Security Engineering (4)

Email: christian.urban at kcl.ac.uk
Office: N7.07 (North Wing, Bush House)
Slides: KEATS (also home work is there)

last week: buffer overflow attacks

- required some cheating on modern OS
- the main point: no cheating in practice

last week: buffer overflow attacks

- required some cheating on modern OS
- the main point: no cheating in practice
- one class of attacks not mentioned last week

Format String Vulnerability

string is nowhere used:

```
#include<stdio.h>
#include<string.h>

// a program that ”just” prints the argument
// on the command line

int main(int argc, char **argv)
{
   char *string = ”This is a secret string\n”;
   printf(argv[1]);
}
```

this vulnerability can be used to read out the stack and even modify it

Case-In-Point: Android

- a list of common Android vulnerabilities (5 BOAs out of 35 vulnerabilities; all from 2013 and later):

  http://androidvulnerabilities.org/

- a paper that attempts to measure the security of Android phones:

  > *"We find that on average 87.7% of Android devices are exposed to at least one of 11 known critical vulnerabilities..."*

https://www.cl.cam.ac.uk/~drt24/papers/spsm-scoring.pdf

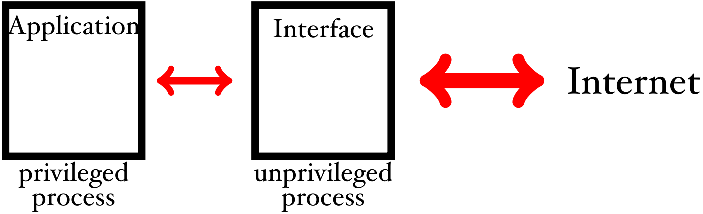
Survey

Two General Counter Measures against BOAs etc

Both try to reduce the attack surface (trusted computing base):

- **unikernels** – the idea is to not have an operating system at all
- all functionality of the server is implemented in a single, stand-alone program
- all functionality an operating system would normally provide (network stack, file system) is available through libraries
- the best known unikernel is MirageOS using Ocaml (`https://mirage.io`)

Network Applications: Privilege Separation



- the idea is make the attack surface smaller and mitigate the consequences of an attack

Access Control in Unix

- access control provided by the OS
- authenticate principals
- mediate access to files, ports, processes etc according to roles (user ids)
- roles get attached with privileges (some special roles: root)

principle of least privilege:
users and programs should only have as much privilege as they need to accomplish a task

Access Control in Unix (2)

- privileges are specified by file access permissions ("everything is a file")
- there are 9 (plus 2) bits that specify the permissions of a file

```
- r--rw-rwx bob staff file
```

directory user group other

Unix-Style Access Control

- Q: "I am using Windows. Why should I care?"
  A: In Windows you have similar AC:

  administrators group
  (has complete control over the machine)
  authenticated users
  server operators
  power users
  network configuration operators

- Modern versions of Windows have more fine-grained AC than Unix; they do not have a setuid bit, but have `runas` (asks for a password).

Unix-Style Access Control

- Q: "I am using Windows. Why should I care?"
  A: In Windows you have similar AC:

  administrators group
      (has complete control over the machine)
  authenticated users
  server operators
  power users
  network configuration operators

- Modern versions of Windows have more fine-grained AC than Unix; they do not have a setuid bit, but have `runas` (asks for a password).

- OS-provided access control can **add** to your security. (defence in depth)

Weaknesses of Unix AC

Not just restricted to Unix:

- if you have too many roles (i.e. too finegrained AC), then hierarchy is too complex
  you invite situations like...let's be root

- you can still abuse the system...

A “Cron”-Attack

The idea is to trick a privileged person to do something on your behalf:

- root:
  `rm /tmp/*/*`

A “Cron”-Attack

The idea is to trick a privileged person to do something on your behalf:

- root:

  ```
  rm /tmp/*/*
  ```

the shell behind the scenes:

`rm /tmp/dir₁/file₁ /tmp/dir₁/file₂ /tmp/dir₂/file₁ …`

this takes time

A "Cron"-Attack

1. attacker (creates a fake passwd file)
   `mkdir /tmp/a; cat > /tmp/a/passwd`

2. root (does the daily cleaning)
   `rm /tmp/*/*`

   records that `/tmp/a/passwd`
   should be deleted, but does not do it yet

3. attacker (meanwhile deletes the fake passwd file, and establishes a link to the real passwd file)
   `rm /tmp/a/passwd; rmdir /tmp/a;`
   `ln -s /etc /tmp/a`

