Trust-Based Security; Or, Trust Considered Harmful

Abe Singer
Consultant
Bloomington, IL, USA
abe.singer@gmail.com

Matt Bishop
University of California at Davis
Davis, CA, USA
mabishop@ucdavis.edu

ABSTRACT

Our review of common, popular risk analysis frameworks finds that they are very homogenous in their approach. These are considered IT Security Industry "best practices." However, one wonders if they are indeed "best", as evinced by the almost daily news of large companies suffering major compromises.

Embedded in these "best practices" is the notion that "trust" is "good", i.e. is a desirable feature: "trusted computing," "trusted third party," etc. We argue for the opposite: that vulnerabilities stem from trust relationships. We propose a a paradigm for risk analysis centered around identifying and minimizing trust relationships.

We argue that by bringing trust relationships to the foreground, we can identify paths to compromise that would otherwise go undetected; a more comprehensive assessment of vulnerability, from which one can better prioritize and reduce risk.

CCS CONCEPTS

• **Security and privacy → Security requirements**.

KEYWORDS

cybersecurity, risk assessment, trust, trust relationships, vulnerabilities

ACM Reference Format:
Abe Singer and Matt Bishop. 2020. Trust-Based Security; Or, Trust Considered Harmful. In *New Security Paradigms Workshop 2020 (NSPW '20), October 26–29, 2020, Online, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3442167.3442179

1 INTRODUCTION

There are a number of risk management frameworks and methodologies commonly used in practice, which form the basis of what is considered computer security "best practices". We review these methodologies below in Section 3.

These traditional approaches assess risk by identifying threats and assets, enumerating vulnerabilities and potential avenues of attack, and labeling the combination "risk." Threats get instantiated through exploitation of vulnerabilities, or weaknesses that when exploited bypass or compromise controls. Then it applies "controls" to eliminate the dangers or reduce the risk to an acceptable level.

NSPW '20, October 26–29, 2020, Online, USA

ACM ISBN 978-1-4503-8995-2/20/10…$15.00
https://doi.org/10.1145/3442167.3442179

In that paradigm, the element of trust is implicitly present — but that notion is not made explicit, nor is it defined. Our paradigm changes this, in that it is based on trust, which for the purposes of this paper we define in terms of reliance:

> ***Anna trusts Bill if Anna relies on Bill, and cannot (or chooses not to) monitor or control whether and how Bill meets Anna's expectations.***

A clear definition of trust leads to an understanding that vulnerability is fundamentally a result of trust, and that conventional vulnerabilities such as software flaws are merely specific instances of misplaced trust. Put more succinctly: trust relationships are the root of vulnerabilities. We discuss this definition, and compare and contrast it with others, in the following section.

Consider how the risk analysis methodologies are used in practice. The methodologies focus on reducing risk by identifying vulnerabilities and then applying controls to ameliorate or eliminate them. In other words, the choice of controls is dictated by the nature and type of vulnerabilities. This might be well and good if one is able to identify all vulnerabilities. But virtually none of those risk analysis methodologies speak to identifying vulnerabilities beyond checking for known vulnerabilities. We argue there are two problems with this omission. First, the search for vulnerabilities in software and hardware is woefully incomplete. In fact, so many vulnerabilities go undetected prior to attack that we have a term for them: "zero-day." Second, many of the vulnerabilities exploited are configuration or architectural problems in the systems, or human or organizational problems, none of which are related to flaws in software and hardware, and hence not detectable by vulnerability analysis tools and techniques. This suggests an explanation of why organizations continue to fail at security, that the "best practices" actually mislead one into thinking that they've identified their risks, and argues for a better approach. However, we hear frequent announcements of compromises and data breaches of very large organizations such as Equifax [22], RSA [29], and Target [46], ones with the ability to devote more than enough resources to cybersecurity, and that we would expect to understand and ameliorate the risks they have taken on.

This stems from the current operational paradigm of security and risk assessment being to determine what equipment to purchase and how to configure and install both the computing infrastructure (networks, etc.) and the equipment. At that point, one makes configuration or architectural decisions, and optimizes the design and configurations based on those decisions. The design and configuration creates trust relationships at various levels: lower levels such as software configurations, and higher levels such as integrating a third-party service or partnering with a particular vendor. In other words, the risk assessment is performed first, and then the system is architected and set up based on that assessment. Trust relationships may be considered, but this is unusual, especially at lower levels.

Our paradigm is different than zero-trust networking. Zero-trust networking [23], generalized into zero-trust architecture [41], defines "zero-trust" as "the term used to describe various cybersecurity solutions that moved security away from implied trust based on network location and instead focused on evaluating trust on a per-transaction basis" [41, p. 4]. This approach to enterprise architecture addresses particular instances of where trust is to be placed, and when; its primary focus is on transaction-level authentication and dynamic authorization. While a zero-trust architecture may reduce particular types of trust relationships, it is not a risk analysis methodology, nor does it address the general evaluation of trust relationships.[1] Indeed, the paradigm we propose below might provide a foundation for improving such an architecture.

We argue that the current approach is inadequate because it does not make trust relationships sufficiently explicit. Indeed, rather than have the controls define the trust relationships, as is currently done, we identify the trust relationships from the architecture of the system and the goals of the security policy. We then use security controls to minimize the number of these relationships, taking into account the effects of these relationships being misplaced.

The usual practice has controls determine what entities can and cannot do, and strengthening or weakening them as the threats, and hence the security policy, evolves. Instead, we propose identifying and evaluating trust relationships as the basis of the threat modeling, and then based on the evaluation, defining controls to monitor or reduce trust as much as possible. Vulnerabilities then come simply from trust relationships that cannot be fully monitored and controlled. Indeed, we argue that failure to identify and address trust relationships is the root of vulnerabilities; that "best practices' " overlooking of these trust relationships has resulted in a failure to properly address risk.

Our new paradigm revolves around the concept that violations in trust relationships are the root of vulnerabilities. Specifically, our paradigm addresses the lack of recognition of trust relationships in "best practices." We advocate a practice of risk analysis built around the recognition of trust relationships, and using them to guide selection and configuration of security controls.

In this new operational paradigm, trust relationships get identified when the system is designed and built and configured, and those relationships are made explicit. Then installation and configuration are based upon the trust relationships. Where there is trust, there is a recognition that one or more parties in the relationship are vulnerable to the others. When the trust is deemed inappropriate, controls are applied to reduce that trust. Security is considered both during design and development, and the controls get implemented based on the trust relationships.

Framing risk analysis from the perspective of trust has several advantages:

- We can identify paths to compromise that would otherwise go undetected;
- We make the exposure due to trust relationships explicit, and bring them to the foreground where they can be acknowledged; and
- Addressing vulnerabilities in terms of trust can be more easily understood by non-technical people (such as executive management).

Returning to the popular risk management approaches, their failure to acknowledge trust relationships means that none provide practitioners with the tools they need to fully identify what is actually at risk. This failure is structural, in that trust relationships are not addressed as risks. Indeed, one can argue that all vulnerabilities (except perhaps weaknesses in cryptosystems) are the result of trust relationships gone awry. Some examples are:

- A malicious package in RubyGems designed to steal cryptocurrency adds an entry to the Windows registry to ensure it is run even if the system reboots [12];
- Buffer overflows, the perennial problem that is always supposed to disappear "next year", arise because the programmer trusts that indices will never go out of bounds;
- Input validation problems arise because the programmer trusts that the input will be valid and supplied as expected; and
- Access controls based on IP addresses (such as used in firewalls and network services) rely on IP addresses not being spoofed or forged.

We next explore what we mean by "trust", and examine both current risk analysis methodologies and "best practices" in detail. We then apply our paradigm to several real world compromises. We conclude by discussing related work and future directions.

2 WHAT IS TRUST?

There are many definitions of trust. Our definition of trust is based upon reliance. One entity (Anna) trusts another entity (Bill) if Anna believes that Bill will meet some requirement (action or characteristic) that she does not control in some way and in a context in which it affects Anna in some way. Informally, Anna trusts Bill if she relies on Bill in some way, and must assume Bill will perform as expected, meaning she cannot or does not control or monitor whether or how Bill meets her expectations. Call this *reliance trust*.

To expand upon this, when Anna trusts Bill:

- Anna expects (trusts) Bill to behave in a certain way.
- Anna does not control or monitor (must trust) how Bill behaves.
- Anna is relying on (trusting) Bill to behave in that way.

Anna is now vulnerable to (trusts) Bill; should Bill misbehave, Anna cannot stop Bill, nor even detect the misbehavior. Hence (reliance) trust begets vulnerability.

Note that we variably say here that Anna "cannot," "does not," or "chooses not" to monitor or control. Whether the lack of control and monitoring is involuntary or voluntary, the vulnerability is the same. The only difference comes in our paradigm for evaluating the trust relationship; whether the capability exists to reduce trust.

Most of the definitions of trust focus on the social interactions among groups of people. Golembiewski and McConkie note about trust ([24, p. 133], as cited in [32]):

> Trust implies some degree of uncertainty as to outcome.
> Trust implies hopefulness or optimism as to outcome.

Marsh [32] expands on this:

[1] Neither [23] nor [41] actually define "trust" or "trust relationship."

