OS Security

Martin Děcký, Vojtěch Horký

DEPARTMENT OF DISTRIBUTED AND DEPENDABLE SYSTEMS
http://d3s.mff.cuni.cz/

CHARLES UNIVERSITY IN PRAGUE
FACULTY OF MATHEMATICS AND PHYSICS

Computer Security

- Security *in large*
  - Knowledge of potential threats
    - Cost of information
    - Information flow
    - Cost of assets
    - Cost of time and resources
  - Security policies
    - Human factor
    - Guidelines
    - Best practices

Computer Security (2)

- Security implementation
  - Security mechanisms
    - Authentication
      - Verification of identity (via credentials)
    - Authorization
      - Verification of access permissions
    - Auditing
      - Backward verification of actions
    - Cryptography
      - Information secrecy, information integrity
    - Steganography
      - Information hiding

Golden Rules of Security

In the end of the day everything is reducible to and relies on physical security.

Even the best security mechanisms cannot win against flaws in security policies.

Security is not a product.

An attacker might not target the strongest nor the most obvious part of the system.

Basic OS Security Mechanisms

- Physical separation
  - Important for backups, certification authorities, etc.
- Temporal separation
  - Avoiding covert channels
- Logical separation
  - Virtualization, kernel/user mode
  - Memory management (segmentation, paging)
- Cryptographic separation
  - Shared medium communication

Steganography

- Embed data into a photograph, audio or video
- Demo
  - Embed secret text into a photograph
  - Difficult to recognize
    - Whether steganography was used at all
    - Difference between files minimal
  - Extraction protected by password
  - `steghide`

Cryptography

- Cryptographic hashing
  - with or without a secret key
  - MD4, MD5
  - SHA-1, SHA-256, …

Hash demo

- MD5 is not enough
  - What is 5f4dcc3b5aa765d61d8327deb882cf99?
  - Search for it with Google
- Databases for other hashes exists too
  - http://md5-database.org/sha256/

Cryptography (2)

- Substitution ciphers
  - primitive, easy to attack with brute-force
  - Ceasar
  - Vigenère table (demo)
- One-time pads
- Transposition ciphers
  - Enigma, Hagellin, …

Cryptography (3)

- Symmetric-key ciphers
  - Feistel network
    - iterated ciphering
  - DES
    - probably most widely spread
    - short key, not considered safe
  - IDEA
    - considered safe
  - Rijndael
  - …

Cryptography (4)

- Public/private-key ciphers
  - based on trapdoor functions
  - Merkle-Hellman
    - knapsack problem, broken
  - RSA
    - factoring problem
  - Elliptic curves
    - discrete logarithm problem (algebraic groups)
  - …

Cryptography (5)

- Pseudo-random number generators
  - prevent computation of the next number
  - might be based on asymmetric ciphers
- Random number generators
  - observe „truly“ random events
  - combine more sources
    - network traffic, I/O latency, system timer, …
  - testing for randomness
    - sequence 111111 or 01010101 is random, but...

Cryptography (6)

- Random number sources in Linux
  - /dev/random
    - blocking
    - „better“ randomness
  - /dev/urandom
    - non-blocking („u for unlimited randomness“)

Authentication

- Verification of identity
  - User, task, network service, etc.
  - Credentials
    - Name/password, passphrase, one-time password
      - Plain text vs. hash
      - Challenge/response
      - Error hiding, exponential latency

Authentication (2)

- Verification of identity
  - Credentials
    - Tokens, certificates, smart tokens
      - Issued by a 3rd party trustworthy authority
    - Biometrics
      - Fingerprint, retina, DNA, face, voice, keyboard typing profile
    - Ownership of a private key (asymmetric cryptography)

Authentication (3)

- Verification authority
  - Usually central
    - Credential database
    - Implicit trust
  - External
    - Explicit trust
- Impersonation
  - Successful authorization to perform an action can lead to identity change
    - SetUID mechanism
  - Inherent issue of central/external authority with no or just symmetric cryptography

Authentication (4)

- Pluggable Authentication Modules (PAM)
  - API for verification of identity
    - Originally implemented in Solaris, very common in Linux
    - Dynamic configuration of authentication methods for different programs
  - Several groups
    - Account management (users/groups creation, deletion, etc)
    - Authentication management (authentication methods)
    - Password management (updating stored credentials)
    - Session management (custom actions after sucessful authentication)
  - Several categories of pluggable modules
    - Requisite, required, sufficient, optional

Configuring PAM – `/etc/pam.d`

- Each service has its own file (chpasswd, sudo, …)
  - Chains (what to verify in which order)
  - Facility
    - authentication (establishing credentials)
    - account management (is account available?)
    - session management (session set-up and tear-down)
    - password management (change authentication token)
  - Control flags (required vs. sufficient)
  - Module name (and arguments)

```
auth            sufficient   pam_rootok.so
auth            required     pam_unix.so
account         required     pam_unix.so
session         required     pam_unix.so
password        required     pam_unix.so sha512 shadow
```

