Software Engineering Institute

Principles of Trust for Embedded Systems

David A. Fisher

March 2012

TECHNICAL NOTE
CMU/SEI-2012-TN-007

CERT® Program

http://www.sei.cmu.edu

Carnegie Mellon

This material is based upon work funded and supported by the United States Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Department of Defense.

This report was prepared for the

Contracting Officer
ESC/CAA
20 Shilling Circle
Building 1305, 3rd Floor
Hanscom AFB, MA 01731-2125

NO WARRANTY

THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN “AS-IS” BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

* These restrictions do not apply to U.S. government entities.

Table of Contents

Abstract

The development of trusted systems is a long-standing, elusive, and ill-defined objective in many domains. This paper gives substance and explicit meaning to the terms *trust* and *trustworthy* as they relate to automated systems and to embedded systems in particular. Principles of trust are identified. Some of their implications for software engineering practice and for the design of hardware-based trusted computing platforms are also discussed.

1 Background

Trusted computing is a common goal in many automated domains. Those involved in current efforts to achieve trusted computing typically believe that, regardless of what is meant by *trusted*, their efforts will require a secure infrastructure and development of correct application codes that extend security to enable trusted applications. A *trusted computing platform* is a computing infrastructure that provides a variety of hardware-supported security functions. Although trust, trusted, and trustworthiness are never defined, it is hoped that trusted computing platforms and resulting improvements in the security in computing infrastructure and applications will enable trustworthy applications and systems.

Hardware-based trusted computing platforms include security chips, such as the Trusted Platform Module (TPM). The TPM is an inexpensive passive hardware chip that can perform hashing, generate random numbers and asymmetric cryptographic keys, store keys with confidentiality and integrity, confirm its own identity, and perform asymmetric signing, encryption, and decryption on small blocks of storage [TCG 2007]. TPMs provide these primitive security functions without the vulnerabilities inherent in purely software solutions. In theory, they can be extended to almost any well-understood security function, except perhaps availability, through a multilevel chain-of-trust bootstrapping process [Arbaugh 1997].

Effective use of trusted computing platforms requires provably correct code at every level of implementation, from bios startup and boot load to the end applications. It requires strict and careful maintenance of chains of trust. It is a brittle process in which a mistake at any level voids the results at all subsequent levels. Success is impractical in all but the simplest systems. In certain situations, dynamic roots of trust [McCune 2008] can be used to reduce the number of levels. Even then, security is assured only if

- the application does not depend on external components
- one assumes that implementation bugs do not create exploitable vulnerabilities
- neither hardware nor software supply chains have been compromised
- the system cannot be compromised by physical attacks

Trusted computing has become a recognized domain of computing with several annual conferences and workshops [TCI 2010, Trust 2010]. Hardware-based trusted computing platforms have been widely deployed, if not fully utilized. The focus, however, remains on security, specifically infrastructure security that leaves issues of trust to software engineering at the application level. Software engineering, unfortunately, has either ignored issues of trust or interpreted them simply as fulfilling application requirements rather than a driver in determining appropriate requirements.

The work in this report follows from previous examinations of the capabilities and limitations of trusted computing platforms [Fisher 2011] and of business models and implications for trusted platforms.[1] In this paper I attempt to give more substance to *trusted*, *trust*, and *trustworthy* in the context of automated systems. I identify principles of trust and discuss the implications of the principles for both trusted computing platforms and software engineering.

Although some of the findings may also apply to other domains, this report focuses on embedded systems applications. An *embedded system* is any automated system that involves the use of sensors, control devices, and network connections; provides dedicated special-purpose capabilities that are complex and adaptive; and has power, weight, or other constraints that limit CPU performance, memory capacity, and communications bandwidth. Because embedded systems must interact with the physical world in real time, time is an omnipresent aspect of embedded systems and must be a primary consideration in user expectations and development decisions [Duranton 2006].

