David Goldschlag
Carl Landwehr
Michael Reed
Naval Research Laboratory
Code 5540, B16, R241
4555 Overlook Avenue SW
Washington, DC 20375-5337
(202) 767-2389

Chapter 17
Agent Safety and Security

Introduction

Automobiles have proven to be a wonderful invention, and people all over the world depend on them. But people also recognize that cars used improperly can cause injuries. Cars can also serve as hiding places for car bombs.

It's hard to imagine a software agent that could cause physical harm to anyone -- it's only software, after all. But what if that software is controlling an electrical appliance, say a coffee maker? Could a control failure cause the coffee maker to overheat and start a fire? We hope not; the coffee maker should in any case have passed an Underwriters' Laboratory test to assure that it won't start a fire even if it fails catastrophically.

It's easy to imagine a software agent that damages data stored in computers, however: usually, it's called a virus. Of course, agents are supposed to be friendly and useful, not malicious and destructive. And they presumably operate in a constrained environment of some sort. But to be useful, an agent must be able to operate flexibly and dynamically: it may, for example, need to determine where to go next in search of some particular piece of data. It may need to store results or to send messages back to its initiator. It certainly will require some computing cycles on every site it visits. (We use the term *applet* for agents that are imported from a remote site for strictly local execution; *agent* encompasses both applets and mobile agents.)

An agent's initiator lends some aspects of his persona to it. The agent consumes resources, and these will eventually have to be accounted for. In addition, the agent may request access to resources that may not be public. The initiator must (explicitly or implicitly) delegate to the agent the authority to make such requests, if they are to be honored.

If agent technology is widely adopted, it will reinforce current trends that are decentralizing information and information-based services. The advent of the World Wide Web has shown how individuals can become producers, as well as consumers, of information. But the decentralization of the Web can also make it difficult to find and assemble information. Agents that can travel around the Web independently and locate interesting information and resources increase the utility of the decentralized system and reduce the need for services focused at single sites.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE **1996**	2. REPORT TYPE	3. DATES COVERED **00-00-1996 to 00-00-1996**
4. TITLE AND SUBTITLE **Chapter 17. Agent Safety and Security**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Naval Research Laboratory,Code 5540,4555 Overlook Avenue, SW,Washington,DC,20375**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S)
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT **Approved for public release; distribution unlimited**		
13. SUPPLEMENTARY NOTES		
14. ABSTRACT		
15. SUBJECT TERMS		

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**17**	

What kinds of problems do we need to worry about in an environment of decentralized information systems, where agents operate flexibly and dynamically with the authority delegated to them by their initiators? Here are a few:

Privacy: Although most people recognize that many organizations hold computerized records about them, the current isolation and inaccessibility of those systems acts to preserve a degree of privacy. If these systems make themselves available to a common network supporting an agent interface, it will become much easier to extract records from different systems about the same individual.

Rogue Agents: A Trojan horse is a program that uses the privileges of the user who invokes it for unintended purposes, such as mailing the victim's password to the program's author. A virus can be thought of as a Trojan horse that attaches itself to another program or file and, when it is executed, propagates itself to other files or possibly other systems. A worm is an independent program that propagates itself within a system or between systems. Both Trojan horse and virus carry negative connotations; they are malicious. A worm may or may not be. An agent is, in these terms, indistinguishable from a worm. Anyone who operates a system that supports agents must ask whether an agent could conceal malicious software, such as a virus or Trojan horse. If this is a possibility, the next question is whether the damage such rogue agents could cause can be limited sufficiently that the benefits gained from friendly agents are worth the risk.

Access Control: Even if an agent is friendly, most systems will need to enforce an access control policy that distinguishes among different agents. For example, a system that stores airline flight reservations should return information about flights on which I am booked to an agent running on my behalf, but not to others. This means that the agent must be authenticated by the system from which it requests services.

These concerns are serious, but not entirely new. Although few users may realize it, PostScript is in fact a programming language, and printing a PostScript document really means sending a PostScript program to the printer for execution. A faulty or malicious PostScript program can affect the printing of subsequent documents. Furthermore, if displayed on a workstation with a naive PostScript previewer, it may compromise system security. Automated Teller Machines (ATMs) provide a limited interface to banking systems with many more capabilities than the ATM supports; the user's button presses program the banking system through a constrained interface. Database queries can similarly be viewed as programs in the query language that are executed interpretively by the database management system.

People have, on occasion, attacked systems through these interfaces with varying degrees of success. Nevertheless, we have learned to live with the risks posed by PostScript printers, ATMs, and database management systems. The rest of this chapter addresses the question of how we can assure that the technology of software agents does not expose us to unanticipated or unacceptable risks.

Three Categories of Security Concerns

Since agents are only computer programs, it is reasonable to ask why security concerns about agents differ from security concerns about more common application programs, like word processors, spreadsheets, and schedulers. The answer resides in two observations: An agent infrastructure encourages the sharing of programs as well as data, and this sharing of programs greatly magnifies the threat of malicious programs like viruses. Also, an agent infrastructure may allow users to run their agents on machines they would not normally be able to access, and this access makes those machines and the data on them more vulnerable to attack.