4. root now deletes the real passwd file

A "Cron"-Attack

1. attacker (creates a fake passwd file)
```
mkdir /tmp/a; cat > /tmp/a/passwd
```
2. ro

   rm

   > To prevent this kind of attack, you need additional policies (don't do such operations as root).

   should be deleted, but does not do it yet
3. attacker (meanwhile deletes the fake passwd file, and establishes a link to the real passwd file)
```
rm /tmp/a/passwd; rmdir /tmp/a;
ln -s /etc /tmp/a
```
4. root now deletes the real passwd file

Infamous Security Flaws in Unix

- `lpr` unfortunately runs with root privileges; you had the option to delete files after printing ...

Infamous Security Flaws in Unix

- `lpr` unfortunately runs with root privileges; you had the option to delete files after printing ...
- for debugging purposes (FreeBSD) Unix provides a "core dump", but allowed to follow links ...

Infamous Security Flaws in Unix

- `lpr` unfortunately runs with root privileges; you had the option to delete files after printing ...
- for debugging purposes (FreeBSD) Unix provides a “core dump”, but allowed to follow links ...
- `mkdir foo` is owned by root

```
-rwxr-xr-x 1 root wheel /bin/mkdir
```

it first creates an i-node as root and then changes to ownership to the user’s id
(race condition – can be automated with a shell script)

Infamous Security Flaws in Unix

- `lpr` unfortunately runs with root privileges; you had the option to delete files after printing ...
- for de... Unix provides a “cor...
- `mkdir` ...

> Only failure makes us experts.
> – Theo de Raadt (OpenBSD, OpenSSH)

```
-rwxr-xr-x 1 root wheel /bin/mkdir
```

it first creates an i-node as root and then changes to ownership to the user’s id
(race condition – can be automated with a shell script)

Subtleties

- Can Bob write `file`?

```
-  r-- rw- rwx  bob staff file
```

directory user group other

Subtleties

- Can Bob write `file`?
- What if Bob is member of `staff`?

```
- r-- rw- rwx bob staff file
```

directory user group other

Login Processes

- login processes run under UID = `0`

```
ps -axl | grep login
```

- after login, shells run under UID = user (e.g. 501)

```
id cu
```

Login Processes

- login processes run under UID = `0`

```
ps -axl | grep login
```

- after login, shells run under UID = user (e.g. 501)

```
id cu
```

- non-root users are not allowed to change the UID — would break access control
- but needed for example for accessing `passwd`

Setuid and Setgid

The solution is that Unix file permissions are 9 + 2 Bits: **Setuid** and **Setgid** bits

- When a file with setuid is executed, the resulting process will assume the UID given to the owner of the file.
- This enables users to create processes as root (or another user).
- Essential for changing passwords, for example.

```
chmod 4755 fobar_file
```

```
$ ls -ld . * */*
drwxr-xr-x ping staff  32768 Apr  2 2010 .
-rw----r-- ping students  31359 Jul 24 2011 manual.txt
-r--rw--w- bob  students   4359 Jul 24 2011 report.txt
-rwsr--r-x bob  students 141359 Jun  1 2013 microedit
dr--r-xr-x bob  staff     32768 Jul 23 2011 src
-rw-r--r-- bob  staff     81359 Feb 28 2012 src/code.c
-r--rw---- emma students    959 Jan 23 2012 src/code.h
```

members of group staff: ping, bob, emma
members of group students: emma

	manual.txt	report.txt	microedit	src/code.c	src/code.h
ping					
bob					
emma					

Discretionary Access Control

- Access to objects (files, directories, devices, etc.) is permitted based on user identity. Each object is owned by a user. Owners can specify freely (at their discretion) how they want to share their objects with other users, by specifying which other users can have which form of access to their objects.
- Discretionary access control is implemented on any modern multi-user OS (Unix, Windows NT, etc.).

Mandatory Access Control

- Access to objects is controlled by a system-wide policy, for example to prevent certain flows of information. In some forms, the system maintains security labels for both objects and subjects (processes, users) based on which access is granted or denied. Labels can change as the result of an access. Security policies are enforced without the cooperation of users or programs.
- This is implemented in banking or military operating system versions (SELinux).

