Access Control and Privacy Policies (4)

Email: christian.urban at kcl.ac.uk
Office: S1.27 (1st floor Strand Building)
Slides: KEATS (also home work is there)

Survey: Thanks!

- "Would be good, if you provide more detailed explanations. I feel your slides are not as structured as they could be."
- "Please consider reference book chapters to cover core subject areas."

Survey: Thanks!

- "Would be good, if you provide more detailed explanations. I feel your slides are not as structured as they could be."
- "Please consider reference book chapters to cover core subject areas."
- "The homework questions don't come directly from the slides. So must go look things up."
- "Could you please put the homework answers online, perhaps just before the exam. That's late enough where we should have done it and if not, we're screwed already then."
- "Could you provide a brief basic answers to sheets for reference and not to be relied on."

last week: buffer overflow attacks

D-Link Wifi Router, BOA

As a proof-of-concept, the following URL allows attackers to control the return value saved on the stack (the vulnerability is triggered when executing "`/usr/sbin/widget`"):

```
curl http://<target ip>/post_login.xml?hash=AAA...AAABBBB
```

The value of the "hash" HTTP GET parameter consists of 292 occurrences of the '`A`' character, followed by four occurrences of character '`B`'. In our lab setup, characters '`B`' overwrite the saved program counter (`%ra`).

Discovery date: 06/03/2013
Release date: 02/08/2013

http://roberto.greyhats.it/advisories/20130801-dlink-dir645.txt

D-Link Backdoors

D-Link router flaw lets anyone login through "Joel's Backdoor":

> If you tell your browser to identify itself as Joel's backdoor, instead of (say) as Mozilla/5.0 AppleWebKit/536.30.1 Version/6.0.5, you're in without authentication.
>
> "What is this string," I hear you ask?
> You will laugh: it is

D-Link Backdoors

D-Link router flaw lets anyone login through "Joel's Backdoor":

> If you tell your browser to identify itself as Joel's backdoor, instead of (say) as Mozilla/5.0 AppleWebKit/536.30.1 Version/6.0.5, you're in without authentication.
>
> "What is this string," I hear you ask?
> You will laugh: it is
>
> `xmlset_roodkcableoj28840ybtide`

October 15, 2013

http://www.devttys0.com/2013/10/reverse-engineering-a-d-link-backdoor/

CVE-2014-0476 `chkrootkit` vulnerability 4 Jun'14

Hi,
we just found a serious vulnerability in the chkrootkit package, which may allow local attackers to gain root access to a box in certain configurations (`/tmp` not mounted noexec). Steps to reproduce:

- Put an executable file named `update` with non-root owner in `/tmp` (not mounted noexec, obviously)
- Run chkrootkit (as uid `0`)

Result: The file `/tmp/update` will be executed as root, thus effectively rooting your box, if malicious content is placed inside the file.
If an attacker knows you are periodically running chkrootkit (like in `cron.daily`) and has write access to `/tmp` (not mounted noexec), he may easily take advantage of this.

http://seclists.org/oss-sec/2014/q2/430

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

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

```
$ ls -la
-rwxrw-r--    foo_file.txt
```

Login Process

- login processes run under UID = `0`

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

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

```
id cu
```

Login Process

- 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
```

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).

Discretionary Access Control

In its most generic form usually given by an Access Control Matrix of the form

	/mail/jane	edit.exe	postfix
jane	r, w	r, x	r, x
john	∅	r, w, x	r, x
postfix	a	∅	r, x

access privileges: **r**ead, **w**rite, e**x**ecute, **a**ppend

```
$ 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					

Mandatory Access Control

- Restrictions to allowed information flows are not decided at the user's discretion (as with Unix `chmod`), but instead enforced by system policies.
- Mandatory access control mechanisms are aimed in particular at preventing policy violations by untrusted programs, which typically have at least the same access privileges as the invoking user.

Mandatory Access Control

- Restrictions to allowed information flows are not decided at the user's discretion (as with Unix `chmod`), but instead enforced by system policies.
- Mandatory access control mechanisms are aimed in particular at preventing policy violations by untrusted programs, which typically have at least the same access privileges as the invoking user.

