Computer Security
&
Access Protection

topics:

Overview and Terminology

Security requirements

Threats, adversaries and intruders

Attacks from outside the system

Attacks from inside the system

Security holes

Protection mechanisms

Trusted systems

Trust
Security
Protection

?

translation of terms:

Authenticity:	**Authentizität**
Availability:	**Verfügbarkeit**
Confidentiality:	**Vertraulichkeit**
Denial of Service:	**Dienstverweigerung**
Integrity:	**Datenintegrität, Schutz gegen unautorisierte Veränderung**
Intruder, Adversary:	**Eindringling, Angreifer, Gegner**
Privacy:	**Datenschutz**
Protection:	**Zugriffsschutz**
Security:	**(Informations-) Sicherheit (Betriebssicherheit= safety)**
Security threat:	**Bedrohung**
Trust:	**Vertrauenswürdigkeit**

Definitions:

Trust is a property within a social organization with respect to handling information. Trust defines the requirements and the resulting policies defined by an application area concerning the proper usage of information in the temporal and functional domain. It reflects the flow of information in an organization and is specified in terms of rules between authorization of subjects and clearance of information.

Security is the property of an information processing system. Security defines the requirements useful for an owner and user of information to protect it against security threats. Basic requirements which have to be assured in spite of intentional and malicious attacks are the confidentiality, integrity, availability and authenticity of information.

Protection is the set of hardware and software mechanisms to enforce security in a system.

Access Control

Trusted System:

Mandatory access control.

Rules defined by organization policy.

Secure System:

Discretionary, user defined access control.

Rules defined by individual user.

Goal: Flexibility, Expressiveness, Least Privilege.

Protection System:

Mechanisms in the hardware, firmware and the operating system to enforce access specifications.

Security vs. Privacy

Security protects data against misuse by individuals.

Privacy protects individuals against the misuse of data.

Security is a necessary but not a sufficient condition for trust and privacy !

requirements for security

Confidentiality:	data should not be read by unauthorized parties.
Integrity:	data should not be changed by unauthorized parties.
Availability:	data should be accessible when they are needed.
Authenticity:	the identity of subjects may not be forged

Orange Book Security Classes

highest

security levels

A1: Verified Design

B3: Security Domains

B2: Structured Protection

B1: Labeled Security Protection

C2: Controlled Access Protection

(C1+ object reuse + audit)

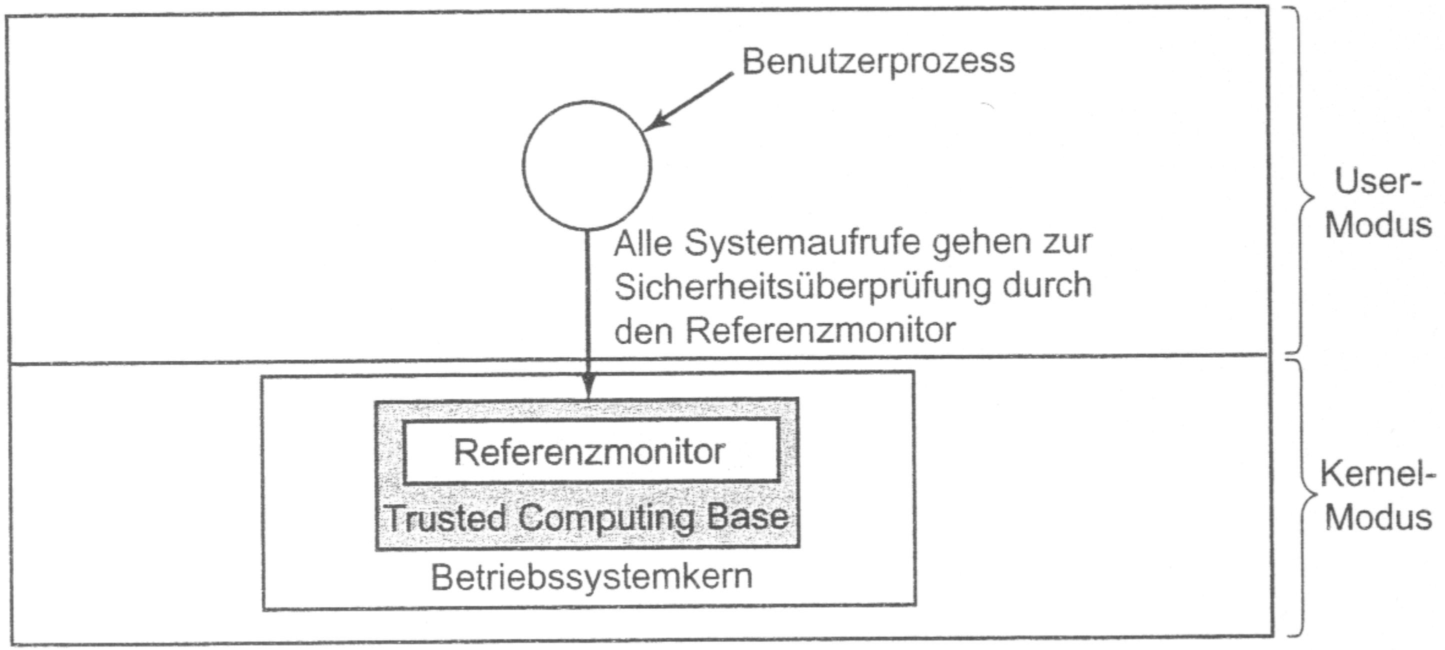
C1: Discretionary Access Control

D: Minimal Protection

lowest

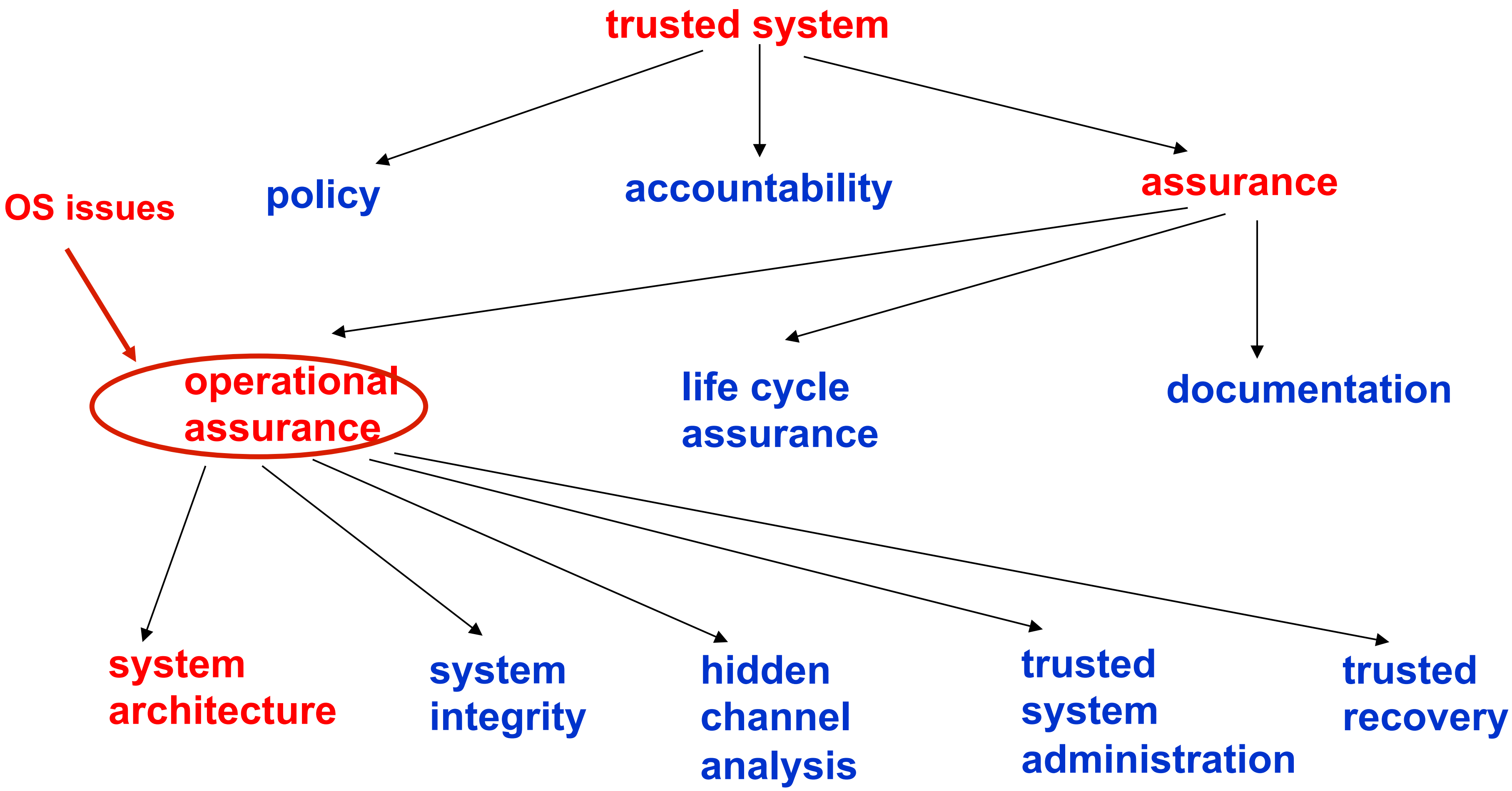
Components of the security model

- Reference monitor: an abstract machine that mediates all access control decisions
- Reference validation mechanism (RVM): an implementation of a reference monitor
- Security kernel: software+ hardware that implements a reference monitor
- Trusted Computing Base (TCB): all protection mechanisms that enforce security policy
- Target of Evaluation (ToE): the subject of evaluation (system or product)

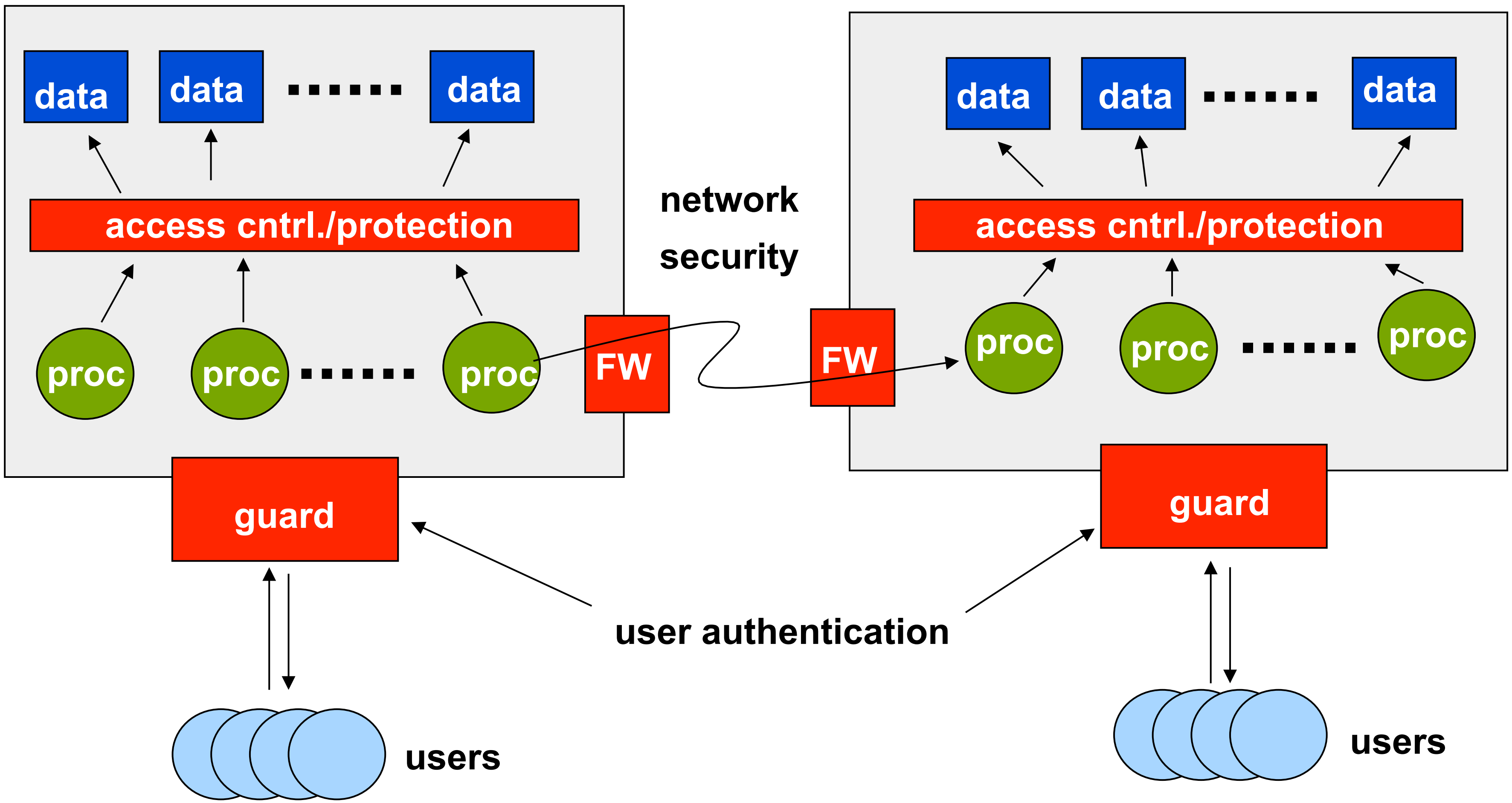

Benutzerprozess
Alle Systemaufrufe gehen zur Sicherheitsüberprüfung durch den Referenzmonitor
User-Modus
Referenzmonitor
Trusted Computing Base
Betriebssystemkern
Kernel-Modus

structuring requirements

acc. DoD Orange Book

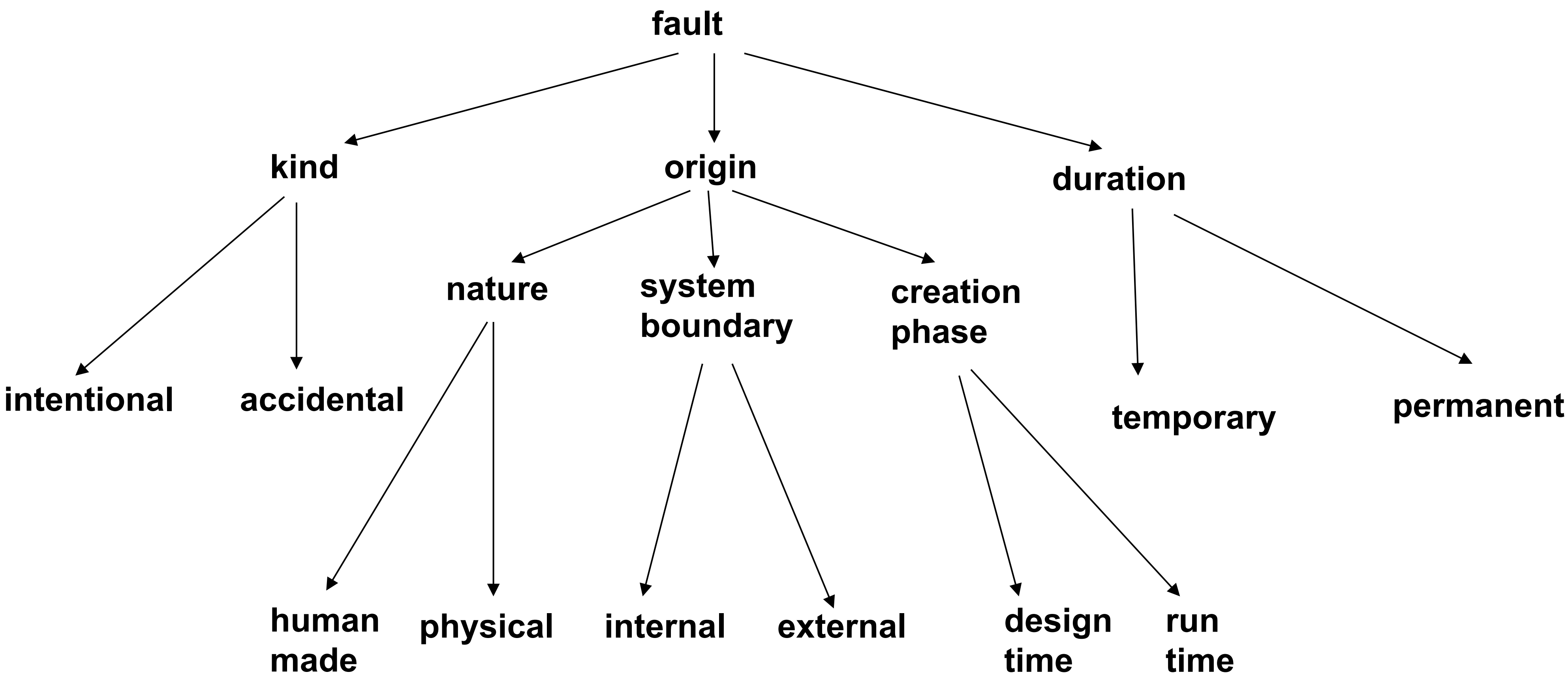


system vulnerabilities



classification of threats

a threat emerges from a fault in some system component or a fault by some user of the system



