and if they are identically configured they may represented a vulnerable "monoculture." There have even been calls for creating a cybersecurity version of the US Center for Disease Control to monitor malware outbreaks and provide immunizations.

In general, this metaphor works well according to the criteria we have been using. Everyone understands public health and a good deal of the reasoning one might follow in the public health domain will not lead you astray in the cybersecurity domain.

However, it may have the side effect of making people think they are dealing with a natural system, one in which they can't easily alter the infrastructure (cells, organisms) and can only react. Cybersecurity resides in artificial, engineered systems and the threats against it are intentional and man-made. Regarding it as a natural system may steer us away from engineered solutions that would be much more difficult to accomplish in natural systems.

2.6 Cloud

Although "cloud" computing is not specifically a cybersecurity-related metaphor, its widespread use requires a comment. Where did we get the idea that computing is somehow similar to large volumes of water vapor flitting across the sky? I don't know for sure, but I suspect this metaphor gestated in the countless slides (including many I have shown myself) in which a large collection of network links and nodes representing computers and communication paths was drawn as a fuzzy cloud in order not to have to represent the full details and complexity of the network it was to represent. Data sent from one system to another would leave its source, enter the network "cloud," and emerge from some other part of the cloud to be delivered. It is a short step from having the cloud represent a network to having it represent the attached computing resources as well.

But is this a helpful metaphor? What would security be for a cloud? To a meteorologist, a cloud may be a complex, structured object, but I doubt whether reasoning about meteorological clouds will yield much insight about computational ones.

3. BUILDINGS AND BUILDING CODES

The metaphor I want to promote is that of software systems, and more broadly, computing systems, as buildings. It's hardly a new idea. Computers are designed objects, artifacts, and people have written about the organization of both the physical machine structure, the instruction set, and the organization of the software running on the machine in terms of "architecture" for a very long time, if not from the beginning. Returning to *The Mythical Man-Month*, one of Brooks's key points is the need for conceptual integrity in system design, and he explicitly draws parallels with the design of European cathedrals to illustrate his points [1, p. 42 ff.].

Reasoning about building construction doesn't of course carry directly over to software, but the parallels are significant and useful. Physical buildings of any size require design documents and specifications. They must tolerate natural phenomena (gravity, wind and weather, earthquake, fire), they are subject to inspection during construction, and they are tested before they can be occupied. Software and computing systems (perhaps especially cyber physical systems) are specified, designed, implemented, inspected, and tested. They must tolerate the perils of the environment in which they are intended to be used.

But aren't software and computing systems much more complex and much more dynamic than buildings? Yes. Nevertheless, if we are to have confidence that a software system meets its requirements, as we surely want to have in systems on which critical infrastructure relies, those systems must have a structure and a mode of accommodating change that we can understand and reason about.

Throughout the world, the primary mechanism that has arisen to assure that buildings and collections of structures are safe and useful to their occupants and communities is the building code. Building codes in general specify constraints of various sorts on how a building may be constructed. They can incorporate all kinds of requirements relating to design, construction materials, gas, electrical, and plumbing systems, and more. They are not generally drafted by governments but rather by professional societies of various sorts, motivated by a variety of interests. They are periodically updated as society, technology and risk perceptions change. They gain the force of law only when they are adopted by municipalities, who may choose to adopt a model code directly or with modifications motivated by local interests.

I argue that the notion of a building code, and particularly a building code for critical infrastructure software security, is one that modern society needs to adopt in order to assure that future generations will have a cyberinfrastructure that can meet the demands society is imposing on it. Such a code can provide a way for us to capture what we have learned about how to build and how to inspect software to support critical needs in the face of attack. It can be developed incrementally and can be adopted where needs are most urgent. It can be tailored to domains where critical functions and threats differ. It can be updated as our understanding improves, as better methods are developed.

Further, I want to persuade you to take an active role in helping to develop such a code. To help convince you of the importance of this task, I first provide some historical context about buildings and building codes.

3.1 Buildings and Foundations

Everyone knows about the great pyramids of Giza, outside of Cairo. There is a somewhat less famous pyramid about 30 miles south of Cairo called the Bent Pyramid, built by Sneferu about 2600 BCE. The name comes from the fact that the lower part of the pyramid rises at an angle of about 54 degrees, but the top section rises at a shallower angle of about 43 degrees, giving the pyramid a "bent" aspect. The reason for the change in angle is thought to be that another nearby pyramid being built at the steeper angle collapsed while this one was under construction, causing a change of plan.

The pyramids, the Acropolis, the Roman Forum, and most of the great cathedrals of Europe were built before Galileo and Newton laid the foundations for modern physics. Today we are building computing systems of unprecedented complexity and connectedness, but we are mostly building them without the benefit of scientific foundations as useful and strong as Galileo and Newton provided for mechanics. Instead, we build systems, see if they (or similar ones under construction) fall down, revise, and repeat.