  Simple example: Air Gap Security. Uses a completely separate network and computer hardware for different application classes.

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.
- Meta-Rule: All principals in a system should have a sufficiently high security level in order to access an object.

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.
- Meta-Rule: All principals in a system should have a sufficiently high security level in order to access an object.

This restricts information flow $\Rightarrow$ 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

  $\Rightarrow$ Bob should lower his security level

Biba Policy

Data Integrity (rather than data confidentiality)

- 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 confidentiality)

- 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

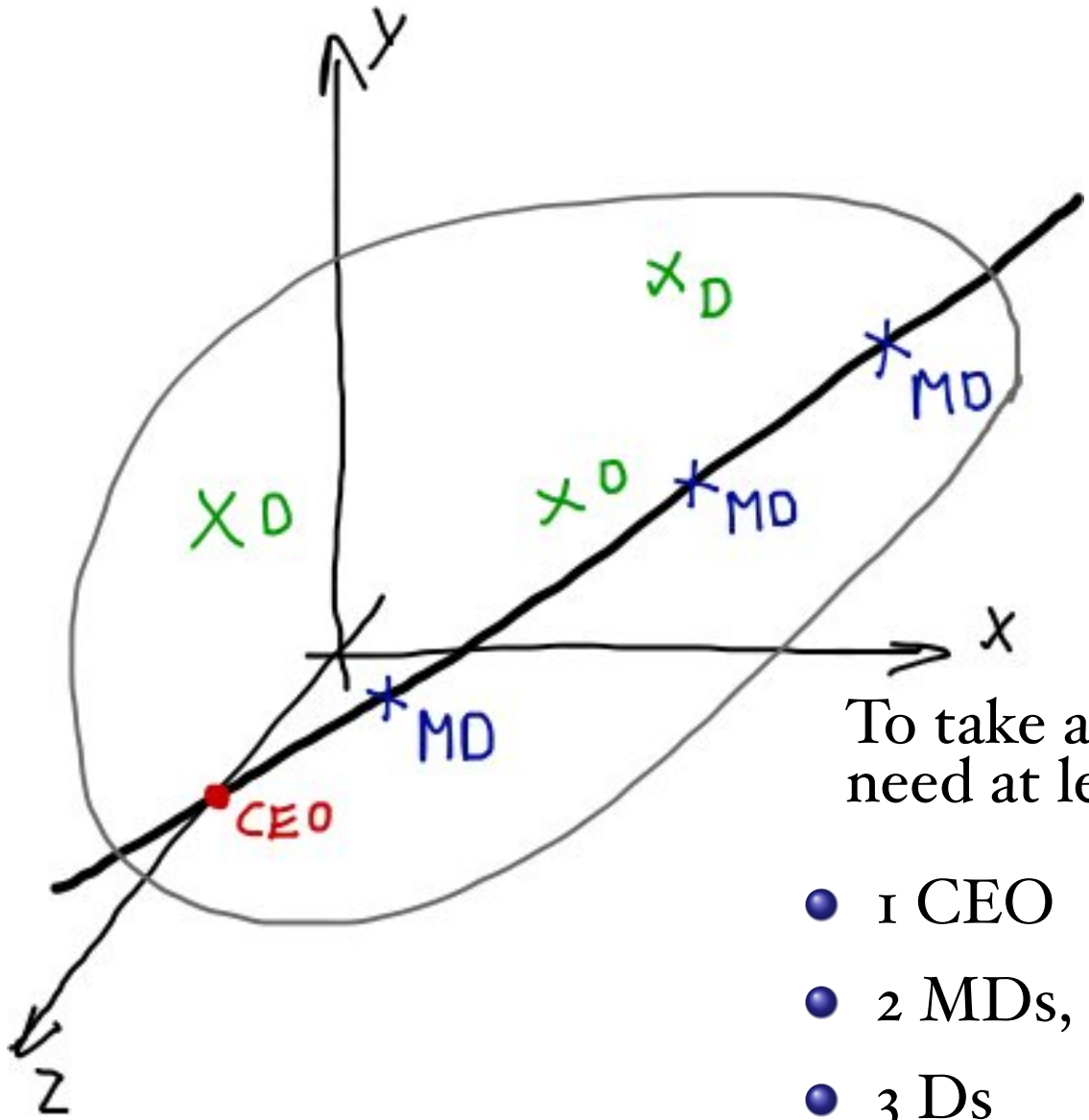
Security Levels (2)

- 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...

