Trustworthy Computing

Microsoft White Paper

Craig Mundie – Senior Vice President and CTO, Advanced Strategies and Policy
Pierre de Vries
Peter Haynes
Matt Corwine
Microsoft Corporation

October 2002

The following is a revised version of the paper on Trustworthy Computing we published in January 2002. It represents our synthesis of the vast amount of valuable input we have received on the subject since the original paper saw the light of day. To everyone who offered their thoughts and help: many thanks.

Why Trust?

While many technologies that make use of computing have proven themselves extremely reliable and trustworthy—computers helped transport people to the moon and back, they control critical aircraft systems for millions of flights every year, and they move trillions of dollars around the globe daily—they generally haven't reached the point where people are willing to entrust them with their lives, implicitly or explicitly. Many people are reluctant to entrust today's computer systems with their personal information, such as financial and medical records, because they are increasingly concerned about the security and reliability of these systems, which they view as posing significant societal risk. If computing is to become truly ubiquitous—and fulfill the immense promise of technology—we will have to make the computing ecosystem sufficiently *trustworthy* that people don't worry about its fallibility or unreliability the way they do today.

Trust is a broad concept, and making something trustworthy requires a social infrastructure as well as solid engineering. All systems fail from time to time; the legal and commercial practices within which they're embedded can compensate for the fact that no technology will ever be perfect.

Hence this is not only a struggle to make software trustworthy; because computers have to some extent already lost people's trust, we will have to overcome a legacy of machines that fail, software that fails, and systems that fail. We will have to persuade people that the systems, the software, the services, the people, and the companies have all, collectively, achieved a new level of availability, dependability, and confidentiality. We will have to overcome the distrust that people now feel for computers.

The *Trustworthy Computing Initiative* is a label for a whole range of advances that have to be made for people to be as comfortable using devices powered by computers and software as they are today using a device that is powered by electricity. It may take us ten to fifteen years to get there, both as an industry and as a society. This is a "sea change" not only in the way we write and deliver software, but also in the way our society views computing generally. There are immediate problems to be solved, and fundamental open research questions. There

are actions that individuals and companies can and should take, but there are also problems that can only be solved collectively by consortia, research communities, nations, and the world as a whole.

Setting the Stage

History

Society has gone through a number of large technology shifts that have shaped the culture: the agrarian revolution, the invention of metalworking, the industrial revolution, the advent of electricity, telephony and television—and, of course, the microprocessor that made personal computing a reality. Each of these fundamentally transformed the way billions of people live, work, communicate, and are entertained.

Personal computing has so far only really been deployed against white-collar work problems in the developed world. (Larger computer systems have also revolutionized manufacturing processes.) However, the steady improvement in technology and lowering of costs means that personal computing technology will ultimately become a building block of everybody's home and working lives, not just those of white-collar professionals.

Progress in computing in the last quarter century is akin to the first few decades of electric power. Electricity was first adopted in the 1880s by small, labor-intensive businesses that could leverage the technology's fractional nature to increase manufacturing productivity (that is, a single power supply was able to power a variety of electric motors throughout a plant). In its infancy, electricity in the home was a costly luxury, used by high-income households largely for powering electric lights. There was also a good deal of uncertainty about the safety of electricity in general and appliances in particular. Electricity was associated with lightning, a lethal natural force, and there were no guarantees that sub-standard appliances wouldn't kill their owners.

Between 1900 and 1920 all that changed. Residents of cities and the fast-growing suburbs had increasing access to a range of energy technologies, and competition from gas and oil pushed down electricity prices. A growing number of electric-powered, labor-saving devices, such as vacuum cleaners and refrigerators, meant that households were increasingly dependent on electricity. Marketing campaigns by electricity companies and the emergence of standards marks (for example, Underwriters' Laboratories (UL) in the United States) allayed consumer fears. The technology was not wholly safe or reliable, but at some point in the first few years of the 20th century, it became safe and reliable enough.