About 800 years after Sneferu, Hammurabi's famous code of laws included what we might consider the first building code:

> §229 If a builder build a house for someone, and does not construct it properly, and the house which he built fall in and kill its owner, then that builder shall be put to death. [10]

This would be in the category of a "performance code" today: it doesn't tell you how to build the building, but if it doesn't stand up, you are liable.

At this writing, we are somewhere between Sneferu and Hammurabi with respect to building codes and liability for software and computing systems in general. There are efforts in progress to try to develop more scientifically rigorous foundations for software and security engineering, but system construction proceeds without them. Software producers have so far avoided general liability for their products and systems, though the advent of cyber physical systems may bring change in this area.

3.2 How Do Building Codes Arise?

The creation of building codes seems to be stimulated by disasters. Here are a few disasters of different kinds and how they affected building codes that I've been able to glean from the worldwide web.

3.2.1 Fire: London, 1666

The great fire of London, documented in Pepys diary, burned from September 2-5, 1666 and destroyed some 430 acres, approximately 80% of the city, including 13,000 houses, 89 churches, and 52 guild halls [11]. The fire led to the London Rebuilding Act of 1666 and further legislation that aimed to limit new construction to be faced with brick and imposed other measures designed to reduce the likelihood of large fires [12]. These acts are commonly cited as the earliest laws regulating construction in London, although it is also reported that thatched roofs were banned as early as 1212, again to reduce the danger of fire [13]. Reading the earlier act, it appears that enforcement was not strict.

3.2.2 Earthquake: Santa Barbara, 1925

A month after the disastrous San Francisco earthquake of 1906, scientists and engineers banded together to form the Structural Association of San Francisco and concluded that well-braced wooden buildings secured strongly to their foundations could have withstood that quake. Although (perhaps for economic reasons) the city fathers did not add specific earthquake-resistance requirements to the building code, ordinances were passed approving the use of reinforced concrete and requiring steel framing in any new brick construction In June, 1925, Santa Barbara suffered a severe earthquake that leveled most of its downtown and led to the first earthquake provisions in any California municipal building code. [14, 15, 16].

3.2.3 Hurricane: Okeechobee 1928

Miami was hit by a powerful hurricane in 1926, and another Category 4 hurricane struck further north in 1928, in the Lake Okeechobee area. The storm caused thousands of deaths along with widespread and severe property damage. Buildings made of brick, stone, or concrete survived better than others and those with shutters had much less damage to windows. These observations led to stronger building codes [17,18]. It is worth noting, however, that in the aftermath of Hurricane Andrew in 1992, more than a half century later, deficiencies in building codes and enforcement remained a major issue in South Florida [19].

3.2.4 Construction errors: Kansas City, 1981

The collapse of a suspended walkway in the atrium of the Hyatt Regency Hotel in Kansas City killed 114 people and injured more than 200 others in July 1981. The cause was ultimately determined to be a change from the original design in the way the walkways were suspended. Instead of a single rod bearing two levels of walkways, the lower walkway was suspended from the upper walkway. This change, proposed by the steel company building the structure and approved by the engineering firm responsible for the design, led directly to the disaster. The investigation concluded that the fundamental problem was lack of proper communication between the designer and the steel company. The responsible engineers were convicted of gross negligence and unprofessional conduct; they lost their engineering licenses and their memberships in the American Society of Civil Engineers. The engineering firm was not found criminally negligent, but it lost its license to be an engineering firm [20,21]. As it turned out, even the original design was deficient with respect to the local building code requirements.

3.2.5 Malicious Attack: Oklahoma City Bombing

The bombing of the Alfred P. Murrah federal building in Oklahoma City, April, 1995, is reported to have destroyed or damaged 324 buildings and claimed 168 lives. The effect on building construction was first to trigger the installation of Jersey walls (add-on security) to many existing Federal buildings and to add new requirements for Federal buildings that they have deep setbacks from surrounding streets to reduce vulnerability to truck bombs. Other recommendations for design of new Federal buildings drew on features also used for earthquake protection [21,22].

3.2.6 Discussion

The preceding examples illustrate how disasters can stimulate the creation of building codes and other kinds of regulations that aim to assure the safety of public and private structures in the face of hazards. However, as is well-documented in [16] with respect to California earthquakes, it often takes repeated disasters and diligent work by safety advocates to stimulate public policy. The history of hurricane damage in Florida illustrates the importance not only of having codes but of enforcing the codes that are on the books. For those interested in more details, Robert Ratay provides several specific and relatively recent examples of structural failures (including one bridge construction failure on the Baltimore-Washington Parkway near the National Security Agency) that have triggered changes in codes, standards, and practices in structural engineering [23,24].