In a computing system without agents, security mechanisms are usually meant to protect both the computing system and the other users and their data on that system from another user and his programs. Extending this goal to mobile agents requires protecting the rest of the world from the initiator and his agents, no matter which target machine the mobile agent runs on. In a safe agent system, however, it is also essential to protect the initiator and his agent from the rest of the world and to protect the initiator from his agent. These three security concerns, and some of the hazards that can occur in each area are summarized in List 17.1.

A. Dangers to the world from an agent/initiator

- "impostor" agents acting without proper initiator authorization
- malicious behavior by authorized agents (e.g., gaining access to host system and retrieving or modifying data without authorization)
- accidental errors in authorized agents

B. Dangers to agents/initiators from the world

- world corrupts agent, which continues to run but not as intended
- valid agent action is "replayed"
- world extracts private data from agent (e.g., credit card number)

C. Dangers to the initiator from an applet

- applet executing with initiator's authorization damages initiator's data
- applet discloses initiator's data to third party without authorization
- applet initiates unintended activities (e.g., packet echoing, outbound connections) without initiator's authorization

List 17.1: Some Possible Hazards from Agents.

We must protect the rest of the world from the agent and its initiator. Agents may try, either maliciously or accidentally, to retrieve of modify data they should not access. Agents may also use other resources, including computing cycles, and so degrade system operation for other users. The protection is obtained in two ways: A machine must be able to determine who an agent's initiator is, in order to determine what resources that agent is entitled to. And the agent's behavior must then be restricted to those resources.

The first category of hazards focuses on the risk to a computing system from programs and users, and the needed protection mechanisms are similar to those used in a distributed computing environment. An agent, however, must also protect its private data from the machine that it runs on. In a distributed environment, although the system typically has access to all of the data on the system, the system is trusted not to abuse that access. Can an agent trust the systems that it runs on in a similar way? For example, if an agent carries its initiator's credit card number to a target machine, can the target machine be trusted not to keep a copy of that credit card number? What prevents a target machine from replaying an agent, to repeat a valid action? An initiator might be happy to make one purchase, but not to make several. Finally, what prevents a target machine from corrupting an agent, to change its behavior so it does something that the initiator did not intend? The agent and its initiator must be protected from the rest of the world. The hazards in this second category arise because an agent needs to have authority delegated to it by its initiator. But this delegated authority must be used only as the initiator intends.

The third category of hazards applies to applets that a user imports to run on his own machine. These programs may have both desirable and undesirable behaviors. How can a program that should have access to a user's data be prevented from damaging that data? How can such a program be restricted from passing a user's data to third parties? Protection mechanisms that protect an initiator from his applets may also be used to protect a user from virus-infected programs. Mechanisms for providing this protection must be both safe and convenient.

These three categories of security concerns map nicely onto the two sorts of agents: mobile agents and applets. Both are meant to run with the initiator's privileges. Mobile agents are sent by the initiator to *someone else's* computer. Applets are programs that the initiator imports from another computer and intends to run on his machine. A safe environment for mobile agents requires extending basic safety mechanisms to a distributed world to protect the rest of the world from the mobile agent. More difficult to envision are the new mechanisms to protect mobile agents and their data from the rest of the world. Applets pose the third challenging problem: The initiator might want an applet to run on his machine like any of his other programs, unless the applet begins to do things it was not advertised to do. Detecting when this erroneous or malicious behavior begins is, in general, impossible.

Protection Concepts

The basic idea behind standard safety mechanisms is not to grant all programs complete access to the machine. For example, certain parts of memory may not be readable or writable. Or a microprocessor's instructions may be split into two sets, one accessible to all programs, the other only to programs running in *supervisor* mode (this mode is usually assigned to the operating system, the OS). These restrictions help limit the damage that a program can cause, either accidentally or maliciously, and help make the computing environment more robust. For example, a significant enhancement of Windows95 over Windows 3.1 and MS/DOS is that OS memory space is isolated from application program memory space. This reduces the likelihood of an application crashing the Windows95 machine. Windows NT also isolates applications from each other, limiting each application's ability to interfere with others; this makes the system even more stable. In a multi-user environment, with a file system shared among many users, the OS may provide mechanisms for controlling how different parts of the file system are accessible to each program.

These mechanisms enforce programs' *privileges* Privileges may define which files a program has access to and how it may access them, may control a program's use of resources like printers, tape drives, and CPU time, and may state whether a program is running in supervisor mode.

The principle of *Least Privilege* is fundamental to security. Any program should be granted only the smallest set of privileges that it should need in order to accomplish its task. Assigning a program this smallest set of privileges reduces the likelihood that malicious or accidental behavior will damage the computer system. Attempts to violate the assigned privilege will be detected and prevented.

How are privileges enforced? How can one run a program with a reduced set of privileges? A useful metaphor is the notion of a *safe box*, which is a constrained execution space from which a program cannot escape. The safe box limits the access a program has to the machine's file system and other resources.

One way to build a safe box is to run a program under an interpreter. Instead of counting on the machine to detect an illegal operation (i.e., one which violates the privileges granted to the program), the interpreter can monitor the execution. Interpreters provide finer control over the detection of privilege violations and more flexibility in the assignment of privileges. For example, consider an applet. If the applet were run directly on the machine by the initiator, the applet would inherit the initiator's privileges. Running the applet under an interpreter permits further reducing those privileges.

