

Implementing Enterprise Security Policies

Technical White Paper



© 2002 Sun Microsystems, Inc. All rights reserved.
4150 Network Circle, Santa Clara, California 95054 U.S.A

RESTRICTED RIGHTS LEGEND: Use, duplication, or disclosure by the government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 and FAR 52.227-19.

The product described in this manual may be protected by one or more U.S. patents, foreign patents, or pending applications.

TRADEMARKS

Sun, Sun Microsystems, the Sun logo, Sun Startup Essentials, Trusted Solaris, JVM, and Java are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

THIS PUBLICATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS PUBLICATION COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THE PUBLICATION. SUN MICROSYSTEMS, INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS PUBLICATION AT ANY TIME.



Please
Recycle



Adobe PostScript

Contents

Introduction	1
The Landscape	3
In Front of Our Noses	4
Views of Security	5
Basic Principle	5
The Enterprise Security Policy Specification	6
Physical Security	6
Internal Threats	7
Disaster Planning	7
Periphery Security	7
Firewalls	8
Denial of Service	8
Protocol Security	10
Privacy and Encryption	11
Secure Protocols	12

Identity and Authentication.....	13
Application Security.....	14
The Use of Java™ Technology	14
Bridging the Gap	15
A Methodology for Security Policy Assurance	15
Applying this Approach.....	17
Proposition	19
Testing the System Against the ESPS	20
Summary.....	21
References.....	23

Foreword



Engineering has always been concerned with *predictability*, the expectation for development efforts to result in products which perform desired functions with acceptable reliability and availability on time and at acceptable cost. However, most branches of engineering experience strict physical and technical limitations that impact what can be economically achieved. As a result, strong emphasis is placed on estimation and calculation, such as the amount of materials involved, the strengths of structures, and whether multiple purposes can be served by any single construction.

The use of the word *architecture* in software design and hardware infrastructure is analogous to its use in civil engineering. Architecture is the science, and art, of bringing together all major elements of an intended construction or reconstruction so as to fulfill the primary purposes of the project. The goal of architecture is to simultaneously fulfill both primary and secondary project objectives while satisfying an engineering sense of correctness, appropriateness, and elegance. When designing a new IT hardware and software infrastructure, such as that needed by Web sites that utilize the capabilities of the Internet to promote interest to other businesses as well as customers, similar estimations and calculations must be done to ensure sufficient capacities are available without compromising economy.

There are many disciplines to engineering. Those involved in meeting performance, reliability, availability, security and other requirements — the techniques collected under the term *architecture* — are among the least understood. As a result, IT system architects often learn their craft on the job,



from mentors, and through experience. Consequently, architects often fall prey to two tendencies: they tend to be effective only in developing systems seen before, and every problem is made to fit the architect's preconceptions.

Learning the art of architecture on the job has undesirable consequences. Basing all knowledge on mentoring and experience requires years observing and imitating more seasoned professionals. Eventually, such experience leads to good architectural judgement — but good judgement often comes from learning from mistakes. This slow and error-prone approach is unacceptable in a world where the need for new systems is great. Architecture must be taught to budding architects, and seasoned professionals need better tools with which to evaluate decisions without requiring systems to be built.

To be successful, architecture must be transformed into a rigorous process using quantitative mathematical tools. In an effort to help this process, Sun has developed a series of white papers on *Quantitative Architecture* — the discipline of developing architectures using rigorous techniques that enable architects to predict the non-functional properties of a system to ensure, from the very beginning, that the system meets its goals.

In order to develop a rigorous and quantitative approach to architecture, it is important to understand the disciplines that enable predictions to be made regarding CPU power, network bandwidth, response times, and so on. Toward that end, the white papers address the following topics:

- *Introduction to the Elements of Web Application Architecture*, introduces the sequence of considerations that go into what may be termed modern *dot-com* or *Web-based* architectures, and explains some of the terms and principles that apply.
- *What is Architecture?*, discusses the design processes that lead to an appropriate architecture that meets design and business goals.
- *Simplified Performance Estimation*, presents a simple and approximate performance estimation method for early capacity planning which enables architectural decisions to be made with confidence.
- *Understanding Growth*, discusses ways to make the growth problem easier to understand, and aims to facilitate the finding of good estimates of the effects of growth and alleviate some scalability concerns.



-
- *Understanding Statistics and Queuing — Probability, Distributions and Statistical Models*, delves deeply into the bases of these kinds of computations, making evident the assumptions in identifying the various kinds of scenarios in which it is possible to obtain useful, and interesting, results.
 - *Understanding Capacity and Scalability*, presents both capacity and scalability in a basic mathematical model, and shows how common architectural decisions affect them.
 - *Implementing Enterprise Security Policies*, discusses security policies and the potential for machine verification and other techniques.

This series is based on the Sun Startup EssentialsSM program, a class devoted to teaching entrepreneurs and their colleagues the steps needed to establish and operate business-to-client or business-to-business Web-enabled sites. Discussions with students revealed that quantifying many decisions enabled good architecture — and that the basis of the methodology is rarely fully understood. These papers aim to bridge the gap, making the underlying theory accessible and able to serve as a firm foundation, and demonstrating, by example, how these foundations support resulting hardware and software architectures.



Implementing Enterprise Security Policies



A primary obstacle to building secure Web-based systems is the problem of stating, specifying, and testing the properties that make the system secure. These properties are always based on a statement of what is and is not permitted. To build a commercial system with an enforceable security policy, loosely-stated security policies must be transformed into clear and testable specifications of security-related behaviors. This paper provides a definition of security, surveys the security landscape, and describes a proposed methodology that bridges the gap between security policy and secure implementation through a construct called the Enterprise Security Policy Specification (ESPS). Construction of the ESPS is integrated with the analysis of a system using use-cases and object-oriented analysis in the style of Jacobson.

Introduction

Writers often joke they can sound intelligent about an unread document by asking, “Who is the intended audience?” Similarly, the question, “What do I have to do to make my system secure?” can be answered by saying, “It depends on your security policy.”

Much of the history of security originated in military and defense, with security built around impregnable physical defenses, a hierarchy of authority and “need to know,” and the use of expensive encryption and coding mechanisms, one-time codes, and couriers. Much of this was formalized in the requirements for computer systems for use by these agencies — the Trusted Computer System Evaluation Criteria (TCSEC), a publication of the United States Department of Defense (DoD 5200.28-STD), as interpreted by the National Computer Security



Center (NCSC) [10] in other artifacts of the “Common Criteria” world. A set of books, each designated by a different color, and collectively known as the Rainbow series [15], explains and extends these criteria.