> Trust is thus strongly linked to confidence in some thing, be it the person to be trusted, the environment, or whatever it is that the desirable outcome is contingent upon. We arrive at the concept of trust as choosing to put ourselves in another's hands, in that the behaviour of the other determines what we get out of a situation.

In terms of social trust, Marsh says [30] that "trust concerns a positive expectation regarding the behavior of somebody or something in a situation that entails risk to the trusting party." This is analogous to our approach, but we frame it in the negative: trust of an entity, whether social or computing, creates risk. It is then a matter of whether that risk is acceptable.

Despite similarities, social trust differs from reliance trust. Social trust arises from many intangible and sometimes irrational factors such as reputation, past performance, and personal relationships. Further, these factors do not apply when the entity being trusted has been coerced in some way.

Summing up, when people say "we trust" something, they mean there is no danger from it. It is important to realize that many uses of the word "trust" in technical work are reformulations of this notion of social trust, not reliance trust. For example, the word "trust" appears in four places in the MITRE ATT&CK Matrix [33]. All refer to social trust, in which an entity evaluates whether a third party or technology will function as expected. The controls or entities in question in each place can be monitored and controlled. Contrast this with our definition, which requires there be no capability provided to monitor or control.

In what follows, we will refer to social trust explicitly, and without the modifier, "trust" refers to reliance trust. The reader is cautioned not to draw incorrect conclusions arising from applying the notion of social trust to what follows. This, indeed, is a problem with may new paradigms, which unfortunately must co-opt existing words due to the universe of thought in which the concepts they represent are to be applied [36].

In high assurance work, trust recurs in many ways — trusted third party, trusted path, trust management, and trusted computing base, to name a few. The instantiation of trust for each of these is presented differently, but ultimately mean that an entity relies on the mechanism to work exactly as expected, with no additional functionality — and the dependent entity is unable to verify or monitor the mechanism to detect any corruption.

A key observation in this context is that when one trusts something, one relies upon it to function in some way without the ability to prevent a malfunction. If the reliance is well placed, then the system functions correctly. But if the reliance is misplaced, the system may not function correctly. And if there are controls and monitoring to identify problems, the component is not fully relied upon — the reliance is distributed over the component, controls, and monitoring instruments. In essence the reliance is distributed.

As an example, consider the trusted computing base (TCB). This underlies the subsystems of a "trustworthy" computer. Such a base is "trusted" to some degree because either it has been verified (to some degree), or the system relies on the base even though it has not been verified. In the former case, TCB trust devolves to assuming proper behavior of the verification techniques, tools, and people involved in the verification. A program or subsystem cannot validate these. So the system with the TCB relies on them to both work correctly and be used correctly; this is a case of distributing the reliance among multiple components (the TCB, the verification tools, and the humans who used those tools). In the latter case, the reliance is of necessity, because were it unreliable and therefore not used, the system could not function. Again, a program may detect some inconsistencies, but in general cannot determine if the TCB is corrupt because anything used to validate results from the TCB requires using that base — and therefore relying upon that which is being analyzed to determine how reliable it is. Note that one could say it cannot be relied upon and still use it, but then nothing the system does can be relied upon.

Another classic example of trust comes from Thompson's "Trusting Trust" paper [50]. In it, Thompson designed an attack postulated by Karger and Schell [28]. He doctored the compiler so when it compiled the login program, it added code to accept a fixed, "magic" password giving immediate access to the system to whoever used it. As the compiler added the code, examining the login program source would lead one to believe the program was not corrupted. He then doctored the compiler once more, so that it would insert the code to doctor the login program whenever the compiler was compiled. He recompiled the compiler and removed the modified source code of the compiler, replacing it with the source code for the old compiler. Now, even if one were suspicious and examined the compiler source, one would not detect the malicious code; and even if the compiler source were recompiled, the compiler would still be corrupted with the malicious code. The point here is that a component of the system, the compiler, that programmers rely upon was compromised in a way that could not be detected. So the "trust" or reliance upon the compiler here is out of necessity. Programmers needed the compiler. They did not verify it, and no verification technique that involved examining the source code would find the corruption.

This view of trust, specifically as what is relied upon, leads immediately to the identification of where vulnerabilities in the system and its operation may be located. That which can be checked, monitored, or controlled need not be relied upon, but the reliance is distributed over the mechanisms doing the checking, monitoring, or controlling. Those mechanisms are typically not checked, controlled, or monitored themselves. That which cannot be checked, monitored, or controlled must be relied upon, and are where the system is vulnerable to failures or attacks. This immediately identifies the components which, if the reliance is misplaced, can be used to compromise the system. We therefore argue that trust relationships, or more properly reliance relationships, are the root of vulnerabilities.

Consider TCP/IP network security. Networked systems are susceptible to spoofing because they accept connections based on IP addresses or network segments. These connections are often relied upon even though it is well known that they can be spoofed, simply because for a long time there was no alternative.[2] As cryptography became more widely used, and in particular with the development and deployment of TLS, DNSSEC and IPsec, or DNS over HTTPS, that trust could be transferred to the cryptographic keys (and their management) and to the managers and operators of both the DNS server(s) involved and the end hosts involved. Thus, something that

[2]The attacks by Kevin Mitnick against the San Diego Supercomputer Center as described in Takedown [44] were among the earliest seen TCP spoofing attacks.

could be easily spoofed need no longer be relied upon. But more locations (and third parties) are now relied upon, and if an attacker could compromise one of them, spoofing would still be a threat. Trust — reliance — had not been eliminated, merely shifted.

These examples demonstrate the utility of reliance trust. Indeed, if we replace "trust" with "rely upon, and hence are vulnerable to attacks from," the risk entailed becomes explicit. Our reasoning is that, in this context, "trust" is an assumption that the trusted entity will (or will not) act in some way. Indeed, penetration testers begin by looking for these assumptions, and then try to ensure the trust is misplaced, compromising the entities that rely upon those components.

Consider again Thompson's compiler. There, the users of the system, and developers working on that system, trusted the compiler, so an attacker would look to sabotage that mechanism. And indeed, that is exactly what Thompson did. The phrasing "the users were vulnerable to attacks from the compiler" makes plain the threat. Consider the vulnerabilities due to input validation failures: SQL injection, shell metacharacter exploits, buffer overflows. These vulnerabilities result from the programmer relying on the (often anonymous) client to provide the expected input.

For example, the Heartbleed vulnerability in OpenSSL exploited a particular implementation of the RFC 6520 Heartbeat extension [43], in which the sender sends a "heartbeat" request consisting of a text string and an integer indicating the length of the string. The receiver was then supposed to transmit back to the sender the same string. However, the receiver erroneously trusted that the sender would always specify a length that matched the actual length of the heartbeat payload. The result was that the receiver sent back not only the string received, but whatever additional data happened to be left in the buffer from previous use, and even data in memory beyond the end of the buffer. As the data transmitted before the heartbeat dealt with the management of X.509 private keys, the exploit allowed attackers to obtain private key information.

We should note there are degrees of reliance — one may rely completely on a component, and less completely on another component, perhaps because its functioning can be checked. That means there are degrees of vulnerability. The less acceptable the trust, the less acceptable relying on the component, and the more one will want to control and monitor. This, at least in theory, lessens the degree of vulnerability from that source. So this supports our thesis.

We turn now to applying this point of view to development and operations. Normally, both have security controls that protect their work or operations. Given those tools, one can determine what is not controlled or monitored, which identifies what it trusted, and implicitly declares that trust acceptable. The view we have proposed above suggests an inverse approach. First, identify what is relied upon. Then determine whether that reliance can be reduced with appropriate controls, and apply those controls as necessary to reduce the reliance to an acceptable level. The remaining trust is what is commonly called "accepted risk."

Again, return to Thompson's compiler. On a general purpose system it cannot be controlled or monitored. Thus, it, and any controls built using it, or that depend on it, must trust the compiler and are therefore vulnerable to it. Similarly, network traffic can be monitored and controlled, but the destination cannot control or verify the source IP address that is embedded in the packet, and hence that must be relied upon (whether we like it or not), absent some sort of independent confirmation (such as cryptographic authentication of the source). Consider a user connecting to a remote authenticated network service from a workstation not managed by the organization running the server (e.g. a researcher at a university using their university managed laptop to log in to a service at a national lab). The service cannot determine who is in control of the remote system, nor whether the person who entered the valid credentials is actually the authorized person. The service has neither control of the workstation directly nor over the security policies applied to the workstation, and no ability to monitor. The service has to rely upon the remote system being under control of the authorized user and has not been compromised.

Consider an organization that decides to run a sensitive application and/or store sensitive data on a cloud service. Cloud services basically consist of virtual machines (VMs) and networks owned and managed by a third party, the "provider." The organization cannot monitor the network or system of the cloud provider as thoroughly as if it were hosted in-house.[3] The provider may also be hosting VMs of other customers on the same hardware. Thus the organization has to rely on the VM software (chosen by the cloud provider), the cloud provider network, any other co-hosted VMs,[4] and most importantly, all the cloud provider employees and contractors who have administrative access to the VM host and the endpoints they use for administration (which could very well be home or personal systems) to not violate the agreed upon security policy. The organization has to further rely on the cloud provider having properly vetted those with access. Thus, the organization has a large number of trust relationships between their cloud instance, the vendor, and other cloud customers. Our paradigm predicts a day will come when a large cloud provider suffers a catastrophic failure of that trust, and many subscribers will no longer be able to rely on that cloud.