3.3 Building Codes Today

Several kinds of building codes are in use in the United States and around the world today. The Uniform Plumbing Code, which originated in Los Angeles in 1948, published its fourth edition in 2012. There is also a National Standard Plumbing Code, first published in 1933, which is updated annually by the Plumbing, Heating, and Cooling Contractors (PHCC) Association. These codes are being reshaped today to enable and control graywater use. There is a National Electrical Code published by the National Fire Protection Association (NFPA) and updated every three years. There is also an International Building Code (IBC) which was established in 1994 as an organization dedicated to developing a single set of comprehensive and coordinated national model construction codes. In general, these are "model" codes which gain the force of law only when they are adopted by states, regions, or municipalities to govern construction within their jurisdictions. They may be adopted in whole, in part, or with modifications to suit local needs. The latest versions of the codes, like many industrial standards, are public but not available for free; the 2012 IBC is available electronically for about $100 at this writing. Older versions of the codes may be found online at no charge [25]. Public interests, insurance companies, building trades, architects, engineers, and builders all participate in the process of creating and updating these codes.

3.4 Building Code Enforcement

Building codes typically are adopted by municipalities or other civil jurisdictions and in this way can gain the force of law. Construction of a building cannot begin until a building permit is issued by the local authorities, and the permit will not be issued without a set of plans that have been stamped and signed by a licensed architect or professional engineer, who is expected to assure the plans conform to the applicable codes. A building

cannot be occupied until it passes inspections carried out by employees of the governing entity throughout the construction process to assure that it satisfies the applicable building codes. A building inspector requires some training, and care must be taken to control conflicting interests of the builder, inspector, owner, and community.

3.5 Building Codes and Security

As the examples above illustrate, building codes have been motivated more by safety concerns than security. Of course the structure of forts and castles have always had security against physical attack as a primary consideration (and touring them with an eye to the security measures is fascinating) but resistance against intrusion or physical attack has not generally been a primary concern for modern building codes. Indeed, the placement of large boulders and Jersey barriers in front of public buildings following the 911 attacks provides graphic (and concrete!) examples of add-on security.

4. SOFTWARE SYSTEMS AS BUILDINGS

Software has long been described in architectural terms. As noted above, Brooks uses the design of European cathedrals to explain what he means by the conceptual integrity of a software system. Further, he discusses the architecture of software systems and the importance of the role of the system architect [1, p. 41 ff,]. Like many homes today, much software is designed by the builder and assembled from components drawn from diverse and little-examined international supply chains, without the benefit of an architect.

Software is often described in terms of layers, with the hardware instruction set providing the foundational layer, providing support for the higher layers, just as the foundation of a building supports the entire structure. Already in 1968, Djikstra described the T.H.E. multiprogramming system in terms of hierarchical layers, with each new layer building on the layers below [26].

As Parnas subsequently observed [27,28], software can be described in terms of different kinds of hierarchies, including the "uses" hierarchy, resource ownership and allocation hierarchies, protection hierarchies, and more. Buildings too can be said to display several kinds of hierarchies. Beyond the obvious ordering of floors, the heating, ventilating and air-conditioning systems, the electrical systems, the plumbing systems, the security systems, and others typically display a branching structure in which some components are foundational and others depend on them to provide services to the occupants.

Today we speak of cyberspace as a place in which we spend time. The many software systems that make up this place need protections analogous to those our physical dwellings require: systems that can detect intruders, safe escape routes in case of natural disaster (perhaps an earthquake may be likened to a hard drive crash), means to restore the structure and contents when a disaster occurs. Further, the roof and walls need to be kept in good repair and patched when cracks appear.

5. SOFTWARE BUILDING CODES IN USE TODAY

Some systems that include substantial software control have been subject to regulatory control for many years. The National Academy of Sciences (NAS) report *Software for Dependable Systems: Sufficient Evidence?* in 2007 reviewed then-current certification practices for avionics software, medical software, and security [29]. These are perhaps the closest things to building codes for software that are currently in use.

5.1 FAA

The Federal Aviation Administration (FAA) certifies aircraft for flight safety and the software affecting flight safety is included in this certification. A special committee (SC-145) of the Radio Technical Commission for Aeronautics (RTCA) first developed a document, *Software Considerations in Airborne Systems and Equipment Certification*, document DO-178, in 1982. Since then it has been updated twice, as DO-178B in 1992 and DO-178C in 2011. The NAS report notes "At least in comparison with other domains (such as medical devices), avionics software appears to have fared well inasmuch as major losses of life and severe injuries have been avoided," although it goes on to observe that the basis for some of the required testing procedures seems to be poorly justified and that static analysis of the software revealed many remaining "serious, safety-related defects" [29, pp. 34-35].