While important, these issues do not always apply to commercial installations. Companies cannot ask someone to prove a *need to know* before granting them access to a Web site. Where authority and *need to know* do apply, such as when users access bank account details, it is unrealistic to send couriers to account owners, at regular intervals, to give them new codes and keys to ensure account access remains secure. Such thinking defeats the objectives of the information highway, just as the use of a man with a red flag to warn of on-coming automobiles defeated the usefulness of automobiles.

Companies want to distribute information as easily and as effectively as possible, keeping restrictions to a minimum. Unfortunately, many commercial security policies are uninformative — they are composed of motherhood statements (“information about individuals will remain private”), vague guidance (“employees are required to protect sensitive data”), and threats (“all violations of this security policy will lead to immediate dismissal and legal action”). But, does this matter?

The increase in articles in the popular press, trade magazines, and on-line e-zines concerning the security of the Internet suggests there is, in fact, a real problem. Security has three aspects, each of concern on the Internet, or in any Information Systems (IS) application.

- *Authentication*, supporting the need to evaluate authorization. Authentication identifies the actor, person or system component acting for itself or on behalf of a person requesting the system perform an operation. Authorization decides whether the actor identified may have that operation completed.
- *Privacy*, ensuring that no communication can be overheard, copied, or played back, enabling the meaning to be extracted by any actor unauthorized to do so. Ideally, truly private communication hides the fact that any communication occurred.
- *Integrity*, assuring that any data stored in an information system or transmitted by any means of communication has not been altered or corrupted in any way, either accidentally or deliberately, other than by actors authorized to do so.

Attempts to circumvent any of the protections put in place to support the above guarantees are termed *attacks* on the security of the system.



Another kind of attack, which does not lead to data loss or corruption, and, in fact, does no permanent harm to the system, is the *Denial of Service* (DoS) attack. While not strictly a security issue in the above sense, limiting the effects of a DoS attack is treated as a security attack, since the effects and defenses have much in common.

Another issue which falls under the purview of security requirements is that of *monitoring, logging* and *management*. Just as the price of freedom is eternal vigilance, it is also true that security only comes from detecting attempted attacks, taking further defensive measures, and, in the most egregious cases, putting the attacker out of business. For this reason, no security regime is complete without the following:

- A means to know when attacks are being made
- A method of recording pertinent information in order to discover what was being attacked and, hopefully, by whom
- A way of changing the security regime in such a way as to avoid repetitions of those attacks

What popularly published articles do show is:

- No information is completely safe from unauthorized access, copying, or change
- New ways of attacking IS installations and applications are being found all the time
- The dangers from carelessness, ineptitude, and lack of training are as great as the dangers from those attacks launched with malicious intent.

This paper focuses on what can be done to state an effective security policy, take the steps to enforce it within an IT system, mitigate the risks of a security breach through careful architecture, and apply systematic methods of risk evaluation before attacks occur.

The Landscape

Everyone has reason to be interested in Internet and IS security. Those providing IS services, including the Internet, can only attract and keep customers by assuring security. Users need to know their money will not be stolen, their identity cannot be stolen¹, their personal information will not be inappropriately

1. An identity is stolen when someone obtains the means to pretend to be another person. With a stolen identity, a person can obtain access to email, credit and other important information, possibly leading to the destruction of a good reputation.



divulged, and so on. On the other hand, customers do not want security concerns to inhibit them from using prevalent and convenient IS and Internet services.

In Front of Our Noses

The fact that the popular press is becoming more interested in reporting on security issues reveals that security is becoming critical to IS and Internet systems. The following items illustrate the breadth of the problem.

- As detailed in a well-circulated Computer Emergency Response Team (CERT)¹ advisory [5], current implementations of the TCP/IP protocols, used for moving data on the Internet [11], have inherent flaws. These flaws can be exploited for attacks for DoS and man-in-the-middle² attacks, enabling attackers to overhear conversations and causing parties to reveal information that they would normally only reveal to each other.
- Almost every week, there are reports of new *viruses*, programs that copy themselves and infect other programs. Viruses then perform bad actions, from sending funny messages to everyone in the infected system's email address lists, to changing the behavior of programs to alter data or delete files. The more prevalent a mail program, the more often it gets attacked.
- Similarly, Web-system *hacks* [23], from "script-kiddy" Web site defacement, to concentrated information warfare attacks from active adversaries of the US Government, are becoming common. In fact, hacks are so common that simple defacements are usually not even reported in the press. Recent advertising on TV attempts to sell services by illustrating what could happen, using a fictional, but realistic, launch of a Web site sponsored by U.S. cheese makers.

1. CERT is operated by the Carnegie Mellon Software Institute as a clearing house of security attacks. CERT issues "security advisories" that describe the attacks as well as defensive measures.

2. In a man-in-the-middle attack, communication is intercepted between two parties, and the actor in the middle pretends to be one party when communicating with the other party.



- Claims abound that steganographic¹ codes are being used by U.S. adversaries to conceal information on cracked Web sites for covert, high bandwidth communications. Most notably, this has been reported as a method used by Osama bin Laden's associated terrorist groups.

These examples illustrate that IS and Internet security is indeed the critical issue professionals have long since claimed, and is, perhaps, the critical issue that may prevent wide public acceptance of Internet-based business.

Technological approaches are being used to try to prevent black-hats from exploiting security flaws. These range from system sanitation, the updating and patching of known flaws, to intrusion detection, to the deployment of more highly trusted operating environments and implementations, like the Trusted BSD[®] and Trusted Solaris[™] environments [27].

Views of Security

This paper consider four broad views of security:

- Physical
- Periphery
- Protocol
- Application

The security problem can be sliced different ways, and is sufficiently complex that even experts disagree on the relative importance of different aspects [21]. Since the intention here is to introduce the kind of thinking that goes into seeking solutions, the above division of views will suffice. However, every IT system will exhibit different properties, and different owners or operators will have different concerns. As a result, the analysis and treatment of each IT system will be different, and the best available advice should be sought. Nevertheless, this paper conveys the need for caution, and should impart the necessary skepticism about easy solutions to the need for security policies, no matter how subdivided.

1. Steganographic codes hide one message inside another innocuous message. In written codees, this may involve the hiding of a message in plain sight by writing a note, where the hidden message is found by reading, for example, every sixth letter of the body of the note. One computer method involves hiding the message bit-wise in the least significant bit of each pixel of a graphic.



Basic Principle

There is a basic, underlying principle to all security activities. The effort to breach security must be, and need only be, more expensive than the value of that which is being protected. Information tends to decrease in value as it gets older — what a business was planning last year is of little value, and plans for next year are far more interesting. It is only necessary to protect information while the effort to obtain it exceeds its remaining value.