acc. J.C. Laprie: Dependability: Basic Concepts and associated teminology, 1990

classification of threats

example 1: threats caused by intentional (malicious), human-made faults

system boundary		creation phase		duration		threat
internal	external	desing time	run time	perm.	temp.	
	x		x	x		Intrusion
	x		x		x	Intrusion
x			x	x		Virus
x		x	x	x		Trojan Horse
x		x		x		malicious logic

classification of threats

example 2: threats caused by accidental faults

	system boundary		creation phase		duration		threat
	internal	external	desing time	run time	perm.	temp.	
physical	x			x	x	x	denial of service
physical	x			x	x	x	loss of integrity
physical	x			x	x	x	loss of confidentiality
human made	x		x		x		loss of integrity
human made	x		x		x		loss of confidentiality

by software or

hardware design faults

classification of adversaries

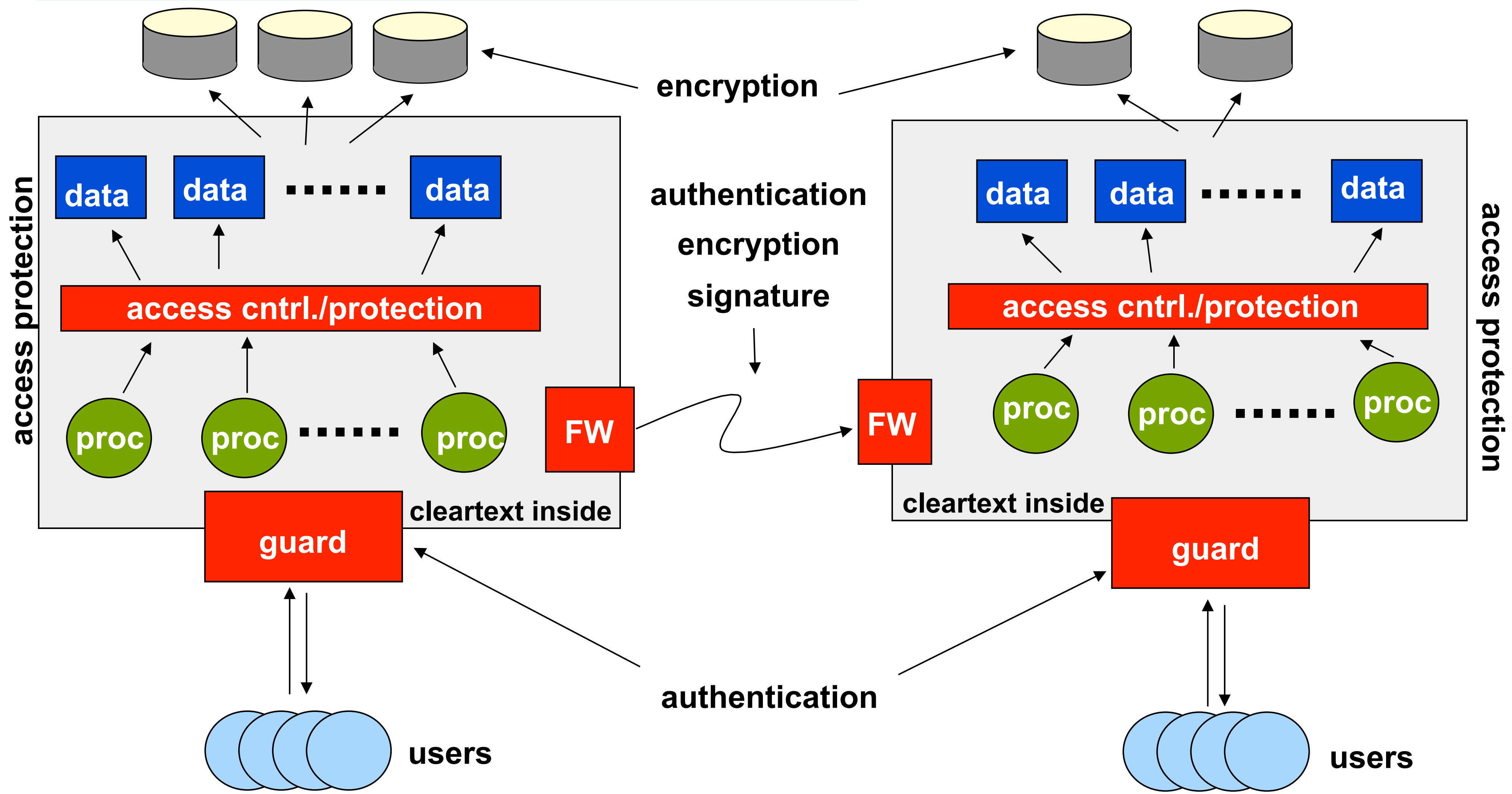
- occasional non-expert intruders
- expert insiders, unauthorized experienced hackers hacking the system
- expert insiders which have authorized access to the system
- espionage (military and company systems)
- sabotage (military, intelligence sevices, companies
- higher forces: Fire, flood, earthquakes
- faults and bugs in the computer and the network
- just humans: e.g. disk with highly confidential data on the garbage etc.

Authentication

Access Protection

Encryption

Encryption, authentication and access protection



Definitions

Authentication: Ensures that a user is the one she pretends to be. Authentication is based on a secret or on a non-forgable identifier. On a standard OS, authentication is enforced by the "Login" procedure.

On the OS-level, processes act on behalf of a user. A process usually obtains the privileges which are granted for the user.

Access Protection: Ensures that an authenticated users (or the processes acting on behalf of the user) only have access to exactly the items they are allowed to use. This is enforced by memory and file protection mechanisms.

attacks from outside of the system

The login procedure

```
LBL>telnet elxsi
ELXSI AT LBL
LOGIN: root
PASSWORD:root
INCORRECT PASSWORD, TRY AGAIN
LOGIN: guest
PASSWORD: guest
INCORRECT PASSWORD, TRY AGAIN
LOGIN: uucp
PASSWORD: uucp
WELCOME TO THE ELXSI COMPUTER AT LBL
```

Stoll 89

Tabelle 15.2 Beobachtete Passwortlänge

Länge	Anzahl	Anteil an Gesamtheit
1	55	0,004
2	87	0,006
3	212	0,02
4	449	0,03
5	1.260	0,09
6	3.035	0,22
7	2.917	0,21
8	5.772	0,42
Gesamt	13.787	1,00

From: William Stallings:Betriebssysteme, Prinzipien und Umsetzung, 4. Auflage, Pearson Studium, 2003

Tabelle 15.3 Passwörter, die aus einer Probe von 13.797 Konten geknackt wurden [KLEI90]

Passwortart	Suchgröße	Anzahl der Treffer	Erratene Passwörter in Prozent
Benutzer-/Kontoname	130	368	2,7%
Zeichenfolge	866	22	0,2%
Zahlen	427	9	0,1%
Chinesisch	392	56	0,4%
Ortsnamen	628	82	0,6%
Gebräuchliche Namen	2,239	548	4,0%
Frauennamen	4,280	161	1,2%
Männernamen	2,866	140	1,0%
Ungewöhnliche Namen	4,955	130	0,9%
Mythen und Legenden	1,246	66	0,5%
Shakespearesch	473	11	0,1%
Sportbegriffe	238	32	0,2%
Science-Fiction	691	59	0,4%
Filme und Schauspieler	99	12	0,1%
Comics	92	9	0,1%
Berühmte Menschen	290	55	0,4%
Redewendungen und Muster	933	253	1,8%
Nachnamen	33	9	0,1%

Tabelle 15.3 Passwörter, die aus einer Probe von 13.797 Konten geknackt wurden [KLEI90]

Passwortart	Suchgröße	Anzahl der Treffer	Erratene Passwörter in Prozent
Biologie	58	1	0,0%
Wörterbuch des Systems	19.683	1,027	7,4%
Rechnernamen	9.018	132	1,0%
Mnemonik	14	2	0,0%
King James-Bibel	7.525	83	0,6%
Verschiedene Wörter	3.212	54	0,4%
Jiddische Wörter	56	0	0,0%
Asteroide	2.407	19	0,1%
GESAMT	62.727	3,340	24,2%

From: William Stallings:Betriebssysteme, Prinzipien und Umsetzung, 4. Auflage, Pearson Studium, 2003

passwd security

/etc/passwd holds a list of <name, encoded passwd>

passwd guessing: prepare a list of common passwd, encoded passwd
read the /etc/passwd from some computer
compare encoded passwd
on match > store <name, passwd>

salt: create entries: <name, random number, encoded passwd>
to obtain a match, the cracker has to generate b^n (b=base n=exponent) versions of each passwd.

better passwd: longer names, not in a dictionary, numbers, special characters

one-time passwd: only used once. (Lamports algrithm to generate the list)

Def. One-Way-Function

Definition: One-Way Function

Informally, a function *f* is a one-way function if

1. **The description of *f* is publicly known and does not require any secret information for its operation.**
2. **Given *x*, it is easy to compute *f*(*x*).**
3. **Given *y*, in the range of *f*, it is hard to find an *x* such that** $f(x) = y$

   More precisely, any efficient algorithm solving a P-problem succeeds in inverting *f* with negligible probability.

The existence of one-way functions is not proven. If true, it would imply $P \neq NP$ **. Therefore, it would answer the complexity theory NP-problem question of whether all apparently NP-problems are actually P-problems. Yet a number of conjectured one-way functions are routinely used in commerce and industry. For example, it is conjectured, but not proved, that the following are one-way functions:**

1. **Factoring problem for randomly chosen primes *p, q*.**
2. **Discrete logarithm problem. (given b (mod p) and b^n (mod p) find n)**
3. **Discrete root extraction problem. This is the function commonly known as RSA encryption.**
4. **Quadratic residue problem.**

Used e.g. in password encryption, Public Key Cryptography, Digital Signatures, ...

Eric W. Weisstein. "One-Way Function." From *MathWorld*--A Wolfram Web Resource. http://mathworld.wolfram.com/One-WayFunction.html

Scientific American August 1977, Martin Gardener, Column "Mathematische Spiele".

He claimed that it would take "millions of years" to break the code.

N=114 318 625 757 888 867 669 235 779 976 146 612 010 218 296 721 242 362 562 561 842 935 706 935 245 733 897 839 597 123 563 958 705 058 989 075 147 599 290 026 879 543 541

26.November 1994 the factors were found by a group of 600 volunteers.

426 bits, Range: 10^{129}

N = 3490529510847650949147849619903898133417764638493387843990820577

x 32769132993266709549961988190834461413177642967992942539798288533

Smallest not yet factored product of two primes (Dec. 2011):

RSA-210=245246644900278211976517663573088018467026787678332759743414451715061600830038587216952208399332071549103626827191679864079776723243005600592035631246561218465817904100131859299619933817012149335034875870551067

696 Bits, Range: 10^{210}

RSA-704 (704 Bits, 212 decimal digits).... a cash prize of US$30,000 was offered for a successful factorization

Simon Singh: "The Code Book", 1999

One-time password

It is only possible to calculate an old, already used password.

Initialize with P0 = f(P1)

P1 = f(f(f(f(s))))

P2 = f(f(f(s)))

P3 = f(f(s))

P4 = f(s)

↑ Pn-1 is easy to calculate

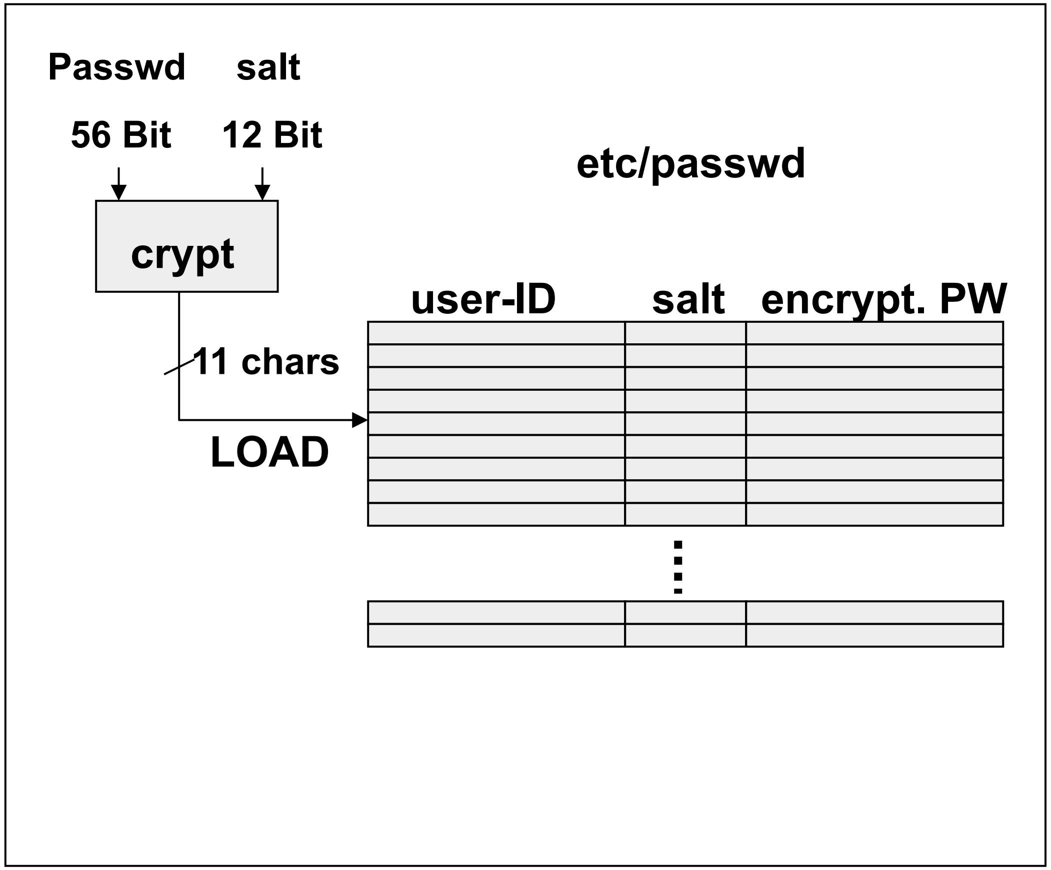
↓ Pn+1 is difficult to calculate

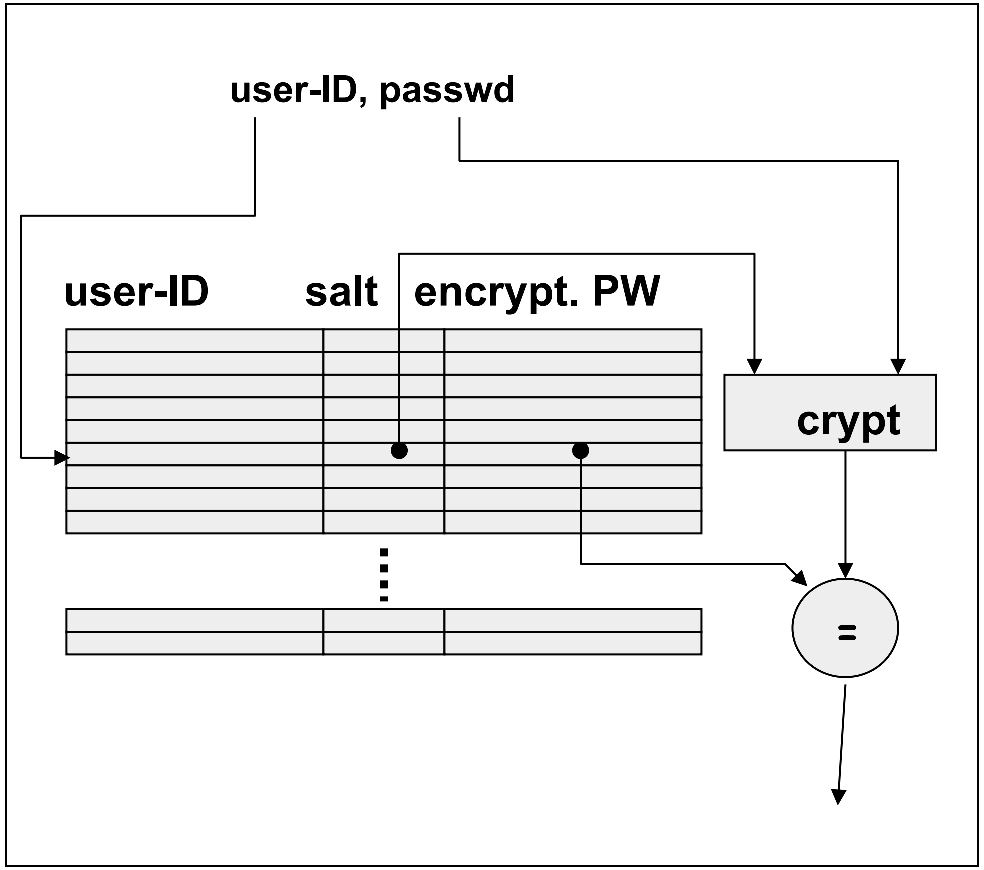
Only the user knows the secret s.