Unfortunately, programs run under an interpreter run more slowly. A clever compromise between flexible assignment of privileges and efficient execution is the *Virtual Machine* approach. In a virtual machine, all non-supervisor mode machine instructions run directly on the microprocessor. Supervisor mode instructions trigger an interrupt which is trapped by the virtual machine interpreter. This interpreter, which runs in supervisor mode, decides whether the program has the privilege to execute the instruction, and takes the appropriate action. Virtual machines introduce overhead only when interpreting supervisor mode instructions.

Another way to reduce the overhead introduced by an interpreter is to reduce the amount of program monitoring that needs to be done at runtime (dynamic analysis), by analyzing the program before it runs. This is called static analysis. If certain conditions can be proved never to occur, then the interpreter need not check those conditions, and the interpreter will run more efficiently. For example, if it is possible to determine that type violations do not occur, then the interpreter need not do runtime type checking.

Virtual machines can also be used to separate one program from another. This is one technique for isolating domains. A program runs in a domain. Each domain is linked to a user and inherits the privileges associated with that user.

How do programs in different domains communicate? Two domains may share a buffer. Or the operating system (or the safe box) can pass messages between programs, and will be responsible for protecting the privacy of those messages. If the programs are running on different machines, then the messages must be passed between machines.

Cryptography -- Privacy

How can the confidentiality of data be protected when it moves over public channels? One way is to encrypt the data. Encryption uses an encryption algorithm and a key to change some input, called plaintext, to some output, called ciphertext. The ciphertext looks like a random string of bits to observers who do not know the appropriate decryption algorithm and key. There are two classes of encryption/decryption algorithms: symmetric and asymmetric. Symmetric algorithms, like DES (the Digital Encryption Standard), use the same algorithm and key for encryption and decryption. Imagine that Alice and Bob want to communicate privately using DES. They would agree to use a single key K. If Alice wanted to send a message M to Bob, she would encrypt M using the DES algorithm initialized with key K, producing the ciphertext. Let's call this ciphertext DES(K,M). To anyone who does not know K, DES(K,M) appears to be a random string. However, Bob can decrypt the message by applying DES again to the ciphertext using key K. The result, DES(K,DES(K,M)), is the original plaintext, M. Bob can reply to Alice using DES with the same or another key they have agreed to use. Anybody who knows these keys can listen in on Alice and Bob's conversation, so these keys must be kept secret.

Symmetric encryption/decryption algorithms like DES use an already existing shared secret (the key K) to efficiently share new secrets (the message M). But, how can Alice and Bob choose a key if they do not already share some secret? This is called the key distribution problem. Until the early 1970's, when public key cryptography was invented, keys were distributed by some other means (e.g., a trusted courier).

Public key cryptography is also called asymmetric encryption, because different keys are used for encryption and decryption (i.e., the parties do not share a key). Instead, everyone has two keys, a public key (P) which is widely published, and a private (secret) key (S) which the owner keeps secret to himself. In a public key scheme like RSA, if Alice wants to send a private message M to Bob, she looks up Bob's *public* key, P_Bob, and encrypts M, using the public key algorithm RSA, to produce the ciphertext RSA(P_Bob,M). Bob can decrypt this ciphertext by applying RSA again, using his *private* key S_Bob. The result, RSA(S_Bob,RSA(P_Bob,M)), is M. Notice that when using RSA, unlike DES, Alice is able send Bob a private message, without sharing a secret with Bob.

There are two problems with asymmetric crypto-systems. The first is that they are relatively expensive to use. It is not practical to encrypt and decrypt long messages using public key cryptography (one source estimates that RSA, the only widely used public key algorithm is, at its fastest, about 1000 times slower than DES, a widely-used symmetric key algorithm). Therefore, public key cryptography is usually used by two parties to choose a new key that will be used later to encrypt messages using some efficient symmetric crypto-system. The process of key distribution and private communication is illustrated in Figure 17.2:

