Access control

PA193 – Secure coding

Petr Švenda
Zdeněk Říha
Faculty of Informatics, Masaryk University, Brno, CZ

CROCS
Centre for Research on Cryptography and Security

Access control - introduction

In the fields of physical security and information security, access control is the selective **restriction of access** to a place or other resource. The act of accessing may mean consuming, entering, or using. Permission to access a resource is called authorization.

Source: Wikipedia

Access control - authentication

- Process on behalf of a user
- User authentication as a prerequisite of access control
- 3 ways to authenticate users
  - Something they know (PINs, passwords)
  - Something they have (smartcards, tokens)
  - Something they are (biometrics)

Access control paradigms

- Discretionary (DAC)
  - Owner of the object can set the access rights as he/she wishes
  - The most common approach
- Mandatory (MAC)
  - The access rights are restricted by an additional policy
  - Multi Level Systems (MLS)
  - Not very common or only with a limited functionality
- Role-based
  - Access rights finely defined for a role
  - Used in information systems and database systems

DAC – Access Control Matrix

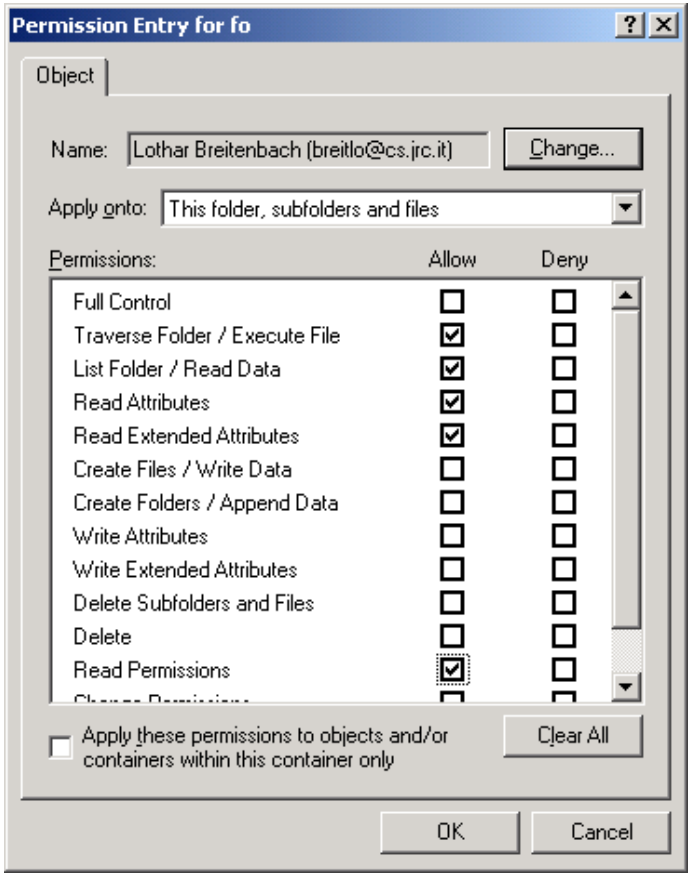
	OBJECTS / SUBJECTS	A	B	C	D	E	F	G	H	J	K	L
Group 1	Alex	W	W	W	R	R	R	R	R	R	R	R
	Brook	R	W	W	R							
	Chris	R	W	W	R	R						
	Denny	R	W	W	R	W	R					
Group 2	Eddie	R	R	R	W	W	W					
	Fran	R	R	R	R	W	W					
Group 3	Gabriel	R	R	R			R	W	W	R		
	Harry	R						W	W	R	R	R
	Jan							W	W	W		
Group 4	Kim	R									W	W
	Lee	R									W	W
	Meryl	R									W	W

Notes:
R Read
W Write and read

Discretionary access control

- Access control matrix
- Matrix too big in practice
- Stored by columns
  - Access control lists (ACL)
  - Access control elements (ACE)
- Stored by rows
  - Capabilities lists
  - Not so common

DAC – ACL/ACE in MS Windows



Mandatory Access Control

- Typical example is the Bell-LaPadula model
- Classification/categorization of data & users

- Data – classified by confidentiality
  – Unclassified, confidential, secret, top secret
- Users – classified by trustworthiness
  – Unclassified, confidential, secret, top secret

MAC – Bell-LaPadula - Policy

- The simple security property
  - a subject at a given security level may not read an object at a higher security level (**no read-up**).
- The star-property
  - a subject at a given security level must not write to any object at a lower security level (**no write-down**).