P0 is stored in the (server-) computer.

Pn is given by the user. The system applies the function f n times and compares the result with P0. If it matches, login is granted.

Password mechanisms in Unix





more authentication

challenge-response

chip card + PIN

- magnetic (~ 140 Bytes, costs 0,1 -0,5 €)
- memory cards (~1 KB, ~1 €)
- smart cards (8bit CPU, 16 KB ROM, 4 KB EEPROM, 512 Bytes RAM,
  9600 bps communication channel)

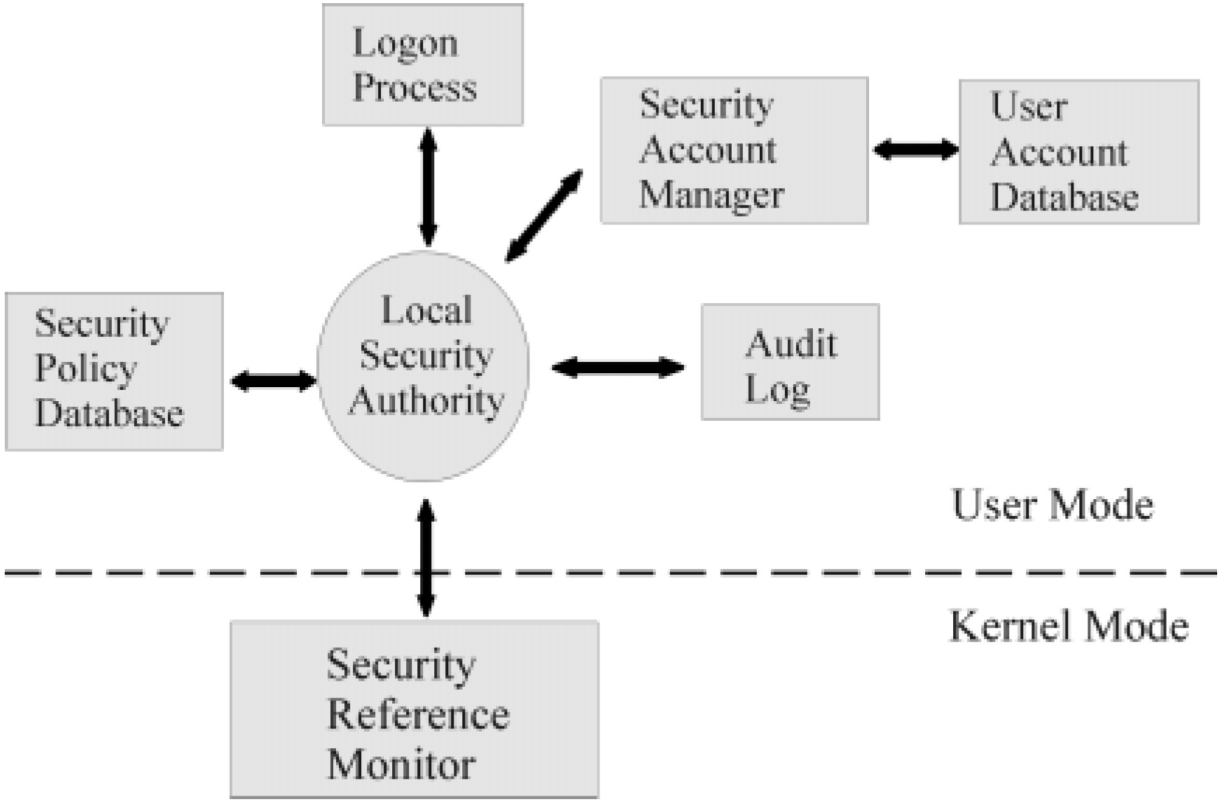
biometric authentication

Security and access protection in W2K

- secure login and antispoofing
- discretionary access control
- privileged access control
- process address space protection
- prevention of data leaks by zeroing all new pages before loading
- security auditing

overall NT security model

http://www.ciac.org/ciac/documents/CIAC-2317_Windows_NT_Managers_Guide.pdf



NT is C2 certified

Local Security Authority - LSA

The LSA is the heart of the security subsystem. It has the responsibility of validating local and remote logons to all types of accounts. It accomplishes this by verifying the logon information from the SAM database. It also provides the following services:

- Checks user access permissions to the system
- Generates access tokens during the logon process
- Manages local security policies
- Provides user validation and authentication
- Controls the auditing policy
- Logs audit messages generated by the SRM

The Security Account Manager - SAM

The Security Account Manager:

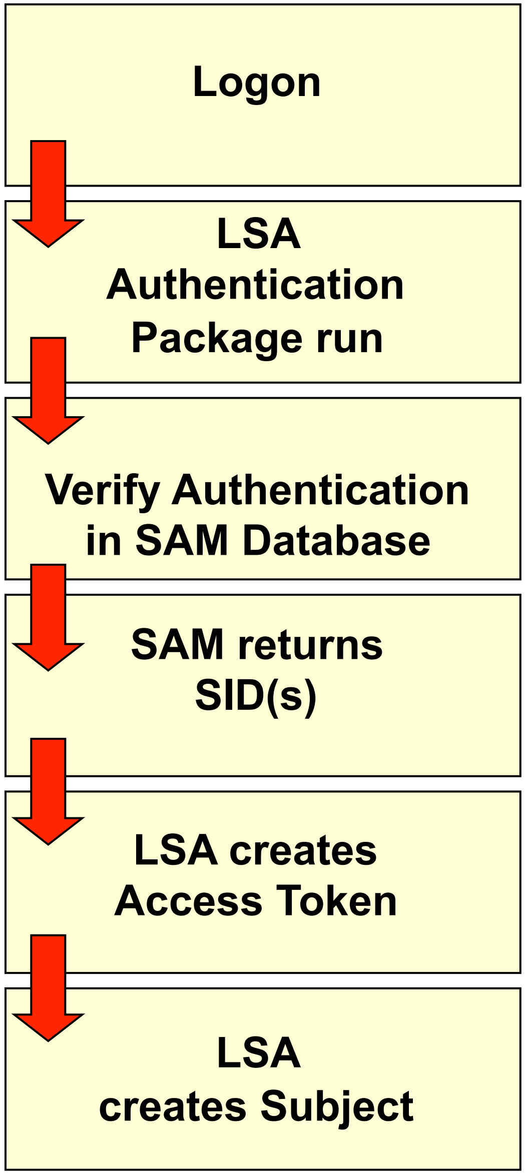
- manages the User Account Database which comprisecall user and group account information.
- provides user validation services which are used by the LSA, and are transparent to the user.
- responsible for checking logon input against the SAM database and returning a secure identifier (SID) for the user, as well as a SID for each group to which the user belongs.
- creates an access token on user logon which includes the SID information along with the user’s name and associated groups.

From this point on, every process that runs under this user's account will have a copy of the access token. When a user requests access to an object, a comparison is made between the SID from the access token and the object’s access permissions list to validate that the user has the correct permissions to access the object.

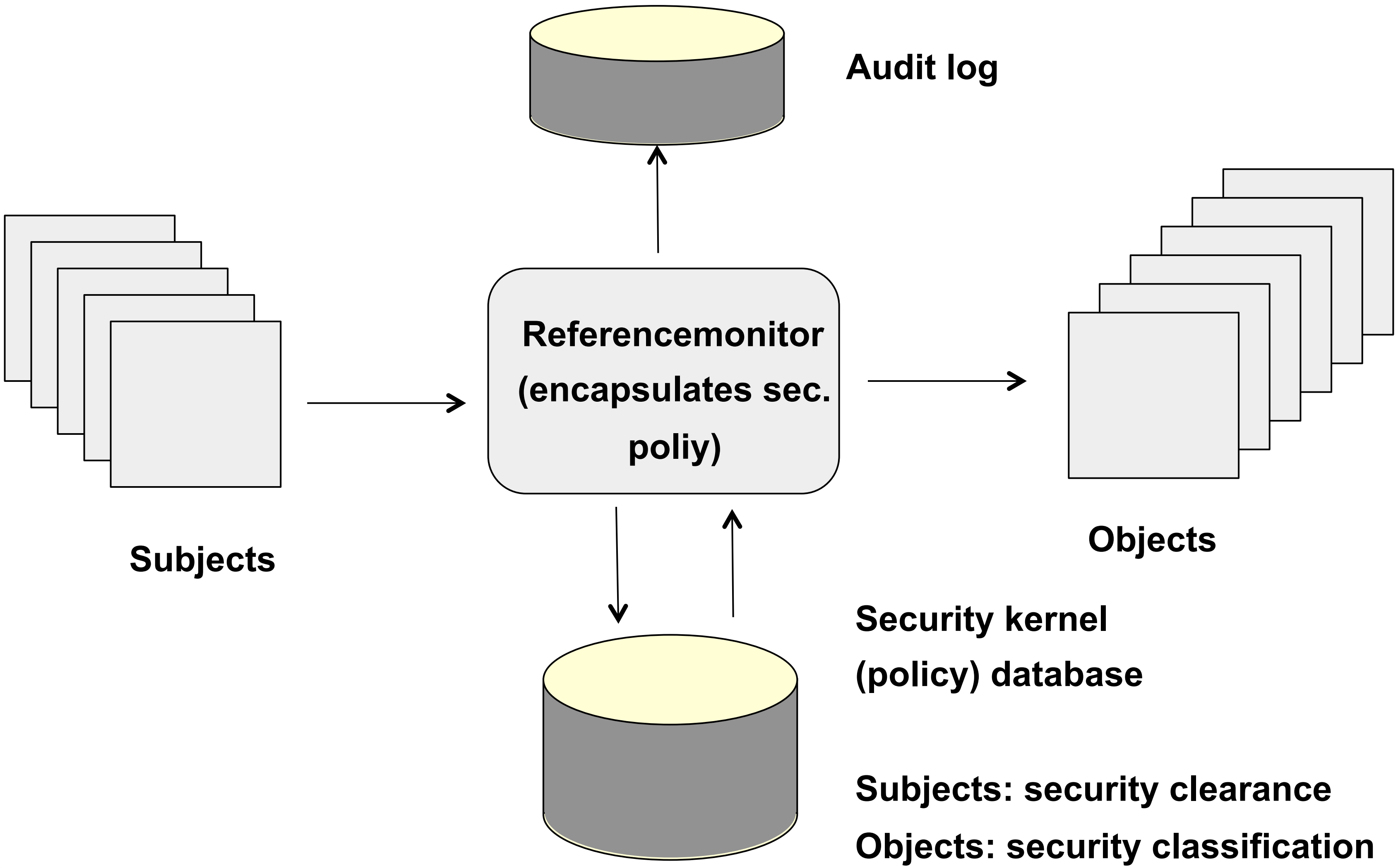
NT logon process

Windows NT logon processes provide mandatory logon for user identification and cannot be disabled.

To protect against spoofing, the logon process begins with a Welcome message box that requests the user to press Ctrl, Alt and Del keys before activating the actual logon screen.



The Security Reference Monitor - SRM



The Security Reference Monitor - SRM

The steps used to determine user access to objects are as follows:

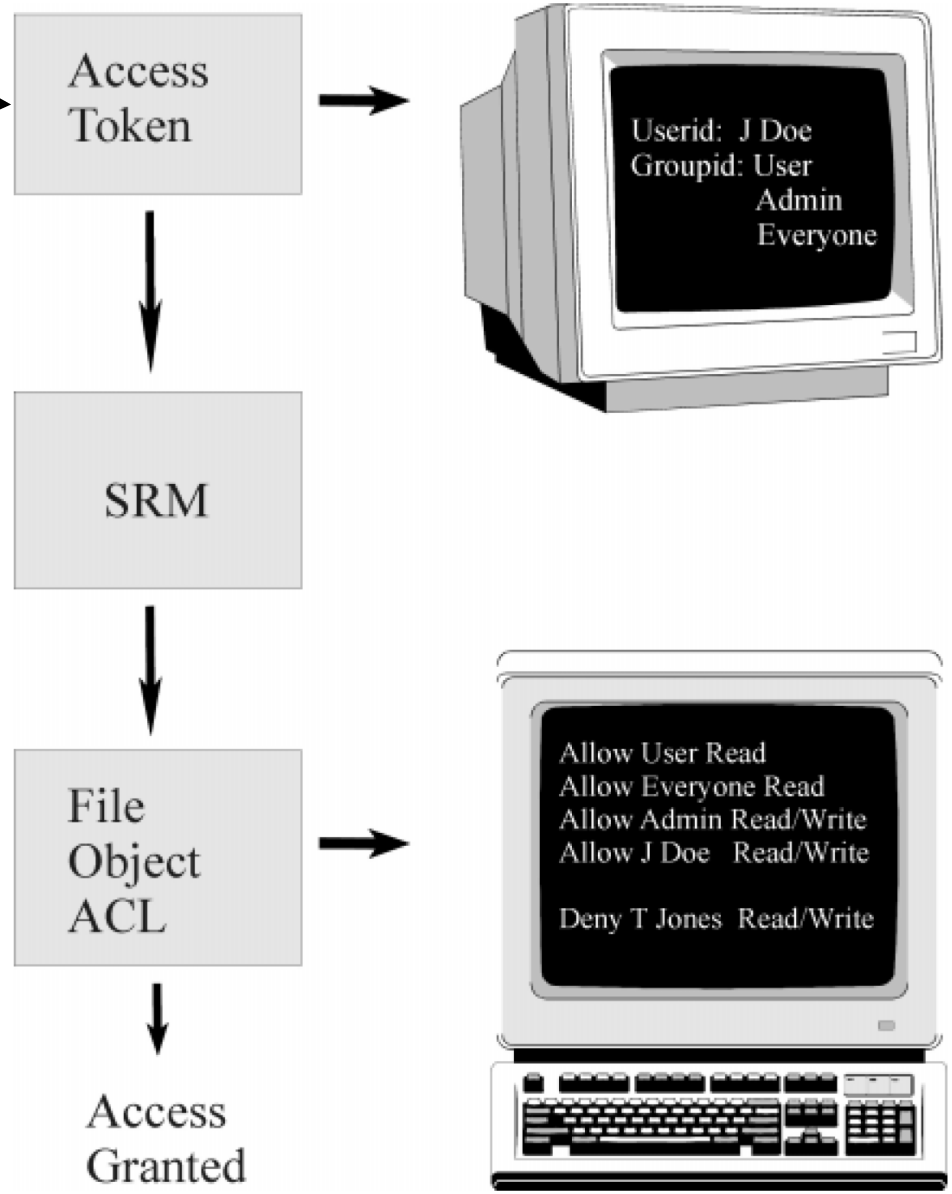
1. When access to an object is requested, a comparison is made between the file's security descriptor and the SID information stored in the user's access token. The user will obtain access to the object given sufficient rights. The security descriptor is made up of all the Access Control Entries (ACE) included in the object's Access Control List (ACL).

2. When the object has an ACL, the SRM checks each ACE in the ACL to determine if access to the object is granted. If the object has no ACL associated with it, SRM automatically allows access to everyone. If the object has an ACL with no ACEs, all access requests to that object will be denied.

3. After the SRM grants access to the object, continued validation checks are not needed to access the particular object. Any future access to the object is obtained by the use of a handle which was created when the access was initially validated.

SRM access validation

LSA creates access token and subject (via SID)



W2K Security Structures

access token

→ owned by a process

Security ID (SID)
group SID
special rights
default owner
default ACL

security descriptor

→ points to the security structures

flags
owner
SACL (System ACL)
DACL (Discretionary ACL)

access control list

→ contains access info

ACL Header
ACE Header access mask SID
ACE Header access mask SID

......

ACL: Access Control List
ACE: Access Control Entry

the access token

header	expir. time	groups	standard DACL	owner SID	group SID	restricted SIDs	privileges

Security ID (SID): The SID is a variable length unique name (alphanumeric character string) that is used to identify an object, such as a user or a group of users in a network of NT/2000 systems.

Expiration time: defines validity interval for the access token (currently not used)

Discretionary Access Control List (D ACL): Default ACL when they are created by a process and no other ACL is specified.