Perhaps the best example of our approach lies in the electronic voting systems people cast ballots on (called a "DRE"[5]). A common physical control is tamper-proof tape, to detect when an attacker has opened the physical box, and presumably tampered with its internals. This is indeed necessary, but tests have shown it is not sufficient. For example, unless the tape is placed properly, an attacker can pry open parts of the system without breaking the tamper-proof tape. Worse, it is possible to order the same tamper-proof tape, with the same numbers, over the Internet, so the tape can simply be broken and then replaced. Similarly, many e-voting systems have anti-virus controls, with the goal of preventing compromise from attacks that modify the data or software. But the utility of anti-virus controls is questionable, as these systems are special-purpose systems and certified with respect to both hardware and software. One is not allowed to download software from the Internet and install it on those systems; indeed, many DREs do not even have network cards. Indeed, the existence of anti-virus might cause one to question their trust of the voting system vendor. [34]

One can view some aspects of social trust similarly. Marsh and Dibben [31] defines lack of trust in three ways. "Untrust" means

[3] Often nothing can be monitored outside the VM instance.
[4] While theoretically isolated, VM breakout vulnerabilities are well documented, and network attacks such as spoofing may occur in a VM environment.
[5] "DRE" is an abbreviation for "Direct Recording Electronic".

the truster expects the trustee not to cooperate with the truster; in terms of reliance, it means that the truster cannot rely on the trustee. "Distrust" means the truster expects the trustee to act in a way inimical to the truster; in terms of reliance, this means the truster can rely on the trustee to act against the truster. "Mistrust" means trust is misplaced; in terms of reliance, it is misplaced reliance. Considering that trust and confidence are directly correlated, this view is unsurprising

The alternate approach proposed here focuses on what must be relied upon, and hence trusted, as opposed to what has vulnerabilities. With respect to software, the trust points are the software development and functioning, delivery, and installation. With respect to the hardware, the trust points are the points at which the hardware can be compromised, which requires direct access to the hardware. With respect to system management, it is the trust relationships created by the architectural and configuration decisions, the privileges granted to the system administrator, and third-party relationships. These lead naturally to mechanisms to reduce trust by controling and monitoring the systems.

3 SURVEY OF RISK ANALYSIS METHODOLOGIES AND BEST PRACTICES

Given that the concept of trust relationships, and consequences of their violation, are not new, one would naturally assume that common computer security best practices would address the establishment of trust relationships in an organization's computing environment. A survey of the risk analysis frameworks, industry requirements, and other recommended best practices finds that, with one exception, trust relationships hardly merit a mention. And that one exception discusses trust relationships at such a high level, and in just one document of many, that it does not percolate down to the actual approach to vulnerability assessment.

As we are focusing on what gets applied in practice, we limited our survey to risk analysis approaches that a *practitioner* would likely use or be required to use, starting with an Internet search. The results of which were: FISMA (FIPS 200 and related documents)[6] [35], OCTAVE [5], SERA [6], TARA [54], Common Criteria [1], PCI-DSS [2], SANS Consensus Server Security Policy [42], and CIS Controls [14]. We reviewed these approaches looking at their guidance on identifying vulnerabilities, for any discussion of trust relationships or a similar concept.

Every reviewed framework has essentially the same theme: identify assets, threats and vulnerabilities, compute risk, and apply controls. The frameworks vary either in how prescriptive they are, or the procedures to be used, or both. For example, FISMA uses a quantitative risk analysis to choose a subset of predefined controls. The OCTAVE framework uses a more open-ended approach and emphasizes participation in the evaluation process by organizational stakeholders. As PCI-DSS deals with a specific situation (processing credit card purchases), it skips the risk analysis step completely and provides an entirely prescriptive set of controls to implement.

Only FISMA, specifically NIST SP800-39, delves into significant discussion of trust relationships. The Common Criteria comes somewhat close in requiring analysis of hypotheses of flaws. Of the rest, we found only one mention of trust relationships [14], without an accompanying definition.[7] We found even that discussion of trust was largely limited to an assumed understanding of "trusted host" or "trusted user."

NIST SP800-39 provides a good working definition of trust relationships, and points out a myriad of ways in which trust relationships can exist. However, the FISMA risk analysis process is described in a grand bouquet of documents, many of which are guidelines that do not need to be followed or even read in order to achieve FISMA compliance.[8] The concepts of trust in SP 800-39 are not carried through to the documents that a compliance plan would focus on, such as NIST SP800-30 and SP800-53. Furthermore, while NIST SP800-39 mentions trust relationships at a variety of levels, the detailed discussion focuses on third-party trust. This focus is further amplified in NIST SP800-37 and NIST SP800-53A, which only mention trust relationships in the context of third parties.

All other trust relationships get addressed by asserting a trusted system is "trustworthy," resulting from the application of the controls outlined in NIST SP800-53. The rhetoric used in NIST SP800-39 implies that such trust relationships are a positive because the system is trustworthy since it is "secure" (in reality, *compliant*). We also highlight here the assumption that the application of controls (*controls-based security*) assumes that the controls provide effective security. Finally, as stated above, many practitioners will not take the time to read NIST SP800-39, let alone translate the information provided into an actual risk assessment; indeed, they may not understand the NIST risk analysis procedure, or be unable to apply it in their environment. In fact, with FISMA's focus on controls, one can largely produce a FISMA compliant plan by following only FIPS-199 and NIST 800-53.

The identification of vulnerabilities is fundamental to performing a risk analysis. One would expect that risk analysis methodologies would provide considerable detail as to how to reliably do so. We found that, in contrast to lengthy discussions of how to identify and enumerate threats, short shrift was given to the identification of vulnerabilities. In almost all cases they essentially say to identify known vulnerabilities, using published catalogs such as the CVE, NVD, CAPEC, and/or using vulnerability assessment tools.

The Common Criteria is somewhat of an exception, in that it explicitly calls for the assessor to identify the design for potential vulnerabilities, using a flaw hypothesis methodology. This may lead to inference of trust relationships, but it is not driven by trust relationships. NIST SP800-39 also discusses architectural and configuration flaws, but again at such a high level that neither are useful to a practitioner not already trained to make such an assessment.[9]

These approaches to vulnerability assessment are fraught with potential for failure. To begin with, they essentially rely on the assessor, tool and/or catalog to be a perfect oracle of all vulnerabilities. More importantly, relying on catalogs of known vulnerabilities

[6] FISMA is technically the legislation mandating the application of FIPS 200 for federal agencies, but the risk analysis methodology using this FIPS is commonly called "FISMA".

[7] We infer from the context that it refers specifically to Microsoft Windows cross-domain authentication.

[8] And will not be read by any practitioner that is trying to complete a risk assessment in their lifetime.

[9] This would appear to be a bootstrapping problem.

(and scanners are essentially an automation of those catalogs) completely overlooks risks created by configuration choices and the trust relationships created from such. For example, NFS servers have a configuration option that blocks mounts from "unprivileged ports" (port numbers greater than 1024). When that option is disabled, a user on any NFS client can masquerade as any user known to the NFS server. The vulnerability created by disabling that option is not cataloged in the CVE nor in CAPEC.

In contrast, an assessment that documents trust relationships would identify the trust an NFS places in clients (and in the network in general) and would naturally consider this as a point of potential vulnerability, and highlight not only the need for proper configuration of the NFS server, but policies for configuration and management of the clients and client networks.

Instead of the short shrift given to vulnerability assessment, the frameworks devote considerable effort to the process of threat assessment, and claim that once threats of concern are identified, one can ignore any other threats (and their corresponding risk). This approach has the same fundamental problem as the approach to vulnerability assessment: it relies on the assessor being a perfect oracle of threats to the organization. We believe that many cases of compromise were the result of failure to identify valid threats, and in many cases the threats were not even of consideration when the assessment was performed.

Now, it might seem that our paradigm merely rotates the shield harmonics;[10] rather than being a perfect oracle of vulnerabilities, one has to be a perfect oracle of trust. We argue that identifying trust relationships does not require the very specific technical expertise that identifying vulnerabilities does. Concepts of trust are more universally understood. Where the details of a trust relationship require more technical expertise, the analyst can enlist someone with the necessary technical expertise.[11]

As an example, consider ransomware. Ransomware first appeared in 1989 [15]. It is similar to the threat of data deletion, as it makes data unavailable, but the victim can, at least in theory, recover the data by complying with the demands embodied in the ransomware (usually, paying some sum of money). It also combines malware with data deletion. But this threat was not considered critical until recently, because ransomware has become more widespread than in the past. In our paradigm, one might identify the data as a valuable asset, that a trust relationship exists with the systems and people authorized to modify the data. A control to reduce that trust might be to archive immutable copies of the data, enabling recovery of corrupted data. The control minimizes the risk from ransomware regardless of existence or awareness of the specific threat.