In the computing space, we're not yet at that stage; we're still in the equivalent of electricity's 19th century industrial era. Computing has yet to touch and improve every facet of our lives—but it will. It is hard to predict in detail the eventual impact that computing will have, just as it was hard to anticipate the consequences of electricity, water, gas, telecommunications, air travel, or any other innovation. A key step in getting computing to the point where people would be as happy to have a microprocessor in every device as they are relying on electricity will be achieving the same degree of relative trustworthiness. "Relative," because 100% trustworthiness will never be achieved by any technology—electric power supplies surge and fail, water and gas pipes rupture, telephone lines drop, aircraft crash, and so on.

Trustworthy Technologies in General

All broadly adopted technologies—like electricity, automobiles or phones—have become trusted parts of our daily lives because they are almost always there when we need them, do what we need them to do, and work as advertised.

Almost anyone in the developed world can go buy a new telephone handset and plug it into the phone jack without worrying about whether it'll work or not. We simply assume that we'll get a dial tone when we pick up a phone, and that we'll be able to hear the other party when we connect. We assume that neither our neighbor nor the insurance

broker down the road will be able to overhear our conversation, or obtain a record of who we've been calling. And we generally assume that the phone company will provide and charge for their service as promised. A combination of engineering, business practice, and regulation has resulted in people taking phone service for granted.

One can abstract three broad classes of expectations that users have of any trustworthy technology: safety, reliability, and business integrity (that is, the integrity of the organization offering the technology). These categories, and their implications for computing, are discussed in more detail below.

Trustworthy Computing

Computing devices and information services will only be truly pervasive when they are so dependable that we can just forget about them. In other words, at a time where computers are starting to find their way into just about every aspect of our life, we need to be able to trust them. Yet the way we build computers, and the way that we now build services around those computers, hasn't really changed that much in the last 30 or 40 years. It will need to.

A Framework for Trustworthy Computing

We failed to find an existing taxonomy that could provide a framework for discussing Trustworthy Computing. There is no shortage of trust initiatives, but the focus of each is narrow. For example, there are treatments of trust in e-commerce transactions and trust between authentication systems, and analyses of public perceptions of computing, but a truly effective approach needs to integrate engineering, policy, and user attitudes. Even just on the engineering side, our scope is broader than, say, the SysTrust/SAS70 models, which deal purely with large online systems.

First, there are the machines themselves. They need to be reliable enough that we can embed them in all kinds of devices—in other words, they shouldn't fail more frequently than other similarly important technologies in our lives. Then there's the software that operates those machines: do people trust it to be equally reliable? And finally there are the service components, which are also largely software-dependent. This is a particularly complicated problem, because today we have to build dependability into an end-to-end, richly interconnected (and sometimes federated) system.

Since trust is a complex concept, it is helpful to analyze the objective of Trustworthy Computing from a number of different perspectives. We define three dimensions with which to describe different perspectives on trust: Goals, Means, and Execution.

Goals

The *Goals* consider trust from the user's point of view. The key questions are: Is the technology there when I need it? Does it keep my confidential information safe? Does it do what it's supposed to do? And do the people who own and operate the business that provides it always do the right thing? These are the goals that any Trustworthy Computing has to meet:

Goals	**The basis for a customer's decision to trust a system**
Security	The customer can expect that systems are resilient to attack, and that the confidentiality, integrity, and availability of the system and its data are protected.
Privacy	The customer is able to control data about themselves, and those using such data adhere to fair information principles

Reliability	The customer can depend on the product to fulfill its functions when required to do so.
Business Integrity	The vendor of a product behaves in a responsive and responsible manner.

The trust Goals cover both rational expectations of performance—that is, those that are amenable to engineering and technology solutions—and more subjective assessments of behavior that are the result of reputation, prejudice, word of mouth, and personal experience. All of these goals raise issues relating to engineering, business practices, and public perceptions, although not all to the same degree. In order to clarify terms, here are examples for the *Goals*:

- Security: A virus doesn't infect and crash my PC. An intruder cannot render my system unusable or make unauthorized alterations to my data.
- Privacy: My personal information isn't disclosed in unauthorized ways. When I provide personal information to others, I am clearly informed of what will—and won't—be done with it, and I can be sure they will do what they promise.
- Reliability: When I install new software, I don't have to worry about whether it will work properly with my existing applications. I can read my email whenever I want by clicking the Hotmail link on msn.com. I never get "system unavailable" messages. The Calendar doesn't suddenly lose all my appointments.
- Business Integrity: My service provider responds rapidly and effectively when I report a problem.