There is a certain relativity to this principle. While organizations may abide by a fairly strict accounting of information value versus cost of information acquisition, value assessments by other agencies may be biased by other elements of their value system. For example, when countries are at war, the enemy's data is highly valued, to the extent that soldiers, and even civilians, sacrifice their lives to obtain information about enemies, their plans, and movements. At the other extreme, the message "fire now!" does not need to be protected in any way — by the time the enemy hears and understands it, it is already too late.

Similarly, hackers tend to place little value on their time and effort compared to the glory of hacking a system. Even if the data on that system has little intrinsic value, the system may still be a target for the enterprising hacker.

The Enterprise Security Policy Specification

This paper references the Enterprise Security Policy Specification (ESPS), each enterprise's equivalent to the TCSEC for the military and defense industries. No attempt is made to describe how to write this specification, as it is impossible to address the range of situations faced when providing on-line, business-to-business (B2B), or business-to-customer (B2C) services. Expert advice must be sought.

Evaluating a system against an ESPS has at least two significant problems:

1. While the objectives of a security policy are reasonably simple to state, it is very difficult to state, specify and test for the properties of a system that can guarantee achieving those objectives. It is also difficult to show that a system is free of known vulnerabilities, making it difficult to classify a system as being more or less secure on this basis.



2. The security or trustworthiness of an actual application's realization cannot be guaranteed based on its environment alone. Trust of the application within its environment is largely determined by trust in the application itself. This trust is only established by demonstrating that the application has properly, uniformly, and without exception, been implemented to meet the requirements imposed by the security policy.

The problem of stating, specifying, and assuring the proper implementation of secure Web-based systems is considered in this paper. By surveying these issues, the breadth of such a specification may be appreciated and understood, fostering recognition of the process characteristics for taking a specification and turning it into effective action. Such a specification is different in many ways from the TCSEC approach, and the characteristics needed in an ESPS are mentioned in context, as each of the views of security are surveyed.

Physical Security

The components of an information system, as well as its infrastructure, must be physically secure. There is no value in using sophisticated networking protocols to prevent data loss by hackers if it is possible for an individual to approach the computer installation and disable the CPU by emptying a bottle of sugary soda into the top of the electronics racks.¹

Internal Threats

All the typical strategies employed for physical security, including locked doors, security personnel, surveillance systems, restricted access to the utility services access points, etc., are applicable to IT systems.

Restricting physical access to critical IT system areas applies as much to the employees of the enterprise as it does to all others. Furthermore, this applies to all possible modes of access, including the simple ability to execute commands on a system [1]. Surveys have shown repeatedly that significant risks to information systems come from inside the organization, and are part of the burgeoning white collar crime wave [4]. The risks run from simple and misguided opportunism to the malicious, and often time-delayed, activities of

1. In an infamous example, this occurred accidentally when an operator brought lunch into a computer room. When it comes to security, sometimes best friends may, unintentionally, wind up being the worst enemies.



disaffected employees. In a famous case, a laid-off employee left a software time-bomb that would not “go off” until the anniversary of his departure. (Similar stories can be found in reference [16].)

Disaster Planning

Another aspect of physical security, as it relates to safe data and continuing service, is dealing with accidents or sabotage that may be inflicted by the external world. This includes everything from overload protection on the electrical supply, duplication of critical components and services, to coping with natural disasters such as flooding and earthquakes. The events of September 11, 2001, have given a new emphasis to the needs and requirements of disaster planning.

Disaster planning is an extensive subject, and its mention here is intended as a reminder of its importance. Furthermore, all backup and disaster recovery plans should be tested repeatedly, and the skills of those chosen to execute the plan must be re-evaluated at intervals consistent with the objectives of the plan. No guarantees can be made about a plan that has never been tried in practice using real equipment and real people.

Periphery Security

In the real world, physical security and periphery security are often just variations on a theme. Physical security is a barbed wire fence around the property, with locked gates. In IT environments, *periphery security* is the equivalent of airport security barriers — it is the first opportunity to determine if a person (or protocol request) has a reasonable right to be there, and is not carrying anything dangerous.

Firewalls

In IT environments, every electronic communication is passed through a *firewall*, a system, or a layer of a system, dedicated to narrowing the allowed reasons for entry, and having broad powers to check the payload of the communication. In networking, communications are addressed to *ports*, numbered access points. For example, HTTP communications to Web sites use port 80. There are other port numbers corresponding to telnet, ftp, remote procedure calls, management information, and so on. Often, Web sites close all ports except port 80. Any communication destined for any other port is arbitrarily rejected. Some firewalls

are further programmed to investigate the content of the HTTP request and, for example, those attempting to invoke the execution of a program, or those trying to transport data formats that may represent executable programs, are also blocked.

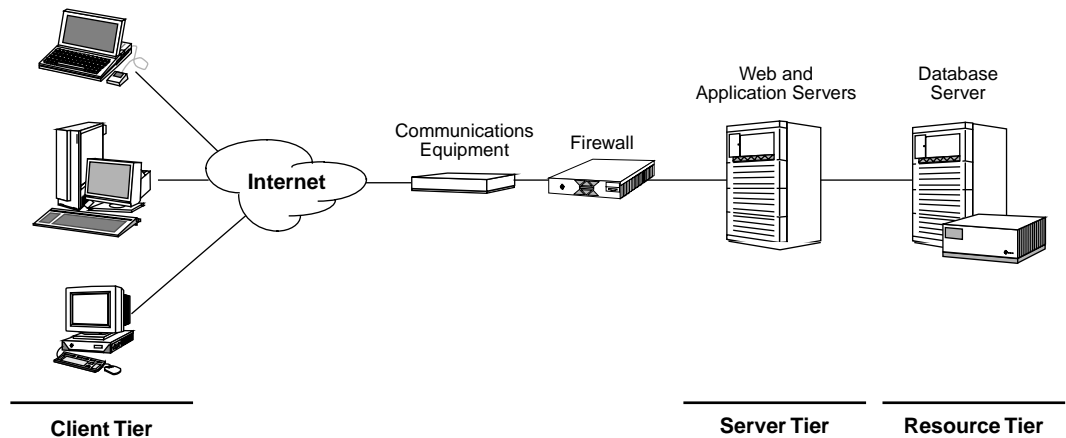


Figure 1 Three tiers, with a firewall

Those attempting to gain entry to an information system must find a way through the firewall. The secret is to detect the attempt through logging and monitoring before the attacker has time to succeed. Given time, a firewall can be breached. All that is needed for defense is sufficient time to react and isolate the firewall system, so that there is nowhere to go. More critical systems often employ a second firewall with a different design to provide even more delay. The most critical parts of the IT system reside behind the second firewall. For responsiveness, less critical components are placed between the firewalls, an area termed the *DMZ*, by analogy with the function of a real demilitarized zone.