5.2 FDA

Medical software, as noted in [29], is less uniformly controlled than avionics software. The Food and Drug Administration (FDA) provides "guidance" for software validation [30] that draws on standard software engineering approaches (up to 2002, when it was issued); although the guidance is not binding it does bear considerable weight. Recently, following a number of demonstrations of security vulnerabilities in medical devices, the FDA has issued its first draft guidance for management of cybersecurity in medical devices [31]. While it includes conventional kinds of guidance for authentication, validation of updates, risk analysis, and the like, it does not address software development practices.

5.3 Security

Of course there is a great deal of experience in the security community with evaluation/certification of software and hardware systems. Some of the other sessions at this conference are celebrating the 30th anniversary of the first release of Trusted Computer System Evaluation Criteria (TCSEC, the "orange book") [32]. Today we live under the Common Criteria [33] and there are separate standards for certification of hardware/software cryptographic modules [34]. While these documents contain much that is valuable from a technical standpoint, it is hard to consider the programs around them as successful overall. The original TCSEC development was undertaken as part of a strategy to encourage vendors to build at least a moderate level of security into their normal product lines so that the government might build on them to reach higher levels of assurance. This strategy didn't succeed for a number of reasons [35]. The Common Criteria scheme has been criticized because it tends to be applied after the fact (a criticism that also applies to earlier TCSEC product evaluations targeted at levels of B2 and below) and focuses primarily on specific security functions rather than the system as a whole. Instead of developing the required documentation as part of the development process, vendors often prefer to hire a third party to prepare those documents for the evaluation laboratories and to handle the evaluation process. "[B]ecause the certification process at economically feasible evaluation levels focuses on the functioning of the product's security features even while real vulnerabilities can occur in any component or interface, real-world vulnerability data show that products that have undergone evaluation fare no better (and sometimes worse) than products that have not" [29, pp. 31-32].

5.4 BSIMM and OpenSAMM

The current "Building Security In Maturity Model" (BSIMM) grew out of an effort to survey practices in commercial firms engaged in software development that have undertaken software security initiatives [36]. The report's authors have surveyed firms and recorded their practices in five rounds, so the current report, from 2013, is referred to as BSIMM-V and includes results from 67 organizations...

As the work developed, the authors identified a set of twelve "practices" organized into four domains (governance, intelligence, secure software development lifecycle (SSDL) touchpoints, and deployment). Practices include, for example compliance and policy, attack models, code review, and penetration testing.

The authors make it clear that this is a descriptive, not prescriptive, activity. They observe what practices are in use and by recording them and providing statistical summaries, they provide a yardstick against which firms can compare their practices with others. The authors do not attempt to measure the effectiveness of the practices undertaken.

The Open Software Assurance Maturity Model (OpenSAMM), developed under the Open Web Application Security Project (OWASP) also identifies a set of twelve practices in its guiding document [38]. The goal of the effort is to "help organizatioins formulate and implement a strategy for software security that is tailored to the specific risks facing the organization" [38,p.3] The approach seems slightly more normative than in BSIMM, in that the basis for the specified practices and maturity levels seems to intuition and general observation/experience rather than observation of particular practices. Nevertheless, there seems to be a good deal in common between the practices prescribed in [38] and documented in [37], and neither of the efforts attempts to assess effectiveness of the practices.

5.5 NIST Cybersecurity Framework

Following the failure of Congress to pass proposed cybersecurity legislation in its 2012 session, the Obama administration pledged to act within its existing authorities to improve cybersecurity in critical infrastructure systems. As part of this effort, the administration issued an Executive Order in February, 2013 [38]. Section 7 of the order directs the National Institute of Standards and Technology (NIST) to "lead the development of a framework to reduce cyber risks to critical infrastructure (the "Cybersecurity Framework")." The framework is to include

> "a set of standards, methodologies, procedures, and processes that align policy, business, and technological approaches to address cyber risks. The Cybersecurity Framework shall incorporate voluntary consensus standards and industry best practices to the fullest extent possible. …"

A preliminary version of the framework was released for public comment by NIST on Oct. 29, 2013; a final version is due out in February, 2014.

At this writing, the most recent draft framework is organized around the core notions of "Identify (assets), Protect, Detect, Respond, Recover" [39]. It is too early to comment on the outcome of the effort, but several public meetings have been held, and it seems clear that the basis for the framework will be limited to methods currently in commercial use. Even if the final framework calls for *best* current practice, current practice is what has led us to the current state. Bringing critical infrastructure software up to "best current practice" would be laudable but it seems unlikely to be sufficient to deal with the threats evident today.