PAM usage

```
#include <security/pam_appl.h>
#include <security/pam_misc.h>

static struct pam_conv conv = { misc_conv, NULL };

int main(int argc, char *argv[])
{
  pam_handle_t *pamh = NULL;
  char *user;
  int retval;

  // ...

  retval = pam_start ("check_user", user, &conv, &pamh);
  if (retval == PAM_SUCCESS)
    retval = pam_authenticate (pamh, 0);        // Is user really himself?
  if (retval == PAM_SUCCESS)
    retval = pam_acct_mgmt (pamh, 0);           // Is user account valid?
  if (retval == PAM_SUCCESS)

  // ...

  pam_end (pamh, retval);
}
```

Authentication (5)

- Kerberos
  - External (central) authority
    - Used for various distributed systems (AFS, Windows Domain)
    - Based on symmetric cryptography
      (authority knows keys of all communication partners)
      - Based on Needham-Schroeder protocol
      - Mutual trust
      - Both the client and the server identity is verified
      - Safe against replay attacks and snooping
    - Authority issues tickets which can prove identity
      - Transfer encrypted by a session key
      - To minimise the problem of stealing unencrypted tickets, each ticket has a limited lifetime (synchronization of clocks)
    - Authority can impersonate any user

Security models

- Military security
  - access rights
  - classification (top secret, secret, confidential, …)
  - compartment
- Lattice
  - generalization of the MSM

Security models (2)

- Bell-LaPadula
  - information transfer
  - simple security property
    - no read-up
  - *-property
    - no write down
- Biba
  - data integrity

Security models (3)

- Chinese wall
  - dynamic model
  - “adviser cannot leak information between competing companies“
- …

Authorization

- Verification of access permissions
  - Whether given subject (user, process, etc.) has the permission to perform given action on a given object
    - Subject identity has to be already established (authentication, explicit anonymous identity)
  - *Mandatory Access Control* (MAC) model
    - Subjects *S*
    - Objects *O*
    - Actions *A*

Access Control Matrix		***Subjects***		
		Alice	**Bob**	**Cecile**
Objects	**file_a**	read	-	write
	file_b	read	read, write	-
	file_c	read	-	-

Authorization (2)

- Usual MAC properties
  - Access control check is performed according to *security policy* on every action
  - Security policy is *enforced* by a central authority (kernel, server) and *controlled* by security policy administrator
    - Subjects (except the administrator) cannot change the policy
      - *Discretionary Access Control* (DAC) model
        - Subjects have a possibility to alter the security policy
          - Usually the security policy actions are controlled by a MAC policy
        - Most systems use both MAC and DAC for various objects

Authorization (3)

- Access Control Lists
  - Maps *objects* to a list of [*subject*, list of *actions*]
    - Unix file access rights
      - Each file/directory (object) is associated with a list

        [owner, *r*/*w*/*x*]
        [group user 0, *r*/*w*/*x*]
        [group user 1, *r*/*w*/*x*]
        ...
        [other user 0, *r*/*w*/*x*]
        [other user 1, *r*/*w*/*x*]
        ...
    - POSIX ACLs
      - Extension of the previous fixed scheme to an unlimited number of users and groups

Authorization (4)

- Access Control Lists
  - Static assignment
    - Subjects are users/groups, not processes
      - Usually special users/groups for specific processes
      - Other mechanisms besides ACLs
  - Scalability
    - Every object has to store all allowed actions
      - Action groups
      - Hierarchy inheritance (NetWare ACLs, Windows ACLs)

Authorization (5)

- Capabilities
  - Maps *subjects* to a list of [*object*, list of *actions*]
  - Quite common in distributed systems (Amoeba, Mach, EROS)
    - Capabilities cannot be directly accessible to subjects (easy to falsificate)
      - Indirect reference into protected storage (Mach)
      - Encryption (Amoeba)
  - Advantage over ACLs: dynamicity
    - Individual processes can be selectively limited
    - Confused attorney problem

Authorization (6)

- POSIX Capabilities
  - Each process has three sets (bitmaps in Linux)
    - Effective set
      - On each action a check is performed
    - Permitted set
      - Capabilities which can be turned on in the *Effective set*
    - Inheritance set
      - Capabilities which are inherited during `exec()` call
  - Sets stored in filesystem (associated with executable files)
    - SetUID mechanism copies a *Forced set* (File Permitted set) into *Permitted set*
  - Capabilities for users
    - Can be set by the login process
      - `pfexec` in Solaris (role-based capabilities setup)

Authorization (7)