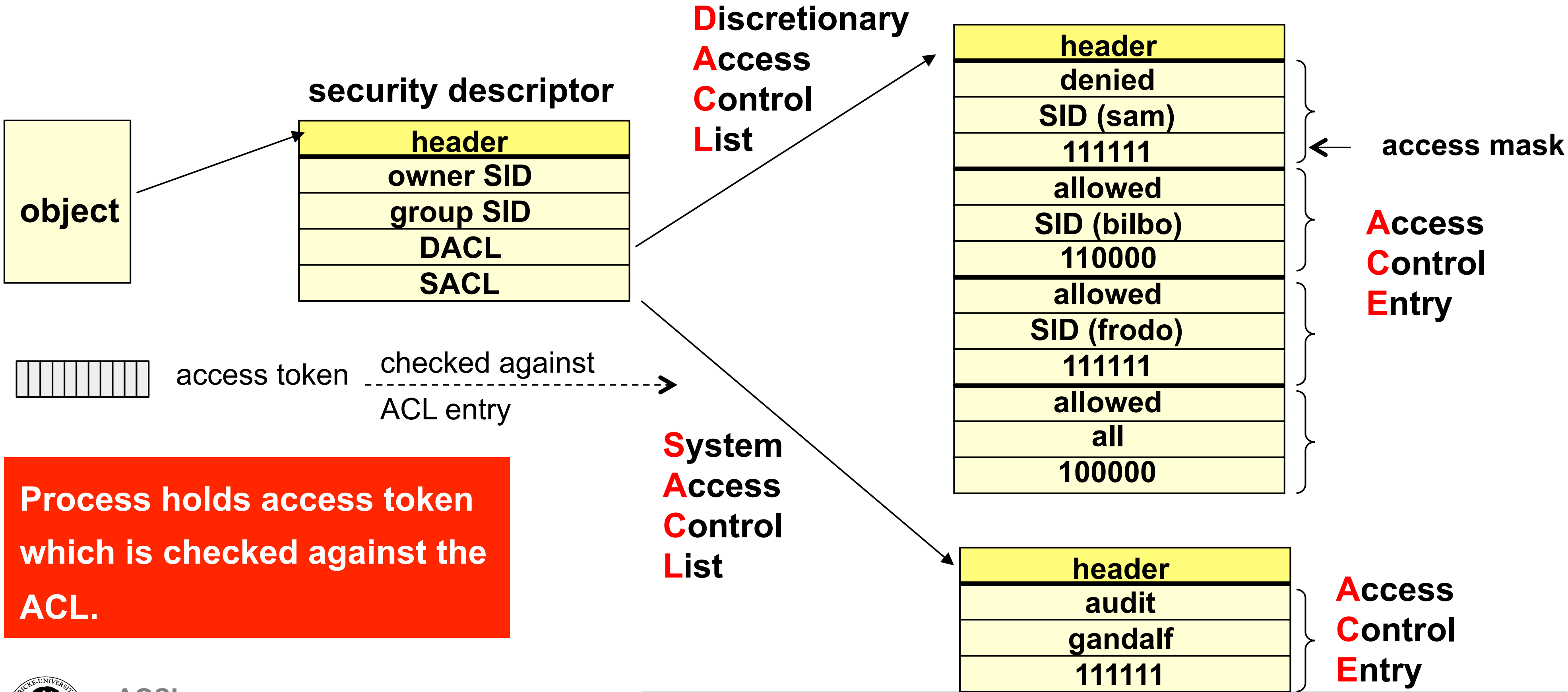
Owner/group SID: indicates the user/group who owns the process.

Restricted SID: enables the cooperation of trusted and non-trusted processes by constraining access for the latter.

Privileges: enable to define "admin rights" in a more fine-grained fashion and associate these with user processes.

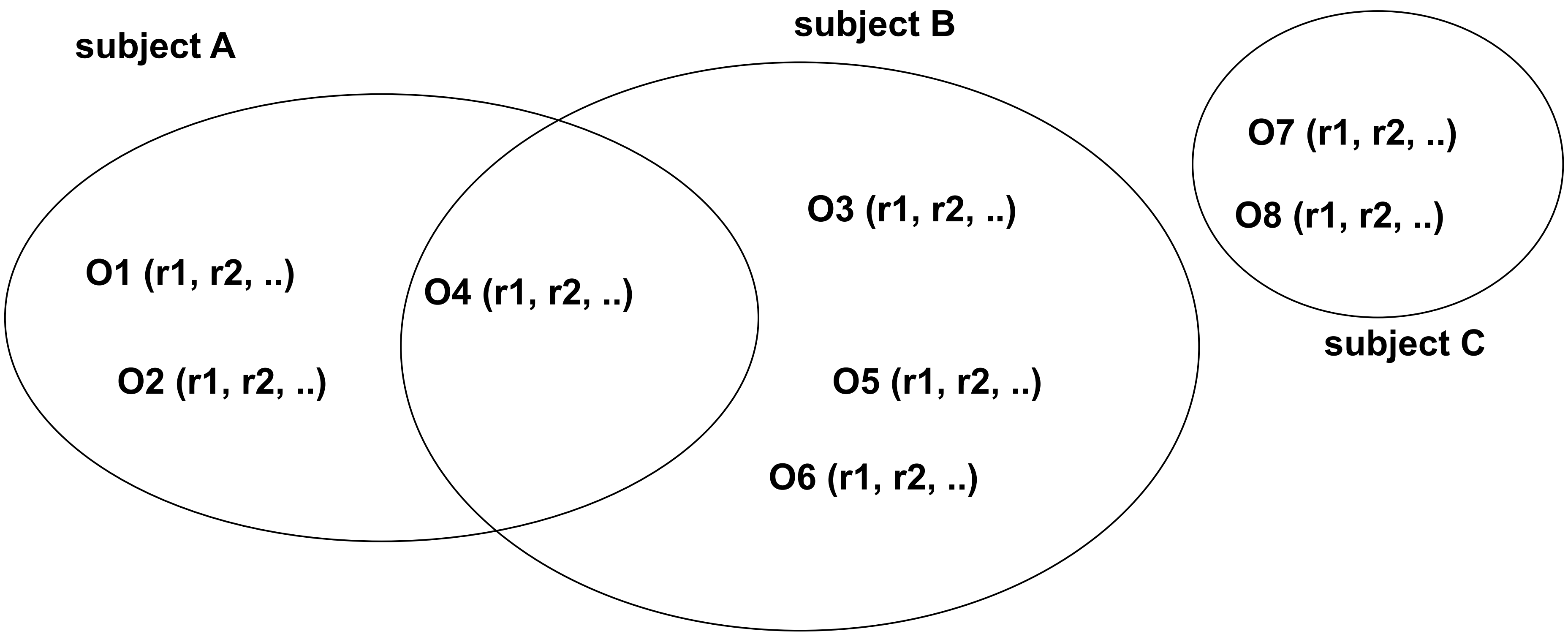
the security descriptor

- is associated with every object
- defines who may access the object with which operation



Process holds access token which is checked against the ACL.

Access control models and least priviledge



Protection Domains define the acess relations between

Active system components: **Subjects**, e.g. users, processes,.. and

Passive system components: **Objects**, e.g. files, devices, ...

The model of access protection

Lampson's model:

paper "Protection" first appeared in *Proc. 5th Princeton Conf. on Information Sciences and Systems*, Princeton, 1971, p 437.

entities are distinguished as:

subjects, taking the active role in the system, and

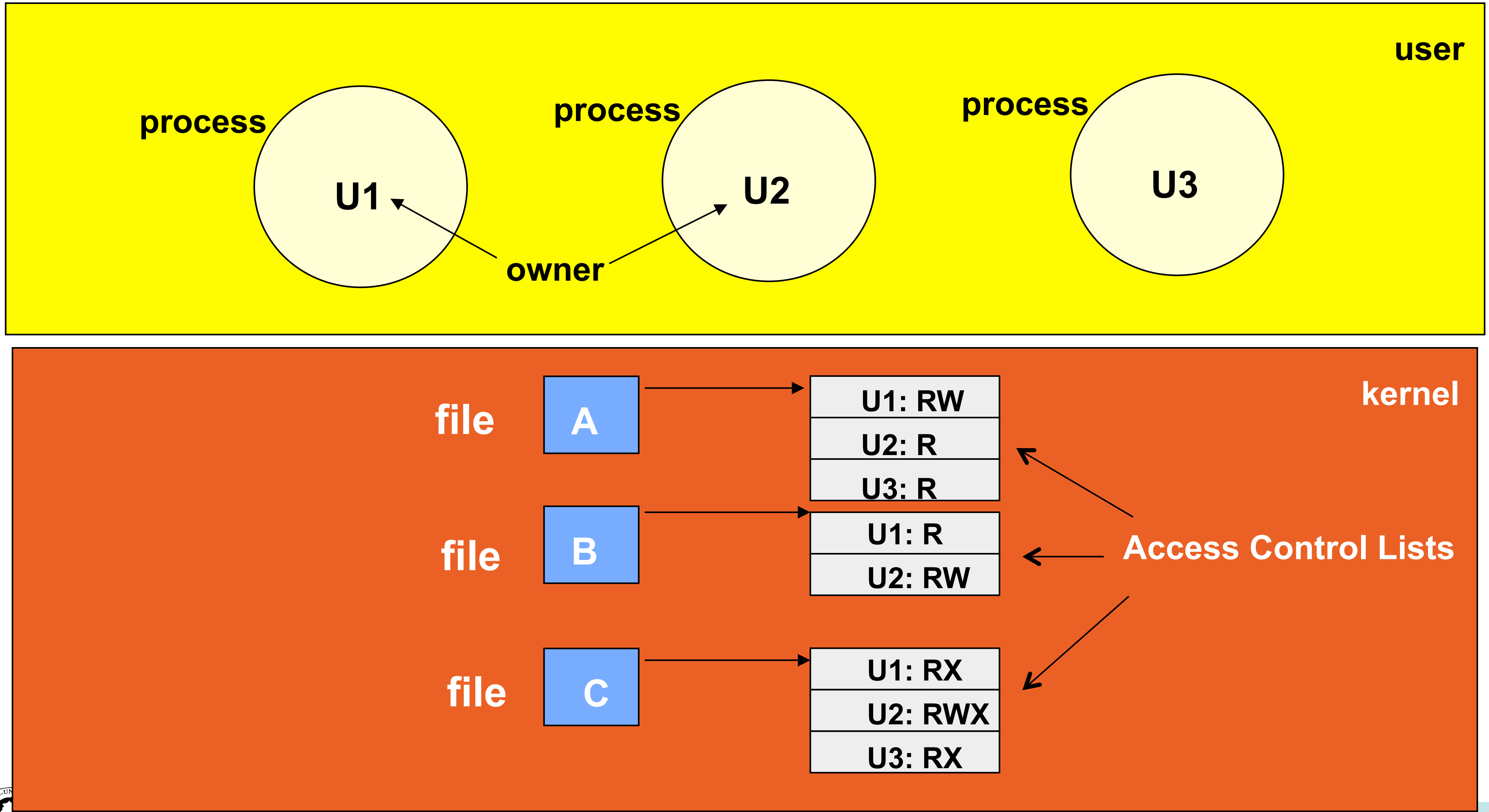
objects, that are passive entities

capability list for s_3

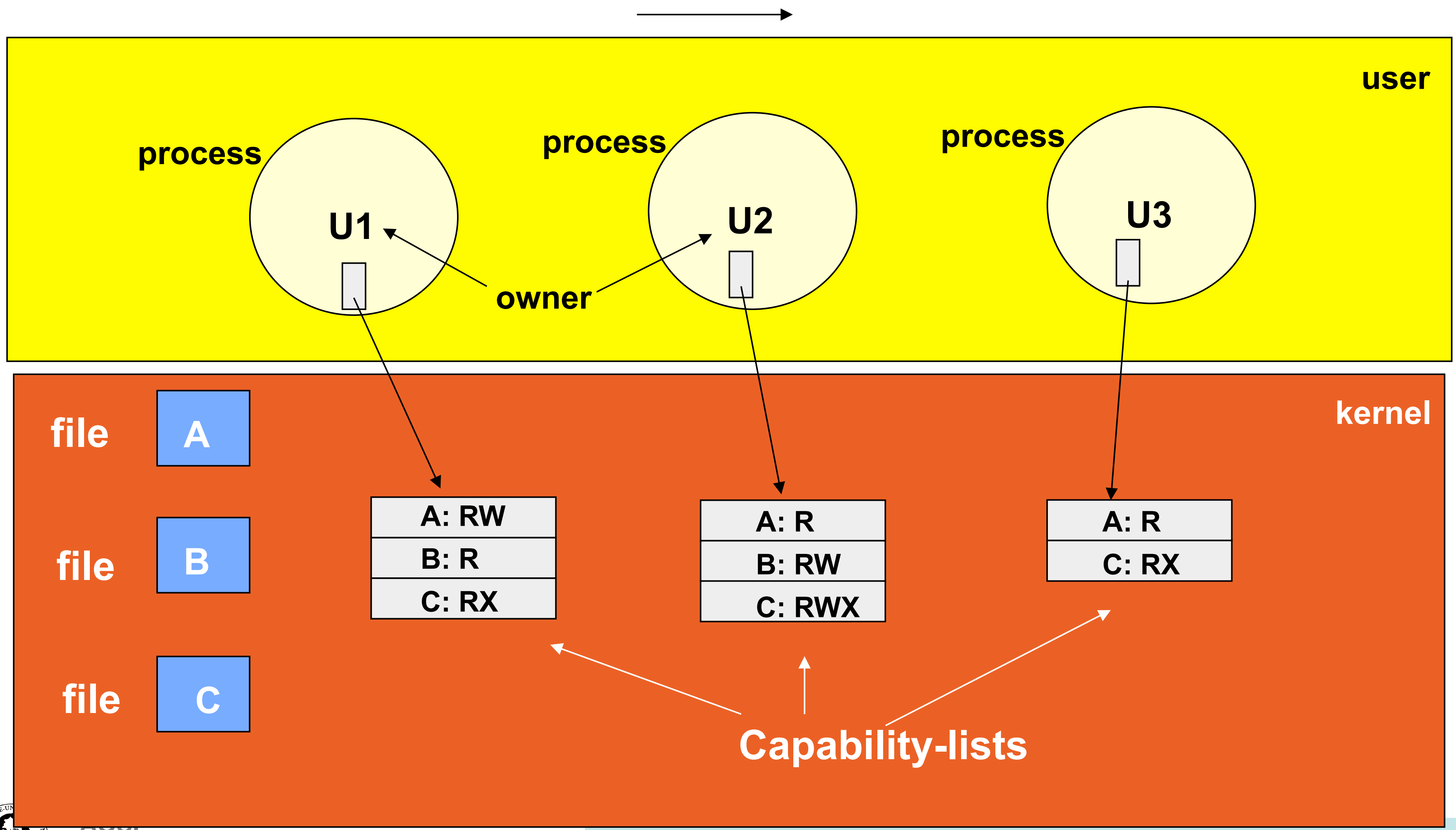
subjects		s_1	s_2	s_3		s_k		s_n
objects	o_1	$R(o_1,s_1)$	$R(o_1,s_2)$	$R(o_1,s_3)$		$R(o_1,s_k)$		$R(o_1,s_n)$
	o_2	$R(o_2,s_1)$	$R(o_2,s_2)$	$R(o_2,s_3)$		$R(o_2,s_k)$		$R(o_1,s_n)$
	o_j	$R(o_j,s_1)$	$R(o_j,s_2)$	$R(o_j,s_3)$		$R(o_j,s_k)$		$R(o_1,s_n)$
	o_m	$R(o_m,s_1)$	$R(o_m,s_2)$	$R(o_m,s_3)$		$R(o_m,s_k)$		$R(o_1,s_m)$

ACL for o_2

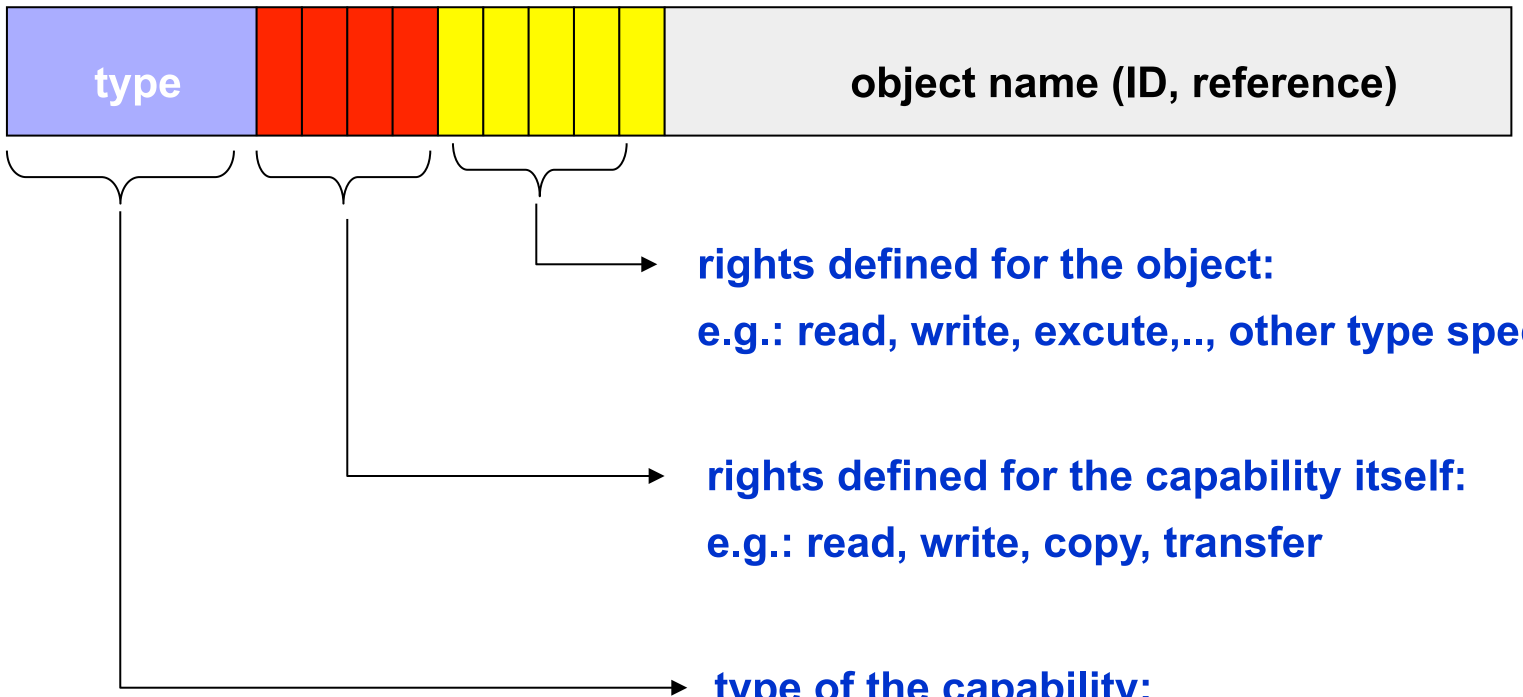
Access Control List (ACL)



Capability List (C-List)



structure of a capability



rights defined for the object:

e.g.: read, write, excute,.., other type specific rights

rights defined for the capability itself:

e.g.: read, write, copy, transfer

type of the capability:

refers to the diffrent system levels on which the cap. is used, e.g. segment cap, file cap, user object..