Mandatory Access Control

- Access to objects is controlled by a system-wide policy, for example to prevent certain flows of information. In some forms, the system maintains security labels for both objects and subjects (processes, users) based on which access is granted or denied. Labels can change as the result of an access. Security policies are enforced without the cooperation of users or programs.

- This is implemented in banking or military operating system versions (SELinux).

- A simple example: Air Gap Security. Uses a completely separate network and computer hardware for different application classes (Bin Laden, Bruce Schneier had airgaps).

Mandatory Access Control

- Access to objects is controlled by a system-wide policy, for example to prevent certain flows of information. In some forms, the system maintains security labels for both objects and subjects (processes, users) based on which access is granted or denied. Labels can change as the result of an access. Security policies are enforced without the cooperation of users or programs.
- This is implemented in banking or military operating system versions (SELinux).
- A simple example: Air Gap Security. Uses a completely separate network and computer hardware for different application classes (Bin Laden, Bruce Schneier had airgaps).
- What do we want to protect: Secrecy or Integrity?

The Bell-LaPadula Model

- Formal policy model for mandatory access control in a military multi-level security environment. All subjects (processes, users, terminals, files, windows, connections) are labeled with a confidentiality level, e.g.

  unclassified < confidential < secret < top secret

- The system policy automatically prevents the flow of information from high-level objects to lower levels. A process that reads top secret data becomes tagged as top secret by the operating system, as will be all files into which it writes afterwards.

Bell-LaPadula

- Read Rule: A principal P can read an object O if and only if P's security level is at least as high as O's.
- Write Rule: A principal P can write an object O if and only if O's security level is at least as high as P's.

This restricts information flow $\Rightarrow$ military

Bell-LaPadula

- Read Rule: A principal *P* can read an object *O* if and only if *P*'s security level is at least as high as *O*'s.
- Write Rule: A principal *P* can write an object *O* if and only if *O*'s security level is at least as high as *P*'s.

This restricts information flow ⇒ military

Bell-LaPadula: **'no read up'** - **'no write down'**

Principle of Least Privilege

A principal should have as few privileges as possible to access a resource.

- Bob (*TS*) and Alice (*S*) want to communicate

  ⇒ Bob should lower his security level

Biba Policy

Data Integrity (rather than data secrecy)

- Biba: **'no read down' - 'no write up'**
- Read Rule: A principal *P* can read an object *O* if and only if *P*'s security level is lower or equal than *O*'s.
- Write Rule: A principal *P* can write an object *O* if and only if *O*'s security level is lower or equal than *P*'s.

Biba Policy

Data Integrity (rather than data secrecy)

- Biba: **'no read down'** - **'no write up'**
- Read Rule: A principal P can read an object O if and only if P's security level is lower or equal than O's.
- Write Rule: A principal P can write an object O if and only if O's security level is lower or equal than P's.

E.g. Firewalls: you can read from inside the firewall, but not from outside
Phishing: you can look at an approved PDF, but not one from a random email

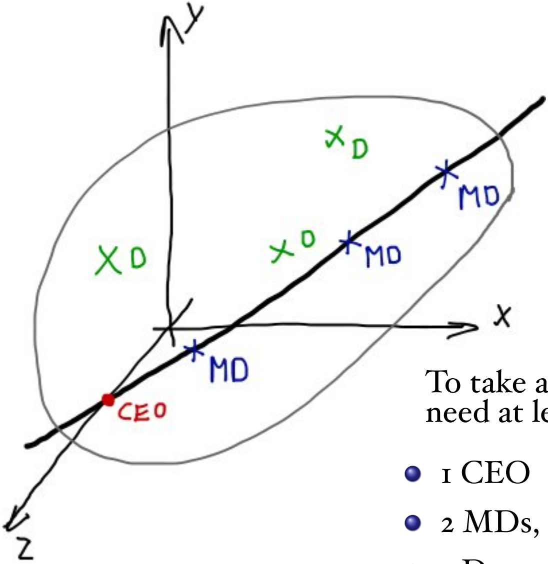
Security Levels (2)

- Bell-La Padula preserves data secrecy, but not data integrity

Security Levels (2)

- Bell-La Padula preserves data secrecy, but not data integrity
- Biba model is for data integrity
  - read: your own level and above
  - write: your own level and below

Shared Access Control



To take an action you need at least either:

- 1 CEO
- 2 MDs, or
- 3 Ds

Lessons from Access Control

Not just restricted to Unix:

- if you have too many roles (i.e. too finegrained AC), then hierarchy is too complex
  you invite situations like...lets be root
- you can still abuse the system...