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...

- policies (a finite system)
  computer system (infinite)

  Q: Does your policy ensure that a tainted file cannot affect your core system files?

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

$$
\begin{array}{l}
A \rightarrow B : \ldots \\
B \rightarrow A : \ldots \\
\quad \vdots
\end{array}
$$

- 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)

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

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.

Authentication Protocols

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

Passwords:

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

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 Protocols

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

Simple Challenge Response:

$$\begin{array}{ll} A \rightarrow B: & N \\ B \rightarrow A: & \{N\}_{K_{AB}} \end{array}$$

Authentication Protocols

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

Mutual Challenge Response:

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

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

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

The attack (let A decrypt her own messages):

$$
\begin{array}{ll}
A \rightarrow E: & N_A \\
E \rightarrow A: & N_A \\
A \rightarrow E: & \{N_A, N'_A\}_{K_{AB}} \\
E \rightarrow A: & \{N_A, N'_A\}_{K_{AB}} \\
A \rightarrow E: & N'_A \ \ (= N_B)
\end{array}
$$

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

The attack (let A decrypt her own messages):

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

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.)

Person-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

Person-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

Person-in-the-Middle

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

Person-in-the-Middle

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

C would have to invent a totally new message

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.)

Binding Attacks

with public-private keys it is important that the public key is bound to the right owner (verified by a certification authority CA)

$$A \rightarrow CA : A, B, N_A$$
$$CA \rightarrow A : CA, \{CA, A, N_A, K_B^{pub}\}_{K_A^{pub}}$$

A knows K_A^{priv} and can verify the message came from CA in response to A's message and trusts K_B^{pub} is B's public key

Binding Attacks

$$A \rightarrow I(CA) : A, B, N_A$$
$$I(A) \rightarrow CA : A, I, N_A$$
$$CA \rightarrow I(A) : CA, \{CA, A, N_A, K_I^{pub}\}_{K_A^{pub}}$$
$$I(CA) \rightarrow A : CA, \{CA, A, N_A, K_I^{pub}\}_{K_A^{pub}}$$

Binding Attacks

$$A \rightarrow I(CA) : A, B, N_A$$
$$I(A) \rightarrow CA : A, I, N_A$$
$$CA \rightarrow I(A) : CA, \{CA, A, N_A, K_I^{pub}\}_{K_A^{pub}}$$
$$I(CA) \rightarrow A : CA, \{CA, A, N_A, K_I^{pub}\}_{K_A^{pub}}$$

A now encrypts messages for B with the public key of I (which happily decrypts them with its private key)

Replay Attacks

Schroeder-Needham protocol: exchange of a symmetric key with a trusted 3rd-party S:

$$
\begin{aligned}
A &\rightarrow S : A, B, N_A \\
S &\rightarrow A : \{N_A, B, K_{AB}, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}} \\
A &\rightarrow B : \{K_{AB}, A\}_{K_{BS}} \\
B &\rightarrow A : \{N_B\}_{K_{AB}} \\
A &\rightarrow B : \{N_B - 1\}_{K_{AB}}
\end{aligned}
$$