4 TRUST-FIRST PARADIGM

We propose a new paradigm of basing security upon trust. The difference between our paradigm and the current one is that the current paradigm takes a threat model, derives requirements, designs, implementations, and so forth, and then might examine trust relationships induced by the implementation and deployment (and in those cases, the trust relationship are more often implied than explicit). Within the threat model and requirements, one makes assumptions about the environment, the use of the system, and the policies and procedures used to manage the system. Our paradigm inverts this. At each step, we ask what assumptions of trust have been made, and whether that trust is (or must be) acceptable. When trust is not acceptable, we then ask how we reduce or eliminate that trust, how do we monitor for violation of that trust, and what assumptions we must make. Then we base our development of threats, requirements, design, and so forth on those assumptions.

As a simple example, consider the architecting of a local area network. Such a network needs to be protected, and one way to do so is to position a guard — a firewall — at the connection between the LAN and the Internet. Doing so assumes the LAN at its site is secure from internal attack, so we note the assumption. The use of such a guard also requires us to trust the guard works, and does not act inimically to us. To determine this, we can ask the vendor about what the guard trusts, and examine those trust relationships. We also explicitly assume that any additional functionality is not inimical to the institution of the LAN, or the LAN itself. Proceeding further, we now know we need to focus on:

- The insider threat (from the assumption that the LAN is secure from inside attack); and
- The firewall's ability to protect the organization and the network (from the assumption that the guard works), and its inability to protect the organization from the payload of traffic allowed through the firewall. Indeed, one might say the firewall is only trusted for the things that it blocks, the allowed traffic being a separate trust relationship (e.g., with the source endpoint). One might even conclude that, if the local network is not trusted, the firewall does not add much value.

As another example, we trust vendors to implement patches correctly, and maintain the changes through future releases. Hence, before installing a patch, examining the trust gives two possible threats: that the vendor can implement the patch incorrectly, or the correctly implemented patch will not interfere with our needs. Many years ago, an anti-virus program given to one university for testing turned out to have a virus in itself. When Microsoft deployed Windows NT Service Pack 2 to improve the security of the system, many people who installed it found it blocked their processes — effectively, a denial of service attack.[12] In another case, a failure to reload pages that a debugger had altered allowed anyone to become the administrative user [8]. The vendor's next minor release fixed the problem. But on the minor release after that, the vulnerability returned. Focusing on threats rather than deriving them from trust would fail to identify these problems.

Furthermore, the entire anti-virus software industry has developed from an implicit lack of trust in software. To the extent that anti-virus software reduces trust in the software it ostensibly protects, the trust has simply been moved to another piece of software — the anti-virus software itself.

[10] https://memory-alpha.fandom.com/wiki/Shield_frequency

[11] One of the authors approaches this by having people knowledgeable about the system (e.g. developers, system administrators) participate in the analysis.

[12] The problem was that the service pack activated an internal firewall that blocked ports used by the processes. Once the firewall's ruleset allowed traffic through those ports, the problems vanished.

This leads to the observation that trust is not binary. Drawn from the social sphere, we may trust our spouse completely. We may trust an acquaintance enough to lend them $5.00, but not $5,000.00. We may trust a complete stranger not at all. The first and last examples are complete trust and no trust. The middle example is partial trust — we trust our acquaintance for small amounts of money, but balk at trusting him for large amounts. Note however that for computer-based trust, often one cannot partially trust.

A system administrator may configure a server, creating a trust relationship with an untrustworthy system without realizing it, or improperly assessing the risks associated with the relationship. This trust relationship gets established at such a low level that it never rises to the attention of the risk assessment group. Thus the configuration decision could cause a cascading effect, and a small configuration error can have a profound impact on security.

This also leads to a second observation. Looking at things through the lens of trust (reliance) enables the assignment of priorities, and in particular when something cannot be relied upon (or, if it must be, when to assume reliability – and here vulnerabilities may arise). This immediately leads to Elgesem's observation [18] that trust management *minimizes* trust, because it reduces what and how trust must be given. We argue that reliance is analogous to trust management here.

In high assurance development, evaluating assurance trust in the product requires the development of assurance evidence that can be evaluated by experts in the area. Experts may weigh the various types of evidence differently; they then iron out their differences to come to a common evaluation level. The point is they must reconcile differing ideas, and values, of trust in both the assurance evidence and the methodology used to gather them.

But trust is ubiquitous; we do not argue here that all trust must be eliminated — ultimately, something must be trusted. We trust in the laws of physics (despite several attempts to repeal the Law of Gravity). We trust vendors who build hardware to use reliable hardware, but the degree of trust is shown in whether we purchase an extended warranty. Again, trust is not binary.

The important aspect of trust is to understand what is being trusted, and what happens if the trust is misplaced. Stanley Kubrick's movie *Dr. Strangelove; Or, How I Learned to Stop Worrying and Trust the Bomb* makes this point in two effective ways. The first is when the fail-safe box on a B-52 failed due to a near hit by a Russian missile. When the recall code is sent, the President and US Air Force believe the bombers have all been recalled because the fail-safe boxes show the bomber crews that they are to return to base. But the bomber with the damaged box continues, much to the surprise of everyone. Here, the fail-safe boxes were trusted to work; when one failed, a bomber carrying nuclear bombs continued on its mission. The story does not end there, as the Russians have developed a "doomsday machine" that will destroy the world should they be attacked.[13] The Russians realize that the US attack was launched by accident, but the doomsday machine cannot be disarmed because, as Dr. Strangelove points out, that would defeat its purpose. Hence the second trust, placed in the belief that no attack can be launched by accident, was misplaced trust, and — more critically — no-one thought of how to handle such a situation. A (somewhat eerily similar) real-life example is the Permissive Action Links put on nuclear missile control systems in the 1960s. The Strategic Air Command worried about trusting NATO allies with US warheads stationed in their respective countries, along with concerns about US generals being able to unilaterally launch missiles. However, a fear of (or, in other terms, a lack of trust in) missile operations properly managing the keys (and rendering themselves unable to arm the missiles) resulted in the PAL passcodes being set to all zeros, and remained that way for over a decade [7].

Trust in the computer security world is often considered to be transitive.[14] Return to the case of the remote user. The user is trusted to prevent their credentials from getting stolen and used. The system administrator of the remote workstation is trusted to not bypass access controls (or allow a malicious third party to do so) and gain access to the user's credentials and/or hijack the authenticated session to the server. The client software and operating system is trusted to enforce access controls and ensure that only the authorized user has access to their local account. The client software and OS developer are trusted to not introduce malware in software updates. The system administrators working for the software developer are likewise trusted.

The transitivity of trust does indeed turn into a case of "turtles all the way down" [37]. Again, our point is simply to emphasize understanding the trust relationships and then decide whether they are acceptable or whether some other solution is more appropriate. For example, when providing remote access to the controls of a very expensive detector for an astrophysics experiment, the organization may choose to allow remote access only from workstations configured, maintained, and monitored by the organization, that do not in turn allow remote access, and conforming to the same security policy as the instrument's local network. By doing so, the organization is reducing the trust in the remote access to the same trust relationships as at the local network.[15]

Security consists of two main categories of actions: monitoring and controlling [9]. This ties directly into our point of trust underlying everything. One can view monitoring as observing trust, and controlling as reducing trust. For example, we trust networks to some degree; otherwise we would not connect our computers to them. But we monitor the network traffic to detect violations of that trust, because we do not completely trust the networks. Similarly, we use firewalls and other mechanisms to reduce the amount of trust we must place in network traffic, transferring some of that trust to the firewall and other mechanisms. So our "trust surface" shifts from the network and network traffic to the controlling mechanisms. And that which we do not monitor or control, we must trust completely. The phrase "trust but verify" is quite apt here, with the understanding that some things are simply infeasible to verify.

5 REAL WORLD COMPROMISES

The problem of identifying trust relationships is at the root of many recent compromises. The compromised organizations either did not

[13] This parallels an effort led by Dr. Strangelove to develop a similar system for the US; the Russians learned about it from a story in *The New York Times*.

[14] In reality, this is not true; but it is usually assumed in practice.

[15] To forestall criticism, we assume that the local network is trusted no more than the remote network; we touch on the problems of local network trust elsewhere.

identify the trust relationships that the attackers exploited, or did not consider what could happen given an existing trust relationship. Several examples will make this point.

5.1 Target Compromise

In 2012, the Target Corporation had millions of customer credit card numbers stolen as the result of compromised point-of-sale (POS) systems [46]. Several key mistakes contributed heavily to the success of the intrusion. The Target network was relatively flat, so that the POS systems were accessible through the company's general network. The POS systems had vulnerabilities exploitable via the network. An HVAC vendor's network was connected to Target's internal network, to provide the vendor with access to Target's internal systems. The attackers compromised the vendor's network, then a host on the Target network, and then compromised the POS system.

The POS systems trusted Target's network, and Target trusted the vendor's network. When selecting the POS system to use, did Target ask the vendor what assumptions the vendor made about the network on which the POS system was to be installed? Did Target consider the connection to the vendor network to be a trust relationship? Perhaps, had Target understood that the POS systems trusted the networks they were connected to, they might have chosen to keep the POS systems isolated from the rest of the network, and impose more restriction on activity from the vendor network, or ask the vendor about the security practices on their network.