Role-based access control

- The role is defined as a set of rights
- Users are assigned to roles

- Example (Oracle DBS)
  - create role vyuka;
  - grant CREATE SESSION, ALTER SESSION, CREATE PROCEDURE, CREATE SEQUENCE, CREATE SYNONYM, CREATE TABLE ... to vyuka;
  - grant vyuka to zriha;

Unix users

- Users in Unix systems are represented by numbers
- User identifier (UID)
- Users are organized in groups
- Each group is identified by Group identifier (GID)
- Mapping of usernames (logins) to UID (numers) is present in /etc/passwd file
  - Read-only access to every user of a system
- Passwords of users (in a hashed form) can be found in /etc/shadow
  - Access only for Administrator(s)

Unix: the file /etc/passwd

- Contains the following fields
  - account – userid
  - password (salt+hash), “x” see shadow, “!” or “*” account locked
  - UID – the numerical user ID
  - GID – numerical primary group ID
  - GECOS – This field is optional...
  - directory – the user's $HOME directory
  - shell – the program to run at login

Sample /etc/passwd file

```
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
news:x:9:13:news:/etc/news:
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
gopher:x:13:30:gopher:/var/gopher:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
nobody:x:99:99:Nobody:/:/sbin/nologin
dbus:x:81:81:System message bus:/:/sbin/nologin
vcsa:x:69:69:virtual console memory owner:/dev:/sbin/nologin
rpm:x:37:37::/var/lib/rpm:/sbin/nologin
haldaemon:x:68:68:HAL daemon:/:/sbin/nologin
pcap:x:77:77::/var/arpwatch:/sbin/nologin
nscd:x:28:28:NSCD Daemon:/:/sbin/nologin
named:x:25:25:Named:/var/named:/sbin/nologin
```

Unix: the file /etc/shadow

- Contains the following fields
  - Login name
  - Hashed password
  - Days since Jan 1, 1970 that password was last changed
  - Days before password may be changed
  - Days after which password must be changed
  - Days before password is to expire that user is warned
  - Days after password expires that account is disabled
  - Days since Jan 1, 1970 that account is disabled
  - A reserved field

Sample /etc/shadow file

```
root:$1$lugkC25v$jsCq2t9ehuqh.3hzCKOeq.:15152:0:99999:7:::
bin:*:13189:0:99999:7:::
daemon:*:13189:0:99999:7:::
adm:*:13189:0:99999:7:::
lp:*:13189:0:99999:7:::
sync:*:13189:0:99999:7:::
shutdown:*:13189:0:99999:7:::
halt:*:13189:0:99999:7:::
mail:*:13189:0:99999:7:::
news:*:13189:0:99999:7:::
uucp:*:13189:0:99999:7:::
operator:*:13189:0:99999:7:::
games:*:13189:0:99999:7:::
gopher:*:13189:0:99999:7:::
ftp:*:13189:0:99999:7:::
nobody:*:13189:0:99999:7:::
dbus:!!:13189:0:99999:7:::
vcsa:!!:13189:0:99999:7:::
rpm:!!:13189:0:99999:7:::
haldaemon:!!:13189:0:99999:7:::
pcap:!!:13189:0:99999:7:::
nscd:!!:13189:0:99999:7:::
named:!!:13189:0:99999:7:::
```

Unix users

- Normally each user is having one account (one UID)
  - Administrator is a special user where UID=0
  - Today administration should be done under standard account and privileges obtained via sudo
  - Some distributions block root login via GUI
- Services
  - Nobody
    - running multiple services (daemons) under a single identity…
  - Each service introduces a separate user
    - nfsnobody, mysql, avahi, named, …
- Android
  - For each application a new user is used

