Turtles All The Way Down:* A Clean-Slate, Ground-Up, First-Principles Approach to Secure Systems

Sean Peisert
UC Davis and Berkeley Lab
California, USA
peisert@cs.ucdavis.edu

Ed Talbot
UC Davis
California, USA
edward.talbot@gmail.com

Matt Bishop
UC Davis
Calfornia, USA
bishop@cs.ucdavis.edu

ABSTRACT

In this paper, we present a set of security requirements for critical systems, fundamental premises that those requirements would entail, and ideas for implementations that would instantiate those premises. We discuss the overriding requirement guiding our paradigm: that "first principles" reflects the only real security strategy, where first principles are ideally provable, often measurable; and at minimum, possible to order and bound. These principles allow us to take into account that many security policies may be even be in conflict, and as such, proofs, measures, and ordering gives an analyst (or even better, an automated system) the metrics that one needs in order to make informed decisions about how to resolve conflicts. We demonstrate several metrics that enable this, including state replication, data slicing, collusion, and information theory.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection; H.1 [**Information Systems**]: Models and Principles

General Terms

Design, Management, Measurement, Reliability, Security, Verification

Keywords

Assurance, continuous authentication, dynamic access control, fault tolerance, first principles, formal methods, identity binding, insiders, metrics, perfect privacy, probabilistic computing, security policy, trust negotiation

*With appreciation to Stephen Hawking [18].

NSPW'12, September 18–21, 2012, Bertinoro, Italy.

1. INTRODUCTION

> *As cocaine dealers, crane operators, and tweeting politicians know, every venture involves risk. You can't eliminate it, so you limit your exposure. Drug dealers hire drug mules; politicians hire publicists. Banks hire risk managers.* [37]

Security is based on risk management. Risk management implicitly involves a tradeoff between the magnitude of the risk and the cost of managing that risk. Some critical systems have risks of virtually unlimited magnitude. Modern tools of computer security, including firewalls [44] and anomaly detection systems are inadequate to defend against threats posed against those systems. While false positive and false negative rates of anomaly detection systems can be adjusted in concert with the value of the resource they are defending, anomaly detection cannot efficiently detect non-secure behavior on most computing systems and networks [15, 53]. Also, metrics for anomaly detection systems are notoriously unreliable [2, 38]. Although it is possible to estimate the value of "zombie computers" and the value proposition of running a spam campaign to sell fake Rolexes and Prada handbags [33], what is the value of a nuclear reactor or a missile guidance system? The economic argument fails when the value is virtually limitless or no meaningful value can be assigned. *It is a fantasy* to think that our current security methods have any chance of protecting such systems [51, 52].

This fantasy is protected and promoted by an elaborate and pernicious mythology based solely on existing practice [56]. Security analysts, system administrators, and adversaries all use the same resources. Thus, they pose similar threats to computer security. Consider the "insider," a trusted systems administrator by day who sells stolen information at night.

The answer to securing these systems is not a better anomaly detection algorithm. Metrics without a formal basis (e.g., IDS tests) are useful for many systems, but inadequate for critical systems. The critical systems that run aircraft carriers do not need to have the same software designed to run Minesweeper, Firefox, and Microsoft PowerPoint. Indeed, critical systems used to be more complex than consumer systems, but now the opposite is true. Thus, general-purpose operating systems such as Windows and Linux systems are neither appropriate nor necessary for critical systems [31]. They also cannot be secured because access controls cannot be precisely constructed for non-trivial systems [24].

Historically, the time and monetary cost of formal verification has been prohibitive, and so formal methods were intractable for useful systems [6, 16]. However, times have changed. First, increased connectivity has increased the need to protect critical systems. Second, computational resources make verification more feasible especially for special-purpose systems. Third, advances in hardware enable us to restrict programs and supporting software so that, while we cannot trust that which we have not written ourselves [58], the *combination* of the untrusted program and a trusted sandbox allows some degree of trust in the combination. Thus, formal methods are practical [6] for designing and implementing secure systems. They are simply costly [39, 51]. Systems with risk of essentially unlimited magnitude require such expense. They require an assured, trusted computing base [10, 46] of both hardware and software. This paper argues that such a system *can* be constructed from dynamic collections of special-purpose, formally verified systems.

One complication is that most systems interact with people, so requirements must take people into account [8]. Thus, studies in human behavior [19] are essential for helping to guide and evaluate the requirements by complicating the application of basic principles [47]. For example, applying the principle of least privilege must not force users to extreme measures to do their work.

In this paper, we leverage a number of metrics including fault tolerance, diversity, and various social science studies to create a metric for a subset of specific security goals particularly useful for critical systems. Such metrics are founded on scientific first principles akin to the physical specifications of the cyber-physical devices that the systems may control. We do not seek an overarching set of "science of computer security metrics" because we feel that that many metrics held up as such are not based on first principles—e.g., those that make claims about intrusion detection system efficacy.

The overriding requirement is that security for systems requiring high assurance must be designed in from the beginning of system development, using "first principles." Existing strategies for managing classified information in multilevel or otherwise compartmentalized [7] environments may be layered on top of these principles. They may also conflict with them. Indeed, existing strategies serve a specific purpose in a particular environment. But such a purpose may not hold in today's environments.

2. PRINCIPLES AND ASSUMPTIONS

The heart of the following first principles are the notions of "policy," "data," and "resource." They underlie the security of the system.

1. Data has an associated policy, as do resources. When a resource accesses data, the resource must satisfy the data policy and the data must satisfy the resource policy. In a sense, the data and resources are each encased in a "computational container" that restrict what it can do, or what can be done with it. This roughly corresponds to the ring access controls for data segments in Multics, where a process accessing the data segment had to meet requirements set *on the data segment* [48]. Specifically, the calling segment's ring level had to lie in the data segment's access bracket.
2. Resource and data policies require balancing security properties (confidentiality, integrity, and availability) in measurable ways, so that the mix of properties implemented can be compared to the desired mix given in the system requirements. The measurements need not be quantitative or precise; in some cases, simply bounding, comparing, or ordering elements is sufficient. For example, instructions and data *replicated* widely are likely to be both more available and have higher integrity, but data *sliced* (for example, using onion routing [12]) and stored widely will be more likely to be confidential because it would be harder to obtain every slice.
3. Systems must be able to enforce *perfect privacy.* That is:
   (a) Privacy: users control all data relating to them.
   (b) Anonymity: others, including adversaries, are provably unable to extract useful data.
   (c) Non-persistence: no trace remains of data transferred or deleted.