Protocols

- Other examples: Wifi, Http-request, TCP-request, card readers, RFID (passports)...

Protocols

- Other examples: Wifi, Http-request, TCP-request, card readers, RFID (passports)...
- The point is that we cannot control the network: An attacker can install a packet sniffer, inject packets, modify packets, replay messages...fake pretty much everything.

Keyless Car Transponders

- There are two security mechanisms: one remote central locking system and one passive RFID tag (engine immobiliser).
- How can I get in? How can thieves be kept out? How to avoid MITM attacks?

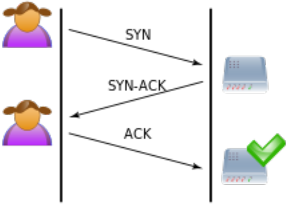
Papers: Gone in 360 Seconds: Hijacking with Hitag2,
Dismantling Megamos Crypto: Wirelessly Lockpicking
a Vehicle Immobilizer

HTTPS / GSM

- I am sitting at Starbuck. How can I be sure I am really visiting Barclays? I have no control of the access point.
- How can I achieve that a secret key is established in order to encrypt my mobile conversation? I have no control over the access points.

Handshakes

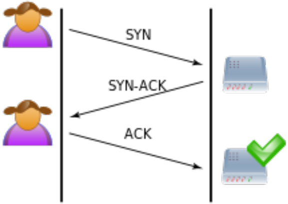
- starting a TCP connection between a client and a server initiates the following three-way handshake protocol:



Alice:	Hello server!
Server:	I heard you
Alice:	Thanks

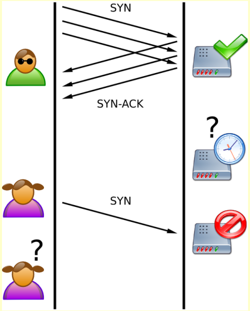
Handshakes

- starting a TCP connection between a client and a server initiates the following three-w handshake protocol:



Alice:
Server:
Alice:

SYNflood attacks:



Protocols

$$A \;\rightarrow\; B : \ldots$$

- by convention A, B are named principals Alice... but most likely they are programs, which just follow some instructions (they are more like roles)

Protocols

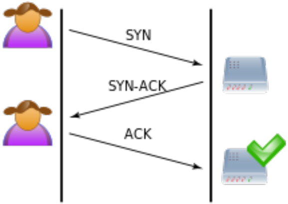
$$\begin{array}{rcl} A & \to & B : \ldots \\ B & \to & A : \ldots \\ & \vdots & \end{array}$$

- by convention A, B are named principals Alice... but most likely they are programs, which just follow some instructions (they are more like roles)

- indicates one "protocol run", or session, which specifies some order in the communication
- there can be several sessions in parallel (think of wifi routers)

Handshakes

- starting a TCP connection between a client and a server initiates the following three-way handshake protocol:



Alice:	Hello server!
Server:	I heard you
Alice:	Thanks

$$
\begin{array}{ll}
A \rightarrow S: & \text{SYN} \\
S \rightarrow A: & \text{SYN-ACK} \\
A \rightarrow S: & \text{ACK}
\end{array}
$$

Cryptographic Protocol Failures

Ross Anderson and Roger Needham wrote:

> A lot of the recorded frauds were the result of this kind of blunder, or from management negligence pure and simple. However, there have been a significant number of cases where the designers protected the right things, used cryptographic algorithms which were not broken, and yet found that their systems were still successfully attacked.

Oyster Cards

- good example of a bad protocol
  (security by obscurity)

Wirelessly Pickpocketing a Mifare Classic Card

The Mifare Classic is the most widely used contactless smartcard on the market. The stream cipher CRYPTO1 used by the Classic has recently been reverse engineered and serious attacks have been proposed. The most serious of them retrieves a secret key in under a second. In order to clone a card, previously proposed attacks require that the adversary either has access to an eavesdropped communication session or executes a message-by-message man-in-the-middle attack between the victim and a legitimate reader. Although this is already disastrous from a cryptographic point of view, system integrators maintain that these attacks cannot be performed undetected.