5.2 Stakkatto

Between 2003-2005, an attacker calling himself Stakkatto compromised systems at numerous universities, research facilities, government and military sites, and more [47, 48].[16] These incidents are rich with examples of exploited trust relationships. Stakkatto used a trojaned SSH client to get user credentials as users logged into remote hosts. At some sites, system administrators would log from their workstation, laptop, or home computer directly into systems as the user *root*. Once Stakkatto had installed the trojan on their workstations, he[17] obtained *root* access to the organization's systems without any additional effort.[18] The target systems trusted the system administrators' workstations, which often did not adhere to the same security policies as the systems themselves.

Stakkatto also exploited another trust relationship that often goes unnoticed. He used a 10-year old tool called *nfsshell* [38, 53] to execute the NFS client protocol from the command line of a user-level account. This allowed him a variety of exploits. At a site with shared user home directories, he was able to access files using the UID of any user on the system. At a site that had a filesystem mounted by both a Linux cluster and an AIX cluster, and had used neither the *root-squash* nor the *disable setuid* options, he compromised the AIX system from the (already compromised) Linux system, by writing setuid-to-*root* scripts on the shared filesystem, logging into a compromised user account on the AIX system, and then simply executing the script. These exploits did not make use of any software bugs, but took advantage of the NFS protocol's trust in the NFS clients, and in the latter case, the trust relationship between the two clusters as a result of their shared filesystem and its configuration.

The NFS protocol requires almost complete trust in the clients, and so lets the client assert the numeric UID of the user making a file request. The server has no means to validate the UID provided. This means that the NFS server trusts the client to provide valid UIDs. Furthermore, once a file has been opened and an NFS file handle obtained, mere possession of it gives the type of access requested when the file handle was obtained. So the NFS protocol requires not only that the server trust all of its clients, but also any host that can communicate with it.

5.3 Apache.org Compromise

In 2001, a group of friendly attackers compromised apache.org [3] through a combination of configuration errors, beginning with the document root for the web server and the FTP server (on the same host) being the same directory. An upload of a script via the FTP server could be executed as a CGI program using the web server. When this was done, the system was compromised. The enabling trust relationship in this attack was the decision to trust the web server to execute scripts that could be uploaded via FTP.[19] Did the system administrator who made this decision realize that (s)he was creating a trust relationship between the two services?

5.4 RSA Corporation Compromise

The RSA Corporation compromise of 2011 resulted in theft of the seeds to the SecurID tokens used by their customers to harden access to their own systems [29]. The attacks began with successful phishing attacks against RSA staff ("low priority targets"), which gave intruders command-level access to systems at RSA. The attackers were then able to escalate privileges to *administrator* accounts, and uncover credentials used to access databases containing unencrypted seeds used by the SecurID tokens. The first trust relationship violated was that of trusting users to not get phished[20] followed by trusting users who had access to systems from which sensitive information could be exfiltrated. Next was trusting local users not to exploit privilege escalation vulnerabilities,[21] and finally trusting anyone at RSA to obtain unencrypted access to customer seeds. Our final example of trust here is the trust that customers placed in RSA (a classic "third party") to protect the security of the customer's tokens.[22]

Phishing has become a common point of entry for Internet attacks, and is an interesting example of misplaced trust. Phishing attacks rely on the target user unwittingly handing over credentials to the attackers, or opening a file containing malware which provides the attackers with access to the target user's workstation. From there, the attacker leverages authorizations granted to the user. The "best practice" to combat phishing attacks has been to

[16] Estimates of the number of sites compromised number in the thousands.
[17] Stakkatto was eventually caught; his identity is public knowledge.
[18] Logging in directly as *root* is deprecated for exactly this reason, but it is still done by many of the younger generation of system administrators.
[19] It is likely this trust relationship was not identified during the choice of configuration; that is the point here.
[20] Admittedly a difficult problem, but misplaced trust in users is commonplace.
[21] Also a common oversight, in our observation.
[22] Indeed, some RSA customers did get systems compromised as a result of the stolen keys.

educate the users. But this practice is doomed to failure; it is improbable that all users have enough expertise to understand how to avoid such attacks. Even the users who do must be able to act correctly each and every time without mistake. The organization has a trust relationship that the users will act correctly, but there cannot be an expectation of proper behavior at all times.

The other increasingly common technology used to address phishing is two-factor authentication, typically with one of the factors being a one-time password.[23] When a user successfully authenticates using the two factors, the organization trusts that the entity connecting is actually the user. However, the organization is also trusting the user's endpoint (remote computer), not just the user. The authentication does not validate the trustworthiness of the user's endpoint. All the user has done is prove that (s)he was inputting data at some time in order to authenticate.

5.5 Equifax Compromise

Equifax, a credit reporting agency, is trusted by most individuals in the US to protect their credit information and related personally identifiable information (PII). It failed to do so in 2017 [22]. Equifax reported that intruders had gained access to the records of over 145 million people. The intrusion began through exploitation of a recently announced vulnerability in a web server. Two months after the intrusion began, the attackers began accessing PII information. Similar to the RSA compromise, the attackers not only obtained access to databases with PII through the host they had compromised, but they also found a repository with cleartext usernames and passwords, which they used to obtain access to many more databases. Equifax did not discover the compromise until 76 days after data exfiltration began. Ironically, one thing that contributed to their failure to detect the incident sooner was that the system that they trusted to monitoring network traffic and detect suspicious activity had not been doing so for over 10 months, due to an expired SSL certificate. Equifax also appears to have placed too much trust in hosts on their network to access databases with PII, let alone trust anyone who could access the repository with cleartext credentials.

5.6 Stanford Incident

Finally, we harken back to an incident that began at Stanford University and resulted in the compromise of a number of systems across the ARPANET [40]. An "obscure" mail gateway was compromised as a result of attackers guessing the password of a guest account. The intruders were able to escalate privileges due to default permissions on a directory in *root*'s path; this allowed the guest account to write a script that *root* would then execute. From the *root* account, the intruder could assume the identify of any user on the system. The intruder was able to use the *rlogin* program to compromise user accounts on other systems, on which the users had configured their *rhosts* file to allow access from their account on the compromised system. The intruder repeated this process to compromise machine after machine.[24] The author of a report on the incident notes: "The machine on which the initial break-in occurred was one I did not even know existed, and no one in my department had any control over it." Here users created the trust relationships between independent systems without the knowledge of the respective system administrators.[25]

6 TRUST-BASED ANALYSIS

Here we outline, at a very high level, how a risk analysis might be implemented under this paradigm. We do not intend to provide a complete or even working framework, but merely to illustrate how this approach compares to common practice.

Consider a design for an enterprise network. We begin with identifying assets that, if compromised, can result in damage to the organization. We then query what each of those assets trust; what (or who) has the ability to access or modify the assets, and label them as trusted entities. Those trust relationships are vectors of potential compromise. We then evaluate whether those trust relationships are acceptable, and where not, what controls are required to reduce the trust. [26] While our discussion is presented at a rather high and abstract level, in practice this process would be applied at various levels, taking into consideration the operating system, services and software, hardware, people (users, system administrators), network topology, and so forth.

We then repeat the process, with the trusted entities treated as assets, and identifying what they in turn trust. By doing so we identify transitive trust relationships that can provide a path to compromise our primary assets.

Figure 1 shows this process. Part (a) shows a trusted asset. Part (b) shows how that trust relationship can be handled: it can be eliminated, reduced, or monitored; in the absence of these, it must be accepted. Parts (c) and (d) show one iteration of the process. Part (e) emphasizes the iterative process as "turtles all the way down".

For example, for a computer system that is a primary asset, we would identify the operating system as a trusted entity. We would then identify the system administrator, who has privileged access to the operating system, as a trusted entity with respect to the operating system. Then we would identify any system used by the administrator to remotely access the operating system, and label it as trusted by the administrator. We might then ask whether that trust is acceptable, or what controls on the remote system would be required to make it acceptable.[27]

One might dismiss this paradigm as creating an Everlasting Gobstopper[28] of trust evaluation, each trusted entity leading to a new set of transitive trust relationships. But the same is true for other methods of risk analysis, because (for example) the risk involved in installing software goes through the vendor to the systems the vendor uses and their programmers, and the software they use, leading to the vendors of that software, and so forth. In practice, all risk analysis methodologies have a finite scope and

[23] Note this is what RSA SecurID does.

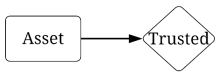
[24] The *modus operandi* of the intruder was eerily similar to that of Stakkatto almost 20 years later.

[25] This was a common practice in the day, and similar to today's users not using passwords to protect ssh keys that enable them to move between systems without further authentication.

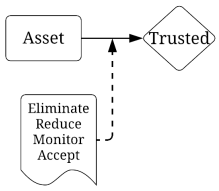
[26] The ability to identify trusted entities may be the first control required. One of the authors has more than once conducted reviews in which he asked "who has the root password to this system?" and was told "We don't know".

[27] Such a trust relationship frequently gets overlooked. One of the authors, while being presented with a detailed design for a HIPAA-compliant environment that was designed for remote contractor access, asked about the requirements for the remote systems and was told "we would expect that they have implemented good security on their systems."