Means

Once the Goals are in place, we can look at the problem from the industry's point of view. *Means* are the business and engineering considerations that are employed to meet the Goals; they are the nuts and bolts of a trustworthy service. Whereas the Goals are largely oriented towards the end-user, the Means are inwardly facing, intra-company considerations. Think of the Goals as *what* is delivered, and the Means as *how*.

Means	**The business and engineering considerations that enable a system supplier to deliver on the Goals**
Secure by Design, Secure by Default, Secure in Deployment	Steps have been taken to protect the confidentiality, integrity, and availability of data and systems at every phase of the software development process—from design, to delivery, to maintenance.
Fair Information Principles	End-user data is never collected and shared with people or organizations without the consent of the individual. Privacy is respected when information is collected, stored, and used consistent with Fair Information Practices.
Availability	The system is present and ready for use as required.
Manageability	The system is easy to install and manage, relative to its size and complexity. (Scalability, efficiency and cost-effectiveness are considered to be part of manageability.)
Accuracy	The system performs its functions correctly. Results of calculations are

	free from error, and data is protected from loss or corruption.
Usability	The software is easy to use and suitable to the user's needs.
Responsiveness	The company accepts responsibility for problems, and takes action to correct them. Help is provided to customers in planning for, installing and operating the product.
Transparency	The company is open in its dealings with customers. Its motives are clear, it keeps its word, and customers know where they stand in a transaction or interaction with the company.

Some examples:

- Secure by Design: An architecture might be designed to use triple-DES encryption for sensitive data such as passwords before storing them in a database, and the use of the SSL protocol to transport data across the Internet. All code is thoroughly checked for common vulnerabilities using automatic or manual tools. Threat modeling is built into the software design process.
- Secure by Default: Software is shipped with security measures in place and potentially vulnerable components disabled.
- Secure by Deployment: Security updates are easy to find and install—and eventually install themselves automatically—and tools are available to assess and manage security risks across large organizations.
- Privacy/Fair Information Principles: Users are given appropriate notice of how their personal information may be collected and used; they are given access to view such information and the opportunity to correct it; data is never collected or shared without the individual's consent; appropriate means are taken to ensure the security of personal information; external and internal auditing procedures ensure compliance with stated intentions.
- Availability: The operating system is chosen to maximize MTBF (Mean Time Between Failures). Services have defined and communicated performance objectives, policies, and standards for system availability.
- Manageability: The system is designed to be as self-managing as practicable. Hotfixes and software updates can be installed with minimal user intervention.
- Accuracy: The design of a system includes RAID arrays, sufficient redundancy, and other means to reduce loss or corruption of data.
- Usability: The user interface is uncluttered and intuitive. Alerts and dialog boxes are helpful and appropriately worded.
- Responsiveness: Quality-assurance checks occur from early on in a project. Management makes it clear that reliability and security take precedence over feature richness or ship date. Services are constantly monitored and action is taken whenever performance doesn't meet stated objectives.
- Transparency: Contracts between businesses are framed as win-win arrangements, not an opportunity to extract the maximum possible revenue for one party in the short term. The company communicates clearly and honestly with all its stake holders.

Execution

Execution is the way an organization conducts its operations to deliver the components required for Trustworthy Computing. There are three aspects to this: Intents, Implementation, and Evidence. Intents are the corporate and

legislative guidance that sets requirements for the design, implementation, and support of the product. Implementation is the business process that operationalizes the Intents. Evidence is the mechanism by which we verify that the Implementation has delivered on the Intent. Some examples:

Intents	• Company policies, directives, benchmarks, and guidelines • Contracts and undertakings with customers, including Service Level Agreements (SLAs) • Corporate, industry and regulatory standards • Government legislation, policies, and regulations
Implementation	• Risk analysis • Development practices, including architecture, coding, documentation, and testing • Training and education • Terms of business • Marketing and sales practices • Operations practices, including deployment, maintenance, sales & support, and risk management • Enforcement of intents and dispute resolution
Evidence	• Self-assessment • Accreditation by third parties • External audit

This problem can only be tackled by working on two parallel tracks.
The first track is the immediate problems—what people read and worry about every day. We need to address known current problems and mitigate currently known weaknesses. This is also a way to learn about the more fundamental problems. We need to be as well-informed as we can about what is really going on and what we can and cannot fix within the constraints of the current systems.