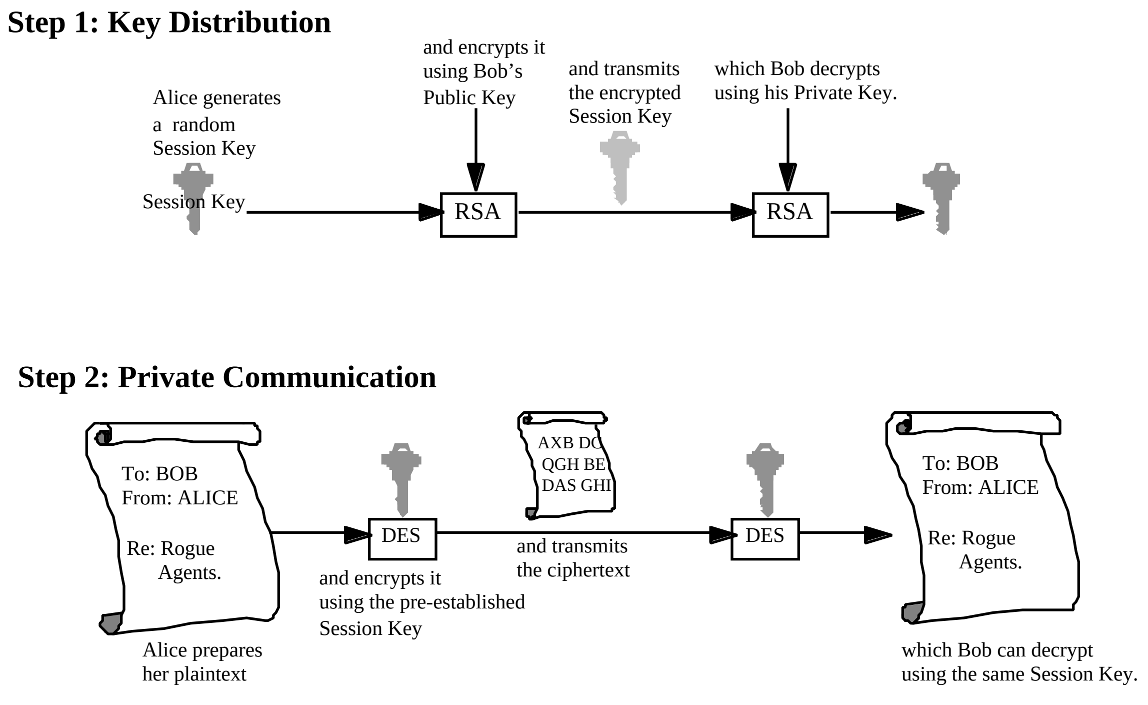
Step 1: Key Distribution
Alice generates a random Session Key
Session Key
and encrypts it using Bob's Public Key
RSA
and transmits the encrypted Session Key
which Bob decrypts using his Private Key.
RSA
Step 2: Private Communication
To: BOB
From: ALICE
Re: Rogue Agents.
Alice prepares her plaintext
DES
and encrypts it using the pre-established Session Key
AXB DC
QGH BE
DAS GHI
and transmits the ciphertext
DES
To: BOB
From: ALICE
Re: Rogue Agents.
which Bob can decrypt using the same Session Key.

Figure 17.2: Key Distribution and Private Communication.

The second problem in asymmetric crypto-systems is the distribution of public keys. How does Alice determine Bob's public key? She must have access to a directory mapping people to keys. This directory must be trusted; if the mapping is not accurate, Alice will end up sharing keys with the wrong party.

Cryptography -- Authentication

Public key crypto-systems can also be used to identify the sender of a message. This is akin to having the sender sign the message and is called a digital signature. RSA can be used to produce digital signatures as well as for encryption, because the public and private keys are interchangeable. The digital signatures can be used to authenticate the sender.

If Alice wants to package a message M so it can be verified to be from her, she would encrypt M using RSA and her private key, S_Alice. The resulting ciphertext, RSA(S_Alice,M), is a digital signature that could only have been produced by Alice, since only she knows the secret key S_Alice. The signed package would include both M and RSA(S_Alice,M). If Bob wants to verify that M was really written by Alice, he "unsigns" the associated digital signature using Alice's public key, P_Alice. The result, RSA(P_Alice,RSA(S_Alice,M)), is M. If someone else tried to forge Alice's digital signature using some other private key, the unsigned message would not match M.

Signatures also provide a means to detect whether a message has been altered after it is signed, since the signature verification will fail if the package is modified. If a conventional checksum were used to detect such modifications, a malicious intruder could alter both the message and the checksum correspondingly, so that the altered checksum would match the altered message. Since the digital signature employs the private key (presumably not available to the intruder), even if the intruder alters the message, he cannot alter the checksum correspondingly. The signature thus forms a *cryptographic checksum*.

Since public key crypto-systems are expensive to use, digital signatures are not usually applied to an entire message. Rather, the message is shortened using a *cryptographic hash function*, like SHA. SHA is an algorithm that takes an arbitrarily long message and produces a short, fixed-size result called a hash. Even small changes in the long message result in apparently random changes in the output, so it is virtually impossible to find another input that produces the same hash. So signing the hash is nearly as safe as signing the original message, but it is much less expensive.

No crypto-system is perfectly secure. Clever attacks exist against some systems, and brute force attacks (try every possible key) exist against all. Secure crypto-systems are those in which brute force attacks are too expensive and for which clever attacks are not known, despite the crypto-system having been carefully analyzed and used for a while.

Combining Protected Domains and Cryptography

Cryptography can help solve two problems associated with domains. Encryption techniques can be used to keep the communication between domains private, especially if the domains are on different machines communicating over a public network. Digital signatures can be used to authenticate the sender of messages between domains and also to authenticate the link between a domain and a user.

Application to Agents

Protection concepts like safe boxes and cryptography are part of a secure agent environment. List 17.3 presents techniques that can be used to reduce the security vulnerabilities of agent systems.

A. Protect the world (remote sites) from an agent

• execute agent in "safe box"
• authenticate arriving agents
• verify agent code (static analysis)
• execute agent code interpretively
• check agent code at run-time (dynamic analysis)
• audit agent actions (as a deterrent)
• employ access control mechanisms

B. Protect the agent/initiator from the world

• place checksums on agent code/data
• encrypt checksums on agent code/data
• employ anti-replay mechanisms
• employ anti-copy mechanisms
• control copying

C. Protect the initiator from an applet

• execute applet in "safe box"
• authenticate arriving applet
• verify applet code (static analysis)
• execution applet code interpretively
• check applet code at run-time (dynamic analysis)
• audit applet actions (as a deterrent)
• employ access control mechanisms

List 17.3: Techniques Addressing Agent Hazards.