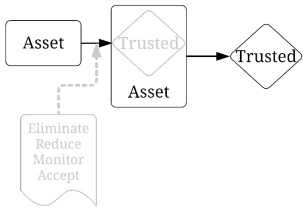
[28] A hard candy that never disappears no matter how hard it is sucked [17].



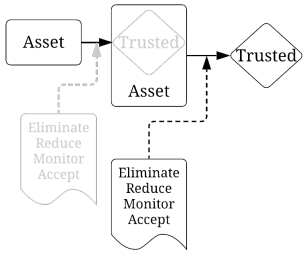
(a) Trust and an asset



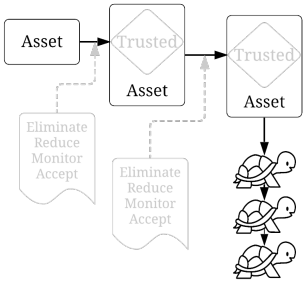
(b) How the trust can be handled



(c) Iterating this process



(d) Handling the iterated trust



(e) Turtles all the way down

Figure 1: The iterative process of analyzing trust relationships

depth; we do not claim that our approach is any different in that respect. However we do claim that it may have a better outcome.

At some point, one may simply declare an entity trusted and move on. The common case of doing so is trusting hardware.[29]

[29] Although recent events with Huawei has some people rethinking that.

The benefit of our paradigm is that, where the scope of the analysis ends, the accepted trust relationships are made explicit, acknowledged, and naturally align on a trust boundary.

6.1 Example: Internet Voting

Using the Internet to cast votes in civic elections provides a good example of our approach. Instead of examining the attack surface to find threats and evaluate risks, we approach the problem in an inverse way. The key issue is, *what* is the attack surface? To answer that question, we examine what needs to be trusted, what can be trusted but monitored, and what cannot be trusted.

A brief description of an Internet voting system will place our paradigm in context. We postulate a voter sitting at their home computer. On Election Day, they connect to their election office server and authenticate as required by their state law. They then are taken to a web page that presents an appropriate ballot. The voter marks the ballot using their input devices, either selecting items in each race or writing in names. They then click on a button to display their votes, allowing them to recheck that they cast their proper votes. At that point, they can either go back and correct any errors, or click another button to cast their votes. Once cast, the server saves both the votes and an image of the filled-out ballot. At the end of the election day (or period, if it extends over multiple days), the election servers tally the votes.

The trust points range over the entire system. The election server's software and hardware must be trusted, as must physical access to that system. The user's home computer must also be trusted to render the ballot correctly and record and transmit the user inputs correctly to the server. The communication between the user and server must be trusted and secret, so an observer cannot determine how the voter voted. (That a particular voter voted is not confidential; indeed, in an in-person election, lists of those who voted are posted at the polling stations every hour.) Indeed, the user's computer must be talking to the legitimate election server. Further, the reporting of the votes must correctly reflect the cast votes, and the user must be able to read the ballot, understand it, and mark it appropriately. Finally, the method of authentication must satisfy the relevant (in the U.S., usually state) law.

Other points of trust are more subtle. First, the voter must be able to access the election server on election day, and do so within a reasonable time. Next, the voter cannot record the cast vote in such a way that they can prove how they voted — this prevents both the selling of votes and people voting a certain way under threat. Third, the authentication must be independent of the casting of the votes, lest the two be combined to breach voter anonymity and ballot secrecy. The results must be correctly reported.

Now that we have identified the "trust surface," we can minimize it. We can reduce our trust in the election server by ensuring it is tested and analyzed by experts — or, even better, developed with high-assurance techniques. But this simply transfers some of the trust to the examination, and in turn the examiners and those who prepare the system for examination. The communication can be secured by a suitable network protocol, such as TLSv1.3, but with anonymous certificates on the part of the voter. This also ensures the user's computer is connected to the genuine election server — and again, the trust is not eliminated but transferred to the

management of keys. The same is true with access; here, the trust is in both system management and on the ability of the election office to handle DDoS attacks. And the software and hardware of the user's home computer is required to ensure the votes are properly recorded and transmitted.

So, the trust surface consists of:

- the voter voting;
- the voter's home computer;
- the voter's authentication information;
- the authentication system or method;
- the election server;
- the connection between the election server and voter's computer;
- the tallying of the votes; and
- the reporting of the votes.

We can now look at ameliorating these. The details are beyond the scope of the paper, but it is instructive to contrast this with in-person voting on paper ballots, the traditional method in most of the U.S. The trust surface is similar:

- the voter voting;
- the voting instrument (marker) working[30]
- the voter's authentication information;
- the authentication system or method;
- the processing and counting of the ballots at Election Central;
- the movement of the ballots from the polling station to Election Central;
- the tallying of the votes; and
- the reporting of the votes.

Here, minimizing the trust surface uses very different methods. For example, one can follow behind the car transporting the ballots from polling stations to Election Central, to ensure they arrive. At least two people are always with the ballots until after they are counted. These differ from the Internet voting control because the observer can watch the process of transport, whereas with Internet voting, one must trust that the communication protocols work correctly and that the voter's workstation is not compromised.

Now consider elections in which voters vote on DREs, and contrast that with the Internet voting process described there. To vote on a DRE, the voter physically goes to a polling station and, after signing in and being validated as a voter, then goes to the DRE and votes. With in-person voting, the voting authority has to trust a relatively small number of machines which (theoretically) have a validated initial state, and compromise of which requires physical access and does not scale. In comparison, Internet voting requires the voting authority to trust as many systems as there are voters, with no means to validate any state, and which can be compromised remotely at scale. It effectively allows voters to each have their own unvalidated voting system. The trust surface is much larger, and there is no means of reducing the trust.

[30] In one jurisdiction that conducted voting on paper, the markers used at one precinct ran out of ink. After that, voters were told the pens used ink that was visible only to the counting systems. This lasted about an hour until a roving inspector arrived. New markers were quickly obtained.

Thus, focusing on trust shows how to develop risk models and attack surfaces. Further, it provides a direct way to compare different systems designed to achieve the same goal — focus on the trust surface.

6.2 Example: Password Policy

As another example, let us look at common password policies through the lens of our approach. We require users to choose a password unique to our system (that is, distinct from other systems they might use), as we do not want to trust that the password will not get exposed by compromise of one of those other sites. But in practice, we can neither prevent the user from actually doing so (controlling), nor detect whether the user has done so (monitoring); we have to trust the user. We also give users requirements for the composition of their password, to make the passwords resistant to dictionary attacks. We can force a user to meet the letter of the requirement; unless the requirement is carefully drawn and completely enforced, we will have to trust that the user's chosen password has a high entropy across the field of all possible user chosen passwords. Yet numerous studies have shown that for both requirements, users routinely violate this trust. For the latter, in addition to other motivations, the user is rarely capable of determining whether their password meets the intended goal.[31] Acknowledging this potential failure of trust, we might choose a different approach that trusts the user less. For example, we might not allow users to create their own password, but instead let the user choose from a small set of randomly generated passwords, to ensure sufficient entropy in the password space [49]. Or we might use a one-time password system, which prevents the user from re-using a password. A side benefit is that, if most sites required users to choose from randomly generated passwords, users would not be able to re-use passwords across sites.

6.3 Framing

The above examples illustrate how our paradigm contributes to better understanding of threats. One's use of language has an impact on understanding. Greenwald [25] showed how the use of specific verbs in security policies affects people's understanding and properly applying the policies. Fichtner *et al.* [20] shows how "framing" of security discussion affects understanding of security issues; by reframing the discussion one can do a better job of getting one's point across.

The concept of "trust" is generally understood by most people at a visceral level, even if they can't provide a good working definition. In the authors' experiences, one of the difficulties with the threat-vulnerability-risk approach is in getting stakeholders, especially non-technical ones, to understand the risk. Viewing "trust" as "relies on" helps. For example, describing a risk as "an attacker could execute a SQL injection attack and override the authorization table" is incomprehensible to the non-technical listener, with a concomitant glazed-over response, or responding in denials such as "why would anyone do that?" or "nobody will ever be able to figure that out." But describing a risk as "the server is relying on people we

[31] This is also a case of the security policy not properly aligned with the goal, but that is a different topic.

don't necessarily trust to access the system in a particular manner" is much easier for the stakeholder to understand and accept.

In our internet voting example above, framing the issues in terms of trust makes it much easier to show how it is infeasible to ensure then integrity of the voting process when the trust surface encompasses millions of systems, with no means to detect let alone prevent compromise. In our password policy example, framing the policies in terms of trusting the user illuminates how the problem with non-compliance is not a matter of tweaking the password rules, but that the policy has resulted in misplaced trust (in the users).

7 RELATED WORK

Trust has been a subject of much study. All definitions seems to have common elements. For our purposes, the key element is that *trust springs from belief in that which we do not, or cannot, monitor* [21]. Castelfranchi and Falcone [13] identify several belief types underlying trust, including a dependence belief (in which one needs a second to perform some action, relies on the second performing some action, or sees relying on the second as better than not doing so) and a disposition belief (in which one believes that a second will carry out what is needed for the first to reach a goal). From our perspective, these two beliefs underlie the notion of trust in computing, because systems depend on what they trust, and assume the trusted entities will behave properly.