   Perfect privacy is bidirectional, but may be asymmetrical. For example, an organization may require privacy of its data ("don't leak it outside the company"), but a user may require privacy of their actions (no audit trail). Or, a person in the United States may have the option of placing a "block" on their credit report, receiving notification when someone has requested their report, or both. It may not be possible for all components of perfect privacy to hold for all parties in all circumstances. It is an open question: under what circumstances can all components hold for all parties?
4. The "system" includes hardware (computational and/or physical), procedures, *and* a set of people who are trusted to correctly carry out those procedures. Thus, formal verification of these procedures using process or workflow modeling is an integral part of validation.
5. All data is bound to an entity, physical or otherwise. Unless anonymity is required, all entities are authenticated at all endpoints. When a person generates or manipulates data, that individual is associated with the data unless the system's overall policy forbids it. For example, in U.S. elections, laws prevent associating a specific person with a specific ballot. In that case, the entity is virtual and unbound to any physical entity.

3. PREMISE

Given the above set of assumptions and principles, one must provide evidence to assure the relevant target audience that the system, and its attendant procedures, conform to the requirements or specification of that system. We next explore several technologies which, when combined, can provide this evidence.

3.1 Formal Verification

Formal verification uses mathematical techniques such as code proofs, model checking [9], and automated theorem proving [4] to verify the correctness of the specification and design of a system. Certain languages and tools extend this verification to the correctness of the implementation of a system. Models of use and maintenance procedures can also be verified [50].

The key problem with formal verification is that it is based on assumptions—about physics, the quality of materials, the

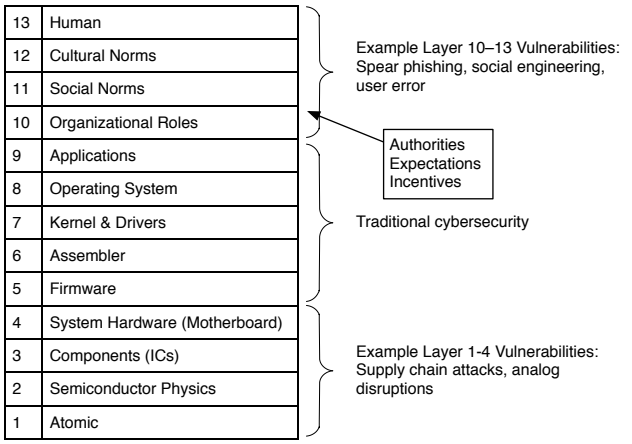

Figure 1: "Full-scope" stack [55].

correctness of the tools used to do the verification, and the correctness of components not checked. For example, even if the physical design is validated, Figure 1 shows that layer still relies on underlying components in layers 1–4. These components may be flawed, and are vulnerable to supply chain attacks. Therefore, to be complete, a formal verification of a system would have to begin at the lowest levels. The system and the verifiers would need to be manually verified until the system and verifiers can be bootstrapped to develop and automatically verify the larger, specified system. Such an approach is possible [34], albeit extremely expensive. It is still limited, however, in that it is able to verify the design but not the *physical components*. It is also essential to use assumptions that extrapolate well to the future, such as physics, rather than assumptions that to not, such as ones involving human behavior [57].

3.2 Fault Tolerance

If a system cannot be proved to be correct at all times, it must be prepared to tolerate failures. This means it must be fault tolerant. Hence, one requirement is that the system have rigorously defined metrics so that faults can be detected and measured. As an (informal) example, a system that provides confidentiality using cryptography could use as a metric the estimated number of operations needed to factor a number into two large primes.

Diverse redundancy can also provide measures of fault tolerance. "Diverse" means the components are written independently and failures are independent of each other, for example as the result of multiversion programming [27] or compiler-generated diversity [14]. For example, a system should have at least $3f + 1$ independent replicas [30] to tolerate up to f faults. This provides a mathematical basis for reasoning about the consequences of fewer replicas.

On the other hand, we cannot simply provide an equation such as $\sum_{i=0}^{N} ReplicatedSystem_i$ because this implies independence in the failure of each system including the interaction of subsystems in their environment [20]. That is, one can imagine that for some $0 \leq j, k < N$ and $j \neq k$; $ReplicatedSubSystem_j$ causes $ReplicatedSubSystem_k$ to be less safe, or indicate a less safe system. However, we can measure both software diversity [25, 36] and independence. For example, we can compare the control paths of each replica, determine how many paths of flow of control there are, and determine how many subsequent elements of a system depend on the outputs from previous elements. This generates an initial set of probabilities that can be used to augment an equation describing the composed system. It also identifies the components that, at an initial cut, are most likely to be the most heavily used. This suggests where to focus the most attention. One can instrument the system to collect runtime information on the paths taken during testing and actual use. As the system runs, those probabilities change, and new computations can be performed as needed. A context-free grammar can be used to model this, with the initial estimates of the probabilities set by the flows of control, and changing as the data accumulates. By comparing the predicted paths and the paths during testing using techniques such as *KL divergence* [28] one can gain confidence that the system is operating as designed.

As discussed in the previous section, formal verification has limits. Fault tolerance also has limits because diversity can be hard to achieve [3]. Therefore, we advocate an approach that combines formal verification and fault tolerance. The fault tolerance aims to tolerate physical failure and/or failure of components that the system is connected to, but which are functionally outside of the system. The formal verification aims to prove the correctness of the parts of the system that it is able to verify. The reason we use both fault tolerance and formal verification is that the two complement each other, the fault tolerance compensating for that which formal verification techniques do not verify, and formal verification reducing the need for fault tolerance on all parts of the system. Moreover, while Byzantine fault tolerance applied to defending against attacks can be limited by the traditional $3f + 1$ threshold model, methods exist to use *adversary structures* to mitigate this [29]. Additionally, some form of graceful transition and/or degredation to a pre-defined failure mode (e.g., fail-safe, fail-soft, fail-hard, fail-over) must be provided in the event that adversary resources exceed the threshold.

While software diversity through N-version programming has been suggested, its value has been questioned due to studies that failed to show significant improvement in reliability [13, 27]. On the other hand, N-version programming need not be used purely for its ability to provide ultra-high reliability, but rather for simply increasing diversity to detect failures [41]. Therefore, the system design must use N-version programming appropriately.