The techniques in (A) for protecting the world from the agent and in (C) for protecting the initiator from an applet are identical. However, (C) is a more difficult problem. If an initiator sends an agent to some target machine, it may be perfectly reasonable to let that agent run at some minimum privilege on that machine. However, when an initiator imports and runs an applet, he may expect it to run with his privileges, except if it begins to abuse those privileges. Safe boxes can be used to solve parts of that problem, because they provide great flexibility in the assignment of privileges to interpreted programs. The privilege set must be carefully chosen to balance security and convenience. One could imagine running an applet under the rule that any file access must be validated by the initiator. However, that rule will probably generate so many queries for the initiator that the initiator will mindlessly validate most of them and violate the security they were intended to provide (or else give up trying to execute the applet and forego its benefits). More difficult to deal with is the fact that a combination of secure actions may be insecure. For example, it may be proper for an applet both to open a channel to another machine and to process some of the initiator's private data. But it may not be proper for the applet to send the private data along that channel.

An agent's privileges may be related to its initiator's privileges. The link between agent and initiator may be authenticated using digital signatures. The initiator signs the entire mobile agent package, so that the target machine can confirm who the initiator is. Acting simultaneously as an encrypted checksum, this signature can also also reveal any tampering of the agent in transit.

Authentication may not be necessary to determine the privileges of some mobile agents, because a target system may choose to grant all mobile agents only some minimum privilege. A free Internet search service may function in this way. However, authentication may still be necessary in order to confirm to whom to return the results.

Agents may use encryption to keep communications private. If a mobile agent wants to send results from the target system back to the initiator in a private way, the agent could encrypt the data using the initiator's public key. Since anything encrypted with a public key can only be decrypted using the corresponding private key, which only the initiator knows, the message will remain private until the initiator decrypts it. This approach implies that the mobile agent knows the initiator's public key. This is not a problem, since the public key is published.

This sort of encryption protects the data from being snooped on in transit between the target and the initiator's machines. The target machine, however, has access to the unencrypted data. This may not be a problem, since the target machine was probably the source of that data.

Encryption does not solve all privacy concerns, especially those relating to (B) protecting an agent/initiator from the world. For example, consider the credit card example described earlier. If a mobile agent is going shopping, it might carry its initiator's credit card number. If the target machine keeps a copy of that credit card number, the target machine could use it for its own purchases.

There are two solutions to this sort of problem. One is to treat the target machine in much the same way that we treat a vendor in a face to face credit card transaction: The vendor is trusted not to keep an extra print of the credit card. Another solution is to say that credit card numbers are not the point, the signature is. The agent should really carry a message from the initiator validating use of the credit card for only the intended transaction. But can the target machine be prevented from repeating that transaction?

The fundamental unsolved problem here is how an initiator can delegate authority to an agent without letting the target system abuse that authority. For example, how can a mobile agent spawn another agent for another target machine and authenticate the spawned agent for its initiator without compromising the initiator's private key? Generating the signature requires the initiator's private key. But if the private key is sent along with the mobile agent to the target machine, its privacy can no longer be guaranteed.

Evolving Network Infrastructure

Both applets and mobile agents need the ability to move from one machine to another, whether it is at the request of the user loading the applet or initiated by the mobile agent. There has been much talk about the benefits and power of the Internet, but very little about its deficiencies or about other networks that exist today.

An important distinction needs to be drawn: the Internet (capital I) is composed of many regional and backbone internets (lower case I) which are interconnected to share data traffic -- no one "owns" the Internet, but many companies and organizations own the regional and backbone internets. A message sent across the Internet from one machine to another may cross many different internets to reach its destination (in fact, parts of the same message may take different paths). The routing of messages from source to destination is handled automatically by the network infrastructure. For a user concerned about the confidentiality and integrity of his message, but lacking control over its routing, the questions of where the message travels and consequently who might see or alter it en route are natural concerns. Many of these concerns arise due to deficiencies in the Internet Protocol deployed today (TCP/IP v4).

In addition to the Internet, many private networks exist today. These private networks may use internet protocols, yet be completely separate from the Internet, or they may employ cryptography to create smaller *virtual networks* within the Internet. While private networks help address some data security concerns, they do not address all concerns -- simply having a private network does not mean you have a safe and secure network.

Security must be carefully considered and implemented for both private networks and the Internet to combat the problems discussed earlier. Deficiencies in the design of TCP/IPv4 require the use of cryptographic algorithms and secure transport protocols to preserve confidentiality and integrity in present internet communications.

These deficiencies and others stimulated a major effort to develop a new generation of Internet protocols; implementations of the successor to IPv4, known as IP version 6 (IPv6) or IP Next Generation (IPng), are underway. IPv6 incorporates security mechanisms at the transport layer and below, reducing substantially the need for additional security at higher protocol layers, but its deployment throughout the Internet will only occur gradually over the next five to ten years.

Cryptographic Algorithms

The need for cryptographic algorithms both for encryption and authentication has arisen in many contexts, but which algorithms are used widely today? There are four areas to consider: key distribution, ciphers, signatures, and hashing. Some of these, such as public key ciphers and signatures, may overlap or employ the same algorithms for multiple purposes. This section provides a brief overview of the primary cryptographic algorithms used today. Readers should check the suggested reading list for details on these algorithms and others.