Denial of Service

The firewall is often the first line of defense against DoS attacks. All requests for service from the Web site, and the IT systems behind it, must first arrive at the firewall. Even if other measures are taken, the firewall must be able to resist DoS attacks to a reasonable degree.



A *DoS attack* is an attack in which spurious requests are made in an attempt to keep a system busy, taking time and resources away from the task of responding to legitimate requests. The most efficient DoS attacks are those in which a complex request is processed all the way to completion, even though the requestor is not a legitimate client.

The *Internet infection era* may have begun with the attack of the “Internet Worm” in November, 1988 [26]. While designed to quietly copy itself from computer to computer, and to run invisibly in the background, a programming error (even viruses can have bugs!) caused it to replicate as fast as possible, consuming all available CPU power on infected machines. System administrators noticed the additional load very quickly.

It is possible to regard the “Internet Worm” as:

- A security violator, as it successfully guessed passwords
- An exploiter of an operating system weakness, as it relied on the mail transmission application being installed with debugging facilities enabled
- A DoS attack, as it rendered infected computers virtually unusable by consuming CPU cycles

When DoS attacks are unable to find spurious requests that consume a lot of resources, they often turn to making a large number of small requests. Consider an airport security analogy. A denial of service attack could be organized by having a number of people with forged boarding passes (good enough to pass the initial inspection at the security barrier) with suspicious items in their hand luggage or pockets present themselves for a security inspection. Each person would need to go through a secondary inspection, using time and resources. Just a few people can create a significant back-up queue of real passengers. To be effective, approximately 500 people would need to help with the DoS attack. At the average airport, 500 imitation passengers, each needing a search time of just one minute longer, could cause delays approaching 10 hours or more. Furthermore, this would cause a ripple effect, as passengers who missed flights lined up at customer service counters to be re-scheduled. In the language of mathematics, the situation would be chaotic.

Of particular interest in the case of firewalls, requests to ports that are closed must be dealt with very quickly, and should not require any host resources to be allocated. In the best case, all remaining bytes in the request should be ignored,



not even stored in memory, as soon as the closed port address is recognized. Network bandwidth is still wasted, but the effect is not amplified by the firewall taking more than the network transmission time to ignore the unneeded request.

Similarly, authentication credentials (e.g., a username and password, account number and PIN) should be checked quickly by the Web server, prior to devoting any effort to satisfying the service request. This may be considered to be a second periphery layer. Apart from the need to reject invalid credentials to prevent an unauthorized actor gaining access to more restricted parts of the system, there is also the need to ensure there are limited effects of DoS by a large number of systems connecting over the Internet and presenting fake credentials.

In some sense, DoS attacks always succeed:

- The only way to completely prevent a DoS attack is to offer no service at all. In this case, it can be argued that just the threat of a DoS attack has succeeded in a total denial of service.
- As long as a service is offered, some amount of available resources must be allocated to rejecting invalid or unauthorized requests. This succeeds in denying some part of the service.
- If simply rejecting such requests cannot be accomplished in a reasonable manner, the usual response is to ration the service — like discounted items at the supermarket being limited to ensure no single customer walks off with the entire supply. In the Web service case, this might involve not allowing a new request from the same user for some number of seconds. For example, many systems impose a delay of 20 to 30 seconds after a failed login, ensuring that someone who is randomly trying passwords cannot make progress too rapidly. Unavoidably, this delays the legitimate user who has simply made a typing error. Rationing the service is itself a partial denial of service — once again, just the threat of a DoS attack has led to reduced service levels.

A broad range of firewall products is available, from software that sits on top of network connections on single systems, to small box solution products suitable for home or small office use, to large systems capable of screening thousands of communications per second. More information on using such products, together, separately, or in multiples, can be found in *Introduction to the Elements of Web Application Architecture*.



Protocol Security

Some communications must be allowed through a firewall — without them there is no Web site or Web service. A *protocol* specifies the sequence of messages needed to communicate, as well as the form and content of the messages at each stage. At issue in any protocol are the opportunities for black-hats to compromise either the privacy or integrity of the protocol and the information being conveyed. The following sections discuss some of the basic problems, considerations, and solutions.

Privacy and Encryption

The question of privacy is usually addressed through *encryption*. The nature of encryption has changed considerably since the introduction of purely mathematical transformations based on the estimated difficulty of certain numerical computations, such as finding the prime factors of very large numbers. No longer are electronic messages encoded by doing letter substitutions, or looking up messages in a codebook, even though these methods remain popular for puzzles in newspapers and magazines.

Advances have been made in *public key* encryption systems, where the number (the key) with which the message is encoded is made public, and is published just like a listed telephone number. This works by requiring a *private key* to decode the message. Each private key corresponds to one public key. The success of this method is the difficulty, based on the known mathematics, of learning the value of the private key from knowing the public key. This can be done. The security depends upon the same logic as was explained above. Decrypting the message takes a long time for large keys, with times estimated in hundreds of years for public keys with two thousand or more bits (600 decimal digits) [25].

Since encoding messages with the very large numbers involved in public key encryption systems takes considerable time and computing power (perhaps 0.1 to 10 seconds), it is too burdensome for messages that may need to be exchanged hundreds of times a second. The solution is to use the public key encryption system to exchange a much shorter number (between 64 and 128 bits, or 20 to 40 decimal digits) that is used for symmetric encryption, a technique in which the same number is used for encryption and decryption. This is termed a *session key* as it is typically used for only a single session between actors. Symmetric encryption with a shorter session key is extremely fast, making it suitable for situations in which messages must be exchanged rapidly. If the protocol includes the undetectable changing of the session key at random intervals, such as forty



or fifty times a day, it can be assured that the effort expended by a black-hat to discover the key is thoroughly wasted. The popular Digital Encryption Standard (DES), designed for banking and other commercial uses, is a symmetric encryption system [8]. Plain DES uses a 56-bit key, offering approximately the same protection as a 384-bit public key. DES is regarded as “weak” encryption, using a symmetric, 56-bit key. Modern usage is “Triple DES”, where the DES algorithm is applied three times, each with different keys.

The use of encryption in Java technology, in particular in the building of Web services, is supported by the Java Cryptography Extension [19].

Secure Protocols

The Secure Sockets Layer (SSL) is a commonly available mechanism for encrypting a communication. SSL encryption is performed at the layer where TCP connections are established from system to system. Being at this relatively low level, SSL addresses only the encryption of the messages between the end points, and does not deal with any of the higher level issues that may be associated with the meaning of the content of the messages. SSL uses a technique called the *Diffie-Hellman key exchange* [25] to establish a session key in such a way to ensure a listener cannot determine the value of an agreed upon key. Work is being done to advance a *Security Architecture for the Internet Protocol* [12], to further secure communications over the Internet Protocol (IP), on which TCP and other Internet protocols are based.