Embedded systems not only include mission requirements that are noncomputational in character, but these requirements typically operate under safety-critical conditions in adversarial environments. Embedded systems may have physical and cyber security requirements and must perform within real-time constraints but in some ways are simpler than general-purpose systems because they involve thousands rather than millions of lines of code. Their complexity and adaptability require software-intensive solutions running on general-purpose processors, while constraints on their computational resources may preclude high-performance processors and the use of general-purpose operating systems. Examples include avionics systems, all but the simplest automated medical devices, embedded automotive systems, and complex supervisory control and data acquisition (SCADA) devices. Unmanned aerial vehicles (UAVs) and other autonomous and remotely controlled vehicles provide especially good examples and serve as the focus of this work [Stansbury 2008, DoD 2010].

A *trustworthy system* is one that fulfills the expectations of its owners and users [Grawrock 2009]. Thus, our research in trusted embedded systems is focused on the design and implementation of embedded systems that can be trusted to fulfill the expectations of their owners and users. It seeks methods that clarify expectations under normal and adverse circumstances, provide dynamic evidence of success or failure, and implement appropriate runtime failure semantics. It recognizes the differences between trust and security and that secure infrastructure is critical to trusted systems whether or not the trusted systems have application-level security requirements.

[1] Andrews, A. D.; Fisher, D. A.; & McCune, J. M. *Gaining Traction for Trusted Computing*. Software Engineering Institute, Carnegie Mellon University, (forthcoming).

2 Introduction

Trustworthiness is the degree to which a system satisfies its owners' and users' expectations. Thus, trustworthiness is system- and mission-specific. Cost-effective trustworthy systems require application- or mission-specific solutions built on special-purpose platforms that recognize the limitations and constraints of differing domains. This means that one-size-fits-all solutions are not optimal and are unlikely to be successful. It also means that trustworthy solutions are impossible without realistic expectations.

Embedded systems operate in the presence of inherent uncertainty, context dependencies, and adversarial certainty. Physical damage, weather conditions, and component maintenance impose uncertainty that is inherent in embedded systems but seldom seen in general-purpose computing systems. In the physical world of embedded systems, all operational environments are adversarial, whether by accident or intent, whether by users, developers, or others, and whether by intelligent or natural adversaries or some combination of these. Systems, or at least their developers, may have to distinguish among a variety of influences including expectations, observations, assertions, assumptions, constraints, and optimization criteria to determine the appropriate actions of an embedded system in response to expected and unanticipated situations.

Trust is the confidence or reliance one has in the integrity, strength, ability, surety, etc. of a product, service, person, or other thing. The degree of trust can vary from one part or aspect of a system to another. Trust can be a function of time and circumstances. For example, my cell phone provides good service except at times when too many others are using the service or I am in a dead zone. Trust depends not only on circumstances but also on our expectations. A computer that is too slow for some may be more than adequate for others who have lesser expectations.

A *trusted system* or *application* is one in which our confidence or reliance is justified. That is, a trusted system is one that we believe will satisfy our expectations and in which that belief is backed by adequate evidence. It is all too easy to make invalid trust decisions in the presence of contrary evidence. Even though my bus arrives 20 minutes late on most days, I continue to treat that evidence as an anomaly and believe it will be on schedule today. We want to believe that the systems and services that we use are trustworthy.

There is an often confusing relationship between security and trust. Although security deals with issues of confidentiality, integrity, and availability of information and services, while trust addresses the confidence one has that a product or service will perform as expected, the terms *security* and *trust* are sometimes equated. The trustworthiness of a product or service can be undermined from below by an adversary exploiting security vulnerabilities in the underlying infrastructure on which it depends. The trustworthiness of an otherwise trustworthy banker might be undermined by taking his family hostage. The trustworthiness of an otherwise trustworthy application might be compromised by exploiting security vulnerabilities in the operating system on which it runs.