Although software diversity itself is unlikely to have a negative effect on reliability, one might postulate that even some diversity is still helpful [25], and so why not use it? The downside is that the requisite voting system makes the entire process more complex, and therefore the benefit of diversity trades off between increasing the complexity of the system and increasing the complexity of verification. While some complexity is unavoidable, our goal is *manageable complexity*. Therefore, we limit our use of diverse redundancy via N-version programming to modules and layers where it provides known and measurable improvements, and where formal methods cannot be applied (e.g., physical components

and other analog failures). Thus, our suggested process is:

1. Formally (and manually) verify the design of the system and verifiers and bootstrap this to develop larger systems. Do this N times, in N different ways, to attempt to increase diversity.
2. Use the N versions of the verifier (each of which has been independently verified, as mentioned above) to automatically verify the system.
3. Use $3f + 1$ redundancy to run the system.
4. (Optional) During testing and/or actual use, compare the predicted paths taken through the system to the paths actually taken, and measure divergence.

Thus we use both formal methods and fault tolerant algorithms.

The fault tolerance metric does not address the consequence of a user taking an action that they are authorized to take—and therefore the system allows them to take—but is nonetheless harmful in some respect. Given the focus on the *user* action for this situation, we can define the "system" to include hardware (computational and/or physical), procedures (including error handling to compensate when errors are made following the procedures), *and* a set of people who are trusted to adhere to those procedures. Thus, formal verification of these procedures using process or workflow modeling is an integral part of validation. We then measure the risk of a compromise of availability and integrity by determining the minimum number of users who must collude in order to take a particular action, and require that for damaging actions, two or more users must collaborate to take such actions. The system must enforce this metric. For example, a nuclear submarine enforces the "two-man rule" by requiring two keys to be turned simultaneously at opposite ends of a room—something that no single person could do [1, §15]. Those keys are "stored in safes locked by combinations in which nobody on board has the combination to those safes, as that combination must come as part of a launch order from a higher, external authority." Thus the two-man rule cannot easily devolve into "one person with a gun." Less prosaically, "Starfleet vessels require authorization from the captain and/or senior officers, with the use of command authorization codes in most cases."[1]

We now present some example metrics to demonstrate some considerations involved in developing appropriate metrics. The actual metrics will depend on the system requirements and environment in which they are used. Note that these equations assume complete independence between replicas, but the equations can be appropriately modified for systems with replicas that are not independent.

Let D be a measure of how different components are as a real number between 0 (identical) and 1 (completely different). Let S be the fraction of the number of components of the system covered by the diverse replicas considered in D (that is, a number between 0 and 1 inclusive). Let f be the number of faults to be tolerated. Let n be the number of replicas of the components. Then the portion P of the system that can be properly and diversely replicated is:

$$P = \frac{n}{3f+1} \times D \times S$$

[1]Auto-destruct: `http://en.memory-alpha.org/wiki/Auto-destruct`

Let M be the number of people controlling or monitoring an object. Let A be the number of these people required to take an action on or with the object that is harmful in some respect. Let S be the fraction of the number of objects considered in M (again a number between 0 and 1 inclusive). Then C, the degree of collusion required to compromise the system, is:

$$C = 1 - \frac{M}{A} \times S$$

Having an M-person rule, therefore, is much like state replication in its ability to provide a statistical measure of the amount of security that an object has.

Thus, one possible measure of availability and integrity is $P \times C$. The *non-security metric* would then be the complement of this measure, which is $(1 - P) \times (1 - C)$. In most critical systems, the cost of the compromise of items left unprotected cannot be expressed in numerical terms, for example, in a monetary value lost due to compromise. However, the components can be *ordered* according to risk and consequence of failure [5].

Measuring *diversity* and *independence* of people in "redundant humans" is beyond the scope of this paper. It is best left to social scientists who can study metrics for motivation, incentive, and background. For example, multiversion programming by students tended not to work, because they learned from the same instructor, whereas multiversion programming a corporate environment *did* work because the programmers had different training [27].

Although state (and human) replication provides protection for availability and integrity, they work against confidentiality. The more systems and people that contain useful and usable information, the less confidential the information is. The simplest way to address this is to distribute encrypted *parts* of the information that cannot be decrypted without each other, and therefore are of no value separately. The more pieces into which this information is divided, the more security against compromise of confidentiality is provided. How to do this division of data varies depending on the data and environment.

One can do this at the cost of availability and integrity by reducing state and human replication and instead dividing information among those machines and humans. Alternatively, one can do this by increasing replication by the "data slicing" factor, which increases confidentiality without reducing availability and integrity. This could be done using Shamir's secret splitting [49, 54] where at least m of n slices must be recovered to reconstruct the data, such that $m < n$. For simplicity in this paper, we assume $m = n$. For example, for a actual slices and s state replicas, there will be a total of $a \times s$ machines supporting the same number of slices and replicas. Given r required slices (data dependent), one data disclosure security metric DD with respect to machine replication is:

$$DD = \frac{n}{3f+1} \times \frac{a}{r} \times \frac{A}{M}$$

where n, f, A, and M are defined as previously. Thus for each replica not "sliced," the system becomes *less* secure with regard to confidentiality, and for each additional slice, the machine becomes *more* secure with regard to confidentiality. By increasing both, then security with respect to all three primary goals can be increased without compromise.

An alternate approach is to use information theory to pro-

vide an upper bound for a metric to measure loss of confidentiality. That metric is simply the amount of data sent via overt channels plus the amount of data sent via covert channels [60], both of which can be automatically measured, monitored, and in some cases, blocked in realtime. This can also capture user actions against confidentiality, as well as the risk of disclosure caused by increased replication (e.g., keeping all of the information in one place), as opposed to slicing it up into pieces. Thus, the upper bound on the *insecurity* metric is for confidentiality is:

$$(1 - DD) \times I \times V$$

where I is the amount of information moved and V the value of that information.

In practice optimal security will be provided both when $3f + 1$ replicas are maintained for security, and S data slices are maintained, resulting in $(3f + 1) \times S$ replicas with a different data slice on each one. Note that undoing the slicing need not necessarily involve reconstituting the original element. Like "lossy compression," the user requesting the data may be able to reconstitute the needed portion of the original element by obtaining a subset of the slices of the data.

We emphasize that the metrics presented above are rough sketches of the analysis that would need to be done. The point is that, given an understanding of the environment in which the system will function and of its function, the metrics can be created. This technique of state machine replication, redundant personnel, and data slicing address Principles 1, 2, and 4 of our list.