The Hyper Text Transport Protocol (HTTP) protocol is used to communicate with Web servers. The *http:* token that precedes a URL indicates to the receiving computer that HTTP is the protocol to be used. This same set of messages may be used over a connection established using SSL, in which case the Secure HTTP (SHTTP) protocol is used. The *https:* token which precedes a URL indicates to the receiving system that the SHTTP protocol is to be used. Most sites dealing with personal information, credit card or banking transactions, and other sensitive information, use the SHTTP protocol. The SSL protocol, originally developed by Netscape Communications, Inc. is now standardized by the Internet Engineering Task Force (IETF) [14]. Implementations are available that have been developed by other organizations [24]. Because of the low-level nature of SSL, it is not a complete solution for higher security applications [29].

While not a complete solution, encryption goes a long way towards ensuring privacy. It is important to note that a black-hat may still compromise integrity by recording messages one day and injecting them into a message stream on a later



day in the hope of confusing or deceiving the actors that are exchanging messages. Steps can be taken to defeat such *replay attacks* by writing sequence numbers into the packets of the message and adding date/time stamps. Additional precautions may be taken, such as sending each message twice, with the recipient acknowledging the message only after two identical copies are received. A more economical and efficient approach is to attach a *digest* to each message [25]. A digest is a large number, usually between 256 and 512 bits, that is based on a complex but fast computation using the message as an input to the arithmetic, such that the given number should only be obtained with exactly the correct message. If the recipient of the message recomputes the digest, a matching value to the one computed by the sender ensures not only that the message has not been tampered with deliberately, but also that it has not been altered by accident, such as an electrical fault, a lightning storm, or a component failure. Digests can be combined with encryption to enable the digital signing of messages as well [9, 25], and such digital signatures are now acceptable by law [28].

Additional steps must be taken to avoid man-in-the-middle attacks. Such steps might include exchanging tokens that can only originate with the legitimate parties to the exchange, and cannot be forged by third parties. This is analogous to people mixing concealed references to private moments into their telephone conversations — anyone intercepting the conversation, and trying to pretend to be the one of the parties, does not understand the significance and cannot come up with the correct response if trying to insert their own answer into the stream.

Creating such protocols and protections requires an expert, and is beyond the scope of this document. For example, this problem has been solved for the Domain Name System (DNS), the world-wide system that translates the name of a system (such as *www.sun.com*) into an actual IP address. Being able to subvert this procedure by being a man-in-the-middle, and supplying false IP addresses for such names, offers enormous opportunities for doing mischief. Fortunately, this is addressed in the description of the DNS Security Extensions [13].

Identity and Authentication

Protocol security also has a role to play in authentication — actors need to present verifiable credentials, and the system must attempt to verify that these credentials have not been stolen. To ensure they are not stolen while being presented, credentials must be kept private. Furthermore, credentials must not be tampered with, either by someone stealing them, or while being transmitted at the time of presentation to a service.



Since encryption, together with the related techniques of generating digests, is the path to privacy and integrity, a correct and verifiable secure protocol is needed. One approach is the use of certificates [7]. The question of *network identity*, the ability to manage, control, and use authentication credentials securely on the Internet, is becoming a more important question. While this problem poses many challenges, organizations are responding [20]. Microsoft Passport is one example solution, as is another under development by the Liberty Alliance Project [22], sponsored by a large number of companies.

Application Security

The heart of the security question is whether the application itself is capable of exercising all of the needed actions to implement the terms of the Enterprise Security Policy Specification. Ultimately, the application software is the crucial gatekeeper of security. The application program is the mediator between the services being offered to users and the requests they generate, and the “crown jewels” of the corporation — its intellectual property, and the contents of its databases.

The application program is the last chance to check on and authenticate the identity of the requestor, verify the permissions, allocate appropriate authorizations, and commit to allowing data to be changed in the permanent database or released to the requestor. In essence, the application is “where the rubber meets the road.”

Since applications tend to be unique, it is important to find a way to ensure the ESPS is being met for a given application. The following sections propose a method. As in all security evaluations, multiple methods should be used to perform the security audit. The following method is constructive — it is applicable while the application is being designed and built. Once built, an application should be subjected to an audit using a different methodology in order to ensure that no door has been left unguarded.

The Use of Java Technology

Security policy implementations tend to be based on a reference monitor approach. In particular, the Java 2 Platform Security Architecture depends on a role-based model, and on the invocation of the appropriate Security Monitor at every point in the code where a potentially security-critical operation can be performed.



There is no question that this architecture is sufficient to ensure that an access-control policy can be enforced. However, making the mapping from the security policy of an organization to the appropriate roles, permissions, and security-critical operations is, at best, very difficult. It is difficult enough that the most common question considered by IT architects and by those considering the use of a Java technology-based application is: “How can we really apply the Java technology-based security model?”

Bridging the Gap

The problem of applying security models has been deeply investigated, and is exemplified in TCSEC. Yet, any TCSEC system must somehow leap from the generic requirement that information must not be disclosed to any entity not cleared to receive it, to an implementation that purports to make this guarantee. At high levels, strong validation or verification of the properties of the implementation is intended to enforce authentication, access control through proper authorization, and the principle of least privilege, as well as to limit the bandwidth of covert channels.

The bridge from requirements to implementation in TCSEC environments comes in two parts:

1. A *descriptive top level specification* that “completely and accurately” describes the implementation of the security policy “in terms of exceptions, error messages, and effects.” (TCSEC 3.2.3.2.2)
2. A *formal top level specification* that restates the descriptive top level specification in a mathematically verifiable form.

Both of these depend on a formal, axiomatic statement of the security policy that can be (at high levels) proven consistent.

This bridge provides a chain of inference from the security policy itself — which may have motherhood problems — to the detailed and rigorous statement of what steps are taken to ensure the realized system enforces the security policy, and from there to the implementation itself for inspection and validation.

For commercial, Web-based systems, this kind of bridge, or chain of inference, is also needed to assure that Web-based systems properly perform within the system’s security policy.



A Methodology for Security Policy Assurance

Clearly, taking a high-assurance TCSEC approach to Web system security is neither desirable nor feasible. On the other hand, providing greater assurance that the Web application actually implements all the security requirements imposed by the security policy is very desirable. The problem, then, is how to do so feasibly.

A feasible and useful method has the following properties:

- It needs to be both explicit and definite. A cookbook that can be followed by system designers in the field without the assistance of security specialists at every turn is ideal.
- It should integrate well into existing methods of analysis and design, particularly object-oriented design, introducing the absolute minimum of new terminology or notation into a process which is already nearly buried in both.
- It must bridge the gap between the initial security policy statement and explicit, testable properties of the final implementation in a way that convincingly demonstrates that the final implementation meets all the requirements of the security policy and the expectations of the application owner.