In an automated system, an application may or may not have application-level security requirements. From a trust perspective, security is like any other application-level quality attribute. The location of a UAV, for example, is not sensitive when flying in civilian air space but may be classified when in a war zone. Automated systems, however, typically depend on certain infrastructure and contextual security assumptions including those that arise from computer system hardware, operating systems, and support libraries. If the integrity of this infrastructure is compromised, the trustworthiness of the application or mission also may be ruined.

The TPM and other hardware-based trusted computing platforms are intended to provide a secure infrastructure on which to run applications and to enable the development of trusted applications. Trusted computing platforms address issues of confidentiality, code integrity, and identity through secure storage of encryption keys and measured code. They do not address issues of trust beyond security. They enable application-level security, but only in the presence of correct security-aware software.

Developers should consider issues additional to security when developing trustworthy applications. Developers need to identify and prioritize the critical functional and quality needs of the system or application. They need to implement software engineering methods and practices that exploit those priorities to maximize the likelihood of the system or application’s mission being fulfilled. Developers also need to assess whether those needs and expectations are being met, determine and collect what constitutes adequate evidence for that assessment, and identify the intended failure semantics when normal expectations cannot be satisfied. *Failure semantics* are the meaning or intent of the actions taken when failures occur [Hoare 1983].

3 Principles of Trust

Certain principles of trust follow from the previous considerations. Principles of trust provide a context and framework for reasoning about issues of trust and trustworthiness. They can guide decisions about how trustworthiness can be measured, managed, and mitigated. They can help in selecting and prioritizing needs, determining what evidence is adequate, and deciding what circumstances constitute failures and what failure semantics are appropriate.

P1. **Trust is essential**. Our inability to observe and correctly interpret everything that might affect us guarantees that some essential influences or aspects will be unknown in any given situation. Furthermore, we live in a dynamic world in which what we know today may not be true tomorrow. No amount of care, concern, or validation can guarantee trustworthiness. Useful operation of anything depends on trust in its design, implementation, and use. Without some degree of trust, nothing can be accomplished and nothing is useful.

P2. **Trust must be evidence-based and never absolute**. Justified confidence is impossible without evidence. Evidence can take many forms but ultimately depends on observations of past performance or quality of outcomes for designers, implementers, and users of the products or services we wish to trust. Testing, validation, and certification are processes that can provide evidence of trustworthiness, as can test use in circumstances similar to those of expected operational use. Evidence of functionality or quality should be given greater credibility than claims by the provider, but rejection of provider claims should not necessarily preclude use of the product. Trust is a measure of confidence relative to the user's, rather than the provider's, expectations. No amount of evidence, however, can guarantee absolute trustworthiness in any given situation. Nothing should be trusted completely, not even ourselves. Any system can fail due to errors in design, implementation, or use, or because of physical damage, incorrect data, misunderstanding, or omissions.

0% .)_______________________(. 100%

degree of trust

Figure 1: The Open Interval of Trust

P3. **Trust should be partitioned by function and context**. Trust can vary within a system depending on the specific function considered or on the time, context, and dynamically changing circumstances of its use. It is unnecessary to completely trust anything. Only the specific functionality, quality, and services needed for the current mission must be trusted. I don't have to trust the doctor's surgical skill to have her as a golf partner. That John gives honest answers to Mary does not necessary mean he will be so forthcoming with others. That the experiment worked in the laboratory does not mean that it will work in the field. That students are punctual on Wednesdays may say little about their tardiness on Fridays.

Evidence of trust is only needed for those functions, qualities, circumstances, and conditions that are expected for the current mission. Confidence of trustworthiness is needed only for that functionality and those qualities of a product or service that are essential to the success of a system or mission. Failure to fulfill other claimed functionality or services may be irrelevant.