Discussion:

how to protect capabilities?

Tagging

Separation

Encryption

Sparse name space

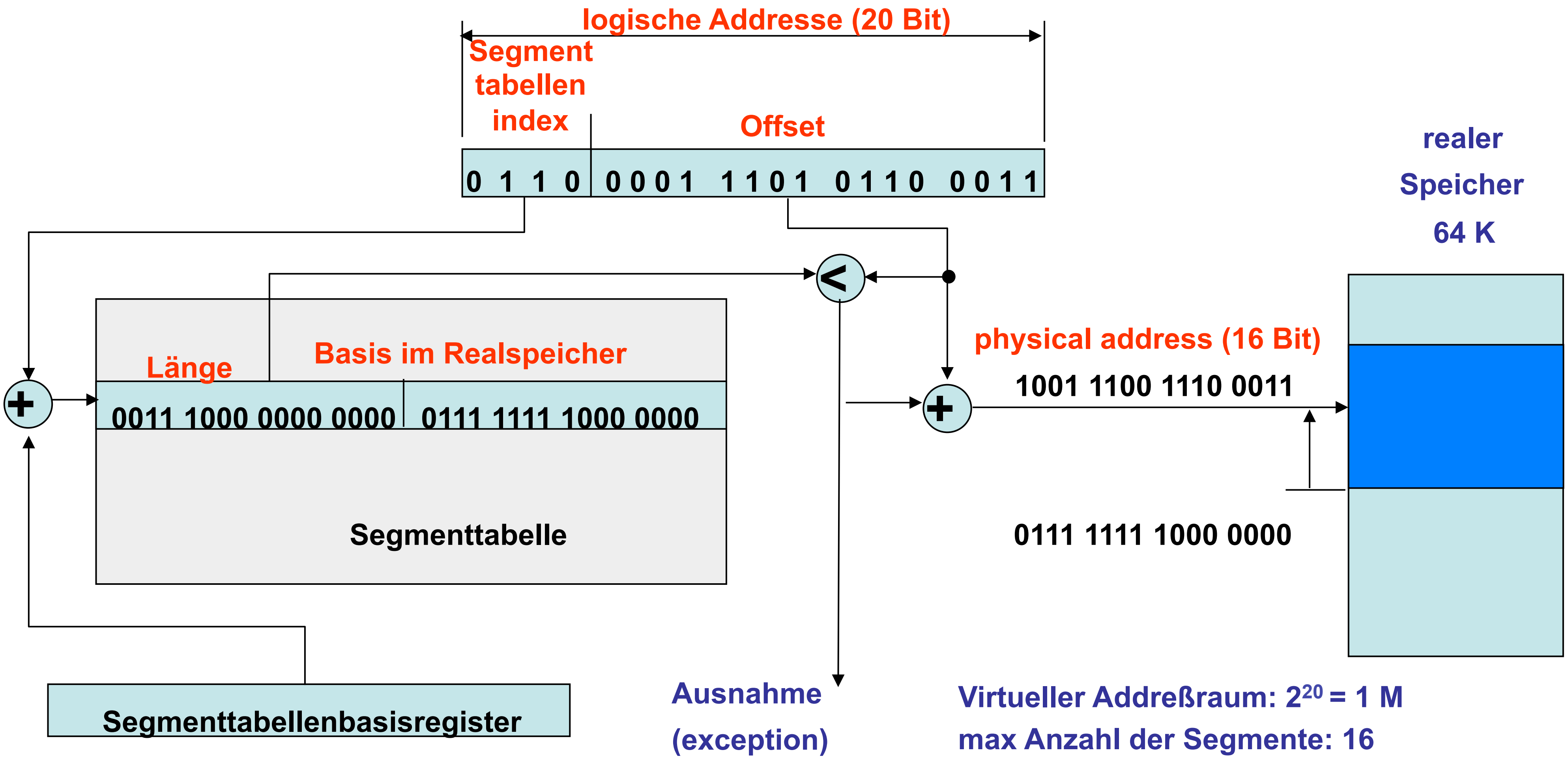
More Problems:

Controlling and confining capability transfer.

Revocation of rights

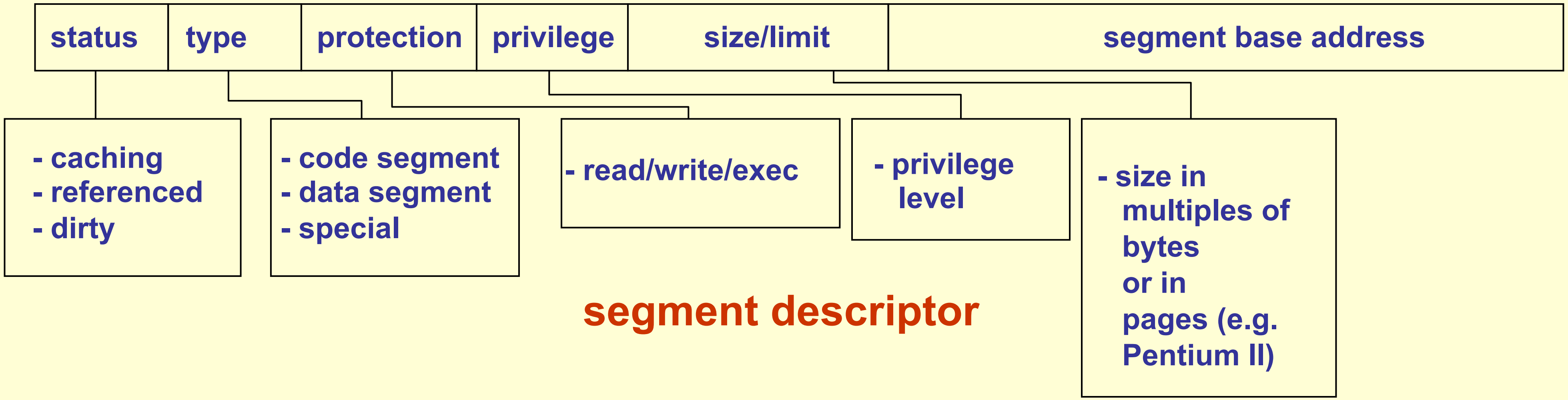
(contradiction of terms according to Roger Needham (1992))

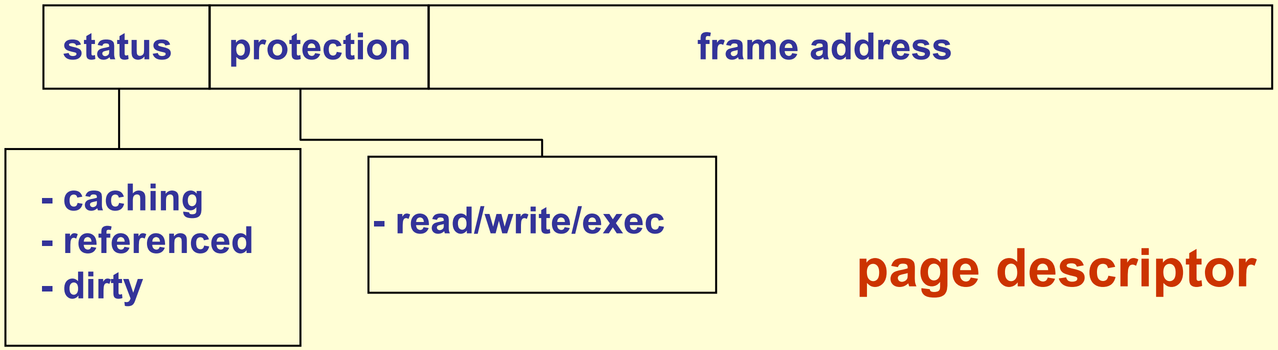
Segmentierter Speicher - Adreßumsetzung



Virtueller Addreßraum: 2^{20} = 1 M
max Anzahl der Segmente: 16
max Größe eines Segments: 64 k

memory protection





Discussion: ACL vs. C-Lists

	ACL	C-List
General mechanism	list based	ticket based
Authentication	every access	on capability creation
Addressability	unrestricted	confined to objects in the C-List
Referencing of objects	extra mechanism	combined mechanism
Transfer of rights	not possible	regulated by specifi rigths
Revocation of rights	easy	problem (possibly not desirable)
Granularity of objects	large objects	small objects

topics:

Overview and Terminology

Security requirements

Threats, adversaries and intruders

Protection mechanisms

Attacks from outside the system

Attacks from inside the system

Security holes

Trusted systems

what cryptography can do for security

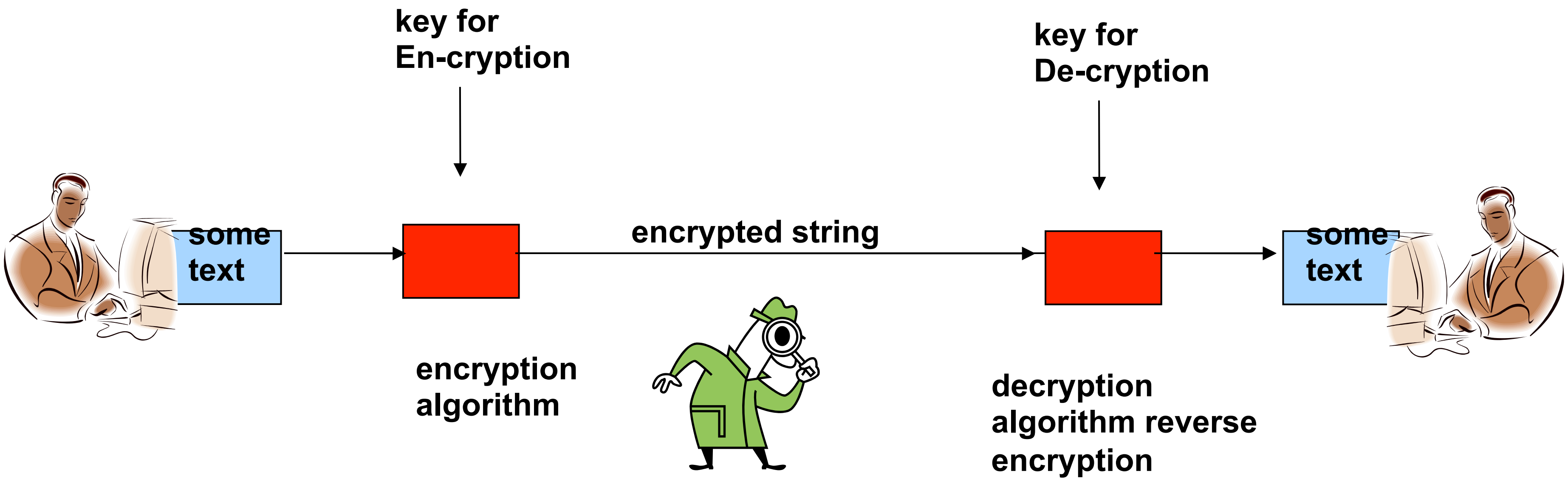
Useful for transmission and storage of data !!

Confidentiality	**encryption of data**
Integrity	**encryption, digital signatures**
Authenticity	**encryption of authentication information**

Mechanisms:

- **one-way functions**
- **cryptographic hash funtions**
- **symmetric cryptosystems with a secret key (DES)**
- **asymmetric cryptosystems with a combination of public/secret key**

Securing the data transfer



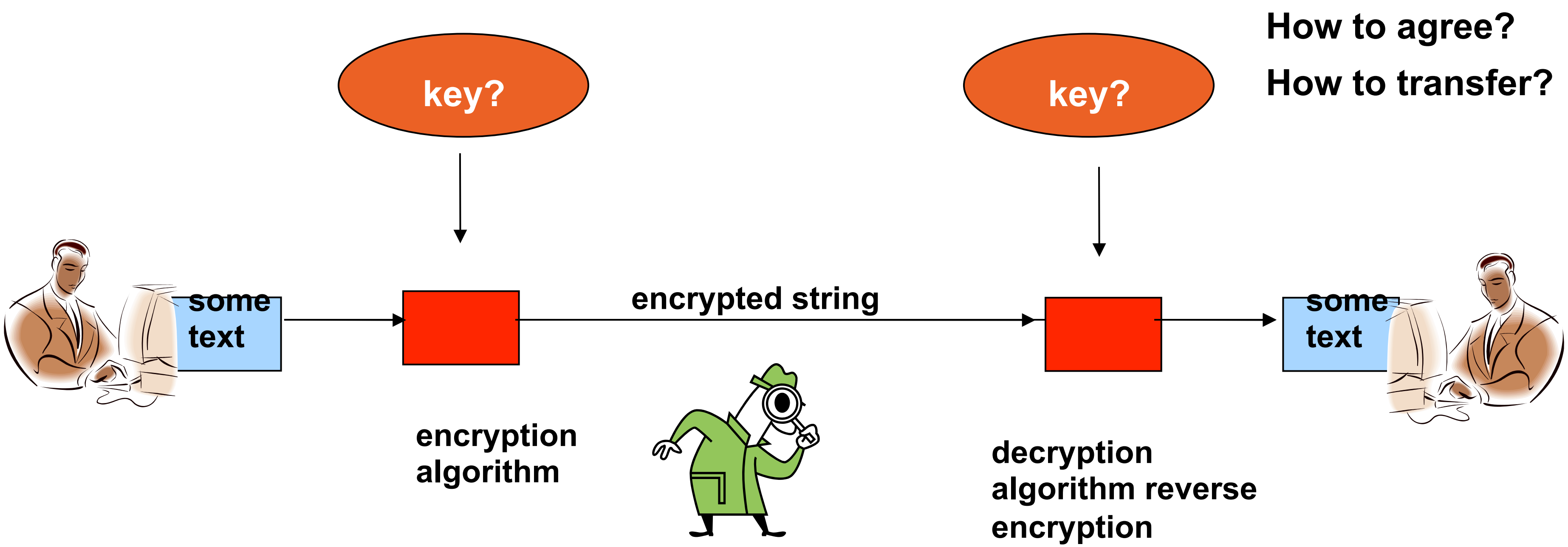
Eavesdropping,

Message faking,

Message corruption,

Masquerade.

Securing the data transfer



Eavesdropping,

Message faking,

Message corruption,

Masquerade.

So far:

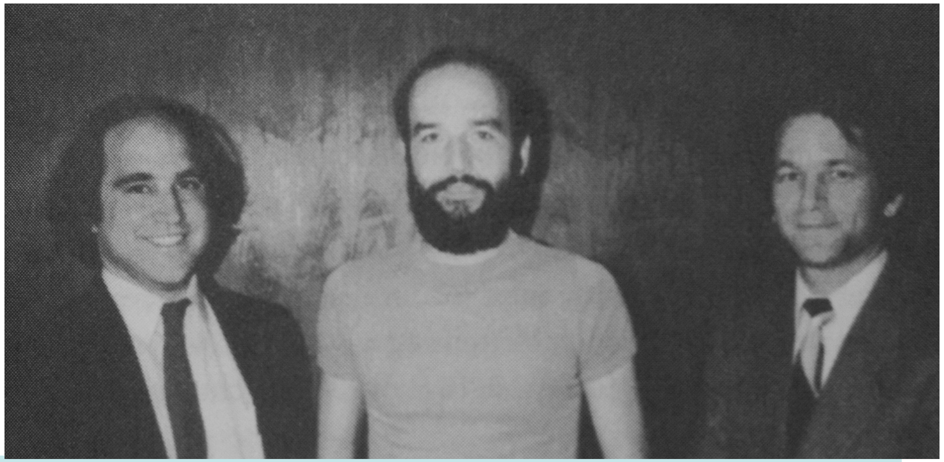
How to create a key not exchanging a secret?