Part of the reason for customer anxiety is that personal computers are now entering areas that they didn't previously worry about. It will be easiest to focus on areas like banking or banking services where such problems are well known and of long standing.

While there is a lot of work to be done through incrementally improving current systems, these efforts will not solve the fundamental problems, some of which are described in the next section.

The computer industry needs to identify and solve the most critical challenges, and fold the solutions in an incremental way into the huge legacy systems that have been built. There will be long technological replacement cycle during which the critical infrastructure systems that society depends on are gradually upgraded to a new and improved status. If these systems already exist, people are not just going to throw them out the window and start over from scratch. So we have to identify critical infrastructure and systems weaknesses and upgrade them on a high-priority basis, and ensure that new infrastructures are built on sound principles.

Fundamental Problems

Policy

Once a technology has become an integral part of how society operates, that society will be more involved in its evolution and management. This has happened in railways, telecommunications, TV, energy, etc. Society is only now coming to grips with the fact that it is critically dependent on computers.

We are entering an era of tension between the entrepreneurial energy that leads to innovation and society's need to regulate a critical resource despite the risk of stifling competition and inventiveness. This is exacerbated by the fact that social norms and their associated legal frameworks change more slowly than technologies. The computer industry must find the appropriate balance between the need for a regulatory regime and the impulses of an industry that has grown up unregulated and relying upon *de facto* standards.

Many contemporary infrastructure reliability problems are really policy issues. The state of California's recent electricity supply crisis was triggered largely by a bungled privatization. The poor coverage and service of US cellular service providers is due in part to the FCC's policy of not granting nationwide licenses. These policy questions often cross national borders, as illustrated by the struggle to establish global standards for third-generation cellular technologies. Existing users of spectrum (often the military) occupy different bands in different countries, and resist giving them up, making it difficult to find common spectrum worldwide.

Processing

Complexity

We are seeing the advent of mega-scale computing systems built out of loose affiliations of services, machines, and application software. The emergent (and very different) behavior of such systems is a growing long-term risk.

An architecture built on diversity is robust, but it also operates on the edge of chaos. This holds true in all very-large-scale systems, from natural systems like the weather to human-made systems like markets and the power grid. All the previous mega-scale systems that we've built—the power grid, the telephone systems—have experienced unpredicted emergent behavior. That is why in 1965 the power grid failed and rippled down the whole East Coast of the United States, and that's why whole cities occasionally drop off the telephone network when somebody implements a bug fix on a single switch. The complexity of the system has outstripped the ability of any one person—or any single entity—to understand all of the interactions.

Incredibly secure and trustworthy computer systems exist today, but they are largely independent, single-purpose systems that are meticulously engineered and then isolated. We really don't know what's going to happen as we dynamically stitch together billions—perhaps even trillions—of intelligent and interdependent devices that span many different types and generations of software and architectures.

As the power of computers increase, in both storage and computational capacity, the absolute scale, and complexity of the attendant software goes up accordingly. This manifests itself in many ways, ranging from how you administer these machines to how you know when they are broken, how you repair them, and how you add more capability. All these aspects ultimately play into whether people perceive the system as trustworthy.

Hardware, Redundancy

We don't yet have really good economical, widely used mechanisms for building ultra-reliable hardware. However, we do have an environment where it may become common-place to have 200+ million transistors on a single chip. At some point it becomes worthwhile to make that into four parallel systems that are redundant and therefore more resistant to failure. The marginal cost of having this redundancy within a single component may be acceptable. Similarly, a computer manufacturer or end user may choose to install two smaller hard drives to mirror their data, greatly improving its integrity in the event of a disk crash.