3.3 Trust Negotiation

Trust negotiation is required for perfect privacy because both sides need a means for authenticating other party of a communication to the desired degree of assurance while revealing no more than necessary or intended. Many technologies exist to accomplish this to the desired degree of assurance, including TrustBuilder [32], zero knowledge techniques [17], and trusted third parties. Such techniques enable negotiation to take place in which a computing system enforces a policy where file sharing is disallowed without explicit consent from the owner. The negotiation considers the action that a user wishes to take, the security policy of the organization, and the privacy of the user. Such a system should be designed and implemented consonant with the integrity and confidentiality metrics of the system (as discussed above) so that the integrity of the negotiation process can be monitored, and accidental disclosure bounded.

3.4 Binding

An entity must be strongly bound to its digital identity. This supports both anonymity (the entity is simply unidentified) as well as accountability (the entity is identified to an appropriate degree). Password-based authentication is known to be insufficient, in large part because of human frailties. Attempts to improve authentication by constraining passwords has increased the security burden on the user while providing little, if any, security improvement [19]. Complex passwords aggravate existing vulnerabilities (such as users writing down the password) while increased computing power has made exhaustive search not just possible but straightforward.

Traditional authentication is based on something you know, something you are, something you have, or where you are. But these have different strengths, depending upon environment. However, in general, "something you are" is harder to fake than "something you know," harder to steal than "something you have," and more certain than "where you are." This suggests relying most heavily on techniques referencing a personal characteristic of the user.

Binding an entity to its digital identity is essential to do not just once at a perimeter, but continuously while it accesses data and resources. Of course, such binding must attempt to minimize susceptibility to masquerade attacks [35]. Dynamic, flexible access control [43] is one such approach.

Traditional access controls have evolved from static and coarse-grained to dynamic and fine-grained. But too little access inhibits usability, effectively creating a denial of service for people trying to do their jobs; and too much access invites breaches of security. Prohibitions fall into two classes. Core prohibitions prevent disaster, and are axiomatic to the system. Ancillary prohibitions, derived from core prohibitions, hinder the ability of an attacker to violate core prohibitions, but are not in and of themselves critical to the security of the system. *Optimistic access control* is a framework in which core prohibitions are always enforced, and ancillary prohibitions are enforced only when a specific threshold is crossed. The threshold may depend upon history, trust, and a variety of other non-binary countermeasures. For example, suppose account `alice` is logged in from person Alice's computer. Trust that `alice` is bound to user Alice might be increased if Alice's office door is open and people are walking by her office frequently (which can be verified by tracking the GPS locations from their cell phone). On the other hand, if Alice's door is closed, then we may have less trust. The access control may require an increase in trust should account `alice` attempt to take sensitive actions on sensitive resources.

Strong, continuous, adaptive entity authentication eliminates the security vulnerabilities inherent with using purely perimeter-based solutions and tokens (such as HSPD-12 badges and smart cards) to mediate the relationship between the human and the computing environment. When the entities are people, widely-available authentication technologies such as facial recognition, voice recognition, GPS, and so forth can provide unambiguous identity as certain and intuitive as in the physical world. Many smartphones incorporate cameras, GPS receivers, accelerometers, and rate gyros. Additional sensors such as galvanic skin response and eye tracking enable continuous real-time user authentication. Early steps in the efficacy of many of these mechanisms have been promising ("Cell phones show human movement predictable 93% of the time.") [21]

A single sign-in event enables access until the user logs out. It is not continuous. The authentication discussed above intelligently fuses sensor data with predictable human behavior and limitations to enable probabilistic confidence that the specific user is at the machine. The confidence can be brought closer to certainty by the use of multiple mechanisms. Conceivably, every keystroke [26] and mouse motion can be automatically and transparently signed indicating the confidence that the machine input was provided by the specific individual. If confidence is eroded due to user inconsistency (such as injury or sickness), it can be restored by requiring more complicated passwords or even more intru-

sive means such as DNA analysis (assuming that technology is available).

To place this into context, a traditional access control matrix is a mapping of a subjects $s \in S$ and objects $o \in O$ to a set of rights $r \in R$ expressed as a matrix A (whose entries are a). When a subject tries to perform an action on an object, the set of rights in the appropriate entry is checked. If the right is present, the action proceeds; otherwise, it is blocked. In this form, the matrix implements a binary decision function where the only possible responses are to allow or block access.

A *conditional access control matrix* contains *functions* of rights rather than rights. Let M be a set of information needed to determine whether a subject s may access an object o. For example, $m \in M$ may be the time of day, a security clearance, execution history, and/or the state of the application or system. Then define a function $f : S \times O \times R \times M \to R$. Each entry in the conditional access control matrix $a[s, o]$ is $f(s, o, r', m')$, where $r' \subseteq R$ and $m' \in M$.

As an example, consider a login mechanism that blocks all logins to an account after n failed attempts. In the above formulation, s would be the user, o would be the account, r would be the (singleton) set of the login right l, m would be the number of previous incorrect login attempts, and

$$f(s, o, l, m) = \begin{cases} l & \text{if } m < n \\ \varnothing & \text{otherwise} \end{cases}$$

Were the function defined (slightly) differently, m could include the environmental parameter of a trusted user vouching for Alice, thereby overriding the $m < n$ check.

The concepts of "risk," "trust," and "consequence" are difficult or impossible to quantify. However, they can often be ordered. Thus, one promising approach approach of characterizing subjects, objects, and actions using attributes [5] and ordering threats accordingly so that bounds and scales of threats can be assessed.

3.5 Summary

In summary, we have described a system in which:

1. Data relates to computational resources by way of a data policy and a resource policy, which must satisfy each other.
2. Provides metric-driven balancing of security components: availability, confidentiality, and integrity.
3. Is capable of enforcing *perfect privacy* on data.
4. Includes hardware, procedures, *and* a set of people who are trusted to adhere to those procedures.
5. There is strong, continuous binding of an individual to data.

4. CASE STUDY: MEDICAL SYSTEMS

In this section we discuss how our approach can be applied to a specific scenario, and discuss metrics related to availability, confidentiality, and integrity, and the tradeoffs.

A medical center has a data policy intended to maximize availability and integrity of medical records but not at the expense of privacy. This policy requires quadruple redundancy: there is always a backup, a backup for the backup, a third- and fourth-level backup. Thus, given constraints imposed by Byzantine fault tolerance, if the system has 4 replicas, it can tolerate the failure of a single system. The privacy requirement means that the environment and nature of the data require that each record be sliced into four pieces and stored separately in order to prevent reconstruction of the data from any single part of the data. Thus, since each record must be replicated four times and sliced into four parts, and given that each replica and slice must be maintained independently, 16 machines are required to maintain the availability, integrity, and confidentiality of the system.