This paper proposes four attacks that can be executed by an adversary having only wireless access to just a card (and not to a legitimate reader). The most serious of them recovers a secret key in less than a second on ordinary hardware. Besides the cryptographic weaknesses, we exploit other weaknesses in the protocol stack. A vulnerability in the computation of parity bits allows an adversary to establish a side channel. Another vulnerability regarding nested authentications provides enough plaintext for a speedy known-plaintext attack. (a paper from 2009)

Oyster Cards

- good example of a bad protocol
  (security by obscurity)
- *"Breaching security on Oyster cards should not allow unauthorised use for more than a day, as TfL promises to turn off any cloned cards within 24 hours..."*

Another Example

In an email from Ross Anderson

From: Ross Anderson <Ross.Anderson@cl.cam.ac.uk>
Sender: cl-security-research-bounces@lists.cam.ac.uk
To: cl-security-research@lists.cam.ac.uk
Subject: Birmingham case
Date: Tue, 13 Aug 2013 15:13:17 +0100

As you may know, Volkswagen got an injunction against the University of Birmingham suppressing the publication of the design of a weak cipher used in the remote key entry systems in its recent-model cars. The paper is being given today at Usenix, minus the cipher design.

I've been contacted by Birmingham University's lawyers who seek to prove that the cipher can be easily obtained anyway. They are looking for a student who will download the firmware from any newish VW, disassemble it and look for the cipher. They'd prefer this to be done by a student rather than by a professor to emphasise how easy it is.

Volkswagen's argument was that the Birmingham people had reversed a locksmithing tool produced by a company in Vietnam, and since their key fob chip is claimed to be tamper-resistant, this must have involved a corrupt insider at VW or at its supplier Thales. Birmingham's argument is that this is nonsense as the cipher is easy to get hold of. Their lawyers feel this argument would come better from an independent outsider.

Let me know if you're interested in having a go, and I'll put you in touch
Ross

Authentication Protocols

Alice (A) and Bob (B) share a secret key K_{AB}

Passwords:

$$B \to A : K_{AB}$$

Authentication Protocols

Alice (A) and Bob (B) share a secret key K_{AB}

Passwords:

$$B \rightarrow A : K_{AB}$$

Problem: Eavesdropper can capture the secret and replay it; A cannot confirm the identity of B

Authentication?

"On the Internet, nobody knows you're a dog."

Authentication Protocols

Alice (A) and Bob (B) share a secret key K_{AB}

Simple Challenge Response:

$$\begin{array}{l} A \rightarrow B : \quad N \\ B \rightarrow A : \quad \{N\}_{K_{AB}} \end{array}$$

Authentication Protocols

Alice (A) and Bob (B) share a secret key K_{AB}

Mutual Challenge Response:

$$\begin{array}{ll}
A \to B: & N_A \\
B \to A: & \{N_A, N_B\}_{K_{AB}} \\
A \to B: & N_B
\end{array}$$

Nonces

1. I generate a nonce (random number) and send it to you encrypted with a key we share
2. you increase it by one, encrypt it under a key I know and send it back to me

I can infer:

- you must have received my message
- you could only have generated your answer after I send you my initial message
- if only you and me know the key, the message must have come from you

$$
\begin{aligned}
A \to B &: \quad N_A \\
B \to A &: \quad \{N_A, N_B\}_{K_{AB}} \\
A \to B &: \quad N_B
\end{aligned}
$$

The attack (let A decrypt her own messages):

$$
\begin{aligned}
A \to E &: \quad N_A \\
E \to A &: \quad N_A \\
A \to E &: \quad \{N_A, N'_A\}_{K_{AB}} \\
E \to A &: \quad \{N_A, N'_A\}_{K_{AB}} \\
A \to E &: \quad N'_A \;\; (= N_B)
\end{aligned}
$$

$$A \to B: \quad N_A$$
$$B \to A: \quad \{N_A, N_B\}_{K_{AB}}$$
$$A \to B: \quad N_B$$

The attack (let A decrypt her own messages):

$$A \to E: \quad N_A$$
$$E \to A: \quad N_A$$
$$A \to E: \quad \{N_A, N'_A\}_{K_{AB}}$$
$$E \to A: \quad \{N_A, N'_A\}_{K_{AB}}$$
$$A \to E: \quad N'_A \;\; (= N_B)$$

Solutions: $K_{AB} \neq K_{BA}$ or include an id in the second message

Encryption to the Rescue?