P4. **Trust must be dynamically confirmed**. Because we live in a dynamic world, the context and conditions of an actual mission are guaranteed to differ from those in which the evidence of trustworthiness was generated. Only dynamic confirmation of expectations can ensure that our trust was justified. Dynamic confirmation also provides an opportunity to take mitigating actions when our expectations are not satisfied. Dynamic evidence through feedback among the constituents of a system is essential where the constituents include all active elements of the system, whether they be human, computational, or mechanical. Otherwise the system is blind to its own state and context and has no way to determine whether failures have occurred and failure semantics should be employed.

P5. **Trust should be proportional to the amount, quality, and relevance of evidence**. The more independent evidence that supports our expectations, the more confidence we can have, especially in the absence of conflicting evidence. The degree of trust should, however, be moderated by the quality and trustworthiness of the evidence. It is often very easy to generate large quantities of evidence of trustworthiness for functionality, quality of service, and circumstances that have little relevance to the current mission, quality needs, and context. Such evidence must be identified and should have minimal influence on our degree of trust. Evidence of intentionally false claims in any area, however, may correlate with untrustworthiness in other areas.

P6. **Trust should be inversely proportional to impact, complexity, and interdependencies of the evidence**. There is little risk in trusting something whose failure would make little difference. For example, I seldom expend energy determining whether the money I receive in change is counterfeit. As the complexity of a product or service increases, so does the likelihood that it will be untrustworthy and the greater the amount of evidence that is needed to generate trust. The complexity of a product or service is sometimes invisible to its users but can be indicated by multiple services in the same product, large numbers of options, incomplete functional or quality specifications, large numbers of visible state variables, or backdoor interfaces to other services. From the user perspective, lack of interdependencies among products, services, or functions, coupled with obvious and understandable interfaces, makes those products, services, and functions seem simple and reduces the amount and kinds of evidence required to provide confidence of their trustworthiness.

P7. **Trust does not inherently require security mechanisms**. Confidentiality is seldom an application-level requirement in embedded systems. Some degree of availability and integrity of information and services are always necessary for trust, but security mechanisms provide only limited support for availability and integrity. Secure contexts and platforms can boost confidence in security but are neither necessary nor sufficient for the availability and integrity of trusted systems. Methods beyond those traditionally used in information security are needed to support expectations for availability and integrity, to provide adequate evidence of their satisfaction, and to enable appropriate responses to failures.

P8. **Trustworthiness is an emergent property that can be composed from untrustworthy components**. Emergent properties arise from the actions and interaction among the constituents of a system and are independent of many of the constituents' details [Fisher 2006]. Confidence in aspects of trust such as availability and integrity can be enhanced through trust mechanisms in which the consumers confirm availability of information or services to provider constituents and providers validate integrity through feedback from consumers. These aspects of trusted systems can be produced from untrustworthy constituents. Trust is thus more analogous to reliability, where reliable systems can be built from unreliable components, than to performance, where high-performance systems require high-performance components. It also means that these aspects of security may be achievable without secure infrastructure.

P9. **Over-specification can weaken or destroy trustworthiness**. The more constraints and requirements imposed on a system, the fewer the number of feasible solutions. If any of those constraints or requirements are not essential to the system's purpose or mission, they likely will reduce cost effectiveness, eliminate the most trustworthy solutions, and may preclude all adequate solutions. Trustworthy systems demand that needs and requirements be limited to those that are essential. Furthermore, emergent effects, in which system-wide properties that are not present in the system's constituents arise from the interactions among the constituents, are possible only in systems where there is significant interaction among the constituents but the constraints remain loosely coupled.

Readers are challenged to test the principles set forth here, to validate or refute them from their knowledge and experience, and to propose modifications where appropriate. Assuming that the principles are sound, the following two sections give a glimpse of some of the implications for software engineering and hardware-based trusted platform infrastructure.

4 Implications for Software Engineering

Trust is about satisfying owner and user expectations. From a software engineering perspective, there are two sides to this equation: having realistic expectations and ensuring that expectations are actually met. Given that little can be guaranteed, the appropriate level of expectations is that likely to be achieved within the available resources. Trust is not only about functionality and quality but also about managing expectations.