Policy trust models evaluate trust based on a given policy [10, 11, 51] and recommendation systems [4, 55]. Tucker [52] presents a graph-based methodology for analyzing trust. Similarly, Huang and Nicol [26] develop an analytic technique to represent trust and risk associated with a public key infrastructure. Marsh presents formalizations of trust that can be used to identify trust/risk mismatches [31]. These techniques could be applied to analyze trust within our paradigm, as we simply shift the analysis of trust in the development cycle.

Karger and Schell [28] examined the trust systems placed in programming infrastructure such as compilers and linkers. As noted above, Thompson [50] presents an example of this trust going wrong.

In his introduction to trust mapping, Tucker comments on risks introduced by trust relationships, and states that some vulnerabilities derive from trust. This is in line with our argument, the key difference being that we assert trust is the cause of all vulnerabilities [52].

SATAN [19, 39], perhaps the first widely distributed vulnerability scanner, developed maps of trust relationships based on the connections machines made to each other and the services they provided. For example, it would detect that the DNS servers were trusted by hosts in the domain, or used as a resolver. It would also detect which systems trusted the authentication mechanisms of other systems, and what the consequences of misplaced trust might be. Unfortunately, to our knowledge, no vulnerability scanner since then has included any identification of trust relationships.

The term *zero trust* [41] refers to ideas and methods for minimizing uncertainty in policy enforcement. This paradigm is simply an extension of an earlier paradigm, before firewalls became ubiquitous. Hosts connected to the ARPANET had to be hardened, because there was no intermediate agent. This minimized trust in the network, instead allocating it to non-network entities, and to those in a degree depending on how much the system managers trusted those remote entities not to be vulnerable to attack, or have vulnerabilities that can be exploited. Our paradigm inverts this; we ask what vulnerabilities may arise by the placement of trust in particular entities. In other words, we ask what is considered reliable enough that we can rely on it. This may involve some of the technologies used in zero trust, or it may involve determining some other level or means (including non-technical[32]) of trust or control. In other terms, we look for the trust boundaries [45] of our system and consider what lies beyond them, that is, how reliable external components are.

As an example of how our paradigm differs from the one is use today, consider the methodology proposed by Jøsang and Knapskog [27]. This paper, like many others, discusses security evaluations as a method for determining trust. Our paradigm is the opposite: determine trust as a basis for security evaluation. This seems simpler, as security rests on assumptions, and those assumptions are based on trust.

8 FUTURE WORK

Future work could expand the paradigm presented here to a full-fledged risk analysis methodology that can get used in practice. A comparative test would demonstrate the efficacy of this framework compared to other, current frameworks. A real-world A/B comparative test would be optimal, but difficult; a simpler approach would be to taking an existing organization that has applied a traditional framework, subject it to a trust-based analysis, and see how the vulnerability assessment and resulting controls differ.

9 CONCLUSION

Organizations have been repeatedly compromised, often to catastrophic effect, either financially or reputationally. These compromises invariably are traced to systems that were improperly maintained or configured, or to attacks from trusted systems or accounts. These arise from misplaced trust — trust that the system configuration would prevent compromises, trust that users of the accounts would not attack the system. So, trust and trust relationships should be among the first aspects of design of systems and networks to be derived and examined. Unfortunately, existing risk management approaches fail to address trust and trust relationships as a key element of risk.

Our paradigm is to use trust relationships as a basis for assessing security and developing security policies and mechanisms to protect systems. Then security can be realistically assessed, and relationships where trust is present but unnecessary (because the trusted component(s) can be monitored or controlled) can provide a basis for acknowledging vulnerabilities that an attacker can exploit. After all, security rests on assumptions; and these assumptions themselves are a result of trust relationships.

Acknowledgements. Matt Bishop gratefully acknowledges support from the National Science Foundation under grants DGE-1303211 and OAC-1739025 to the University of California at Davis.

[32] Indemnification is a common non-technical means of reducing risk due to a trust relationship

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Abe Singer gratefully acknowledges his partner in life and crime, Dr. Uyen Pham, for her support and indulgence in his working on this paper.

REFERENCES

[1] . 2017. The Common Criteria. https://www.commoncriteriaportal.org https://www.commoncriteriaportal.org.

[2] . 2018. Payment Card Industry (PCI) Data Security Standard. https://www.pcisecuritystandards.org/ https://www.pcisecuritystandards.org/.

[3] {} and Hardbeat. 2000. How We Defaced www.apache.org. Bugtraq Mailing List. https://seclists.org/bugtraq/2000/May/54 https://seclists.org/bugtraq/2000/May/54.

[4] Alfarez Abdul-Rahman and Stephen Hailes. 1997. A Distributed Trust Model. In *Proceedings of the 1997 New Security Paradigms Workshop*. ACM, New York, NY, USA, 48–60. https://doi.org/10.1145/283699.283739

[5] Christopher Alberts, Audrey Dorofee, James Stevens, and Carol Woddy. 2003. *Introduction to the OCTAVE® Approach.* Technical Report. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA.

[6] Christopher J. Alberts, Carol Woody, and Audrey Dorofee. 2014. *Introduction to the Security Engineering Risk Analysis (SERA) Framework.* CMU/SEI Report CMU/SEI-2014-TN-025. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA. https://resources.sei.cmu.edu/asset_files/TechnicalNote/2014_004_001_427329.pdf

[7] Steven M. Bellovin. 2006. Permissive Action Links, Nuclear Weapons, and the History of Public Key Cryptography. Invited Talk. https://www.usenix.org/conference/2006-usenix-annual-technical-conference/permissive-action-links-nuclear-weapons-and https://www.usenix.org/conference/2006-usenix-annual-technical-conference/permissive-action-links-nuclear-weapons-and.

[8] Matt Bishop. 1986. *The* adb *Compromise.* Research Memo 86.2. Research Institute for Advanced Computer Science, Moffett Field, CA, USA.

[9] Matt Bishop. 2019. *Computer Security: Art and Science* (second ed.). Addison-Wesley, Boston, MA, USA.

[10] Matt Blaze, Joan Feigenbaum, John Ioannidis, and Angelos D. Keromytis. 1999. *The KeyNote Trust-Management System Version 2.* RFC 2704. https://doi.org/10.17487/RFC2704

[11] Matt Blaze, Joan Feigenbaum, and Angelos D. Keromytis. 1998. KeyNote: Trust Management for Public-Key Infrastructures. In *International Workshop on Security Protocols (Lecture Notes in Computer Science, Vol. 1550)*. Springer Berlin Heidelberg, Berlin, Germany, 59–63. https://doi.org/10.1007/3-540-49135-X_9

[12] Danny Bradbury. 2020. Trove of RubyGems Malware Highlights Software Supply Chain Issues. https://nakedsecurity.sophos.com/2020/04/23/trove-of-rubygems-malware-highlights-software-supply-chain-issues https://nakedsecurity.sophos.com/2020/04/23/trove-of-rubygems-malware-highlights-software-supply-chain-issues.

[13] Cristiano Castelfranchi and Rino Falcone. 2000. Trust Is Much More than Subjective Probability: Mental Components and Sources of Trust. In *Proceedings of the 33rd Hawaii Internaltional Conference on System Sciences*. IEEE Computer Society, Los Alamitos, CA, USA, 1–10. https://doi.org/10.1109/HICSS.2000.926815

[14] Center for Internet Studies. 2019. CIS Controls Version 7.1. https://www.cisecurity.org/controls/ https://www.cisecurity.org/controls/.

[15] CIAC. 1989. *Information about the PC CYBORG (AIDS) Trojan Horse.* CIAC Information Bulletin A-10. Lawrence Livermore National Laboratory, Livermore, CA, USA. http://www.securityfocus.com/advisories/700

[16] Cary L. Cooper (Ed.). 1975. *Theories of Group Processes.* John Wiley & Sons, New York, NY, USA.

[17] Roald Dahl. 2007. *Charlie and the Chocolate Factory.* Puffin Books, New York, NY, USA.

[18] Dag Elgesem. 2006. Normative Structures in Trust Management. In *Proceedings of the 2006 International Conference on Trust Management (Lecture Notes in Computer Science, Vol. 3986)*, Ketil Stølen, William H. Winsborough, Fabio Martinelli, and Fabio Massacci (Eds.). Springer, Berlin, Germany, 48–61. https://doi.org/10.1007/11755593_5

[19] Dan Farmer and Weitse Venema. 1993. Improving the Security of Your Site by Breaking Into It. https://cyberwar.nl/d/1993-FarmerVenema-comp.security.unix-Improving-the-Security-of-Your-Site-by-Breaking-Into-It.pdf https://cyberwar.nl/d/1993-FarmerVenema-comp.security.unix-Improving-the-Security-of-Your-Site-by-Breaking-Into-It.pdf.

[20] Laura Fichtner, Wolter Pieters, and André Teixeira. 2016. Cybersecurity as a Politikum: Implications of Security Discourses for Infrastructures. In *Proceedings of the 2016 New Security Paradigms Workshop* (Granby, Colorado, USA). Association for Computing Machinery, New York, NY, USA, 36–48. https://doi.org/10.1145/3011883.3011887

[21] Diego Gambetta (Ed.). 1988. *Trust: Making and Breaking Cooperative Relations.* Basil Blackwell Ltd., Oxford, UK.