We may have new architectural approaches to survivability in computer systems these days, but it always comes from redundancy. This means you have to buy that redundancy. So people will, in fact, again have to decide: Do

they want to save money but potentially deal with more failure? Or are they willing to spend more money or deal with more complexity and administrative overhead in order to resolve the appropriate aspects of security, privacy, and technological sufficiency that will solve these problems?

Machine-to-Machine Processes

The Web Services model is characterized by computing at the edge of the network. Peer-to-peer applications will be the rule, and there will be distributed processing and storage. An administrative regime for such a system requires sophisticated machine-to-machine processes. Data will be self-describing. Machines will be loosely coupled, self-configuring, and self-organizing. They will manage themselves to conform to policy set at the center.

Web applications will have to be designed to operate in an asynchronous world. In the PC paradigm, a machine knows where its peripherals are; the associations have been established (by the user or by software) at some point in the past. When something disrupts that synchronicity, the software sometimes simply hangs or dies. Improved plug-and-play device support in Windows XP and "hot-pluggable" architectures such as USB and IEEE 1394 point the way toward a truly "asynchronous" PC, but these dependencies do still exist at times.

On the Web, however, devices come and go, and latency is highly variable. Robust Web architectures need dynamic discoverability and automatic configuration. If you accept the idea that everything is loosely coupled and asynchronous, you introduce even more opportunities for failure. For every potential interaction, you have to entertain the idea that it won't actually occur, because the Web is only a "best-effort" mechanism—if you click and get no result, you click again. Every computing system therefore has to be redesigned to recover from failed interactions.

Identity

Questions of identity are sometimes raised in the context of Trustworthy Computing. Identity is not explicitly called out in the framework, because a user does not expect a computer system to generate their identity. However, user identity is a core concept against which services are provided. Assertions of identity (that is, authentication) need to be robust, so that taking actions that depend on identity (that is, authorization) can be done reliably. Hence, users expect their identities to be safe from unwanted use.

Identity is difficult to define in general, but particularly so in the digital realm. We use the working definition that identity is the persistent, collective aspects of a set of distinguishing characteristics by which a person (or thing) is recognizable or known. Identity is diffuse and context-dependent because these aspect "snippets" are stored all over the place in digital, physical, and emotional form. Some of this identity is "owned" by the user, but a lot of it is conferred by others, either legally (for example, by governments or companies) or as informal social recognition.

Many elements of Trustworthy Computing systems impinge on identity. Users worry about the privacy of computer systems in part because they realize that seemingly unrelated aspects of their identity can be reassembled more easily when the snippets are in digital form. This is best evidenced by growing public fear of credit-card fraud and identity theft as a result of the relative transparency and anonymity of the Internet versus offline transactions, even though both crimes are equally possible in the physical world. Users expect that information about themselves, including those aspects that make up identity, are not disclosed in unapproved ways.

People

It's already challenging to manage extremely large networks of computers, and it's just getting harder. The immensity of this challenge has been masked by the fact that up to this point we have generally hired professionals to manage large systems. The shortcomings of the machines, the networks, the administration, the tools, and the applications themselves are often mitigated by talented systems managers working hard to compensate for the fact that these components don't always work as expected or desired.

Many of the system failures that get a lot of attention happen because of system complexity. People make an administrator error, fail to install a patch, or configure a firewall incorrectly, and a simple failure cascades into a catastrophic one. There is a very strong dependency on human operators doing the right thing, day in and day out.

There are already too few knowledgeable administrators, and we're losing ground. Worse, the needs of administration are evolving beyond professional IT managers. On the one hand we are at the point where even the best operators struggle: systems are changing too rapidly for people to comprehend. On the other, the bulk of computers will eventually end up in non-managed environments that people own, carry around with them, or have in their car or their house.

We therefore need to make it easier for people to get the right thing to happen consistently with minimal human intervention. We must aim towards a point where decision-makers can set policy and have it deployed to thousands of machines without significant ongoing effort in writing programs, pulling levers, and pushing buttons on administrators' consoles.

The industry can address this in any of a number of ways. Should we actually write software in a completely different way? Should we have system administrators at all? Or should we be developing machines that are able to administer other machines without routine human intervention?