Expectations can be satisfied only if they are known, documented, and realistic. They can be realistic only if we know what is feasible and affordable. Owners' and users' expectations for a system must be captured and agreed upon. Equally important is agreement on what evidence is required to demonstrate success or failure in meeting those expectations. Lack of evidence of failure does not necessarily imply success. In general, there are three cases to consider depending on whether the expectations are met, not met, or undetermined. The intended responses in each case represent expectations for the system.

Requirements for trusted systems should be composed from only four elements: (1) realistic expectations for normal operations, (2) the evidence required to confirm failure or success, (3) the failure semantics when expectations cannot be satisfied, and (4) the practical constraints of the system implementation. The fourth element might include the cost, performance, and capacity of available resources, as well as real-time constraints. The need to document evidentiary requirements and intended failure semantics may be new.

The value in omitting nonessential desires and specific implementation mechanisms from requirements has long been argued but has had little impact on practice. For trusted systems, it is critical and may explain why so many systems fail to satisfy their owners' and users' expectations. Removing extraneous requirements can improve cost performance, but runtime collection of evidence to confirm trustworthiness may reduce cost performance. A formal specification language for trusted systems could encourage appropriate requirements specification and provide the necessary input for analysis and synthesis tools.

The need for simplicity in trusted systems imposes a need for application architectures that divide the functionality into independent constituents at each level of implementation. These independent constituents can then easily be composed to provide the functionality and quality required at the next level. Each constituent should have restricted access to the state of other constituents on a need-to-know basis. Ideally, each constituent would provide a single functionality with access to other constituents only through parameter-passing mechanisms.

Mitigation strategies in case of failure to meet expectations should seldom be reduced forms of normal requirements. While fail-soft and fail-safe mechanisms may be adequate for some general-purpose systems, they are seldom appropriate for embedded and real-time applications. In safety-critical systems, "do no harm to the user or others" may be the overriding concern. In unmanned military vehicles, self destruction may be an appropriate mitigation in some situations. In automated medical control systems, mitigations might focus on keeping patient parameters within safe ranges and sounding an alarm for human intervention. Conflicting evidence as to the state of a system or its context must be addressed in safe and effective ways. Most important is that the mitigating actions in case of failure reflect expectations of owners and users for the specifics of the situation in light of the available evidence. Mitigating strategies, like other needs and requirements, must be mission- and situation-specific.

Trust requires practical rather than formal correctness. Trustworthy systems must be able to react creditably to unexpected and unanticipated situations, operate in the presence of uncertainty, and deal with failures in provably correct code. Formal correctness can mitigate certain forms of logic and implementation errors but cannot address errors, such as externally imposed state changes, that are not observable within the code. Neither should optimization necessarily remove runtime checks that are redundant or unnecessary from a model checking or proof-of-correctness perspective.

5 Implications for Platform Infrastructure

Hardware-based trusted computing platforms provide a level of secure infrastructure that cannot be achieved in software implementations alone. They also can reduce the concern for otherwise trustworthy applications being undermined through their platform infrastructure.

Existing trusted platforms such as the Trusted Platform Module (TPM), which provide simple security isolation functions that cannot be compromised by software, may be needed to satisfy confidentiality, integrity, and identity confirmation requirements in trusted applications. These platforms are generally affordable in terms of weight and power requirements in embedded systems. Their practical value may be greater in embedded systems and especially in systems built from a trusted systems perspective because application simplicity and reduced distance between platform and application reduce the practical problems of producing correct chains of trust. The use of separate chips for security, however, imposes vulnerabilities that would be absent if the hardware security functions were on the same chip as the central processing unit (CPU). In commodity applications, they also can add prohibitively to system costs. Use of trusted computing platforms and improvements in CPU architectures could facilitate trusted computing solutions.