Key distribution is a crucial problem in the use of cryptography -- the most powerful crypto-systems will fail if their keys are compromised. Asymmetric, or public key, crypto-systems permit open distribution of public keys, but they present another problem: How can one be certain that the public key for Alice (P_Alice) is *really* Alice's public key and not someone elses? Alice may *personally* present her public key to some trusted, well-known authority (perhaps a government or independent agency of some sort), who could in turn sign the key. If everyone signs in person with the trusted authority, then one need know only the public key for the trusted authority to verify the integrity of any other public key. This system is commonly referred to as a *hierarchy of trust* since it may be abstracted to more than one level of signing described here. The signed public keys, or *certificates*, are signed by the trusted, well-known authorities, or *certifying authorities*.

Symmetric crypto-systems cannot use certificates to distribute keys in this way, because the keys would lose their value if exposed to public view. Instead, a hierarchy of keys can be established so that lower level keys (e.g. session keys) are encrypted using higher level keys. Only the highest level keys then need be distributed manually, but compromise of a higher level key implies compromise of all lower level keys that have been encrypted under it. Another alternative for establishing a shared session key for a symmetric crypto-system that requires no separate communication channel is the Diffie-Hellman (DH) key exchange algorithm. This algorithm cleverly uses the difficulty of calculating discrete logarithms in a finite field, relative to the ease of exponentiation in the same field, to allow two parties to agree on a secret number (the key). It does not, however, solve the problem of authenticating the parties to each other. So, you and a correspondent can agree on a key to use in your symmetric crypto-system, but you need another method to assure yourself of the identify of your correspondent.

Once keys have been distributed, ciphers can be used. Most encryption employs symmetric crypto-systems because of the high computational demands imposed by the exponentiation operations required by current asymmetric crypto-systems. Common symmetric crypto-systems include DES (the Data Encryption Standard), RC4, and IDEA. DES was adopted in the late 1970's by the US government as a Federal Information Processing Standard (FIPS) for handling unclassified data. It uses a 56-bit key and is widely regarded as secure but possibly not unbreakable. It is noteworthy that it has withstood all public attacks for more than 20 years, though the computing power available to attackers has advanced substantially during that period, and 56-bit keys are now considered too short by some. "Triple DES", in which two 56-bit keys are used in conjunction with three applications of the DES algorithm, represents an effort to satisfy those concerns without abandoning the basic DES algorithm. RC4 is a proprietary, variable length key algorithm of RSA Data Security, Inc., which claims that this algorithm is as strong as DES (given the same size key) and is ten times faster. A version using a 40-bit has been licensed for export by the U.S. government, but such short keys are increasingly vulnerable to brute-force attacks. The algorithm has not been made public. The International Data Encryption Algorithm (IDEA) is a public (though patented) algorithm that uses 128-bit keys and is used for data encryption by the widely distributed Pretty Good Privacy (PGP) software.

The only asymmetric crypto-system in wide use today is RSA. RSA's security is based on the assumption that factoring large numbers is extremely difficult. By the nature of the algorithm, RSA public and private keys can be of any length (typically, they are from 512 to 1024 bits) and are created by knowing the product of two large prime numbers.

For many agent applications, the confidentiality that ciphers can provide may be less important than the assurance of data integrity and authenticity that can be established by hash algorithms and digital signatures. RSA has become the De facto standard for digital signatures in both the US and international communities. While RSA is very attractive because it serves as both an cipher and a digital signature, its use in the US must be licensed under the patents on the RSA algorithm. Products that use RSA for digital signatures may also face export restrictions because of the ability of RSA to serve as a cipher algorithm. An alternative is the Digital Signature Standard (DSS), proposed by NIST for the signing of unclassified documents. DSS uses the Digital Signature Algorithm (DSA), and the Secure Hash Algorithm (SHA), to sign data. These algorithms and standards were designed so that they provide signatures but cannot easily be converted to use as ciphers, so that products incorporating them should not face the export barriers that ciphers do. In addition to SHA, another commonly used hash algorithm is MD5, developed by Ron Rivest. MD5 produces a 128-bit digest from an arbitrary length message, while SHA produces a 160-bit output.

FORTEZZA is the future of US government cryptography for unclassified data handling. Built with the components developed for the Clipper project, FORTEZZA is a tamper resistant PC Card that supplies a complete cryptographic engine keyed to an individual via an access PIN -- possession of both the card and the PIN is necessary for proper operation. The card is capable of doing bulk encryption with the classified SKIPJACK algorithm, it can sign data with DSS, and handles key distribution via signed X.509 certificates.

Secure Transport Protocols

Cryptography is nothing more than a tool or building block for safe and secure communications. Despite the abundance and availability of ciphers, digital signatures, hash algorithms, very few are used in deployed protocols. With the advent of electronic commerce and mobile agent technologies, it has become essential that safe and secure communications become widely deployed. While a host of small, proprietary systems exist that may be secure, only a handful of widely adopted standards have recently become available.