In this particular environment, there are situations in which medical records are the inputs to medical devices such as surgical or life support equipment. Such systems are critical systems, and many operate in a SCADA fashion. Like any other SCADA system, they can be compromised at the supervisory control, network, or remote terminal unit level. As such, end-to-end validation and verification [11] of all process control actions taken by the operator is essential. But, for these systems, availability and integrity might supersede confidentiality (privacy) at times (such as when the system is controlling a patient's vital functions).

Consonant with societal trends, suppose also that the security policy supports "perfect privacy." That is, patients truly "own their own medical records"—patients must grant permission to any physician to see his records, or to any medical system that must access the patient's records. If the physician accessing the records is the patient's own personal physician, then no security violation has taken place because the patient has already granted her access. If the physician is not, as may occur in a "break-the-glass" emergency such as a patient being in cardiac arrest and must activate a system requiring the patient's records, then under this (hypothetical) policy, the physician needs to authenticate and immediately document the reason for viewing the records (strong authentication), or a second physician may provide their credentials to avoid documenting the reason for one hour, or three physicians together may provide their credentials to avoid documenting the reason at all.

The reason for the three possibilities is the problems attendant on creating documentation. First, if the audit logs are not bound to the objects they refer to, they are extra objects that can be attacked separately from the documents, and hence decreases security by increasing the attack surface. Second, the availability of such logs implicitly violates user privacy. Therefore, this method of computing must enable auditing but log data in such a way that it is bound to the relevant objects. Further, when audit data involving a user is collected, the user should know that it is being collected, agree to its collection, know under what conditions the data may be read and used (that is, that the data is protected from unauthorized snooping), and that the data is part of the object being tracked, itself.

So if documentation by the physicians is provided, then it is kept as a cryptographic append-only "watermark" as part of the medical record itself (to maintain provenance and attribution for the event), and not contained in external logs (which may well be non-persistent and difficult to attribute). This "watermark" contains the data and the metadata, thereby limiting the spread of audit logs to other locations that may compromise privacy of the users involved (in this case, the physicians and patients). Sharing this documentation requires similar methods. Suppose that the records can be sent outside the hospital to authorized viewers. By watermarking the file to indicate that the person who authenticated to the system and sent the file was a specific individual, that identity is bound to the distributed

data because the watermark cannot be altered without destroying the data in the record. Further, the transmission requires a handshake that demonstrates that the recipient also authenticated. The recipient's identity is also part of the watermark, providing strong authentication.

This system seems to provides a means for balancing availability, confidentiality, and integrity, and provides a dynamic, flexible means for doing so, based on events external to the computer system (e.g., number of physicians involved). Note that this policy addresses many but not all of our stated requirements. For example, patients are able to maintain perfect privacy, but what if a physician wishes to be anonymous? In this case, there is a conflict between user control of the record and anonymity of the physician, and therefore, perfect privacy cannot be maintained by all parties.

5. IMPLEMENTATION IDEAS

The paradigm that we have proposed is not easy to practice. However, we assert that there are a variety of ways to implement such a paradigm. We propose a set of solutions that addresses the tradeoffs in a way that particularly addresses security policies of critical systems. For example point-to-point communication is particularly vulnerable to DDoS attacks. Wireless spread spectrum techniques mitigate this problem in wireless networks by spreading normal narrow band signals over a much wider band of frequencies thus forcing jammers who do not know such spread pattern to invest much more effort to launch attacks. In the same way, the techniques that we have presented here can require adversaries to invest significant resources to compromise the integrity of computer networks.

Typically, the first step in the execution of a computational task is to secure the resources necessary to complete the task. This static approach to computing has several drawbacks: power is consumed while waiting for all the necessary resources to be secured, and assured availability demands that these resources are captive, which makes them an attractive target.

Dynamic computing uses a just-in-time approach to resource allocation. No computing resources are dedicated to the job until all resources have been scheduled and are available when needed. This enables the allocation of processing, storage, and networking to be optimized on the fly for different criteria. Dynamic computing seeks "design-arounds" to reduce certain vulnerabilities. For example, optimization for processing performance may demand low network latency. Optimization for security may eschew performance (and availability) in favor of randomization to reduce signal-to-noise ratio and increase confidentiality.

Other, modern approaches to secure computing, using innovative, dynamic techniques have been proposed and implemented. Self cleansing intrusion tolerance [40], fluid information systems [45], and crafting secure systems from the gates up [59] are all ideas related to our own. However, self-cleansing intrusion tolerance and fluid information systems both use some variation of an "exquisite knowledge" approach which assumes that humans or agents operating under the direction of other humans can detect malevolent activity and steer computing chores away from compromised resources. The exquisite-knowledge based assumption is that the human has the ability to detect malevolent activity. History (and billions of dollars of research) has shown that this simply is not the case.

Oracle based approaches depend on an incorruptible element to assure security in what are essentially computing systems as they exist today. They assume that such an incorruptible oracle exists and then build extensions to that system that can accomplish useful computing tasks. "Crafting secure systems from the gates up" calls this oracle the "lease unit." The assumption is that the lease unit cannot be corrupted so all security critical functions point back to this lease unit. The Intel Trusted Platform Module (TPM) architecture [22] is another example of an oracle-based approach. The drawback to the oracle-based approach is that the adversary simply attacks the oracle or its inputs. Once either has been compromised, the adversary has control over the computation resource under the control of the oracle.

However, one distinction is that these techniques all look essentially like current technology, and as such, all have similar weaknesses to current pathologies as existing systems. In contrast, we advocate a computing system that is designed from the ground up to be *incapable* of revealing any meaningful data in a way that conflicts with its policy. Extracting useful computing from a first-principles-secure system requires knowledge external to the computing system itself. Separating computing resources from the information being protected assures that the protection is applied only to the critical information. The computing resources simply become reservoirs and conduits (infrastructure) of convenience. Our proposed dynamic computing systems use entirely different paradigms (such as signal-to-noise management) to assure security. These alternative paradigms are designed from the ground up to both tolerate and expose malevolent activity.

Dynamic Data Storage.

Dynamic computing also enables novel approaches to data storage. For example, since the data required for a computation need only be available at the moment the computation is executed, no "data warehousing" function (typically a dedicated data center) is required. In contrast, data storage is optimized around other criteria.