CPU architectures could eliminate some of the current security vulnerabilities inherent in direct memory access (DMA) devices that have

- unrestricted access to memory
- privileged hardware modes that are exempt from normal isolation boundaries
- access to uninitialized memory that may contain information without a need to know
- information leaks through processor registers during task switches
- buffer overflows that are enabled by the absence of affordable bounds checking

More important from a trust perspective, hardware and software infrastructure could provide mechanisms for runtime support of trustworthy applications. Parallel hardware could allow evidence checks to be performed concurrently with normal operation to maximize performance [Bratus 2008]. Exception processing could be initiated without entering privilege mode. And, isolation of constituents could be provided through always-present, functionally-based, fine-grained access mechanisms [Organick 1972, Levin 1975]. Together these approaches may be able to eliminate privileged mode entirely in favor of fine-grained access mechanisms everywhere. The referenced papers provide a baseline of ideas for development of more effective trusted systems technology.

6 Summary and Conclusions

A trusted system is one that, in operational use, satisfies the expectations of its owners and users. This report includes principles that can guide development decisions for trusted systems. Trusted systems require simple, understandable designs and implementations without extraneous requirements. They require evidence-based runtime decisions that consider not only normal execution but intended failure semantics for abnormal situations.

The focus in this report is on embedded applications that

- must operate with computer hardware that is more limited in speed, capacity, and bandwidth than general-purpose systems
- involve sensors, control devices, and communications
- have application-specific requirements that often include safety, real-time performance, and weight and power constraints
- operate in adversarial environments
- unlike general-purpose systems, do not have to deal with multiple applications of unknown character

Fulfilling the vision of trusted embedded systems will require software engineering practices that focus on trust and trustworthiness, not only in implementation, but in requirements generation and software architecture. Embedded systems offer an ideal context for trustworthy systems because they lack the inherent complexity of general-purpose systems and have greater access to sensor and control devices that can provide needed feedback as evidence of success and failure.

Certain modifications and enhancements to CPUs and operating systems of trusted computing platforms could also reduce the cost, improve the performance, and reduce the security risks in trusted systems. These include direct support for common trust mechanisms and elimination of certain avoidable security vulnerabilities.

References

URLs are valid as of the publication date of this document.

[Arbaugh 1997]

Arbaugh, W. A.; Farber, D. J.; & Smith, J. M. “A Reliable Bootstrap Architecture.” *Proceedings of IEEE Symposium on Security and Privacy*, 1997.

[Bratus 2008]

Bratus, Sergey; Locasto, Michael E.; Ramaswamy, Ashwin; & Smith Sean W. *New Directions for Hardware-assisted Trusted Computing Policies (Position Paper)*. Dartmouth College, 2008. http://www.cs.dartmouth.edu/~sws/pubs/berlin.pdf

[DoD 2010]

Under Secretary for Defense (Acquisition, Technology and Logistics). *Department of Defense Report to Congress on Addressing Challenges for Unmanned Aircraft Systems*. September 2010. http://www.acq.osd.mil/sts/docs/2010-uas-annual-report.pdf

[Duranton 2006]

Duranton, Marc. “The Challenges for High Performance Embedded Systems.” *IEEE Proceedings of the 9th EUROMICRO Conference on Digital System Design (DSD’06)*. 2006.

[Fisher 2006]

Fisher, David A. *An Emergent Perspective on Interoperation in Systems of Systems* (CMU/SEI-2006-TR-003). Software Engineering Institute, Carnegie Mellon University, 2006. http://www.sei.cmu.edu/library/abstracts/reports/06tr003.cfm

[Fisher 2011]

Fisher, David A.; McCune, Jonathan M.; & Andrews, Archie D. *Trust and Trusted Computing Platforms* (CMU/SEI-2011-TN-005). Software Engineering Institute, Carnegie Mellon University, 2011. http://www.sei.cmu.edu/library/abstracts/reports/11tn005.cfm

[Grawrock 2009]

Grawrock, David. *Dynamics of a Trusted Platform*. Intel Press, 2009.