6. WHAT ELEMENTS WOULD A BUILDING CODE FOR SOFTWARE ENTAIL?

Suppose we accept the utility of the architectural metaphor for software systems. Can thinking about the codes created to control buildings help us identify the elements needed for creating a code to control the development and deployment of software with desired security properties? Building codes are typically concerned with maintaining public safety, health, and welfare. General areas of concern include:

- structural integrity: building integrity must be maintained in the face of hazards of its location, such as high winds, heavy rains, lightning, earthquakes
- fire safety: prevention, detection, limits on propagation, safe exit, and in larger structures, support for firefighters
- physical safety of the occupants: door and stairway dimensions and handrails, spacing of balusters
- water use: plumbing regulations, sufficiency of supply and sewer/septic, effective sanitation, water conservation
- energy use: insulation, heating systems, lighting, electrical power, energy conservation

Correlates for a building code for software security might include:

- Structural integrity: requirements on software and system integrity, tamper resistance ability of system isolation mechanisms to resist attack
- Fire safety: use of "fireproof" materials (e.g., safe coding standards), domain separation to limit propagation of attacks, intrusion detection, recovery mechanisms
- Physical safety: avoid the equivalent of sharp corners and unprotected drop-offs in the security architecture: be sure security mechanisms are easy to use and understand
- Water and energy usage: information flow control mechanisms, limitations on resource usage for security functions, protection against system becoming a source of denial-of-service attacks (exerting excessive resource demands on networks)

Constructing an actual building code for building code is far beyond the scope of this paper. Moreover, the construction of such a code must inherently be a group activity, something that builds consensus gradually among various stakeholders.

Given the effort that has been expended over the past decades on the Orange Book and the Common Criteria, what would be different about such an effort? Would it just yield a relabeling of the voluminous documents we've already created? I don't think it has to be that way, though it makes sense to leverage activities like BSIMM and the nascent NIST cybersecurity framework as much as possible.

The NRC report on software for dependable systems [29], advocates the development of explicit dependability claims for systems and the development of evidence and arguments based on

the evidence, to support a dependability case for a system[1]. Although the report observes that "it makes little sense to invest effort in ensuring the dependability of a system while ignoring the possibility of security vulnerabilities" [29, p. 19], it in general excludes security concerns from its scope. But it makes one other observation of particular interest here:

> "As is well known to software engineers (but not to the general public), by far the largest class of problems arises from errors made in the eliciting, recording, and analysis of requirements. A second large class of problems arises from poor human factors design…
>
> "Security vulnerabilities are to some extent an exception; the overwhelming majority of security vulnerabilities reported in software products – and exploited to attack the users of such products – are at the implementation level. The prevalence of code-related problems, however, is a direct consequence of higher-level decisions to use programming languages, design methods, and libraries that admit these problems. In principle, it is relatively easy to prevent implementation-level attacks but hard to retrofit existing programs."

The TCSEC and CC generally approach security from the top down: define the security policy, apply it to the specifications, identify security functions, provide assurance that the security functions work as intended. It's a logical approach. But the attacks that we suffer from generally don't aim to pick the locks on the doors in our systems; rather they probe at the weak building materials in the walls that weren't part of the security argument at all. They submit inputs that, because of low-level implementation errors, entirely change the transition function of the system and then open the doors from the inside.

This observation that the vast majority of the exploitable and exploited vulnerabilities in today's systems are not the result of requirements or design flaws, but simple implementation oversights, seems to offer a direction for a building code for critical infrastructure software. At least use development practices that minimize, and, where possible, rule out these kinds of flaws. Use programming languages that make buffer overflows infeasible. Use static and dynamic analysis techniques on software for which source code is available to confirm the absence of whole classes of flaws. The software won't be perfect but it will be measurably better.

One of the advances in the past decade has been the incorporation of hardware in a wide range of systems to provide a root of trust, yet today that hardware is vastly underused. It would seem to make sense for a building code to require that code for critical infrastructure systems take advantage of built-in mechanisms such as Trusted Platform Modules to help assure at least the initial integrity of software loads.

We need to be wary also of creating a code that requires masses of highly trained building inspector equivalents. That's where we ended up with the TCSEC, and the Common Criteria don't seem to have helped a great deal in this respect. Our building code should not try to do a great deal more than can be done with automated support. It can evolve over time as technology advances in this respect.

[1] Similar approaches have been explored with respect to security [40] but not widely adopted.

We also need to recognize that we may want to use software that was developed by others and for which source code may not be available. We need to understand what we can assure about such software and how to use it without depending on properties we can't assure. That may mean certifying the strength of walls we build around it, or providing alternative means of computing the same result as a check, for example.

There is a great deal more to be said about what could and what should be put into a building code for critical infrastructure software. I don't pretend to have all the knowledge and wisdom to create such a code myself; moreover this seems to me inherently a group activity, as already noted.