Optimization for security may seek to utilize "cloud" storage facilities to effectively hide data files. In this example, a data file maybe bulk encrypted, packetized, and each packet may be stored on some combination of independent cloud providers. Redundant storage of each of these packets enables increased confidence in the integrity of the data through statistical methods, Byzantine fault tolerance, or simple voting approaches. The location of all of these packets is tracked through the use of a Dynamic Data Tracking Table (D2T2; analogous to a File Allocation Table or FAT). Recovery of the original data file can be accomplished by recalling each packet (while checking the integrity of each packet for increased confidence), and then decrypting. Packet recall can be sped up using techniques such as "bit torrent." Absent the tracking information in the D2T2, an adversary will have a very difficult time reconstructing the original data file.

File sharing is accomplished by recalling the secret key associated with the file, decrypting and re-encrypting that one using the sharer's public key, and sending that encrypted chunk out to the cloud. The re-encrypted file is then packetized and returned to the cloud and the tracking information in the D2T2 is passed to the recipient. The recipient can now recall and reconstruct the file using their private key. Mul-

tiple recipients can share a single file stored in the cloud by using each of their keys during file transfer re-encryption.

Dynamic Data Storage also provides opportunities for strong forensics by enabling the recipient to maintain records that answer two questions. First, the recipient can prove authorization to receive the file through the ownership of the tracking information with his public key in the D2T2. Second, the recipient can show that he obtained the file through legitimate means by retaining the information used to recall the packets.

The D2T2 is larger and far more complicated than a FAT. Adoption of this approach will, however, reduce overall data storage volume because redundant copies of large files need no longer be stored by individuals. Instead of storing the MP3 file for each song on their playlist, individuals need only store the D2T2 entry for their songs which can then be recalled and played as needed. Use of Dynamic Data Storage will also enable record companies to reduce piracy of their intellectual property by making songs available to the consumer while controlling the distribution channel.

Dynamic Processing.

The proliferation of systems that utilize embedded microcontrollers without fully understanding the security limitations of these systems is a great cause for concern. For example, a self-parking system for a car may use an embedded microcontroller to orchestrate all of those moves. The embedded microcontroller brings with it a multitude of vulnerabilities that have been well characterized and documented. The fact that this microcontroller is linked to the audio system that enables the user to update his MP3 files automatically every time the car is parked at home erodes confidence even further.

The emergence of RISC computing years ago was a reaction to the complexity (and inherent compromises) in a CISC machine. The move to RISC was only based on performance considerations and didn't go far enough. Moving away from embedded microcontrollers, everything that can be executed using a special purpose machine should be executed using a special purpose machine [23]. The technologies to do so exist today. The only thing keeping us from using special purpose machines everywhere possible is the paradigm that we bring to the problem.

Typically, an embedded system is built around an FPGA. The designer instantiates a microcontroller IP core on that FPGA and then uses the remaining gates to execute the specific functions required for the application. Then code is written to link the entire system together. Use of the microcontroller IP core introduces a general purpose computing element with all of its inherent vulnerabilities. Another way to approach an embedded system using an FPGA would be to use a finite state machine (FSM). The FSM would be designed to execute the proper functionality, and then the FSM would be instantiated on the FPGA. The FPGA is field programmable so it can be rewritten in the field. This means that the functionality can be dramatically changed any time the user desires. This approach can be expanded to enable general purpose computing. One could envision a "cloud of gates" which can be configured to perform the user's specific function and then wiped clean and returned to the cloud as an alternative to the "cloud of computers" that we currently think of when we say cloud computing.

Notionally, the functionality of the "cloud of gates" would work like this: First, the user prepares a "processing package" consisting of data and executable packaged to execute on a reconfigurable computing platform (e.g., an FPGA). Then the processing package "certifies" the platform for processing and a bootloader establishes the necessary executable on the platform. Next, the data is processed using the executable and the result is transmitted to the user. Finally, the computing platform is freed for next processing package. In this way everything that can be executed on a special purpose machine is executed on a special purpose machine. And all of the vulnerabilities that go with general purpose computing can be eliminated. One could think of this as *zero instruction set computing* (or "ZISC").

There are enormous problems associated with "place and route" that drives users to microcontroller IP cores. There will be a cost-benefit tradeoff (at least if the beginning) until ZISC design tools are widely available. The place-and-route problem associated with ZISC can be mitigated by enforcing redundancy and diversity of ZISC cores as described in Section 3.2. ZISC cores would be at a higher complexity level than "gates" but not as high as an x86 microcontroller. Of course, ZISC cores also would be less capable than x86 microcontrollers, but sufficient for specific tasks. Note that ZISC cores cannot be modified outside the machine: the FPGAs cannot be reprogrammable, and need to be discarded and replaced (or set up so they can only be programmed on the original machine). For larger processing problems where a designer needs to maintain a high level of certainty, "probabilistic computing" offers an alternative. The designer can break the problem down into provable-core-sized pieces and present each piece of the processing problem to a cloud of independent, ZISC cores. By examining the consensus results returned from each independent core, the designer can obtain probabilistic confidence in the result. Thus, probabilistic computing enables the designer to gain high confidence in the result while not requiring that the system be fully formal methods provable.

Dynamic Networks.

Another opportunity for dynamic computing is networks. Typical homeowners use a single ISP that handles all of their residential network traffic. That traffic travels in serial fashion out of the network, and, in response, traffic is returned on the same network. Even if the data is encrypted, an adversary with access to the pipe is able to collect all bits and, given enough time and computing, decrypt every message sent and received. So this approach is vulnerable both to breach of confidentiality and also denial of service (DoS).

Dynamic computing applied to networks would suggest a "Meta ISP" to provide provide assured, secure data communications. The Meta ISP may choose to contract with as many different ISPs as is necessary to assure that data transmission is secure and robust. The Meta ISP may choose to do bulk encryption of data before transmission, and then packetize this encrypted data, and then commutate it over as many different regular ISPs as is necessary to assure that security and reliability standards are met. The Meta ISP would reassemble the packets and decrypt the message at the other end. Ignoring the endpoints in this limited-scope example, a man-in-the-middle attack would be fabulously difficult. A DoS on any single ISP may affect data transmission rates but shutting data down completely would require

that every ISP be disrupted. The National Broadband Map[2] indicates that the typical urban or suburban home is served by 6 different ISPs. So, a Meta ISP can be accomplished using existing ISPs. The same approach can be used to provide greater bandwidth during normal operations.