➡ **Merkle, Hellman, Diffie**

How to simplify key exchange and distribution?

Idea: Whitfiled Diffie

➡ **solved by Rivest, Shamir, Adleman**



Sicherer Schlüsselaustausch ohne physisches Treffen. Unmöglich??

Idee Martin Hellman (1976) (Diffie-Hellman-Merkle-Verfahren).

Sicheres vereinbaren von Schlüsseln- kein Austausch einen Geheimnisses.

Entsprechung: Schlüssel in Kiste legen. Kiste mit einem Schloss versehen, an Adressaten schicken. Adressat bringt weiteres Schloss an und schickt die Kiste zurück. Eigenes Schloss wird entfernt und Kiste wieder an Adressaten. Der kann nun sein Schloss entfernen, die Kiste öffnen und den Schlüssel entnehmen. Folgende Kisten müssen nur noch mit dem entsprechenden Schloss gesichert sein.

Problem: Vereinbarung von Schlüsseln erfolgt synchron mit einem konkreten Partner.
umständlich, mehrere Nachrichten müssen ausgetauscht werden, Partner muss gemäß des Protokolls antworten.
Symmetrischer Schlüssel, d.h. ver- und entschlüsseln wird mit demselben Schlüssel durchgeführt.

Public-Key Verfahren:

Idee: Whitfield Diffie

Asymmetrischer Schlüssel. Ver- und Entschlüsselung mit unterschiedlichen Schlüsseln.

Öffentlicher Schlüssel zum Verschlüsseln, Privater Schlüssel zum Entschlüsseln.

Entsprechung: Jeder der eine Nachricht an A bekommt eine Menge von Schnappschlössern. Nur A hat den entprechenden Schlüssel. Wenn das Schnappschloss eingerastet ist, kann nur A die Kiste öffnen.

Gesucht: eine Einwegfunktion, die eine solche Asymmetrie unterstützt. Sie muss sich z.B leicht umkehren lassen (Falltürfunktion) z.B. im Gegensatz zu Passwd-Verschlüsselung.

Erste Veröffentlichung der Idee: 1975 (Diffie)

Rivest 1977 hat die Idee. Unter RSA veröffentlicht (Ronald Rivest, Adi Shamir,Leonard Adleman)

Verschlüsselung: $C = K^e \pmod{(p \cdot q)}$ **Alice ist bekannt sind: p, q, e und K**

Öffentlich sind: $N = p \cdot q$ **und e**

Entschlüsselung: $K = C^d \pmod{(p \cdot q)}$

d berechnet sich aus e,p,q mit: $d \cdot e = 1 \pmod{\varphi N} = 1\ ((\bmod\ ((p-1) \cdot (q-1)))$

Def. Cryptographic Hash-Function

A ***hash function H*** is a transformation that takes an input m and returns a fixed-size string, which is called the hash value h (that is, $h = H(m)$). Hash functions with just this property have a variety of general computational uses, but when employed in cryptography, the hash functions are usually chosen to have some additional properties.

The basic requirements for a **cryptographic hash function** are as follows.

The input can be of any length.

The output has a fixed length.

$H(x)$ is relatively easy to compute for any given x.

$H(x)$ is one-way.

$H(x)$ is collision-free.

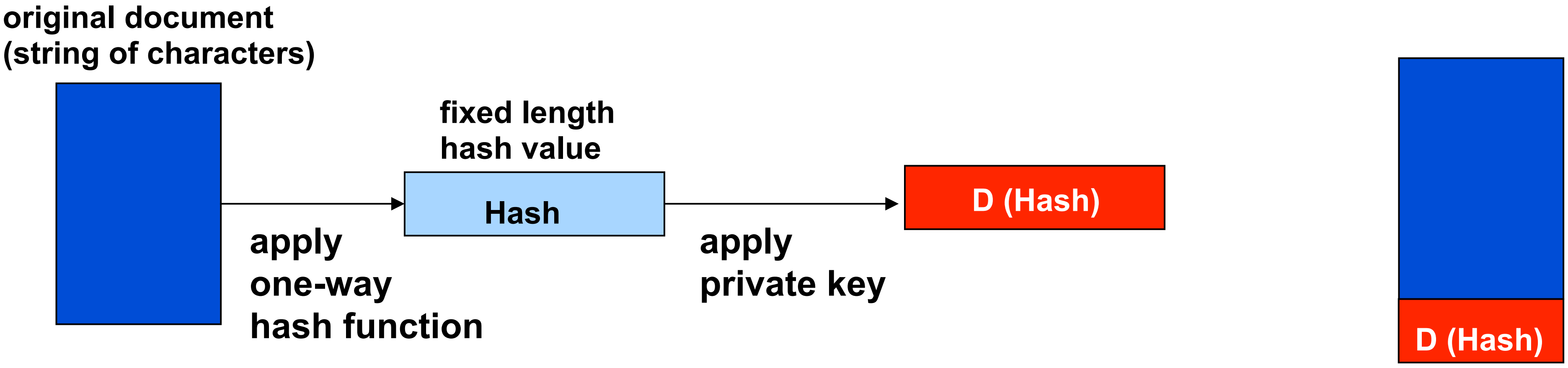
A hash function H is said to be ***one-way*** if it is hard to invert, where ``hard to invert'' means that given a hash value h, it is computationally infeasible to find some input x such that $H(x) = h$.

If, given a string x, it is computationally infeasible to find a string y not equal to x such that $H(x) = H(y)$, then H is said to be a ***weakly collision-free*** hash function.

A ***strongly collision-free*** hash function H is one for which it is computationally infeasible to find any two strings x and y such that $H(x) = H(y)$.

(http://www.rsasecurity.com/rsalabs/node.asp?id=2176)

Example: Digital Signatures



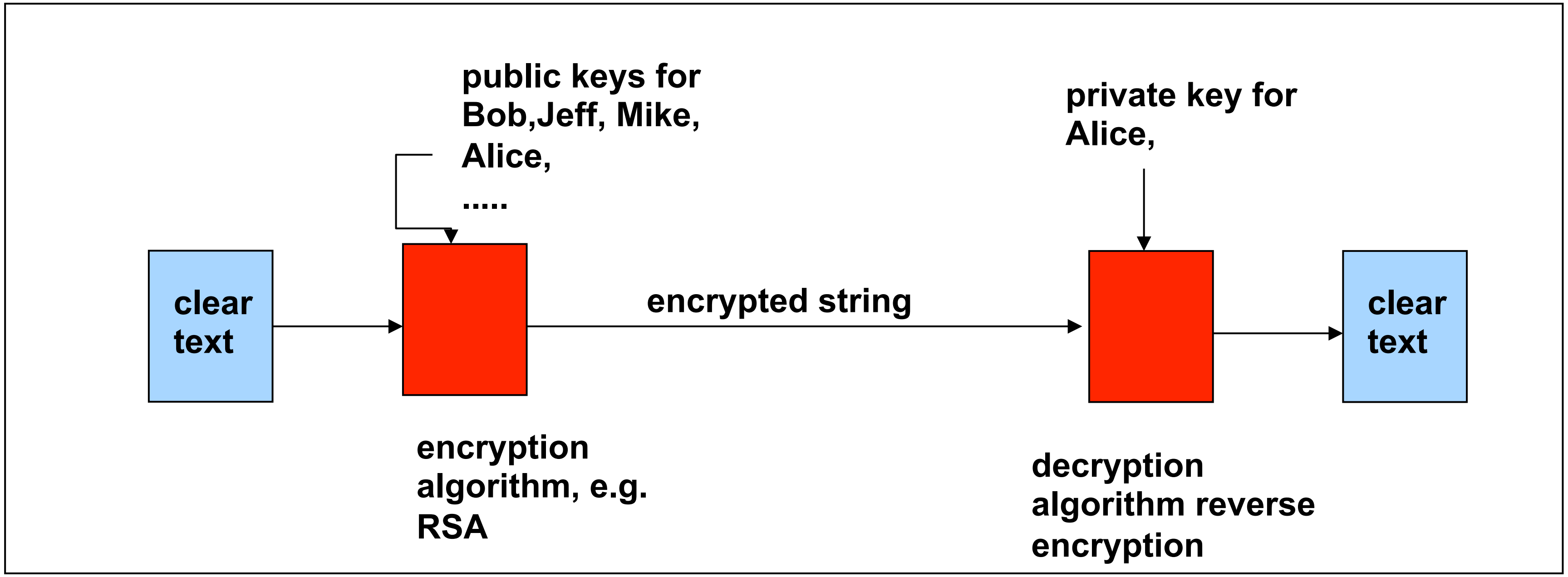
- Receiver calculates the hash value for the document string.
- Receiver applies the public key of the sender E(D(Hash)) to obtain Hash. *
- Then both values are compared and must match.

*Note:it is required that E(D(Hash)) = Hash = D(E(Hash)) !!! This is not true for all encoding functions!

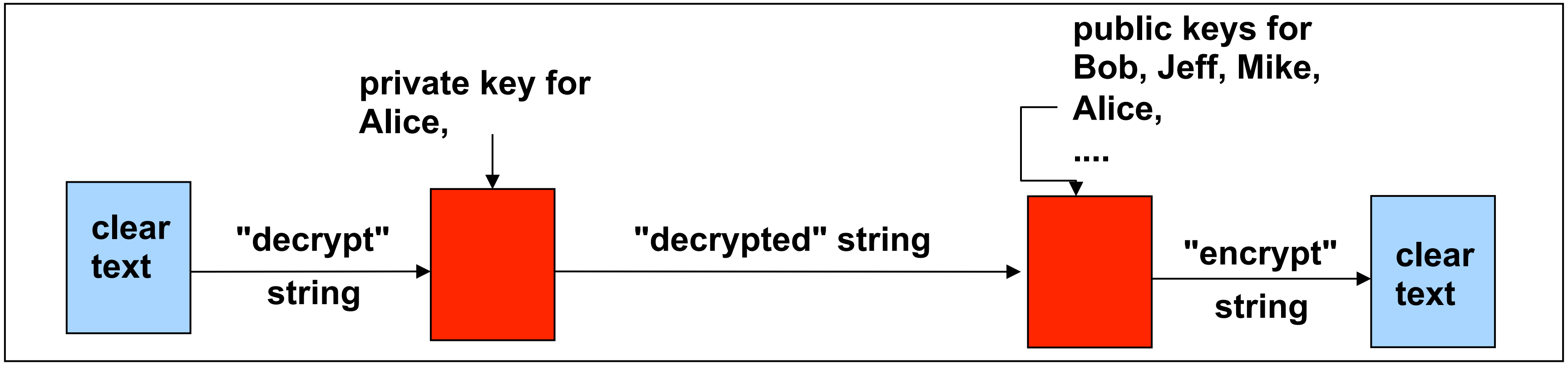
What has to be guaranteed:

1. Integrity of document: this can be checked because the document cannot be changed without changing the hash function ("weakly collision" free property)
2. Authentication of sender: if the document AND the hash value are changed, then applying the public key of the sender to (D(Hash)) will not deliver a correct result.

Public key and Digital Signatures



Public Key: document can only be read by Alice



Signature: everyone can read the doc.

http://www.sans.org/dc2000/wileyhacker.pdf

Attacks to the system

SANS Institute
Washington, D.C. Conference
July 7, 2000

"Hunting the Wily Hacker'

May 1988 vol. 31. No. 5 **COMMUNICATION** 484 **OF THE ACM**

STALKING THE WILY HACKI

An astronomer-turned-sleuth traces a German trespasser on our military netv through operating system security holes and browsed through sensitive databases. Was

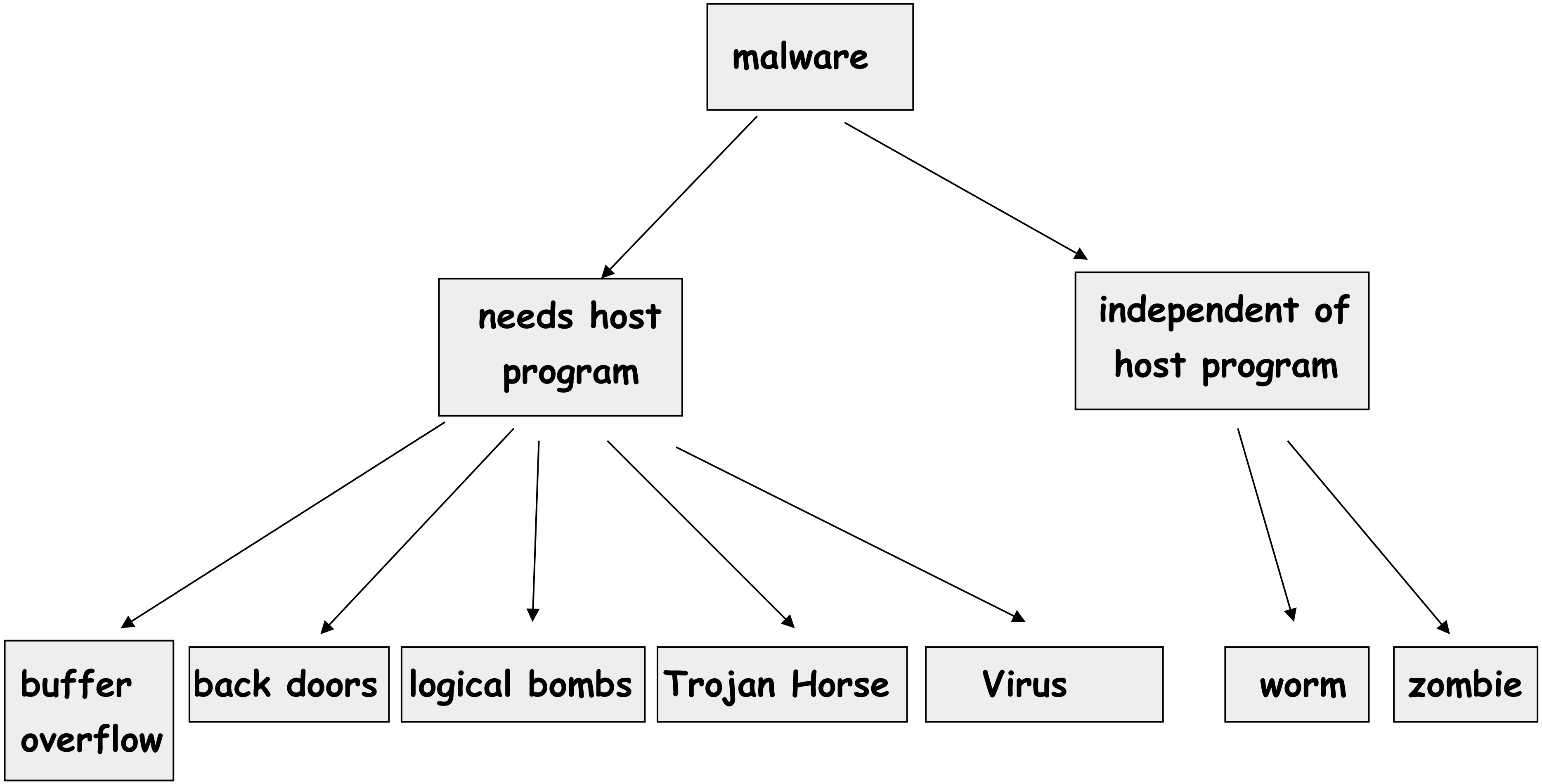
CLIFFORD STOLL

Craig W. Sorum
Supervisory Special Agent
FBI Headquarters
202.324.0322
craig.sorum@fbi.gov