Replay Attacks

Schroeder-Needham protocol: exchange of a symmetric key with a trusted 3rd-party S:

$$
\begin{array}{l}
A \to S : A, B, N_A \\
S \to A : \{N_A, B, K_{AB}, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}} \\
A \to B : \{K_{AB}, A\}_{K_{BS}} \\
B \to A : \{N_B\}_{K_{AB}} \\
A \to B : \{N_B - 1\}_{K_{AB}}
\end{array}
$$

at the end of the protocol both A and B should be in the possession of the secret key K_{AB} and know that the other principal has the key

$$A \to S : A, B, N_A$$
$$S \to A : \{N_A, B, K_{AB}, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$$
$$A \to B : \{K_{AB}, A\}_{K_{BS}}$$
$$B \to A : \{N_B\}_{K_{AB}}$$
$$A \to B : \{N_B - 1\}_{K_{AB}}$$

$$A \to S : A, B, N_A$$
$$S \to A : \{N_A, B, K_{AB}, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$$
$$A \to B : \{K_{AB}, A\}_{K_{BS}}$$
$$B \to A : \{N_B\}_{K_{AB}}$$
$$A \to B : \{N_B - 1\}_{K_{AB}}$$

compromise K_{AB}

$$A \to S : A, B, N_A$$
$$S \to A : \{N_A, B, K_{AB}, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$$
$$A \to B : \{K_{AB}, A\}_{K_{BS}}$$
$$B \to A : \{N_B\}_{K_{AB}}$$
$$A \to B : \{N_B - 1\}_{K_{AB}}$$

compromise K_{AB}

$$A \to S : A, B, N'_A$$
$$S \to A : \{N'_A, B, K'_{AB}, \{K'_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$$

$A \rightarrow S : A, B, N_A$

$S \rightarrow A : \{N_A, B, K_{AB}, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$

$A \rightarrow B : \{K_{AB}, A\}_{K_{BS}}$

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

$A \rightarrow B : \{N_B - 1\}_{K_{AB}}$

compromise K_{AB}

$A \rightarrow S : A, B, N'_A$

$S \rightarrow A : \{N'_A, B, K'_{AB}, \{K'_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$

$I(A) \rightarrow B : \{K_{AB}, A\}_{K_{BS}}$ replay of older run

$A \rightarrow S : A, B, N_A$

$S \rightarrow A : \{N_A, B, K_{AB}, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$

$A \rightarrow B : \{K_{AB}, A\}_{K_{BS}}$

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

$A \rightarrow B : \{N_B - 1\}_{K_{AB}}$

compromise K_{AB}

$A \rightarrow S : A, B, N'_A$

$S \rightarrow A : \{N'_A, B, K'_{AB}, \{K'_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$

$I(A) \rightarrow B : \{K_{AB}, A\}_{K_{BS}}$ replay of older run

$B \rightarrow I(A) : \{N'_B\}_{K_{AB}}$

$I(A) \rightarrow B : \{N'_B - 1\}_{K_{AB}}$

$A \to S : A, B, N_A$

$S \to A : \{N_A, B, K_{AB}, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$

$A \to B : \{K_{AB}, A\}_{K_{BS}}$

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

$A \to B : \{N_B - 1\}_{K_{AB}}$

compromise K_{AB}

$A \to S : A, B, N'_A$

$S \to A : \{N'_A, B, K'_{AB}, \{K'_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$

$I(A) \to B : \{K_{AB}, A\}_{K_{BS}}$ replay of older run

$B \to I(A) : \{N'_B\}_{K_{AB}}$

$I(A) \to B : \{N'_B - 1\}_{K_{AB}}$

B believes it is following the correct protocol, intruder I can form the correct response because it knows K_{AB} and talks to B masquerading as A

Time-Stamps

The Schroeder-Needham protocol can be fixed by including a time-stamp (e.g., in Kerberos):

$$
\begin{array}{l}
A \rightarrow S : A, B, N_A \\
S \rightarrow A : \{N_A, B, K_{AB}, \{K_{AB}, A, T_S\}_{K_{BS}}\}_{K_{AS}} \\
A \rightarrow B : \{K_{AB}, A, T_S\}_{K_{BS}} \\
B \rightarrow A : \{N_B\}_{K_{AB}} \\
A \rightarrow B : \{N_B - 1\}_{K_{AB}}
\end{array}
$$