These properties are roughly identical to another more general problem that has plagued software development — the problem of capturing user requirements and restating them in explicit and testable ways, and constructing a convincing argument showing that all the system's functional requirements have been met. Conventional requirements methodologies allow the convincing argument to be built to any required degree of rigor, including the construction of formally verified assurance mappings that are built from one level of abstraction to the next.

The problem of requirements capture, however, is another issue. One successful approach is that of stating the requirements in the form of *use cases*, user-based descriptions of what the user expects to happen.

Use case modeling can be used to address the initial problem of developing an explicit statement of the security policy in a form that can be applied to development. To do so, organizations must:

- Develop an explicitly stated ESPS by examining the actors used in the use case modeling, and extract the role and permission information that the actors have or need in order to initiate the use case.



- Logically examine the rigorous top-level specification, in order to verify that the security policy as stated in the ESPS is complete and consistent.
- Generate the security-related test cases based on the specifications in the ESPS, including both black box (what happens outside the application when the policy is violated) and white-box (the predicted, and real, call traces within the application for a particular test scenario) testing as needed.

Analogous to the TCSEC approach, a rigorously stated bridge is formed between user expectations and testable implementation requirements.

Applying this Approach

A short summary of the use-case modeling method is needed to continue describing this approach [17, 18]. In the initial stage of use-case modeling, a line is drawn that encloses the conceptual system and separates it from the outside world. This establishes the system context, and in fact, the graphical version of this is called a *context diagram* (Figure 2).

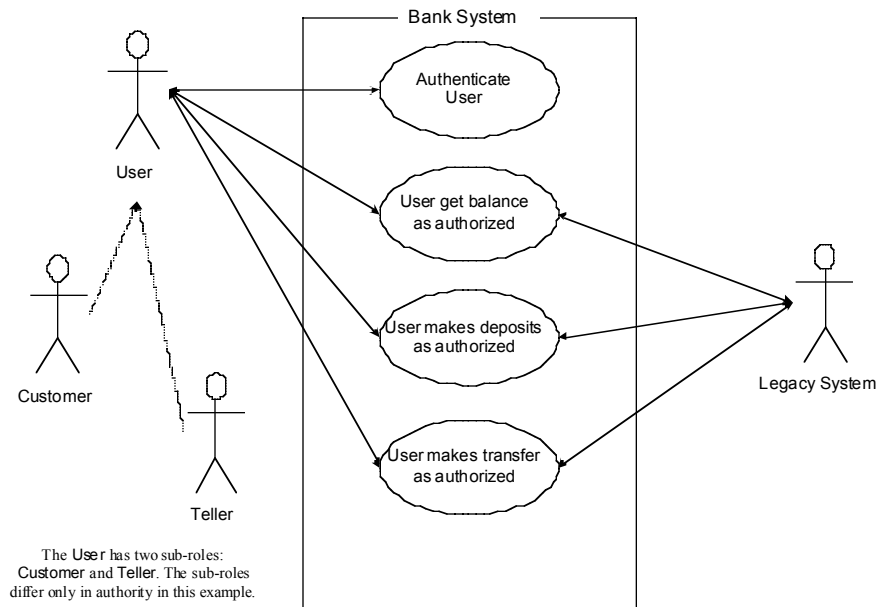


Figure 2 Example Context Diagram



Outside of this enclosure, each entity that can interact with the system is identified. These entities are known as *actors*, anything outside the system under consideration that can interact with the system. A *use case* is a sequence of stimuli initiated by the actor that leads to responses that produce value to the actor.

Use cases are stated in a number of ways, from simply stating use in one sentence, to statements with specific values (user scenarios), to highly detailed descriptive forms [6]. A good, simple form is the user story as used in the Extreme Programming style [2]. Consider the following user story.

A Bank Customer Opens A Balance Page

An on-line bank customer selects the appropriate account from the list of possible accounts. The account balance information is displayed, including the balance as of the last statement, the current account ledger balance, and the current available balance (ledger balance plus deposits available but not yet credited minus current debits not yet processed). An Account Details link, when selected, displays the Account Details page.

User stories are analyzed in various ways. Of interest is the enterprise (or business domain) object modeling step. An Enterprise (or Business Domain) Object Model identifies all the objects that are named in the user stories, as well as the methods for each object in terms drawn from the user stories — and no others.¹

The Enterprise Object Model is the key point at which the enterprise security policy modeling can begin to be integrated. When the enterprise model is completed (all known use cases have been analyzed), two short lists of model entities result:

- The list of actors
- The list of enterprise-level objects

Included with each enterprise-level object is the complete set of all methods identified as being required by any actor. Only the objects contain or represent state, in the sense that only the objects can have any history in the model. This

1. No “programmer thinking” should be allowed to intrude in the business domain object modeling step, as this injects details of the “implementation domain model”, such as the *kind* of Java technology-based object that is to be used. It is much too early to bind the modeling to this level of detail.



leads to a natural and probably correct identification of actors with subjects, and the objects have the properties they would have in a Bell-LaPadula model. The Bell-LaPadula model is the security object model required by the U.S. Department of Defense for software implementations to demonstrate meeting the requirements of the TCSEC [21].

Proposition

Any system security-related behavior is expressed at the time an actor attempts to invoke a method on an object. Since the only behaviors of the modeled abstract system are those that take place when an actor invokes a method of a domain object, this proposition meets the requirements of Jacobson's definitions. Therefore, based on this proposition, the Enterprise Security Policy Specification can be defined. An *enterprise security policy specification* is a specification of the security-related behavior of the system when an actor invokes a system method.

Consider the previous example use case. As well as the actor Bank Customer, a banking system including a Web-based customer component would also need two additional actors:

- Anonymous User, the actor representing a Web access of the banking system by someone not yet recognized as a customer
- Bank Teller, the actor for over-the-counter tellers in a full-service branch

The distinction between an Anonymous User and a Bank Customer is simply whether or not the user has been authenticated as a customer by the defined mechanism. The appropriate section of the ESPS include a statement such as:

```
if user role is Anonymous User
then
    user invokes open customer account list -> exception
    user invokes authentication -> authentication process
fi
```

In fact, this can be expressed in a way exactly analogous to the Bell-LaPadula model by simply constructing a matrix of

(actor × fully-qualified-method)



and specifying the behavior of the system in each location in the matrix. This matrix is a table, with all the actors listed down the left-hand side, and all the methods that may be invoked in the application across the top. In essence, each box at the intersection of a row and column must be filled with the correct information. The table may be sparse (not all actors may be able to reach all methods). However, this evaluation must be made and verified.