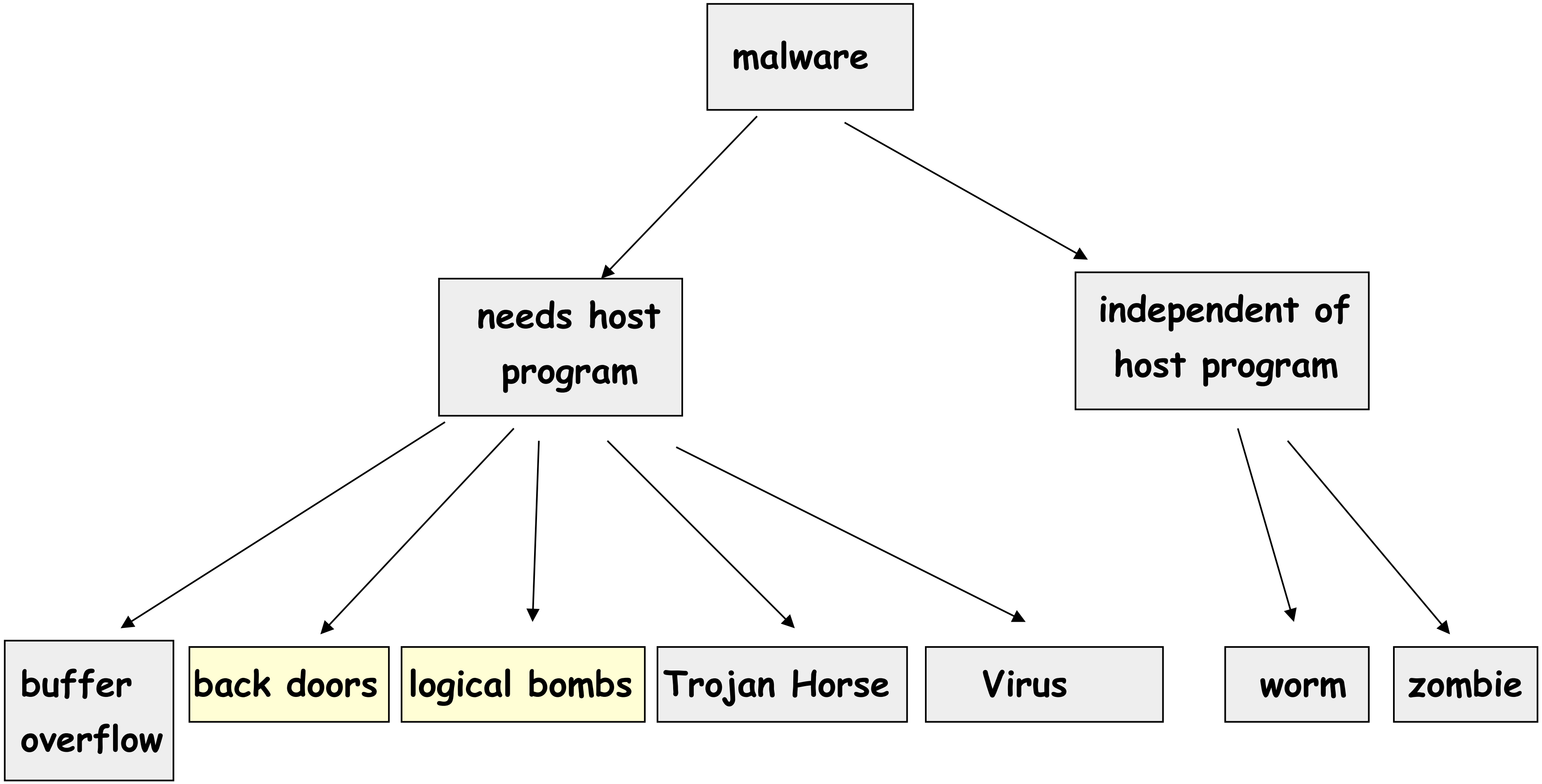
Gary Harter
Special Agent
FBI-Washington Field Office
703.762.3024
gharter@leo.gov

Thank you for your attention!

attacks to the system



attacks to the system



hidden back doors

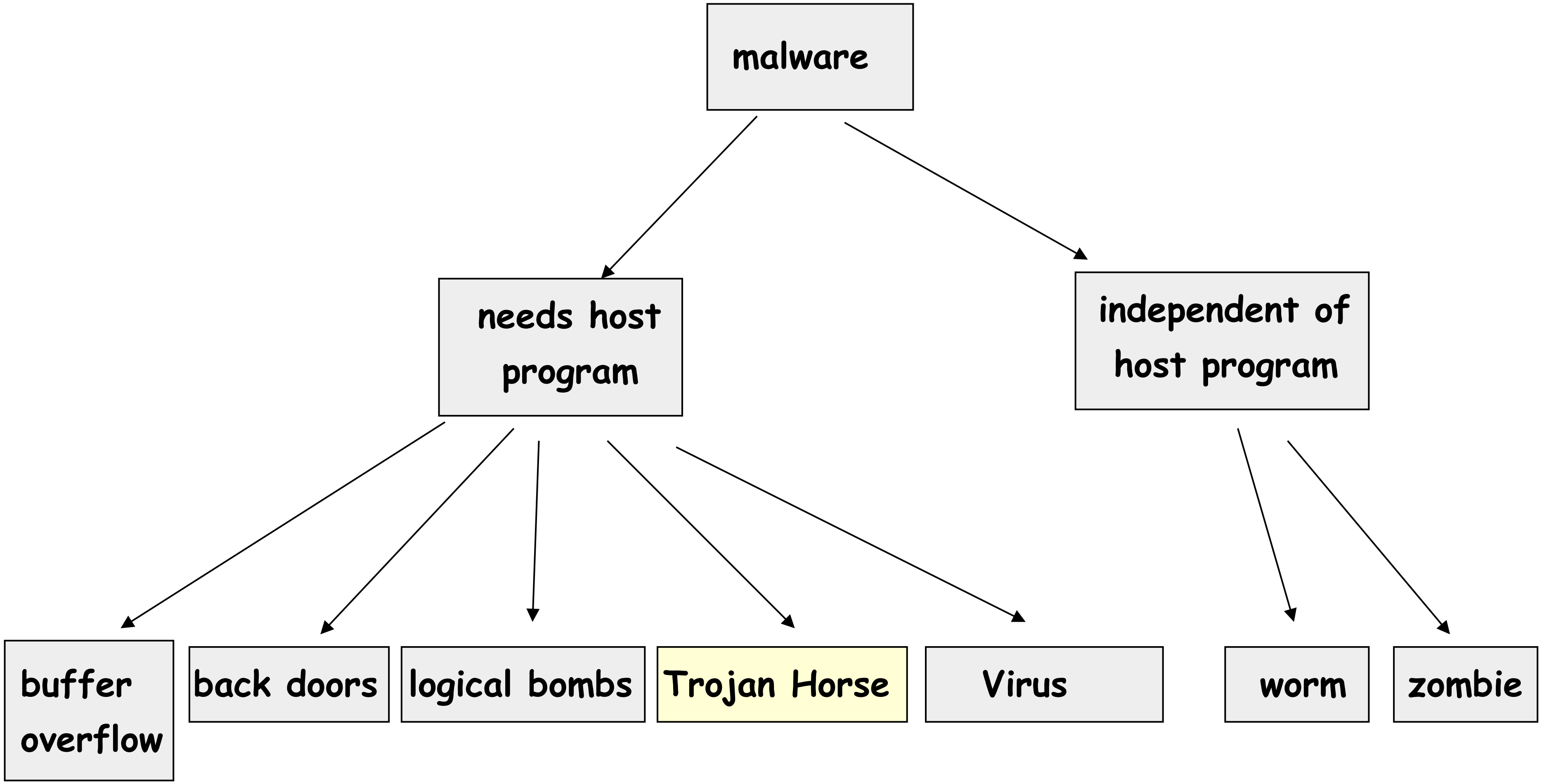
nomal code

```
while (TRUE) {
        printf("login: ");
        get_string(name);
        disable_echoing();
        printf("password: ");
        get_string(password);
        enable_echoing();
        v=check_validity(name,password);
        if (v) break;
}
execute shell(name);
```

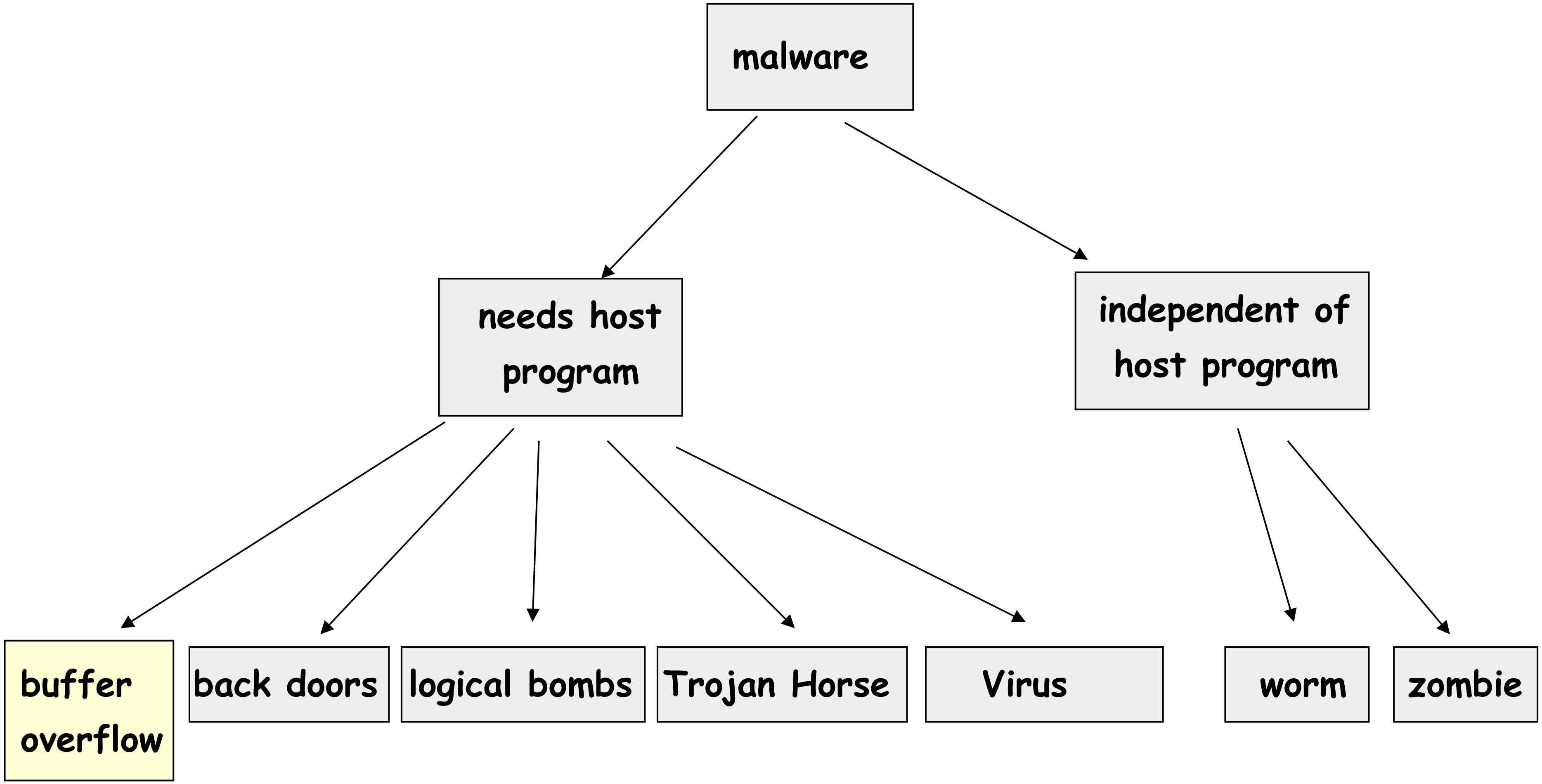
code with back door

```
while (TRUE) {
        printf("login: ");
        get_string(name);
        disable_echoing();
        printf("password: ");
        get_string(password);
        enable_echoing();
        v=check_validity(name,password);
        if (v II strcmp(name, "z!5%zy?" == 0) break;
}
execute shell(name);
```

attacks to the system



attacks to the system



buffer overflow

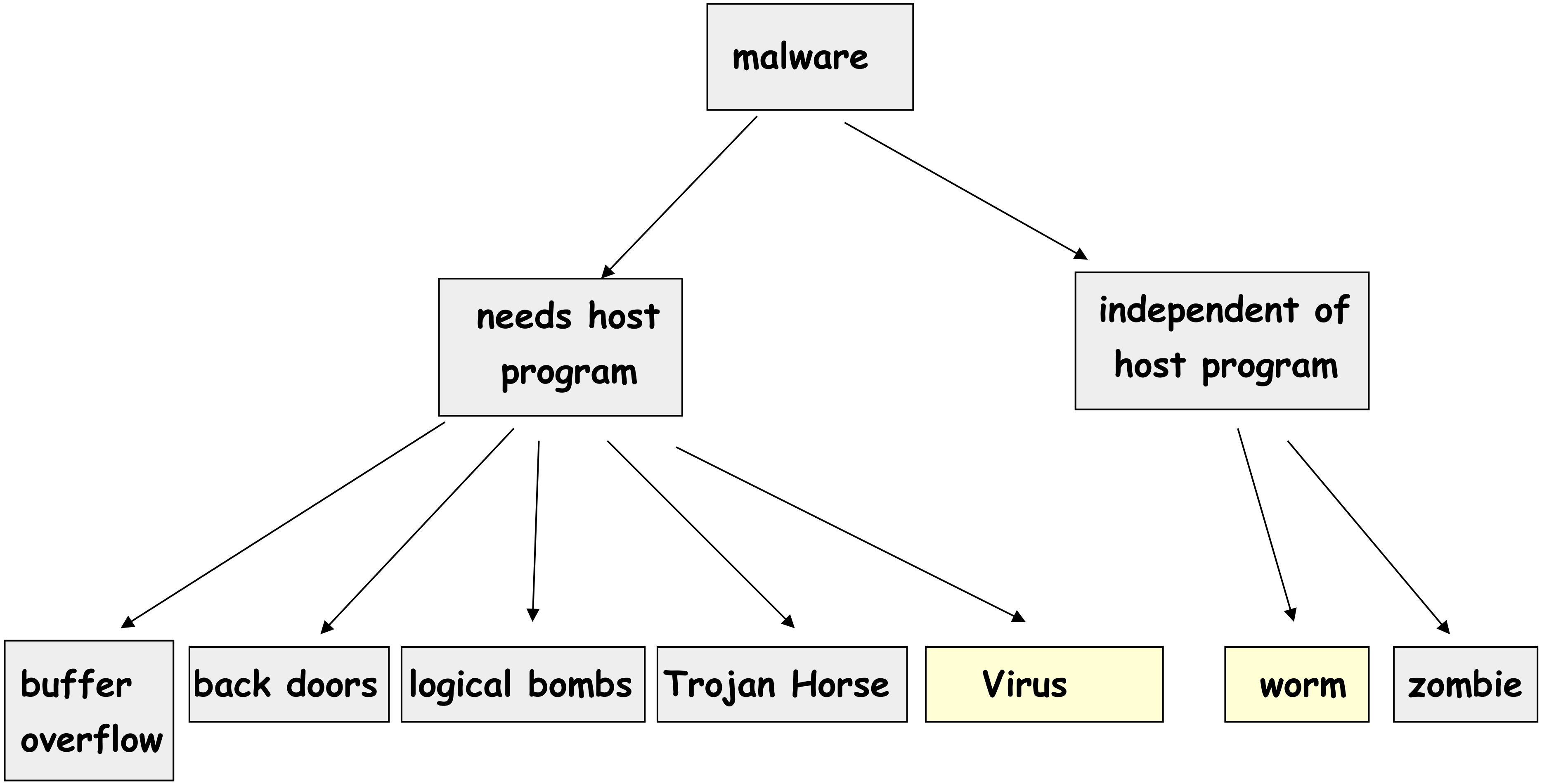
Problem: C-Compiler doesn't check index bounds on arrays.

example:

```
int i;
char c[1024];
i=12000;
c[i]=0;
```

Effect: overwrites a byte that is 10976 Bytes away from the index bound.

attacks to the system



attacks from outside of the system

Viruses and Worms:

Virus: needs host prgram which is explicitely invoked and executed by a user

Worm: autonomous program which acts completely independent from a user.

Hoax: needs (fooled) user to perform action

- **attack over the network**
  (or any infected storage device for virus)
- **transfer executable code to the victim machine**
- **often as e-mail attachment (virus)**
- **replication and distribution by the infected machine**

Geschichte der Computerviren

1950er Bell Labs entwickeln ein experimentelles Spiel, in dem die Spieler gegenseitig ihre Computer mit Schäden verursachenden Programmen angreifen.

1975 John Brunner, Autor von Science-Fiction-Romanen, entwickelt die Idee von einem „Wurm“, der sich in Netzwerken verbreiten kann.

1984 Fred Cohen führt in einer Dissertation den Begriff „Computervirus“ für Programme mit den entsprechenden Eigenschaften ein.

1986 Der erste Computervirus, *Brain*, wird angeblich von zwei Brüdern in Pakistan geschrieben.

1987 Der Wurm *Christmas tree* legt das weltweite IBM-Netzwerk lahm.

1988 Der *Internet worm* verbreitet sich im US-DARPA-Internet.

1992 Der *Michelangelo* -Virus sorgt weltweit für Panik, obwohl nur wenige Computer infiziert werden.

1994 *Good Times*, der erste richtige Virenhoax, erscheint.

1995 Der erste Dokumentenvirus, *Concept*, erscheint.

1998 *CIH* oder *Chernobyl* ist der erste Virus, der Computer-Hardware beschädigt.

1999 *Melissa*, ein Virus der sich selbst per E-Mail weiterleitet, verbreitet sich weltweit. *Bubbleboy*, der erste Virus, der einen Computer allein durch das Lesen einer E-Mail infiziert, erscheint.