Netscape's SSL (Secure Socket Layer) and Microsoft's extension of SSL, PCT (Privacy Communication Technology) are gaining widespread acceptance in the World Wide Web communities. Both SSL and PCT were designed to be used with any socket-based communication services (Telnet, FTP, HTTP, etc.) and therefore have direct applicability to both applets and mobile agents. While neither is currently being used to provide security for applets or mobile agents, it is an inevitable progression as security is refined for both the applet and mobile environments. The cryptographic components that can be used in SSL and PCT are summarized in Table 17.4 below.

General Magic's Telescript currently does not use either SSL or PCT, but instead implements a system based upon: RC4 with a 40-bit key for bulk encryption; Diffie-Hellman key exchange of 512-bits for session keys; and RSA with 512-bit keys for authentication between personal communication devices and the cloud (network infrastructure).

Keying:	RSA, DH, FORTEZZA
Hash:	MD5, SHA, DES (Davies-Meyer variant in PCT only)
Cipher:	RC2, RC4, IDEA, DES, 2-DES (PCT only), 3-DES, FORTEZZA
Signature:	RSA/MD5, RSA/SHA, DSA/SHA

Table 17.4: Cryptographic components that can be used in SSL and PCT.

Applet Languages

Many different languages could be used to create applet technologies, but some are better suited than others. It would be possible to create applets with UNIX C-shell scripts, for example, but these would be neither widely portable nor able to accomplish much due to the limited nature of the scripting language. What follows is a look at some of the more powerful languages that could be used to create applet technologies. A comparison of the different applet and mobile agent environments is presented below in Table 17.5.

Tcl

The Tcl scripting language has many attractive strengths which are also its weaknesses. Tcl is an interpreted language, so it is relatively invulnerable to machine-code dependent flaws and the interpreter can help enforce system safety. Unlike Java, applets are not stored in an intermediate byte code format, but as the original source code. While this is advantageous from a security perspective because it allows the user to read and verify the code, it incurs speed penalties and sacrifices source code privacy. The richness of the Tcl scripting language -- vanilla Tcl has direct access to all resources the user owns -- allows designers to build powerful applets, but it is the antithesis of execution within a safe box. A restriction of Tcl, called Safe-Tcl, has been introduced that removes key system calls to improve security - at the expense of some of the language's power. Safe-Tcl's rich language and operating environment allows for rapid prototyping of reasonably powerful applets.

Java

Sun's Java is a powerful applet language that is rapidly being accepted as an industry standard and integrated into many World Wide Web browsers. Java is a platform-independent, byte-code interpreted language; this discussion only covers the applet version of Java, and not Java as a general application level programming language. Java applets are downloaded from the World Wide Web and interpreted as executable programs by the Java environment. While this appears to be the same as Tcl/Tk, it differs because the code downloaded has been compiled from its native C++ like language into byte code. This offers Java applet programmers some degree of privacy since it is very difficult to regenerate the original native C++ like source code from the byte code. While the user loses the ability to examine the downloaded code for security problems prior to execution, he gains some assurances that the applet is safe by the constraints imposed by the Java environment: pointers do not exist, the filesystem may only be accessed if explicitly authorized by the user via a popup window, network sockets can only be opened to the applet's home machine, and both static and dynamic security checks are performed. While these constraints may not appear important, they deny a malicious applet author the necessary tools to write a worm, a virus, and most (but not all) Trojan horses. These initial design decisions and wide deployment of the language make Java a better base for secure applet development than either Tcl or Safe-Tcl.

Visual Basic / OLE plug-ins

Applets from Microsoft's Visual Basic / OLE plug-in use digital signatures to authenticate the originator of the applet. While this mechanism, properly used, can offer a level of trust and authentication unmatched by the other applet environments, other flaws in the applet environment design may undermine this security. For example, OLE plug-ins are native machine code with full access to all running applications, the operating system, and even the underlying hardware. A virus or worm could easily be implemented and spread by an applet. While the authentication mechanism is worthwhile, it cannot provide the scope of protection that users need to protect them from errant or malicious applets.

Mobile Languages

There are far fewer languages that can be used to create mobile agent environments than their applet counterparts. This is due, in part, to the necessity of an entire environment and support system that applets neither require nor need. A comparison of the different applet and mobile agent environments is presented below in Table 17.5.

Telescript

General Magic's Telescript offers the only real mobile agent language today. While the environment is still undergoing development, Telescript clouds exist today in widely deployed networks like AT&T's PersonalLink service. Like Java, the Telescript language is a byte-code interpreted language that closely resembles C++. The security focus of Java is more narrow than that of Telescript: In Telescript, mobile agents are cryptographically signed for auditing and credential purposes; credentials can be created and limited by the administrative domain the agent is visiting, the machine the agent is visiting, by the agent programmer, or by the agent itself at runtime; mobile agents are transferred from machine to machine with their entire state, via cryptographically secure channels; and a safe box exists to protect the host machine from the mobile agent. The only aspect of safety and security not explicitly addressed in Telescript is the protection of the mobile agent from a hostile environment, but this can be partially accomplished via other methods within the agent language.

	Safe Tcl	Java	VB/OLE	Telescript
Cryptographic Checksums:	N	N	Y	Y
Access Control:	Y	Y	N	Y
Resource Limitations:	N	N	N	Y
Auditing:	N	N	N	Y
Accounting:	N	N	N	Y
Static Verification:	N	Y	N	N
Dynamic Verification:	Y	Y	N	Y
Type Restrictions:	N	Y	N	Y
Interpreter:	Y	Y	N	Y
Anti-replay:	N	N	N	N

Table 17.5: Security Aspects of Applet and Mobile Agent Environments.