- $A \rightarrow B : \{A, N_A\}_{K_{AB}}$ encrypted
- $B \rightarrow A : \{N_A, K'_{AB}\}_{K_{AB}}$
- $A \rightarrow B : \{N_A\}_{K'_{AB}}$

Encryption to the Rescue?

- $A \rightarrow B : \{A, N_A\}_{K_{AB}}$ encrypted
- $B \rightarrow A : \{N_A, K'_{AB}\}_{K_{AB}}$
- $A \rightarrow B : \{N_A\}_{K'_{AB}}$

means you need to send separate “Hello” signals (bad), or worse share a single key between many entities

Protocol Attacks

- replay attacks
- reflection attacks
- man-in-the-middle attacks
- timing attacks
- parallel session attacks
- binding attacks (public key protocols)
- changing environment / changing assumptions

- (social engineering attacks)

Public-Key Infrastructure

- the idea is to have a certificate authority (CA)
- you go to the CA to identify yourself
- CA: “I, the CA, have verified that public key P^{pub}_{Bob} belongs to Bob”

- CA must be trusted by everybody
- What happens if CA issues a false certificate? Who pays in case of loss? (VeriSign explicitly limits liability to $100.)

Man-in-the-Middle

"Normal" protocol run:

- A sends public key to B
- B sends public key to A
- A sends message encrypted with B's public key, B decrypts it with its private key
- B sends message encrypted with A's public key, A decrypts it with its private key

Man-in-the-Middle

Attack:

- A sends public key to B — C intercepts this message and send his own public key
- B sends public key to A — C intercepts this message and send his own public key
- A sends message encrypted with C's public key, C decrypts it with its private key, re-encrypts with B's public key
- similar for other direction

Man-in-the-Middle

Potential Prevention?

- *A* sends public key to *B*
- *B* sends public key to *A*
- *A* encrypts message with *B*'s public key, send's **half** of the message
- *B* encrypts message with *A*'s public key, send's **half** of the message
- *A* sends other half, *B* can now decrypt entire message
- *B* sends other half, *A* can now decrypt entire message

Man-in-the-Middle

Potential Prevention?

- A sends public key to B
- B sends public key to A
- A encrypts message with B's public key, send's **half** of the message
- B encrypts message with A's public key, send's **half** of the message
- A sends other half, B can now decrypt entire message
- B sends other half, A can now decrypt entire message

Under which circumstances does this protocol prevent MiM-attacks, or does it?

Car Transponder (HiTag2)

1. C generates a random number N
2. C calculates $(F, G) = \{N\}_K$
3. $C \rightarrow T$: N, F
4. T calculates $(F', G') = \{N\}_K$
5. T checks that $F = F'$
6. $T \rightarrow C$: N, G'
7. C checks that $G = G'$

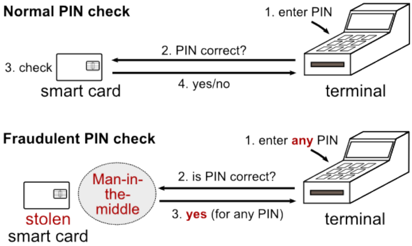
Car Transponder (HiTag2)

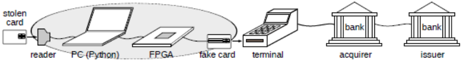
1. C generates a random number N
2. C calculates $(F, G) = \{N\}_K$
3. $C \rightarrow T$: N, F
4. T calculates $(F', G') = \{N\}_K$
5. T checks that $F = F'$
6. $T \rightarrow C$: N, G'
7. C checks that $G = G'$

This process means that the transponder believes the car knows the key K, and the car believes the transponder knows the key K. They have authenticated themselves to each other, or have they?

A Man-in-the-middle attack in real life:

- the card only says yes to the terminal if the PIN is correct
- trick the card in thinking transaction is verified by signature
- trick the terminal in thinking the transaction was verified by PIN





Problems with EMV

- it is a wrapper for many protocols
- specification by consensus (resulted unmanageable complexity)
- its specification is 700 pages in English plus 2000+ pages for testing, additionally some further parts are secret
- other attacks have been found

Protocols are Difficult

- even the systems designed by experts regularly fail
- the one who can fix a system should also be liable for the losses
- cryptography is often not the problem