2000 Der *Loveletter-Virus* ist der bisher „erfolgreichste“ Virus. Im selben Jahr tritt der erste Virus für das Palm-Betriebssystem auf, allerdings werden keine Anwender infiziert.

2001 Ein Virus, der angeblich Bilder der Tennisspielerin Anna Kournikova enthält, infiziert Tausende Computer weltweit.

2002 David L Smith, Autor von *Melissa*, wird von US-Gerichten zu 20 Monaten Haft verurteilt.

2003 Der *Blaster*-Wurm verbreitet sich mit Hilfe einer Sicherheitslücke in der Software von Microsoft im Internet. Gemeinsam mit dem E-Mail-Virus *Sobig* macht er den August 2003 zum bisher schlimmsten Monat der Virenvorfälle.

2004 Die Schöpfer der *Netsky*- und *Bagle*-Würmer wetteifern, welcher Wurm wohl die größeren Auswirkungen hat.

http://www.sophos.de/sophos/docs/deu/comviru/viru_bde.pdf

ich hatte die Datei auf der Festplatte und habe sie inzwischen gelöscht!

---Ursprüngliche Nachricht---
From: "a friend"
To: "a friend"
Subject: Achtung Viruswarnung Adressbuch - DRINGEND (fwd)

---Ursprüngliche Nachricht---
From: "Gasthof Alpenhof" <gasthof.alpenhof@rolmail.net>
Habe heute diese Virusmeldung bekommen und den Virus in meiner Datei auch gefunden! Bitte die Anleitung zum Löschen befolgen!

Grüße Renate

> > Ich hoffe, dass Ihr diese Nachricht rechtzeitig erhaltet. Der Virus verbreitet sich von Adressbuch
> > zu Adressbuch, also bitte gleich nachschauen. Er ist in der Tat von
> > Norton und McAfee (und AntiVir 9x) nicht auffindbar. Er schlummert etwa
> > 14 Tage auf dem Rechner, aktiviert sich dann selbst und löscht sämtliche
> > Daten auf der Festplatte.
> >
> > Die Anweisung zu seiner Entfernung ist recht einfach:
> > 1. Auf "Start" klicken, dann auf "Suchen", dann auf Dateien/Ordner
> > 2. In der Suchmaske "sulfnbk.exe"eintippen - so heißt die Virusdatei
> > 3. Bei "Suchen in" muß die Festplatte drin stehen, in der Regel C:
> > 4. Suche starten
> > 5. Wenn diese Datei auftaucht (sie hat ein häßliches schwarzes Icon)
> > - AUF KEINEN FALL ÖFFNEN
> > 6. Mit der rechten Maustaste den Dateinamen anklicken - Löschen
> > drücken
> > 7. Bei der Rückfrage ob die Anwendung tatsächlich in den Papierkorb
> > verschoben werden soll, Ja drücken
> > 8. Auf den Desktop gehen und den Papierkorb öffnen
> > 9. Die Datei "sulfnbk.exe" im Papierkorb suchen und mit der rechten
> > Maustaste löschen
> >
> > Wenn Sie/Ihr die Datei auf Eurem Rechner gefunden habt, sendet diese
E-Mail
> > an alle Kontakte in Ihrem/ Eurem Adressbuch, weil der Virus über das
> > Adressbuch verbreitet wird.
> >Danke!

Sorry!!!!

---Ursprüngliche Nachricht--- From: "Dr. S" <--------------->

To: <----------->,"'GK'" <gk>

Subject: "Hoax" (eben kein Virus)

AW: Von Renate - Achtung Viruswarnung Adressbuch - DRINGEND (fwd)

> Ich hatte die Datei auf der Festplatte und habe sie nun gelöscht! .. selbst schuld ..

Bei dieser Nachricht handelt es sich um einen sogenannten Hoax, die Weitergabe der Nachricht ist das Problem, die u.g. Datei ist ein normaler Bestandteil von Windows (z.B. W'98) und dient der Wiederherstellung langer Dateinamen. Wobei ich vermute, dass genug Psychologen in diesem Verteiler sind, die eine derartige sich selbst erfüllende Prophezeiung (die Datei hat wirklich jeder ..) erkennen können ... > > 2. In der Suchmaske "sulfnbk.exe"eintippen - so heißt die Virusdatei

Dr. med. Dipl.-Psych. S,

Zentrum für Telematik im Gesundheitswesen

What a virus can do:

- **Slow down of E-Mail**. e.g. *Sobig*.
- **Theft of confidential data. e.g.***Bugbear-D*
- **Website-attacs from YOUR computer**. e.g. *MyDoom*
- **Misuse of YOUR computer by others.**
- **Change of data**. e.g.*Compatable*
- **Deletion of data**. e.g. *Sircam* Wurm
- **Disable hardware**. *CIH* oder *Chernobyl*
- **Jokes**. e.g.*Netsky-D*
- **Display text messages**. e.g. *Cone-F*
- **Loss of credibility**.
- **Embarrasment**. e.g. *PolyPost*

Virus species

kind:

- companion
- overwriting virus
- parasitic virus
- macro virus
- source code virus

components:

- user programs
- system programs
- device drivers

where to hide:

- "cavities" in the program
- interrupt vector area
- in a memory block marked "used"
- boot sector

how to hide:

- stealth virus
- polymorphic virus

virus actions

Infect and hide in a program:

1. **sleep until wake-up by some event**
2. **start code of virus**
3. **search for executable program files**
4. **infect program file**
    - **overwrite code with virus code (overwriting virus)**
    - **leave original functionality but add code (parasistic virus)**
    - **special case "cavity virus".**

Infect and hide in the computer memory (memory resident virus)

hide in the interrupt vector area

modify bitmap of virtual memory or file system

hide in the boot sector of the disk (will not be destroyed by formatting)

Anti-virus techniques

Isolate and identify the virus:

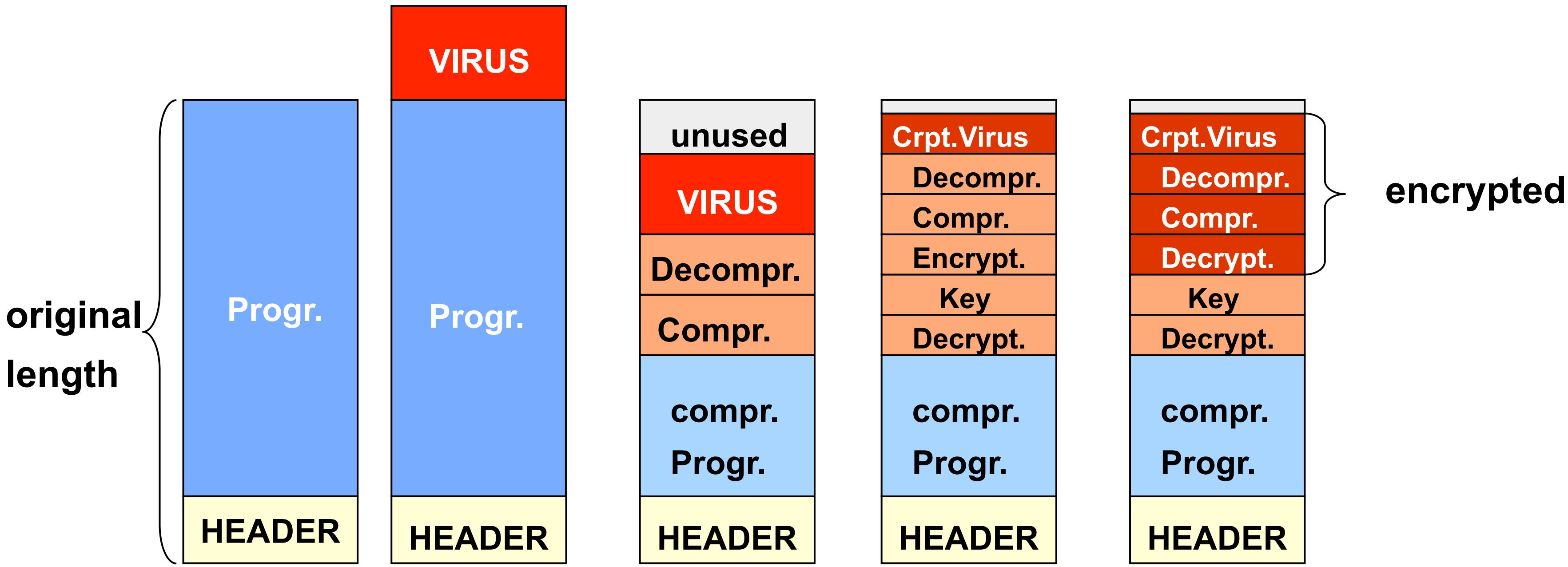
- create a protected environment where the impact of a virus can be tested
- controlled infection of a specific "goat" file. Goal: Isolation of the virus.
- create a listing of the virus code and enter this in a virus database
- isolate the code of the virus kernel and create the virus signature

Function of the virus scanner:

- compare every file on the disk against the data base of viruses
- fuzzy search is required
- use signature to identify viruses
- use creation date to find modifications since the last check
- use length of data to detect infections

Anti-Anti-virus techniques

- setting creation and modification dates
- exploiting compression and decryption techniques to maintain original length and varying the signature.



Anti-Anti-virus techniques

Polymorphic Virus: The many ways to express NOP!

MOV A, R1	MOV A, R1	MOV A, R1	MOV A, R1	MOV A, R1
ADD B, R1	NOP	ADD #0,R1	OR R1, R1	TST R1
ADD C, R1	ADD B, R1	ADD B, R1	ADD B, R1	ADD C, R1
SUB #4, R1	NOP	OR R1, R1	MOV R1, R5	MOV R1, R5
MOV R1, X	ADD C, R1	ADD C, R1	ADD C, R1	ADD B, R1
	NOP	SHL #0, R1	SHL #0, R1	CMP R2, R5
	SUB #4, R1	SUB #4, R1	SUB #4, R1	SUB #4, R1
	NOP	JMP .+1	ADD R5, R5	JMP .+1
	MOV R1, X	MOV R1, X	MOV R1, X	MOV R1, X
			MOV R5, Y	MOV R5, Y

Sophisticated Viruses comprise a Mutation Engine to perform camouflage automatically.

Anti and Anti-Anti-virus techniques

The battle goes on:

How to achieve that an Anti-Virus Program is not infected

Can access to raw disk help the Virus Scanner?

More techniques which don't help:

Integrity Checking

Activity Control

Virus (infection) prevention ?

How to recover from a virus?

Computer Zeitung Nr 11, 10.3.08

Professor Pohlmann: „Trusted Computing bringt Quantensprung"

Die Virenjagd muss kapitulieren

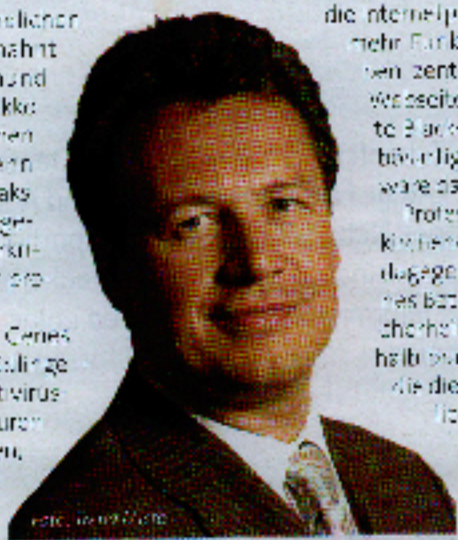
Hannover (ab) – Angesichts stetig steigender Trojaner-Fluten geben Antivirenhersteller offen zu: Die traditionelle Schädlingsjagd auf dem PC ist am Ende. Schutzfunktionen sollen künftig verstärkt aus dem Internet kommen. Deutsche Forscher favorisieren dagegen das Trusted Computing.

Das Virenproblem ist mit herkömmlichen Ansätzen nicht mehr zu lösen", mahnt Trend Micros Cheftechnologe Raimund Genes. F-Secure-Chefforscher Mikko Hypponen stimmt zu: „Wir können nicht weitermachen wie bisher." Denn statt gegen einzelne Viren-Freaks kämpft die Security-Branche heute gegen professionell organisierte Cyberkriminelle, die Tag und Nacht Trojaner produzieren.

Die Malware-Flut steigt daher an: Genes erwartet aktuell 1 500 000 neue Schädlinge pro Monat. Und auch wenn die Antivirus-Hersteller ihre Analyse-Infrastrukturen massiv ausbauen und automatisieren, sei diese Flut kaum noch mit Updates auf alle PCs zu bringen.

„Security muss sich ändern: Von einem Produktverkauf zu einem Service", sieht Hypponen künftig die Internetprovider in der Pflicht. Auch Genes will mehr Funktionen „in die Cloud verlagern." Von zentralen Datenbanken will er infektiösen Webseiten schwebend ständig aktualisierte Black- und White-Lists mit erlaubten und bösartigen Programmen vor, die die PC-Software dann abfragt.

Professor Norbert Pohlmann vom Gelsenkirchener Institut für Internetsicherheit will dagegen tiefer ansetzen: „Es gibt kein modernes Betriebssystem, das eine angemessene Sicherheit gegen aktuelle Angriffe bietet. Deshalb brauchen wir eine Sicherheitsplattform, die die Basisressourcen der Rechner kontrolliert." Das Opensource-Projekt Turaya nutzt dazu TPM-Kryptochips, Mikrokernel-Betriebssysteme und Virtualisierung samt strenger Prozessisolation. Pohlmann: „Ein Quantensprung für die Sicherheit."



Genes: „Virenproblem ist mit herkömmlichen Ansätzen nicht mehr zu lösen."

Mobile Code

Agents, Postscript and Applets

Can we safely execute untrusted code on our computer?

- **Sandboxing**
- **Interpretation**
- **Digital signatures**

Sandbox

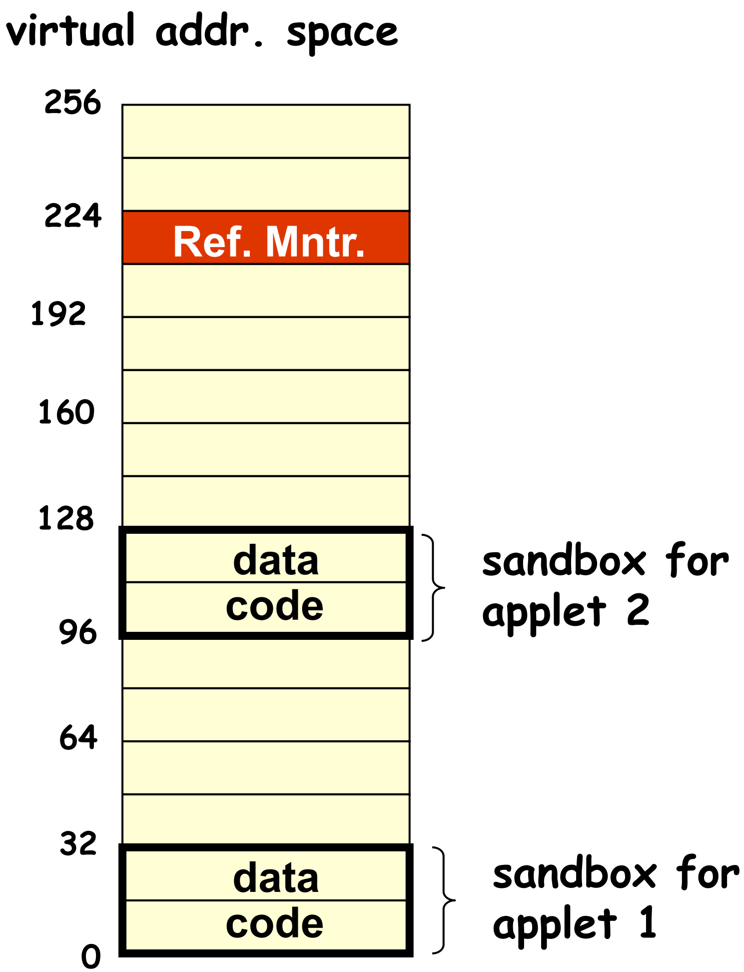
Goal: Separate the virtual address space of a process in areas for trusted and untrusted code.

Problem 1: dynamic jumps where the target address is calculated during run-time.

Solution: Check every "JMP (Rx)" whether its jump target is inside the sandbox.

Problem 2: system calls.

Solution: all system calls are checked by the reference monitor.



Damage due to poor protection and user ignorance:

> 10^{12} $ / year ??

Concept	1995	Word Macro	4 month until widely distr.	$50 Mio.
Melissa	1999	e-mail W-Macro	4 days until widely distr.	$385 Mio.
Love Letter	2000	e-mail Vis.Basic	5 hours until widely distr.	$15000 Mio.

Why not build a trusted and secure computer system ?

Is it possible (with the functionality we are used)?

Is it desirable (or would it be too restrictive) ?

What are the constraints for the user of such a system?