7. CALL TO ACTION

We are told there are tens of thousands of jobs awaiting cybersecurity specialists, and we need to act quickly to train individuals for them [41]. I suspect most of these are not jobs creating new and innovative software products, but rather service jobs for people who configure systems, install patches, monitor systems for attacks, and conduct forensic investigations afterwards. In some cases they are jobs developing attacks on the systems of others. There is nothing dishonorable about these jobs, but they provide graphic evidence of the poor level of software security and system security engineering that we have come to accept in today's commercial products.

We do not lack the means to build systems that are much less vulnerable to attack, but those means will only be applied when those in a position to apply them have the incentive to do so. It is essential to provide that incentive where the systems being delivered underpin our society. Perhaps the Congress will find a legislative means to provide that incentive, but today's highly polarized political climate and industry's ingrained resistance makes that a difficult path. Perhaps the executive branch can use existing regulatory mechanisms to move in that direction, but its tools are weak and the commercial world strongly resists them. Perhaps lawsuits can eventually establish some liability through the courts, but lawsuits are only effective after serious damage has occurred and it would surely be better to avoid disasters than to rely on them to change policy.

The technical community can begin to act on its own by creating a credible building code for critical infrastructure system software that could be adopted by industry and perhaps eventually given legal force through government adoption. Industry can and should participate fully in the development of such a code, which can provide a credible means of self-regulation. The security technical community should collaborate with the software engineering and software quality/dependability/safety communities in this effort, but we should not wait for them to get started. If disasters do occur, this code should be available for adoption.

Another mechanisms the technical community could create would be the equivalent of a board for investigating structural failures. When major cybersecurity failures occur, we need a respected group of technologists who can investigate what happened and extract the lessons engineers of future systems need to learn from those failures. Steve Bellovin has already proposed something similar to this [42]

Research has an important role to play in this process. Ideally, there should be a basis in theory or experiment for each requirement put into the building code. In practice, I suspect there will be substantial consensus on requirements for which the research basis is weak or lacking. I wouldn't omit such agreed-upon practices or architectural features from the code but they

should be recognized as needing empirical or theoretical validation, and the research community should take up the challenge of providing sound evidence to illuminate the issue.

Regulation of any sort is often seen as a brake on technology, an overhead that perhaps must be tolerated, but should certainly be minimized. I want to close with some evidence to the contrary from the *Wall Street Journal*. This conservative bastion recently published a review of a new, fuel-efficient luxury hybrid car with the following paragraphs:

> "The specimen in question is a luxury sedan of goodly size and weight, around 3,700 pounds, with 200 hybrid horsepower, generous leather seats, big audio, exceptional soundproofing, and a cabin filled with fine joinery and brushed alloy trim. All in all, a handsome and sophisticated presentation. This car accelerates from zero to 60 miles per hour in around eight seconds and breaks the beam in the quarter-mile in about 16 seconds, certainly quicker than the last Jaguar E-type I drove.
>
> "This quite uncompromised car … averages 40 miles per gallon in mixed driving, according to the EPA.
>
> "That's astounding. And it's not even the most fuel-efficient car in its class … This yearling moose of a car gets about the same fuel economy as a Ford Fiesta and, if you didn't know [it] was a hybrid, you wouldn't guess.
>
> "You know what's even more astounding? Recall the legions of entrenched industry forces who, two decades ago, swore on their professional lives that increased fuel-economy standards would drive up the cost of automobiles while making them boring and less safe. Yeah, that didn't happen at all.
>
> "In fact, the opposite has happened. Cars have gotten more fuel efficient and more powerful, and quite measurably safer in every type of collision… And, by the way, the car business is booming."[43]

My point is not that you should buy this car, but that regulation, including self-regulation, does not have to imply a drag on technology and creativity. It can, and in this particular case, seems in fact to have served exactly as a spur to creativity, providing engineers and companies the incentive to move their technology in directions that might otherwise have been unexplored and are now providing substantial benefits to society and to the companies that took up the challenge.

I think we should strive to provide the same kind of creative stimulus to the engineering of secure software that we provided to the engineering of fuel-efficient cars, and that if we do, our creative engineers can provide the same kind of results.

I am not naïve enough to think that this will be an easy process or that it will result in systems with perfect security. But we can and must do better, much better, particularly in the area of critical infrastructure software, than we have done to date.

There is no reason we need to view security vulnerabilities as an inevitable disease against which we must routinely invest in weekly or monthly flu shots. These vulnerabilities are engineering defects that have grown to the point where they provide a fertile field for organized crime and a substantial lever to groups or nations who would steal one another's intellectual property and potentially attack each other's critical infrastructures. As engineers and computer scientists, we have a responsibility to act on our knowledge. I propose that we begin to act by creating a credible building code for building code.