In a typical Bell-LaPadula model, the entries in the matrix are just a Boolean *Yes* or *No*, indicating whether or not the actor is permitted to use the method. For the ESPS, and its associated rigorous model, it is more general to map each matrix entry to a general effective computation. This allows the expression of any security policy, including ones that are unlike the Bell-LaPadula model, such as the Chinese Wall model developed for commercial enterprises [3]. Furthermore, this provides an explicit way in which the notion of misuse and abuse cases can be integrated into the methodology as well, by simply introducing black-hat actors. This leads to the explicit identification of all the malicious and nefarious behaviors to be considered, along with the specified action to take when it occurs.

This seems a little strange. How can organizations know what malicious attacks might be made? They cannot — organizations must confront the fact that a system can only be shown to protect itself against attacks that are anticipated.

Testing a System Against the ESPS

To effectively test the system against the ESPS, two items must be verified:

1. The behavior of the system is as specified for each invocation of any method by any actor
2. Every method performs the appropriate action(s) to ensure that the policy is enforced at every invocation

Each of these verifications must consider each kind of security value: authentication and authorization, privacy, integrity, and the ability to resist a DoS attack. Finally, it should consider whether an attack or critical action is being logged or monitored appropriately.

Ideally, all of this should reduce to the same collection of black-box tests — by construction, every method should be invoked only by defined actors. Unfortunately, this may lead to *testing in the negative* — in many cases the act of invoking a particular method and successfully passing the policy restrictions is



indistinguishable from simply invoking a method that does not enforce the policy restrictions. Buffer overrun attacks are the degenerate case. A buffer overrun attack can take place only because the implicit policy of not executing random code passed into the system from outside is violated when the system does not enforce its own buffer-size restrictions. One of the advantages of using a programming language such as Java™ is that buffer-overrun is detected and inhibited by the Java Virtual Machine (JVM™) itself.

An interesting version of this attack arose in a Netscape library which used Java technology. A library flaw allowed a simple Web server, implemented as an applet, to make itself available to any caller. This was a violation of the Java technology-based security policy, which requires an applet to only allow TCP/IP connections to the IP address from which it was served. The problem was that the implementation of *java.net.ServerSocket* did not call the appropriate Security Manager to find out whether or not a connection was permitted.

By specifying the security-related behaviors through the ESPS model, organizations have an opportunity to explicitly state not just the expected behavior, but also to state a simple and testable property of method-call traces that ensures every traced call does in fact attempt to enforce the policy. This property is that every invocation should include a call to the reference monitor or Security Manager between invocation and the return of a result. In a Java environment, this can be implemented very directly by using the facilities of the debugging API to the JVM software.

Summary

The overall security problem revolves around stating the requirements for security in a way that enables them to being verified and tested. The breadth of the problem has been presented by consideration of a set of views of the security landscape. A primary obstacle to building secure Web-based systems (and other IT systems) is that security is usually predicated on a security policy, a statement of what is or is not permissible. In the TCSEC Orange Book, assurance resulted through rigorous review and (at high assurance levels) mathematical verification.

With enough money and time, this approach can be effective. Commercial development never seems to have enough money and time, and the methods used in TCSEC environments require significant, specialized skills. The problem is worsened by the fact that commercial security policies are often not very informative about the specifics of what is to be done and what rules should be enforced.

To bridge the gap between commercial security policies and testable, implementable requirements, the idea of an enterprise security policy specification was introduced. For the evaluation of the necessary qualities of a Web-based application, the ESPS was seen as a matrix into which a specification of the correct security-related behavior for every possible invocation of the system by any entity assuming any defined role can be placed. This matrix is filled incrementally as the system is implemented, simply by asking the question in the analysis of each use case “under what conditions can this enterprise-level method be invoked, and what should be the behavior if a forbidden operation is attempted?”

First, this approach satisfies the original goals of this paper. It is easily turned into a cookbook operation. It also effectively introduces no new notation or terminology, except for the idea of stating the requirement in tabular form that may be directly — almost trivially — integrated into the analysis and design methods commonly used for object-oriented analysis throughout the industry. It leads to explicit direction for implementors that can easily be translated into rigorous test plans for positive correct behaviors, and in the negative that the security policy is enforced in every test.

Second, the security policy matrix provides a basis for a rigorous top-level security model. As this is generalized to allow for any effective decision procedure at each (role, method) pair, it appears at first glance that any realistic chance of formal verification of the model has been eliminated. While application of this approach requires a great deal of work, it appears that a model-checking approach to verification along these lines could be used to effectively check the rigorous specification for consistency and completeness.

Third, the approach is tied to use-case modeling methodology, in which the development of the possible attacks and the system’s behavior under these attacks is integrated into the development of the system as a whole, as well as the use-case behavior of system and security management tools and intrusion-detection procedures.

Security specification and evaluation exist in parallel with the other aspects of quantitative architecture, wherein other non-functional requirements such as capacity, availability, and reliability, are specified, and the system is implemented to predictably meet the specification. As with other parts of the practice of quantitative architecture, the security approach described moves non-functional requirements out into the system behavioral (or use case) model, where they can



be implemented in a clear and explicitly testable form. The outcome should be systems with much greater likelihood of development success, and a much smaller chance of catastrophic failure.

References



Sun Microsystems posts product information in the form of data sheets, specifications, and white papers on its Internet World Wide Web Home page at: <http://www.sun.com>.

Look for abstracts on these and other Sun technology white papers:

Simplified Performance Estimation, Sun Microsystems, 2001.

Implementing Enterprise Security Policies, Sun Microsystems, 2001.

Introduction to the Elements of Web Application Architecture, Sun Microsystems, 2001.

Understanding Capacity and Scalability, Sun Microsystems, 2001.

Understanding Growth, Sun Microsystems, 2001.

Understanding Statistics and Queuing — Probability, Distributions and Statistical Models, Sun Microsystems, 2001.

Other references:

- [1] Daniel J. Barrett and Richard E. Silverman, *The Secure Shell: The Definitive Guide*, O'Reilly & Associates, Inc., February, 2001. ISBN 0-596-00011-1.
- [2] Kent Beck, *Extreme Programming Explained: Embrace Change*, Addison Wesley Longman, Inc., New York, October, 1999. ISBN 0-201-61641-6.