Time-Stamps

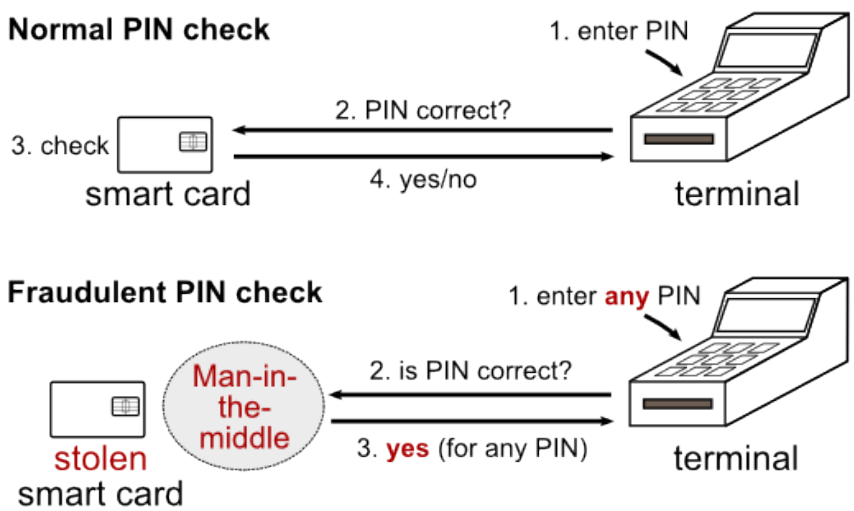
The Schroeder-Needham protocol can be fixed by including a time-stamp (e.g., in Kerberos):

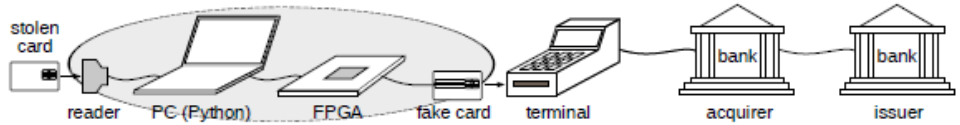
$$
\begin{array}{l}
A \rightarrow S : A, B, N_A \\
S \rightarrow A : \{N_A, B, K_{AB}, \{K_{AB}, A, T_S\}_{K_{BS}}\}_{K_{AS}} \\
A \rightarrow B : \{K_{AB}, A, T_S\}_{K_{BS}} \\
B \rightarrow A : \{N_B\}_{K_{AB}} \\
A \rightarrow B : \{N_B - 1\}_{K_{AB}}
\end{array}
$$

but nothing is for free: then you need to synchronise time and possibly become a victim to timing attacks

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

Problems with WEP (Wifi)

- a standard ratified in 1999
- the protocol was designed by a committee not including cryptographers
- it used the RC4 encryption algorithm which is a stream cipher requiring a unique nonce
- WEP did not allocate enough bits for the nonce
- for authenticating packets it used CRC checksum which can be easily broken
- the network password was used to directly encrypt packages (instead of a key negotiation protocol)
- encryption was turned off by default

Protocols are Difficult

- even the systems designed by experts regularly fail
- try to make everything explicit (you need to authenticate all data you might rely on)
- the one who can fix a system should also be liable for the losses
- cryptography is often not **the** answer

Best Practices

Principle 1: Every message should say what it means: the interpretation of a message should not depend on the context.

Best Practices

Principle 1: Every message should say what it means: the interpretation of a message should not depend on the context.

Principle 2: If the identity of a principal is essential to the meaning of a message, it is prudent to mention the principal's name explicitly in the message (though difficult).

Principle 3: Be clear about why encryption is being done. Encryption is not cheap, and not asking precisely why it is being done can lead to redundancy. Encryption is not synonymous with security.

Possible Uses of Encryption

- Preservation of confidentiality: $\{X\}_K$ only those that have K may recover X.
- Guarantee authenticity: The partner is indeed some particular principal.
- Guarantee confidentiality and authenticity: binds two parts of a message — $\{X, Y\}_K$ is not the same as $\{X\}_K$ and $\{Y\}_K$.

Best Practices

Principle 4: The protocol designer should know which trust relations his protocol depends on, and why the dependence is necessary. The reasons for particular trust relations being acceptable should be explicit though they will be founded on judgment and policy rather than on logic.

Example Certification Authorities: CAs are trusted to certify a key only after proper steps have been taken to identify the principal that owns it.