8. ACKNOWLEDGMENTS

I thank Charles Payne and the ACSAC organizers for the invitation that led me to put these thoughts in print. I also thank Hilary Orman, Susan Landau, and Terrry Benzel, organizers of the GREPSEC workshop, where I presented some initial ideas on this topic, and Bill Scherlis, who chaired discussions on building codes at the 2012 NSF Secure and Trustworthy Cyberspace Principal Investigators meeting. Comments from Jeremy Epstein and Christoph Schuba helped me broaden the coverage and sharpen the analysis. Marvin Michalsen, AIA, provided perspectives on building codes. All opinions and all remaining errors, remain my sole responsibility.

9. REFERENCES

[1] Brooks, F. P., Jr. *The Mythical Man-Month: Essays on Software Engineering*. Addison Wesley, 1975.

[2] Anderson. J. P., ed. *Computer Security Technology Planning Study*. ESD-TR-73-51, Vol. II, AFSC Hanscom Field, Bedford, MA, Oct. 1972 p. 62. Available at: http://seclab.cs.ucdavis.edu/projects/history/papers/ande72.pdf

[3] Brunner, J., *Shockwave Rider*. Harper & Row, 1975.

[4] Shoch, J, and Hupp, J. The "worm" programs – early experience with distributed computations. *CACM 25*, 3 (March 1982) 172-180.

[5] Reynolds, J. The helminthiasis of the Internet. Network Working Group RFC 1135m Dec. 1989. Available at: http://www.ietf.org/rfc/rfc1135.txt

[6] Gerrold, David. When HARLIE Was One. Nelson Doubleday, 1972.

[7] "When HARLIE Was One", Wikipedia article, retrieved 9 Nov. 2013, from http://en.wikipedia.org/wiki/When_HARLIE_Was_One

[8] Cohen, F. Computer viruses: theory and experiments. *Proc. 7th DoD/NBS Computer Security Conference*, 1984, 240-263.

[9] Cheswick. W.R. and Bellovin, S.M. *Firewalls and Internet Security: Repelling the Wily Hacker*. Addison-Wesley, Reading, MA, first edition, 1994. Available for personal use at: http://www.wilyhacker.com/1e/

[10] Harper, R.F *The Code of Hammurabi King of Bablyon*, University of Chicago Press, 1904, p. 81. Available at: http://upload.wikimedia.org/wikipedia/en/4/4e/The_code_of_Hammurabi.pdf

[11] The Great Fire of London, 1666. Luminarium Encyclopedia Project: http://www.luminarium.org/encyclopedia/greatfire.htm

[12] An Act for Rebuilding the City of London, 1666. *Statutes of the Realm, Vol. 5*, 1628-80 (1819), J. Raithby, ed. pp 603-612. Available at: http://www.british-history.ac.uk/report.aspx?compid=47390&strquery=Building%20Act#s5

[13] Regulations for building construction and fire safety, Florilegium Urbanum website. Original source cited as Corporation of London Records Office, Liber Custuarum, f. 52. Translated from Latin, available at: http://users.trytel.com/~tristan/towns/florilegium/community/cmfabr08.html

[14] Earthquake's Impact on Building Codes. Multidisciplinary Center for Earthquake Engineering (MCEER), SUNY Buffalo, web page. Available at: http://mceer.buffalo.edu/1906_Earthquake/industry_impacts/impact-building-codes.asp

[15] Birkland, T. A. *Lessons of Disaster: Policy Change After Catastrophic Events.* Georgetown U. Press, 2006.

[16] Geschwind, C-H. *California Earthquakes: Science, Risk, and the Politics of Hazard Mitigation*. Johns Hopkins U. Press, 2001.

[17] Nelander, B. "The hurricane of 1928: category 4 hurricane scarred Palm Beach," *Palm Beach Daily News*, June 1, 2008. Retrieved from Internet Archive, http://web.archive.org/web/20080920065230/http://www.palmbeachdailynews.com/news/content/specialsections/HURRICANE1928page.html 7 October 2013.

[18] "1928 Okeechobee hurricane." Wikipedia article, retrieved 7 October 2013 from: http://en.wikipedia.org/wiki/Okeechobee_Hurricane#cite_note-27

[19] Bragg, R. "Storm over south Florida building codes." New York Times, May 27, 1999. Retrieved 7 October 2013 from http://www.nytimes.com/1999/05/27/us/storm-over-south-florida-building-codes.html

[20] National Bureau of Standards "Investigation of the Kansas City Hyatt Regency Walkways Collapse". US Department of Commerce. (May 1982).

[21] Perez, A.R. Murrah Federal Office Building. Article in Failures Wiki: Building, Architectural and Civil Engineering Failures and Forensic Practices (overview available at http://failures.wikispaces.com/Home ; this article at: http://failures.wikispaces.com/Murrah+Federal+Building .