Summary and Projections

Along with its benefits, agent technology poses dangers. Many of the dangers are similar to those a user encounters when he or she acquires new software from a bulletin board or a shrink-wrapped package, installs it, and executes it. The new software may malfunction; it may even contain a Trojan horse or virus. The cautious user (a small fraction of all users, one suspects) backs up crucial files and runs a virus scanner on the new software before executing it, but even this strategy does not eliminate the possibility of unrecoverable damage. Few computing platforms at present provide an adequate "safe box" to which new software can be confined. Still, installing a new piece of software is not a daily activity for most users, so the risk may be balanced by the infrequency of the action.

The marriage of World-Wide-Web browsers with languages that make the downloading and execution of new software virtually invisible to the user changes the balance substantially. Agent technology, which implies a substantial rate both of downloading and uploading programs, amplifies the risk. In 1987, it took significant programming skill and knowledge of network infrastructure to craft a program that could move from site to site in the Internet. In 1997, an infrastructure designed so that casual users can build such programs will likely be in place. It is crucial that this infrastructure be able to limit the damage erroneous or malicious agents can cause.

Happily, some of the developers of the new infrastructure have recognized the dangers and are working to limit them. Needed are computing platforms that can confine programs to particular domains, a cryptographic infrastructure that can be used to authenticate agents and their initiators and to assure the integrity of agent code and data, and agent programming languages (including run-time support) that permit agents to perform efficiently and effectively but limit their potential for ill effects. The potential profits from electronic commerce on the Internet are providing a powerful incentive for the development of some parts of the cryptographic infrastructure.

Defining security policies that permit useful functions while prohibiting undesired ones and developing programming languages that can enforce such policies is a major challenge for the users and developers of agent technology. Operations that can do no harm may not be able to do much good, either. It may be possible to strike different balances of security and function in different environments. For example, a company operating a private internet might be able to limit the set of agents introduced into its systems and therefore be willing to grant those agents more privileges than it would if it were to allow arbitrary agents from the public Internet to operate on its systems.

Because this area of technology is advancing so rapidly, we provide pointers to organizations and Web pages where you can probe for the latest developments. At present, Telescript provides the most complete environment for programming mobile agents, and it addresses many important security concerns. The Java language is already a major force in the applet marketplace, and several independent efforts are underway to create a mobile-agent environment and infrastructure for it. If these succeed, Java will probably also be a major factor in agent technology. Separately, researchers at Dartmouth University have extended the Tcl/Tk language, environment, and infrastructure to produce mobile agents. And of course new efforts are likely.

Will the technological vehicles of applets and mobile agents contain compartments where bombs can be easily hidden? We are optimistic that the proliferation of efforts to develop agent technology, the potential size of losses due to fraud, and the increasing public awareness information security issues, will combine to produce, eventually, agent technology that can be used safely. But during this period of turbulent development and competition, users had best inspect their vehicles before driving them.

Reference Material / Further Reading

This section lists references to books and World Wide Web sites relevant to this chapter.

General Reading & References

Cheswick, William R. and Bellovin, Steven M., *Firewalls and Internet Security: Repelling the Wily Hacker*, Addison-Wesley, 1994.

Neumann, Peter G., *Computer Related Risks*, Addison-Wesley, 1995.

Russell, Deborah & Gangemi, G. T., *Computer Security Basics*, O'Reilly & Associates, 1991.

Schneier, Bruce, *Applied Cryptography: Protocols, Algorithms, and Source Code in C,* Second Edition, John Wiley & Sons, 1996.

Cryptography

RSA Data Security, Inc.'s Home Page
http://www.rsa.com/

NIST Federal Information Processing Standards (DES, SHS/SHA, DSS/DSA)
http://csrc.ncsl.nist.gov/fips/

IP Next Generation (IPng)

IP Next Generation (IPng)
http://playground.sun.com/pub/ipng/html/ipng-main.html

IPng Specifications
http://playground.sun.com/pub/ipng/html/specs/specifications.html

Secure Transport Protocols

SSL Version 3.0
http://home.mcom.com/newsref/std/SSL.html

Microsoft Corporation's PCT Protocol
http://www.microsoft.com/windows/ie/pct.htm

Tcl

Tcl/Tk Project At Sun Microsystems Laboratories
http://www.sunlabs.com:80/research/tcl/

Transportable agents at Dartmouth College (Mobile Tcl)
http://www.cs.dartmouth.edu/~agent/

Java

Java: Programming for the Internet
http://java.sun.com/

Frequently Asked Questions - Applet Security
http://java.sun.com/sfaq/

Low Level Security in Java
http://java.sun.com/sfaq/verifier.html

HotJava: The Security Story
http://java.sun.com/1.0alpha3/doc/security/security.html

Visual Basic / OLE

Microsoft Ships Internet Explorer 2.0
http://www.microsoft.com/windows/pr/nov2795.htm

Telescript

Telescript
http://www.genmagic.com/Telescript/index.html

Safety and Security in Telescript
http://cnn.genmagic.com/Telescript/TDE/security.html