- Some of POSIX Capabilities in Linux
  - CAP_CHOWN (change file owner and group)
  - CAP_DAC_OVERRIDE (bypass file permission checks)
  - CAP_IPC_LOCK (permit memory locking)
  - CAP_KILL (bypass permission checks for sending signals)
  - CAP_LINUX_IMMUTABLE (allow setting immutable file attributes)
  - CAP_MKNOD (allow creation of device special files)
  - CAP_NET_ADMIN (allow network devices management)
  - CAP_NET_BIND_SERVICE (allow binding reserved IP ports)
  - CAP_SETPCAP (allow granting and revoking other process' capabilities)
  - CAP_SETUID (allow process UID manipulation)
  - CAP_SYS_ADMIN (permit syscalls such as `mount()`, `swapon()`)

Authorization (8)

- Some of POSIX Capabilities in Linux
  - CAP_SYS_BOOT (allow `reboot()`, `kexec_load()` syscalls)
  - CAP_SYS_CHROOT (allow chrooting)
  - CAP_SYS_NICE (allow raising priority level)
  - CAP_SYS_PTRACE (allow tracing of other processes)
  - CAP_SYS_TIME (allow system clock manipulation)

Authorization (9)

- *Role-Based Access Control* (RBAC) models
  - Maps *roles* to a list of [*object*, list of *actions*]
    - Each *subject* is mapped to a list of *roles*
    - Roles are vertices in an oriented graph (partially ordered set)
      - Role hierarchy
  - Can simulate both MAC and DAC
  - Can be simulated by MAC if the role graph is a tree
  - Many different models and implementations
    - Extensions for *Separation of Duties*
    - Associating roles with global list of actions (capabilities)
    - Extensions for context-sensitive access control
      - *Organization-Based Access Control* (OrBAC)

Authorization (10)

- *Context-Based Access Control* (CBAC) models
  - All previous models use fixed mappings, lists and sets
  - CBAC is dynamic according to the current context
    - Stateful firewalls
      - The permission check is not based only on subjects, objects and actions, but also on the networking context
        - State of the networking (established connections)
        - State history (previous packets, connections, etc.)
        - Data (packet content, application layer state, etc.)

Auditing

- "The independent examination of records and other information in order to form an opinion on the integrity of a system of controls and recommend control improvements to limit risks"
  - Includes examination of
    - system logs
    - backup strategies
    - instructions to handle security breaches
  - …

Auditing (2)

- Keeping track of configuration changes
  - Procedures for making the change and recording the change
  - Versioning the configuration
    - etckeeper, Puppet, …
  - Monitoring for unexpected changes
    - Tripwire, AIDE, …

Auditing (3)

- Checking logs for „odd things“
  - Manually or specialized tools
  - „All system (kernel)“ log
    - Event Viewer, dmesg
  - Unauthorized access
    - `/var/log/auth.log`, `/var/log/secure`
  - Logs of individual services
    - `/var/log/yum.log`
    - `/var/log/httpd/access_log`

Security Certification

- Various classification criteria
  - Trusted Computer System Evaluation Criteria
    - TCSEC/Orange Book
      - Not used since 2000
      - Still considered as important and relatively simple example
    - Classification Levels
      - **D** no security mechanisms
      - **C1** advisory security mechanisms
        - Separation of subjects and objects
        - Subjects are allowed (but not required) to use the mechanisms

Security Certification (2)

- **C2** controlled access
  - Logging of all actions
  - Protection of residual information
  - *OS/400, AS/400 (IBM), OpenVMS VAX 6 (DEC), Windows NT 4.0 (Microsoft)*
- **B1** tagged access control
  - Each object has a security class (level)
  - Each subject has a security clearance (level)
  - Each action is evaluated according to Bell-La Padula security model
  - The implemented security model has a formal description
  - The implementation has been tested
  - *SEVMS VAX 6, ULTRIX MLS+ (DEC), HP-UX BLS 9 (HP), Trusted IRIX/B (SGI), OS1100/2200 (Unisys)*

Security Certification (3)

- **B2** structured access control
  - Verifiable global design
  - Well-defined subsystems
  - Least sufficient permissions principle
  - Security mechanisms enforced also against the hardware
    - Kernel runs in an isolated security domain and periodically checks its integrity
  - Analysis of possible covert channels
  - *Trusted XENIX 4.0 (Trusted Information Systems), Multics*

Security Certification (4)

- **B3** security domains
  - Each subsystem runs in a separate security domain
  - Extensive testing of each access check and action
  - Complete formal description of the design
    - Design based on simple principles
  - Hardened against possible attack vectors
  - Detection of possible threats by audit log examination
  - *XTS-300 (Wang Federal)*
    - Binary compatibility with Unix System V on x86, but requires special hardware
- **A1** formally verified design
  - Formal proofs of security mechanisms consistence
  - Formal verification of conformance between design and implementation
  - Formal analysis of covert channels
  - *Two routers from Boeing and Gemini Computers*

Instead of conclusion

In the end, it always comes down to money.

- ***100% secure system is a delusion***
- ***cost of a security violation vs. cost of hardening the system***

The best policies are useless if users are careless.