6. SUMMARY AND FUTURE WORK

We have presented a set of requirements for critical systems, fundamental premises that those requirements would entail, and ideas for implementations that would instantiate those premises. We have described why current systems—in particular general purpose computer systems—cannot meet those requirements. The overriding requirement guiding our paradigm is the notion that "first principles" reflects the only real security strategy, where first principles are ideally provable, often measurable; and at minimum, possible to order and bound. These principles take into account that many security policies may be even be in conflict, and as such, proofs, measures, and orderings gives an analyst or an automated system the metrics needed to make informed decisions about how to resolve conflicts. We also presented several metrics that enable this, including replication, data slicing, collusion, and information theory.

The requirements that we have presented are by no means the only possible requirements for a "clean-slate, ground-up, first-principles" system. And the implementation ideas that we have presented are by no means the only ways of instantiating such requirements. Many other requirements and implementations are possible. The next steps in this research must involve additional work tying together the various, sometimes disparate components of this system, as well as deep focus on case studies that may help to introduce additional requirements or fine-tune existing (and possibly conflicting) requirements.

Acknowledgements

The authors are standing on the shoulders of giants [42]. We are grateful to the participants of NSPW 2012 for their insightful comments and lively discussions which greatly helped advance the ideas in this paper.

This research was supported in part by the Director, Office of Computational and Technology Research, Division of Mathematical, Information, and Computational Sciences of the U.S. Department of Energy, under contract number DE-AC02-05CH11231. It is also supported in part by the National Science Foundation and the GENI Project Office under Grant Number CNS-0940805, and by the National Science Foundation under Grant Numbers CCF-1018871, CCF-0905503, and CNS-1049738.

Some of this work was completed while Ed Talbot was a Distinguished Member of Technical Staff at Sandia National Laboratories. Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect those of any of the sponsors of this work.

[2]National Broadband Map: `http://broadbandmap.gov`

7. REFERENCES

[1] Assistant to the U.S. Secretary of Defense (Atomic Energy). Nuclear Weapon Accident Response Procedures (NARP). Technical Report DoD 5100.52-M, September 1990.

[2] S. Axelsson. The Base-Rate Fallacy and the Difficulty of Intrusion Detection. *ACM Transactions on Information and System Security (TISSEC)*, 3(3):186–205, Aug. 2000.

[3] A. N. Bessani. From Byzantine Fault Tolerance to Intrusion Tolerance. In *5th Wkshp. on Recent Advances in Intrusion-Tolerant Systems*, 2011.

[4] W. R. Bevier and W. D. Young. The Design and Proof of Correctness of a Fault-Tolerant Circuit. Technical Report SEE N91-17559 09-61, NASA Formal Methods Workshop, 1990.

[5] M. Bishop, S. Engle, D. Frincke, C. Gates, F. Greitzer, and S. Peisert *et al.* A Risk Management Approach to the 'Insider Threat'. In *Insider Threats in Cybersecurity*. Springer, 2010.

[6] B. Blakley. The Emperor's Old Armor. In *Proc. of the New Security Paradigms Workshop (NSPW)*, 1996.

[7] D. F. Brewer and M. J. Nash. The Chinese Wall Security Policy. In *Proc. of the IEEE Symposium on Security and Privacy*, pages 206–214, May 1989.

[8] S. Brostoff and M. A. Sasse. Safe and Sound: A Safety-Critical Approach to Security. In *Proceedings of the New Security Paradigms Workshop (NSPW)*, 2001.

[9] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 8(2):244–263, 1986.

[10] Department of Defense. Trusted Computer System Evaluation Criteria (in the *Trusted Computing Base (TCB)* glossary entry). Technical Report DoD 5200.28-STD, December 26, 1985.

[11] Department of Health and Human Services. Guidance for Industry Cybersecurity for Networked Medical Devices Containing Off-the-Shelf (OTS) Software. Technical Report 1553, January 14, 2005.

[12] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router. In *Proceedings of the 13th USENIX Security Symposium*, 2004.

[13] D. E. Eckhardt Jr., A. K. Caglayan, J. C. Knight, L. D. Lee, D. F. McAllister, M. A. Vouk, and J. P. J. Kelly. An Experimental Evaluation of Software Redundancy As a Strategy for Improving Reliability. *IEEE Transactions on Software Engineering*, 17(7):692–702, 1991.

[14] M. Franz. E unibus pluram: Massive-Scale Software Diversity as a Defense Mechanism. In *Proc. of the New Security Paradigms Workshop (NSPW)*, 2010.

[15] C. Gates and C. Taylor. Challenging the Anomaly Detection Paradigm: A Provocative Discussion. In *Proc. of the New Security Paradigms Workshop (NSPW)*, pages 21–29, Sept. 2006.

[16] V. D. Gligor. Analysis of the Hardware Verification of the Honeywell SCOMP. In *Proceedings of the IEEE Symposium on Security and Privacy*, 1985.

[17] S. Goldwasser, S. Micali, and C. Rackoff. The Knowledge Complexity of Interactive Proof-Systems.

In *Proc. of the 17th Annual ACM Symposium on Theory of Computing*, 1985.

[18] S. W. Hawking. *A Brief History of Time*. Bantam Books, 1988.

[19] C. Herley. So Long, and No Thanks for the Externalities: The Rational Rejection of Security Advice by Users. In *Proc. of the 2009 New Security Paradigms Workshop (NSPW)*, 2009.

[20] H. M. Hinton. Under-Specification, Composition and Emergent Properties. In *Proceedings of the 1997 New Security Paradigms Workshop*, pages 83–93, 1997.

[21] F. Hsu, H. Chen, and S. Machiraju. WebCallerID: Leveraging Cellular Networks for Web Authentication. *Journal of Computer Security*, 19(5):869–893, 2011.

[22] Intel Trusted Computing Group. Trusted Platform Module (TPM) Specifications, Version 1.2, 2011.

[23] C. E. Irvine and K. Levitt. Trusted Hardware: Can It Be Trustworthy? In *Proc. of the 44th Annual Design Automation Conference*, 2007.

[24] A. K. Jones and R. J. Lipton. The Enforcement of Security Policies for Computation. In *Proc. of the 5th Symposium on Operating System Principles*, 1975.

[25] F. Junqueira, R. Bhagwan, A. Hevia, K. Marzullo, and G. M. Voelker. Surviving Internet Catastrophes. In *Proceedings of the USENIX Annual Technical Conference*, Anaheim, CA, April 2005.

[26] K. S. Killourhy and R. A. Maxion. Comparing Anomaly-Detection Algorithms for Keystroke Dynamics. In *IEEE/IFIP International Conference on Dependable Systems & Networks (DSN)*, 2009.