[22] Wikipedia entry, "Oklahoma City Bombing," http://en.wikipedia.org/wiki/Oklahoma_City_bombing

[23] Ratay, R. T. "Changes in Codes, Standards and Practices Followign Structural Failures, Part 1: Bridges," *STRUCTURE Magazine*, Dec. 2010, 16-19.

[24] Ratay, R.T. . "Changes in Codes, Standards and Practices Followign Structural Failures, Part 2: Buildings." *STRUCTURE Magazine*, April. 2011, 21-24.

[25] International Building Code, 2009, Sixth Printing. Available at: http://publicecodes.cyberregs.com/icod/ibc/2009/

[26] Djikstra, E. W.,"Structure of 'THE' Multiprogramming System," *Comm. ACM 11*, 5 (May, 1968), 341-346.

[27] Parnas, D.L. "On the Criteria to Be used in Decomponsing Systems into Modules," *Comm ACM 15*, 12 (Dec. 1972), 1053-1058, reprinted in *Software Fundamentals: Collected Papers by D.L. Parnas*, D.M Hoffman and D.M. Weiss, eds., Addison Wesley, 2001

[28] Parnas, D.L. "On a 'Buzzword': Hierarchical Structure," IFIP Congress 1974, North Holland, 336-339, reprinted in *Software Fundamentals: Collected Papers by D.L. Parnas*, D.M Hoffman and D.M. Weiss, eds., Addison Wesley, 2001.

[29] Jackson, D., Thomas, M. and Millett, L. eds., Committee on Certifiably Dependable Systems, *Software for Dependable Systems: Sufficient Evidence?* National Academies Press, 2007. Accessible at: http://www.nap.edu/catalog.php?record_id=11923

[30] US Food and Drug Administration. General Principles of Software Validation; Final Guidance for Industry and FDA Staff. Issued Jan. 11, 2002. Available at: http://www.fda.gov/medicaldevices/deviceregulationandguidance/guidancedocuments/ucm085281.htm#_Toc517237933

[31] Content of Premarket Submissions for Management of Cybersecurity in Medical Devices - Draft Guidance for Industry and Food and Drug Administration Staff. Issued June 14, 2013. Available at: http://www.fda.gov/medicaldevices/deviceregulationandguidance/guidancedocuments/ucm356186.htm

[32] Department of Defense Trusted Computer Security Evaluation Center, DOD 5200.28-STD, 1985. Available at: http://seclab.cs.ucdavis.edu/projects/history/papers/dod85.pdf available at http://csrc.nist.gov/publications/history/dod85.pdf when US government operating normally.

[33] The Common Criteria for Information Technology Security Evaluation, Part 1, Version 3.1, Rev. 4, September 2009. CCMB-2012-09-001. Part 2, Security Functional Requirements, Part 3, Security Assurance Requirements. All Available at: http://www.commoncriteriaportal.org/cc/

[34] National Institute of Standards and Technology (NIST). Security Requirements for Cryptographic Modules. Federal Information Processing Standards (FIPS) Publication 140-2. May 25. 2001.

[35] Landwehr, C. "Improving information flow in the information security market," in *Economics of Information Security*, L. Jean Camp and S. Lewis, ed., Kluwer, 2004, pp. 155-164. Available at: http://www.landwehr.org/Carl%20E.%20Landwehr/Publications.html

[36] McGraw, G., Migues, S., and West, J. *Building Security In Maturity Model: BSIMM-V*, Oct. 2013. Available at: website. http://bsimm.com/

[37] Chandra, Pravir. *Software Assurance Maturity Model., Version 1.0.* Downloaded 9 Nov. 2013. An OWASP project, available at: http://www.opensamm.org

[38] Obama, B., *Improving critical infrastructure cybersecurity*. Executive Order 13636, February 12, 2013. Federal Register Vol. 78, No. 33, Feb. 19, 2013. Available at: www.gpo.gov/fdsys/pkg/FR-2013-02-19/pdf/2013-03915.pdf

[39] Improving Critical Infrastructure Cybersecurity: Preliminary Cybersecurity Framework. NIST, October 29, 2013. Available at: http://www.nist.gov/itl/upload/preliminary-cybersecurity-framework.pdf

[40] Park, J., Moore, A., Montrose, B., Strohmeyer, B., and Froscher, J. A methodology, a language, and a tool to provide security assurance arguments. NRL/MR/5540-02-8600, Naval Research Laboratory, Feb., 2002. Available at: www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA399505

[41] *Cyber Security Jobs Report*. Abell Foundation and Cyber Point LLC, January 8. 2013. Available from: http://www.ctic-baltimore.com/report.html

[42] Steven M. Bellovin. The major cyberincident investigations board. *IEEE Security & Privacy*, 10(6):96, November-December 2012.

[43] Neil, D. "Lexus ES 300h: A Smooth, Elegant Guilt Eraser," The Wall Street Journal, March 16-17, 2013, p. D1