CptS 464/564
Lecture 26

17th April 2003

Good job on project 3. I think everyone did quite well. The TC assignments went out with today's courier – for some reason my attempt to send them yesterday failed. Matthew and Rick – yours will go out tomorrow.

My intention is to hold the course evaluations at the end of class on Tuesday, April 29th. I'm contemplating a trip to TC on Thursday, April 24th and showing a video that day. Depending on the length of the video I might move the course evaluations to the 24th instead.

Don't forget to retrieve the third homework from the assignments directory.

Security continued

We turn now to mechanisms and techniques that can be used to provide the security properties we talked about last time, beginning with *confidentiality:* protection of information from unauthorized disclosure. The mechanisms mentioned include physical isolation, encryption and authentication and authorization of access.

Threat: an unauthorized person obtains the information. How much damage can be caused by that? Example: an unauthorized person obtains your eecs login password. Potential damage: theft of your work, destruction of your work, perform embarassing or illegal acts; install software that will capture other information that you want to keep private. Turn-key spyware exists and is available to anyone with a search engine to exploit many of the vulnerabilities below.

Vulnerabilities: how could someone obtain your password? Find it written down; get you to tell it to them or watch you using it; get a sysadmin to change your password to something that they know; get it from sniffing the network when you perform a remote login; get it from sniffing the network when you retrieve your POP3 mail; take over a machine that allows logins or mail retrieval; install a keyboard sniffer on your machine; read it from a place you've recorded it (paper or electronic).

Network sniffing is perhaps the easiest of the these vulnerabilities to exploit. For broadcast ethernet it is essentially trivial: any computer attached to the network has the ability to sniff all traffic on the network. For switched ethernet it is somewhat harder: may

require access to devices that are often kept in locked rooms, or the ability to read signals from wires in non-standard ways. Switched fiber-optic networks would be yet harder to sniff. However, it seems prudent to assume that any packet sent on a network may fall into the wrong hands. Encryption is used to prevent the information the packet contains from also falling into the wrong hands. [Example: from the 1960's through the 1990's the U.S. used its submarine fleet to install taps on undersea cables carrying the Soviet Union's military command and control message traffic.]

Basic philosophy of encryption: rewrite a message in such a way that its information content is only accessible to someone who holds a *key:* a critical piece of information (or physical device) without which the message remains gibberish.

Encryption changes the vulnerabilities and the threats. How they change depends on the encryption technology used. Potential vulnerabilities: the messages can be recovered without knowing the key; the key can be recovered from the message; the key can be stolen (how?); ...

Two forms of encryption

- Symmetric-key encryption: encryptors and decryptors use the same key which must remain a secret. Vulnerability: "if something is know by more than one person it is no longer a secret".
- Public-key encryption: encryption and decryption is performed using different keys, one of which is public (i.e. not secret). In theory, the secret key need only be known by one person.

Approaches to encryption in distributed systems

- *Physical circuit encryption* – messages between adjacent routers are encrypted. Messages are decrypted and re-encrypted for the next link along the path. Often peformed by encryption hardware at the phsyical or data link layer. Three problems: a lot of expensive hardware; time taken to decrypt and re-encrypt; a new set of vulnerabilities – each router along the path sees the unencrypted data.
- *Virtual circuit encryption* or *end-to-end encryption*; suppose instead we encrypt the message at the source, before adding the piece of the message that says where it is going. Now the message can travel through the network remaining encrypted the whole way, finally being decrypted at the destination. VC encryption can be done using software at the Application or Transport layer, or by hardware devices. Vulnerabilities: fewer places where the message is exposed; fewer keys that have to be kept secure; secure key distribution *may* be more difficult; header information is exposed everywhere enabling a traffic-analysis threat.

Confidentiality of information in storage is a somewhat different problem from confidentiality of messages. The big difference is that encryption and decryption are widely

separated in time, so retention of keys becomes a much bigger problem for encrypted information storage. (There is an availability vulnerability if keys are forgotten or destroyed. An attacker also may have a much longer time to try to discover the key for stored information.)

Integrity

Protection of information from unauthorized (or undetected) change; mechanisms – cryptographic checksums, redundancy, isolation; authentication and authorization of access. Cryptography also supplies a hint about how to assure ourselves that a message or stored data has not been tampered with. If the information is encrypted, we would expect that change to even a single bit of the encrypted form would entirely destroy the sense of the encrypted information.