- [3] D. Brewer and M. Nash, "The Chinese wall security policy", *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, May, 1982, p206 - 214. (See also: <http://www.gammasl.co.uk/topics/chwall.pdf>.)
- [4] Bureau of Justice Statistics, U.S. Department of Justice. <http://www.ojp.usdoj.gov/bjs/>.
- [5] Carnegie Mellon Software Engineering Institute: Computer Emergency Response Team (CERT) Coordination Center. <http://www.cert.org/>.
- [6] Alistair Cockburn, *Writing Effective Use Cases*, Addison-Wesley Educational Publishers, Inc., October, 2000. ISBN 0-201-70225-8.
- [7] Consultation Committee, *International Telephone and Telegraph (CCITT): Recommendation X509: The Directory—Authentication Framework*, International Telecommunications Union, Geneva, 1989.
- [8] *Federal Information Processing Standards: Digital Encryption Standard (DES)*, Publication 46, January 15, 1977.
- [9] *Federal Information Processing Standards: Digital Signature Standard (DSS)*, Publication 186, May 19, 1994.
- [10] Patrick R. Gallagher, Jr. (Director), NCSC-TG-021, National Computer Security Center, <http://www.fas.org/irp/nsa/rainbow/tg021.htm>, April, 1991.
- [11] *IETF RFC: 793: Transmission Control Protocol, DARPA Internet Program, Protocol Specification*, Information Sciences Institute, University of Southern California, Marina del Rey, California, September, 1981.
- [12] *IETF RFC 2401: Security Architecture for the Internet Protocol (IPSec)*, Internet Engineering Task Force Network Working Group, (Authors: S. Kent, BBN Corp; R. Atkinson, @Home Network), November 1998.
- [13] *IETF RFC 2535: Domain Name System Security Extensions*, Internet Engineering Task Force Network Working Group (Author: D. Eastlake, IBM), March, 1999.
- [14] *Internet Engineering Task Force (IETF): The SSL Protocol, Version 3.0. Internet Draft*, <http://www.netscape.com/eng/ssl3/draft302.txt>, November 18, 1996.
- [15] Federation of American Scientists, *The NSA Rainbow Series*, 1997. <http://www.fas.org/irp/nsa/rainbow.htm>.
- [16] Geek.com, <http://www.geek.com/>.



- [17] Ivar Jacobson, *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison Wesley Longman, Inc., New York, June, 1992. ISBN 0-201-54435-0.
- [18] Ivar Jacobson, Grady Booch, and James Rumbaugh, *The Unified Software Development Process*, Addison Wesley Longman, Inc., New York, January, 1999. ISBN 0-201-57169-2.
- [19] *Java Cryptography Extension 1.2*, Sun Microsystems, Inc. 2002. <http://java.sun.com/products/jce/index-12.html>.
- [20] David P. Kormann and Aviel D. Rubin, *Risks of the Passport Single Signon Protocol*, Computer Networks, Elsevier Science Press, volume 33, p51 – 58, 2000.
- [21] Ronald L. Krutz and Russle Dean Vines, *The CISSP Prep Guide: Mastering the Ten Domains of Computer Security*, John Wiley & Sons, Inc., September, 2001. ISBN 0-471-41356-9.
- [22] Liberty Alliance Project, <http://www.projectliberty.org/>, 2002.
- [23] Stuart McClure, Joel Scambray and George Kurtz, *Hacking Exposed: Network Security Secrets and Solutions*, Osborne/Mc-Graw Hill , September, 1999. ISBN 1-402-81865-3.
- [24] *OpenSSL*, Open SSL Project, 2002, <http://www.openssl.org/>.
- [25] Bruce Schneier, *Applied Cryptography, Second Edition: Protocols, Algorithms, and Source Code in C*, John Wiley and Sons, Inc., New York, 1996. ISBN 0-471-12845-7.
- [26] Eugene H. Spafford, *The Internet Worm Program: An Analysis*, Purdue Technical Report CSD-TR-823, Department of Computer Sciences, Purdue University, West Lafayette, Indiana, December 8, 1988. <http://www.cerias.purdue.edu/homes/spaf/tech-reps/823.pdf>.
- [27] *Trusted Solaris*, Sun Microsystems, Inc., 2002. <http://www.sun.com/software/solaris/trustedsolaris/>
- [28] *Electronic Signatures in Global and National Commerce Act*, Public Law 106-229, US House of Representatives, Session 106, June 30, 2000.



[29] D. Wagner and B. Schneier, "Analysis of the SSL 3.0 Protocol", *The Second USENIX Workshop on Electronic Commerce Proceedings*, p 29 – 40, 1996.



Sales Offices

Africa (North, West and Central): +9714-3366333
Argentina: +5411-4317-5600
Australia: +61-2-9844-5000
Austria: +43-1-60563-0
Belgium: +32-2-704-8000
Brazil: +55-11-5187-2100
Canada: +905-477-6745
Chile: +56-2-3724500
Colombia: +571-629-2323
Commonwealth of Independent States: +7-502-935-8411
Czech Republic: +420-2-3300-9311
Denmark: +45 4556 5000
Egypt +202-570-9442
Estonia: +372-6-308-900
Finland: +358-9-525-561
France: +33-01-30-67-50-00
Germany: +49-89-46008-0
Greece: +30-1-618-8111
Hungary: +36-1-202-4415
Iceland: +354-563-3010
India: +91-80-5599595
Ireland: +353-1-8055-666
Israel: +972-9-9710500
Italy: +39-039-60551
Japan: +81-3-5717-5000
Kazakhstan: +7-3272-466774
Korea: +822-2193-5114
Latvia: +371-750-3700
Lithuania: +370-729-8468
Luxembourg: +352-49 11 33 1
Malaysia: +603-264-9988
Mexico: +52-5-258-6100
The Netherlands: +00-31-33-45-15-000
New Zealand: +64-4-499-2344
Norway: +47 23 36 96 00
People's Republic of China:
 Beijing: +86-10-6803-5588
 Chengdu: +86-28-619-9333
 Guangzhou: +86-20-8755-5900
 Hong Kong: +852-2202-6688
 Shanghai: +86-21-6466-1228
Poland: +48-22-8747800
Portugal: +351-21-4134000
Russia: +7-502-935-8411
Singapore: +65-438-1888
Slovak Republic: +421-7-4342 94 85
South Africa: +2711-805-4305
Spain: +34-91-596-9900
Sweden: +46-8-631-10-00
Switzerland:
 German: 41-1-908-90-00
 French: 41-22-999-0444
Taiwan: +886-2-2514-0567
Thailand: +662-636-1555
Turkey: +90-212-335-22-00
United Arab Emirates: +9714-3366333
United Kingdom: +44-1-276-20444
United States: +1-800-555-9SUN OR +1-650-960-1300
Venezuela: +58-2-905-3800
Worldwide Headquarters:
 Sun Microsystems, Inc.
 4150 Network Circle
 Santa Clara, CA 95054 USA
 Phone: 800-555-9786 or 800-555-9SUN
 Internet: www.sun.com