Programming

Tools

Each of these approaches requires new classes of software. As the absolute number and complexity of machines goes up, the administration problem outstrips the availability and capability of trained people.

The result is that people in the programming-tools community are going to have to think about developing better ways to write programs. People who historically think about how to manage computers are going to have to think about how computers can become more self-organizing and self-managing.

We need to continue to improve programming tools, because programming today is too error-prone. But current tools don't adequately support the process because of the number of abstraction layers that require foreground management. In other words, the designer needs not only to consider system architecture and platform/library issues, but also everything from performance, localization, and maintainability to data structures, multithreading and memory management. There is little support for programming in parallel, most control structures are built sequentially and the entire process is painfully sequential. And that is just in development; at the deployment level it is incredibly difficult to test for complex interactions of systems, versions, and the huge range in deployment environments. There is also the increasing diffusion of tools that offer advanced development functionality to a wider population but do not help novice or naive users write good code. There are also issues around long-term perspectives: for example, tools don't support "sunset-ing" or changing trends in capability, storage, speed, and so on. Think of the enormous effort devoted to Y2K because programmers of the 1960s and 1970s did not expect their code would still be in use on machines that far outstripped the capabilities of the machines of that era.

Interoperability

The growth of the Internet was proof that interoperable technologies—from TCP/IP to HTTP—are critical to building large-scale, multipurpose computing systems that people find useful and compelling. (Similarly, interoperable standards, enforced by technology, policy or both, have driven the success of many other technologies, from railroads to television.) It is obvious and unavoidable that interoperable systems will drive computing for quite some time.

But interoperability presents a unique set of problems for the industry, in terms of technologies, policies and business practices. Current "trustworthy" computing systems, such as the air-traffic-control network, are very

complex and richly interdependent, but they are also engineered for a specific purpose, rarely modified, and strictly controlled by a central authority. The question remains whether a distributed, loosely organized, flexible, and dynamic computing system—dependent on interoperable technologies—can ever reach the same level of reliability and trustworthiness.

Interoperability also poses a problem in terms of accountability and trust, in that responsibility for shortcomings is more difficult to assign. If today's Internet—built on the principle of decentralization and collective management—were to suffer some kind of massive failure, who is held responsible? One major reason why people are reluctant to trust the Internet is that they can't easily identify who is responsible for its shortcomings – who would you blame for a catastrophic network outage, or the collapse of the Domain Name System? If we are to create and benefit from a massively interoperable (and interdependent) system that people can trust, we must clearly draw the lines as to who is accountable for what.

Conceptual models

We face a fundamental problem with Trustworthy Computing: computer science lacks a theoretical framework. Computer security—itself just one component of Trustworthy Computing—has largely been treated as an offshoot of communications security, which is based on cryptography. Cryptography has a solid mathematical basis, but is clearly inadequate for addressing the problems of trusted systems. As Microsoft researcher Jim Kajiya has put it, "It's as if we're building steam engines but we don't understand thermodynamics." The computer-science community has not yet identified an alternative paradigm; we're stuck with crypto. There may be research in computational combinatorics, or a different kind of information theory that seeks to study the basic nature of information transfer, or research in cooperative phenomena in computing, that may eventually form part of an alternative. But, today this is only speculation.

A computing system is only as trustworthy as its weakest link. The weakest link is all too frequently human: a person producing a poor design in the face of complexity, an administrator incorrectly configuring a system, a business person choosing to deliver features over reliability, or a support technician falling victim to impostors via a "social engineering" hack. The interaction between sociology and technology will be a critical research area for Trustworthy Computing. So far there is hardly any cross-fertilization between these fields.

Summary

- Delivering Trustworthy Computing is essential not only to the health of the computer industry, but also to our economy and society at large.
- Trustworthy Computing is a multi-dimensional set of issues. All of them accrue to four goals: Security, Privacy, Reliability, and Business Integrity. Each demands attention.
- While important short-term work needs to be done, hard problems that require fundamental research and advances in engineering will remain.
- Both hardware and software companies, as well as academic and government research institutions, need to step up to the challenge of tackling these problems.