It turns out that integrity with confidentiality is possible using a technique called cryptographic checksums: idea compute a relatively short *checksum* for a message (or stored data item) using a technique such that

- given a checksum and a message it is computationally hard to find another message with the same checksum
- it is hard to find two messages with the same checksum (less important in practice but potentially an indicator of the cryptographic quality of the checksum)

Vulnerability: an attacker can change the message *and the checksum too! – more later*

Another approach to integrity, applicable in some situations: ensure at the lowest levels that the data "makes sense". Often used in database systems in the form of integrity constraints: "if a name is in the database there is a social security number associated with it". This is a very weak form of integrity, but also quite important to avoid triggering vulnerabilities due to program errors.

What is "authorized"?

Authorized includes the notion that a particular person or computational entity (or class of people or entities) is permitted by policy to perform some action. How can such a policy be enforced in a system? We need two things:

- a mechanism that can prevent an entity from taking an unauthorized action (an *authorization* mechanism)
- more importantly, some way of accurately determining the identity of entity attempting to take the action (an *authentication* mechanism)

Techniques for authentication include: what you know(passwords, cryptographic keys), what you have(physical keys, digital keyfobs), or what you are (fingerprints, voiceprints, retinal prints). Even with modest security requirements, so called *two-factor authentication* may be employed: a physical key and a password; a keyfob and a password;... Note the increase in workload for administration.

Back to our concern for integrity: to keep my message and its checksum from both being changed by someone else I can *digitally sign* the message which amounts to encrypting the checksum with a key that only I know.

Chapter 17 and the networking textbooks used in CptS 455 and 555 have many more details on how cryptography is used for confidentiality and integrity, including how authentication can be performed in a distributed system.

Additional mechanisms for security

HW protection mechanisms: priviledged instructions and priviledged operating modes: in priviledged mode the processor executes only programs that have been certified as trustworthy. In unpriviledged mode programs are limited as to the damage they can do (again, in theory!)

Operating systems use the HW mechanisms to create *reference monitors* which check that the entity performing every operation is authorized to perform it. Example:

- in order to write data into a file it must first be opened for writing (every write() operation is checked for this property). In order to open the file for writing a the *file permissions* must indicate that the entity is authorized.
- Vulnerabilities: the file permissions may be modified; permission checking might be bypassed; incorrect authentication.

Expressing access control policies

One general way to express an access control policy is as a matrix:

	Alice	Bob	Carol	Duane
file1	change perms/read/write/execute	x	x	r
file2	r	rw	cr	r
dir1	w	w	w	cwr
file3	append	a	a	r/truncate

In such a matrix, if we read *across a row* we are looking at what is called an *access control list.* Access control lists are typically stored with the object that they protect (the files and dirs in the example). When an entity attempts an operation, the reference monitor checks to see that the entity's entry in the ACL includes that operation.

If we read *down a column* we have what is called a *capability.* Capabilities are handled and stored by the entities (not the protected objects). To perform an operation, an entity

presents the capability along with the request to perform the operation. The reference monitor checks the validity of the capability and checks to see whether the permissions it grants encompass the operation requested. It has to:

- authenticate the entity presenting the capability (we don't want Carol to be able to exercise Duane's priviledges because she was able to get a copy of Duane's capability
- ensure that the capability was created in an authorized way (i.e. the capability's creator is authenticated and had permission to create the capability).

Capabilities are of some interest for distributed systems because they offer a way to allow entities to access objects at sites that have never previously heard of the entities. How?

Randomized algorithms

The homework asks you to investigate the behavior of a randomized algorithm for information propagation in distributed systems. The particular algorithm there is what is known as an epidemic algorithm – it behaves like the spread of an infection in a biological population. It is also known as rumor-spreading.

Randomized algorithms have a number of nice properties concerning security and redundancy that make them appealing in some situations. The way they do it is they make propagation unpredictable and uncorrelated with any other property of the system, thus making it hard for an attacker or even a systematic failure to completely block operation of the system.

In the problem I ask you to look at two subtly different approaches to rumor spreading and simulate the behavior of each. The simulation proceeds in rounds. In each round, each node randomly chooses another node and either "pushes" its own value to the other node or "pulls" the other node's value to itself.

These algorithms are sometimes harder to simulate than to implement for real. The difficulty is in avoiding artifacts introduced by the simulation. In actual operation, these algorithms simply periodically contact another node. There is no worry about the order in which contact occurs or ensuring that contacts occur strictly in rounds.

For simulation though, we want the every node to choose a partner in each round and we'd like the order that they choose not to affect the results. Therefore, at the beginning of each round copy the node state vector and have all the nodes work off the previous round's results in computing this round's results.

Next time: more about randomization in distributed systems.