[Levin 1975]

Levin, R.; Cohen, E.; Corwin, W. P. F.; & Wulf, W. “Policy/Mechanism Separation in Hydra,” 132-140. *Proceedings of the Fifth Symposium on Operating Systems Principle*s. ACM/SIGOPS, University of Texas at Austin, 1975.

[McCune 2008]

McCune, J. M., et. al. “Flicker: An Execution Infrastructure for TCB Minimization,” 315-328. *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008*. Glasgow, Scotland UK, 2008.

[Hoare 1983]

Hoare, C. A. R. and Olderog, E. R. “Specification-oriented Semantics for Communicating Processes.” *Automata, Languages, and Programming; 10th Colloquium, vol. 154 of Lecture Notes in Computer Science*. (Springer, 1983): 561-572.

[Organick 1972]

Organick, E. I. *The Multics System: An Examination of its Structure*. MIT Press, 1972.

[Stansbury 2008]

Stansbury, Richard S.; Vyas, Manan A.; & Wilson, Timothy A. *A Survey of UAS Technologies for Command, Control, and Communication (C3)* (technical report). Center of Excellence for General Aviation Research, June 2008.
http://www.cgar.org/Data/Projects/134/Files/A%20Survey%20of,%20Stansbury%20et%20al.pdf

[Trust 2010]

Trust 2010 Organizing and Steering Committees. *3rd International Conference on Trust and Trustworthy Computing*. Berlin, Germany, June 2010. http://www.trust2010.org

[TCI 2010]

Trusted Computing Institute (TCI). *The Sixth IEEE/IFIP International Symposium on Trusted Computing and Communications (TrustCom10)*. Kowloon, Hong Kong, December , 2010.

[TCG 2007]

Trusted Computing Group, Incorporated (TCG). *TPM Main: Part I Design Principles Specification Version 1.2, Level 2, Revision 103*. TCG, 2007.
http://www.trustedcomputinggroup.org/files/resource_files/646BE624-1D09-3519-ADDA61BE37A21A74/mainP1DPrev103.pdf.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. **AGENCY USE ONLY** (Leave Blank)	2. **REPORT DATE** March 2012	3. **REPORT TYPE AND DATES COVERED** Final

4. **TITLE AND SUBTITLE**
Principles of Trust for Embedded Systems

5. **FUNDING NUMBERS**
FA8721-05-C-0003

6. **AUTHOR(S)**
David A. Fisher

7. **PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

8. **PERFORMING ORGANIZATION REPORT NUMBER**
CMU/SEI-2012-TN-007

9. **SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)**
HQ ESC/XPK
5 Eglin Street
Hanscom AFB, MA 01731-2116

10. **SPONSORING/MONITORING AGENCY REPORT NUMBER**
n/a

11. **SUPPLEMENTARY NOTES**

12A **DISTRIBUTION/AVAILABILITY STATEMENT**
Unclassified/Unlimited, DTIC, NTIS

12B **DISTRIBUTION CODE**

13. **ABSTRACT (MAXIMUM 200 WORDS)**

The development of trusted systems is a long-standing, elusive, and ill-defined objective in many domains. This paper gives substance and explicit meaning to the terms *trust* and *trustworthy* as they relate to automated systems and to embedded systems in particular. Principles of trust are identified. Some of their implications for software engineering practice and for the design of hardware-based trusted computing platforms are also discussed.

14. **SUBJECT TERMS**
Trust, trustworthy systems, trusted platforms, embedded systems, security, failure semantics

15. **NUMBER OF PAGES**
20

16. **PRICE CODE**

17. **SECURITY CLASSIFICATION OF REPORT** Unclassified	18. **SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	19. **SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	20. **LIMITATION OF ABSTRACT** UL