[22] GAO. 2018. *Actions Taken by Equifax and Federal Agencies in Response to the 2017 Breach.* Report to Congressional Requesters GAO-18-559. US Government Accountability Office, Washington, DC, USA. https://www.gao.gov/products/GAO-18-559

[23] Evan Gilman and Doug Barth. 2017. *Zero Trust Networks.* O'Reilly Media, Inc., Sebastopol, CA, USA.

[24] Robert T. Golembiewski and Mark McConkie. 1975. *The Centrality of Interpersonal Trust in Group Processes*, Chapter 7, 131–185. In Cooper [16].

[25] Steven J. Greenwald. 2006. E-Prime for Security: A New Security Paradigm. In *Proceedings of the 2006 Workshop on New Security Paradigms* (Germany). Association for Computing Machinery, New York, NY, USA, 87–95. https://doi.org/10.1145/1278940.1278954

[26] Jingwei Huang and David Nicol. 2009. A Calculus of Trust and Its Appliation to PKI and Identity Management. In *Proceedings of the 8th Symposium n Identity and Trust on the Internet.* ACM, New York, NY, USA, 23–37. https://doi.org/10.1145/1527017.1527021

[27] A. Jøsang and S. M. Knapskog. 1998. A Metric for Trusted Systes. In *Proceedings of the 21st National Information Systems Security Conference*, Vol. 1. National Institute of Standards and Technology, Gaithersburg, MD, USA, 16–29. https://csrc.nist.gov/csrc/media/publications/conference-paper/1998/10/08/proceedings-of-the-21st-nissc-1998/documents/papera2.pdf

[28] Paul A. Karger and Roger R. Schell. 1974. *Multics Security Evaluation: Vulnerability Analysis.* Technical Report ESD-TR-73-193, Vol. II. Electronic Systems Division, Hanscom Air Force Base, Hanscom Air Force Base, MA 01730. https://csrc.nist.gov/csrc/media/publications/conference-paper/1998/10/08/proceedings-of-the-21st-nissc-1998/documents/early-cs-papers/karg74.pdf

[29] John Leyden. 2011. RSA Explains How Attackers Breached Its Systems. https://www.theregister.co.uk/2011/04/04/rsa_hack_howdunnit/ https://www.theregister.co.uk/2011/04/04/rsa_hack_howdunnit/.

[30] Stephen Marsh and Mark R. Dibben. 2003. The Role of Trust in Information Science and Technology. *Annual Review of Information Science and Technology Annual Review of Information Science and Technology Annual Review of Science and Technology* 37, 1 (Jan. 2003), 465–498. https://doi.org/10.1002/aris.1440370111

[31] Stephen Marsh and Mark R. Dibben. 2005. Trust, Untrust, Distrust and Mistrust — An Exploration of the Dark(er) Side. In *Proceedings of the 2005 International Conference on Trust Management (Lecture Notes in Computer Science, Vol. 3477)*, Peter Herrmann, Valérie Issarny, and Simon Shiu (Eds.). Springer, Berlin, Germany, 17–33. https://doi.org/10.1007/11429760_2

[32] Stephen P. Marsh. 1994. *Formalising Trust as a Computational Concept.* Ph.D. Dissertation. Department of Computing Science and Mathematics, University of Stirling, Stirling, Scotland. https://dspace.stir.ac.uk/handle/1893/2010

[33] MITRE. 2020. ATT&CK: https://attack.mitre.org/. https://attack.mitre.org/

[34] Randall Munroe. 2008. Voting Machines: https://xkcd.com/463/. https://xkcd.com/463/

[35] NIST. 2006. *Minimum Security Requirements for Federal Information and Information Systems.* FIPS PUB 200. National Institute of Standards and Technology, Gaithersburg, MD, USA. https://doi.org/10.6028/NIST.FIPS.200

[36] Sean Peisert, Matt Bishop, Laura Corriss, and Steven Greenwald. 2009. *Quis Custodiet ipsos Custodes?* A New Paradigm for Analyzing Security Paradigms with Appreciation to the Roman Poet Juvenal. In *Proceedings of the 2009 New Security Paradigms Workshop (NSPW)*. ACM, New York, NY, USA, 133–144.

[37] Sean Peisert, Ed Talbot, and Matt Bishop. 2012. Turtles All the Way Down: A Clean-slate, Ground-up, First-Principles Approach to Secure Systems. In *Proceedings of the 2012 New Security Paradigms Workshop*. ACM, New York, NY, USA, 15–26. https://doi.org/10.1145/2413296.2413299

[38] Pen Test Partners. 2014. Using nfsshell to Compromise Older Environments. Web page. https://www.pentestpartners.com/security-blog/using-nfsshell-to-compromise-older-environments/ https://www.pentestpartners.com/security-blog/using-nfsshell-to-compromise-older-environments/.

[39] Prabhu Ram and Douglas K. Rand. 1995. Satan: Double-Edged Sword. *IEEE Computer* 28, 6 (June 1995), 82–83. https://doi.org/10.1109/2.392775

[40] Brian Reid. 1987. Viewpoint: Reflections on Some Recent Widespread Computer Break-Ins. *Commun. ACM* 30, 2 (Feb. 1987), 103–105. https://doi.org/10.1145/12527.315716

[41] Scott Rose, Oliver Borchert, Stu Mitchell, and Sean Connelly. 2020. *Zero Trust Architecture.* NIST Special Publication 800-207. National Institute of Standards and Technology, Gaithersburg, MD, USA. https://doi.org/10.6028/NIST.SP.800-207

[42] SANS. 2014. SANS Consensus Policy Resource Community Seerver Security Policy. https://www.sans.org/security-resources/policies/server-security/pdf/server-security-policy https://www.sans.org/security-resources/policies/server-security/pdf/server-security-policy.

[43] Robin Seggelman, Michael Tuexen, and Michael Glenn Williams. 2012. *Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension.* RFC 6520. https://doi.org/10.17487/RFC6520

[44] Tsutomu Shimomura and John Markoff. 1996. *Takedown: The Pursuit and Capture of Kevin Mitnick, America's Most Wanted Computer Outlaw—By the Man Who Did*

It. Hyperion, New York, NY, USA.

[45] Adam Shostack. 2014. *Threat Modeling: Designing for Security.* John Wiley & Sons, New York, NY, USA.

[46] Xiaokui Shu, Ke Tan, Andrew Ciambrone, and Danfeng (Daphne) Yao. 2017. Breaking the Target: An Analysis of Target Data Breach and Lessons Learned. *Computing Research Repository* arXiv:1701.04940 [cs.CR] (Jan. 2017), 1–10. http://arxiv.org/abs/1701.04940

[47] Abe Singer. 2005. Tempting Fate. *;login:* 30, 1 (Feb. 2005), 27–30. https://www.usenix.org/system/files/login/articles/1047-fate.pdf

[48] Abe Singer. 2019. Personal Communication.

[49] Abe Singer, Warren Anderson, and Rik Farrow. 2013. Rethinking Password Policies. *;login:* 38, 4 (Aug. 2013), 14–18. https://www.usenix.org/publications/login/august-2013-volume-38-number-4/rethinking-password-policies

[50] Ken Thompson. 1984. Reflections on Trusting Trust. *Commun. ACM* 27, 8 (Aug. 1984), 761–763. https://doi.org/10.1145/358198.358210

[51] Gianluca Tonti, Jeffrey M. Bradshaw, Renia Jeffers, Rebecca Montanari, Niranjan Suri, and Andrzej Uszok. 2003. Semantic Web Languages for Policy Representation and Reasoning: A Comparison of KAoS, Rei, and Ponder. In *International Semantic Web Conference (Lecture Notes in Computer Science, Vol. 2870).* Springer, Berlin, Germany, 419–437. https://doi.org/10.1007/978-3-540-39718-2_27

[52] Scot Tucker. 2018. Engineering Trust: A Graph-Based Algorithm for Modeling, Validating, and Evaluating Trust. In *Proceedings of the 17th IEEE International Conference on Trust, Security and Privacy in Computing and Communications.* IEEE Computer Society, Los Alamitos, CA, USA, 1–9. https://doi.org/10.1109/TrustCom/BigDataSE.2018.00011

[53] Leendert van Doorn. 1998. NFSShell. https://www.cs.vu.nl/pub/leendert/nfsshell.tar.gz https://www.cs.vu.nl/pub/leendert/nfsshell.tar.gz.

[54] Jackson Wynn, Joseph Whitmore, Geoff Upton, Lindsay Spriggs, Dan McKinnon, Richard McInnes, Richard Graubart, and Lauren Clausen. 2011. *Threat Assessment & Remediation Analysis.* MITRE Technical Report 11-4982. The MITRE Corporation, Bedford, MA, USA. https://www.mitre.org/sites/default/files/pdf/11_4982.pdf

[55] Li Xiong and Ling Liu. 2004. PeerTrust: Supporting Reputation-Based Trust for Peer-to-Peer Electronic Communities. *IEEE Transactions on Knowedge and Data Engineering* 16, 7 (July 2004), 843–857. https://doi.org/10.1109/TKDE.2004.1318566