[27] J. C. Knight and N. G. Leveson. An Experimental Evaluation of The Assumption of Independence in MultiVersion Programming. *IEEE Transactions on Software Engineering*, 12(1):96–109, Jan. 1986.

[28] S. Kullback and R. A. Leibler. On Information and Sufficiency. *Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[29] K. Kursawe and F. C. Freiling. Byzantine Fault Tolerance on General Hybrid Adversary Structures. Technical report, RWTH Aachen, 2005.

[30] L. Lamport, R. Shostak, and M. Pease. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, July 1982.

[31] B. Laurie and A. Singer. Choose the Red Pill and the Blue Pill: A Position Paper. In *Proc. of the New Security Paradigms Workshop (NSPW)*, pages 127–133, September 2008.

[32] A. J. Lee and M. Winslett. Enforcing Safety and Consistency Constraints in Policy-Based Authorization Systems. *ACM Trans. Inf. Syst. Secur. (TISSEC)*, 12(2):1–33, 2008.

[33] K. Levchenko, A. Pitsillidis, N. Chachra, B. Enright, M. Félegyházi, C. Grier, T. Halvorson, C. Kanich, et al. Click Trajectories: End-to-End Analysis of the Spam Value Chain. In *32nd IEEE Symposium on Security and Privacy*, pages 431–446, 2011.

[34] R. J. Lipton and L. Snyder. A Linear Time Algorithm for Deciding Subject Security. *Journal of the ACM (JACM)*, 24(3):455–464, 1977.

[35] T. F. Lunt and R. Jagannathan. A Prototype Real-Time Intrusion-Detection Expert System (IDES). In *Proc. of the IEEE Symposium on Security and Privacy*, pages 59–66, 1988.

[36] M. R. Lyu, J.-H. Chen, and A. Avizienis. Software Diversity Metrics and Measurements. In *Proc. of the Sixteenth Annual Computer Software and Applications Conference (COMPSAC)*, pages 69–78, 1992.

[37] A. Marantz. Dept. of Hoopla, "Odds Are". *The New Yorker*, page 22, April 2, 2012.

[38] J. McHugh. Testing Intrusion Detection Systems: A Critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluations as Performed by the Lincoln Laboratory. *ACM Trans. Inf. Syst. Secur. (TISSEC)*, 3(4):262–294, Nov. 2000.

[39] J. S. Moore, T. W. Lynch, and M. Kaufmann. A Mechanically Checked Proof of the AMD5$_K$86^{TM} Floating-Point Division Program. *IEEE Transactions on Computers*, 47(9):913–926, 1998.

[40] A. Nagarajan, Q. Nguyen, R. Banks, and A. Sood. Combining Intrusion Detection and Recovery for Enhancing System Dependability. In *Proc. of the IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops*, 2011.

[41] L. Nagy, R. Ford, and W. Allen. N-version Programming for the Detection of Zero-Day Exploits. In *IEEE Topical Conference on Cybersecurity*, 2006.

[42] I. Newton. *The Correspondence of Isaac Newton*, volume 1, page 416. Cambridge Univ. Press, 1959.

[43] S. Peisert. Optimistic Access Control and Anticipatory Forensic Analysis of Insider Threats. In *Insider Threats: Strategies for Prevention, Mitigation, and Response*, number 10341 in Dagstuhl Seminar Proceedings. Dagstuhl, 2010.

[44] S. Peisert, M. Bishop, and K. Marzullo. What Do Firewalls Protect? An Empirical Study of Vulnerabilities and Attacks. Technical Report CSE-2010-8, UC Davis, March 2010.

[45] C. W. Probst and R. R. Hansen. Fluid Information Systems. In *Proceedings of the 2009 New Security Paradigms Workshop (NSPW)*, pages 125–132, 2009.

[46] J. Rushby. Design and Verification of Secure Systems. In *Proceedings of the 8th ACM Symposium on Operating Systems Principles*, pages 12 – 21, 1981.

[47] J. H. Saltzer and M. D. Schroeder. The Protection of Information in Computer Systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.

[48] M. D. Schroeder and J. H. Saltzer. A Hardware Architecture for Implementing Protection Rings. *Communications of the ACM*, 15(3):157–170, 1972.

[49] A. Shamir. How to Share a Secret. *Communications of the ACM (CACM)*, 22(11):612–613, 1979.

[50] B. I. Simidchieva, S. J. Engle, M. Clifford, A. C. Jones, S. Peisert, M. Bishop, L. A. Clarke, and L. J. Osterweil. Modeling Faults to Improve Election Process Robustness. In *Proc. of Electronic Voting Technology Wkshp./Wkshp. on Trustworthy Computing (EVT/WOTE)*, 2010.

[51] B. Snow. We Need Assurance! In *Proceedings of the 21st Annual Computer Security Applications Conference (ACSAC)*, 2005.

[52] B. Snow. Our Cyber Security Status is Grim (and the

way ahead will be hard). Synaptic Labs 2012 Annual Reports Video Series, 2012.

[53] R. Sommer and V. Paxson. Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2010.

[54] M. W. Storer, K. Greenan, E. L. Miller, and K. Voruganti. POTSHARDS: Secure Long-Term Archival Storage Without Encryption. Technical Report Technical Report UCSC-SSRC-06-03, Storage Systems Research Center, University of California, Santa Cruz, Sept. 2006.

[55] E. Talbot. Cyber Security Challenges and Opportunities. Keynote Address for the 2011 Workshop on Governance of Technology, Information, and Policies (GTIP), December 2011.

[56] E. B. Talbot, D. Frincke, and M. Bishop. Demythifying Cybersecurity. *IEEE Security and Privacy*, 8(3):56–59, May/June 2010.

[57] N. N. Taleb. *Fooled by Randomness: The Hidden Role of Chance in Life and in the Markets*. Random House, 2001.

[58] K. Thompson. Reflections on Trusting Trust. *Communications of the ACM*, 27(8):761–763, 1984.

[59] M. Tiwari, H. M. G. Wassel, B. Mazloom, S. Mysore, F. T. Chong, and T. Sherwood. Complete Information Flow Tracking from the Gates Up. In *Proc. of the 14th Intl. Conference on Architectural Support for Programming Languages and Operating Systems*, 2009.

[60] C.-R. Tsai and V. D. Gligor. A Bandwidth Computation Model for Covert Storage Channels and its Applications. In *Proc. of the IEEE Symposium on Security and Privacy*, April 1988.