Access control under UNIX

- In Unix a lot of I/O is based on (special) files
- Most of the access control is therefore based on file access
- Each file (or directory) is having
  - Owner (UID) - chown
  - Group (GID) - chgrp
  - Access rights in the form of rwx for the user, group and all others (rwx rwx rwx)
    - R = read (files or the content of directories)
    - W = write (files or add/delete files in directories)
    - X = execute (files or use directories)

UNIX special permissions

- SUID/SGID bit
  - For executable files
  - If executed they run with privileges of the file owner
    - And not the one who executed the file
  - SUID – for the user
  - SGID – for the group
- Sticky bit
  - /tmp directory: permissions are 777 (rwx rwx rwx)
  - Anybody can delete files of anybody else
    - This is not good
  - Sticky bit restricts that only to the owner and administrator

Example: A nice attack against NFS

- NFS = Network file system
- UNIX UIDs are 16-bit on many (older) systems
- NFS uses a 32-bit UID
  - This is a feature for good portability
- NFS server uses UID of remote user for the kernel operations (opening files etc.)
  - Kernel does access control check
- NFS disallows UID 0 (root)
  - For obvious reasons
  - Mapped into 65534 (or –2 [16 bits]), normally the user nobody, before passed to kernel

Source: http://nob.cs.ucdavis.edu/bishop/secprog/sans2002.pdf

Example: A nice attack against NFS

- What if NFS client uses UID of 2^{17}?
- NFS cannot use this directly in the kernel
  - If the kernel works with 16 bit UIDs then the maximum UID is $2^{16} - 1$
- UID is truncated to 16 bits by NFS server
  - As $2^{17} \neq 0$, UID is not remapped
  - 2^{17} gets truncated to 16-bits
  - 2^{17} = 0000 0000 0000 0001 0000 0000 0000 0000
  - So the resulting UID is 0000 0000 0000 0000 (0), and that's root

Source: http://nob.cs.ucdavis.edu/bishop/secprog/sans2002.pdf

Example: UIDs in practice

65535 or greater	• Solaris 2.5 and compatible releases systems running the NFS version 2 software see UIDs in this category truncated to 16 bits, creating possible security problems. • Users in this category using the `cpio` command (using the default archive format) to copy file see an error message for each file and the UIDs and GIDs are set to `nobody` in the archive. • SPARC based systems: Users in this category running SunOS 4.0 and compatible applications see `EOVERFLOW` returns from some system calls, and their UIDs and GIDs are mapped to `nobody` . • IA based systems: Users in this category running SVR3-compatible applications will probably see `EOVERFLOW` return codes from system calls. • IA based systems: If users in this category attempt to create a file or directory on a mounted System V file system, the System V file system returns an `EOVERFLOW` error.

Source: http://docs.oracle.com/cd/E19455-01/805-7228/userconcept-35/index.html

Working with users, UIDs, …

- #include <pwd.h>
- getpwent(), setpwent(), endpwent(), getpwnam(), getpw(), getpwuid(), putpwent()
- Supports also network users
  - NIS, LDAP
- Reentrant versions
  - getpwent_r(), fgetpwent_r(), getpwuid_r(), getpwnam_r()
- uid_t, gid_t

```
struct passwd {
        char   *pw_name;       /* username */
        char   *pw_passwd;     /* user password */
        uid_t   pw_uid;        /* user ID */
        gid_t   pw_gid;        /* group ID */
        char   *pw_gecos;      /* real name */
        char   *pw_dir;        /* home directory */
        char   *pw_shell;      /* shell program */
    };
```

Working with users, UIDs, …

```
#define _GNU_SOURCE
#include <pwd.h>
#include <stdio.h>
#define BUFLEN  4096

int
main(void)
{
    struct passwd  pw, *pwp;
    char buf[BUFLEN];
    int i;

    setpwent();
    while (1) {
        i = getpwent_r(&pw,  buf, BUFLEN,  &pwp);
        if (i)
            break;
        printf("%s (%d)\tHOME  %s\tSHELL  %s\n", pwp->pw_name,
               pwp->pw_uid,  pwp->pw_dir,  pwp->pw_shell);
    }
    endpwent();
    exit(EXIT_SUCCESS);
}
```

Modern Unix

- rwxrwxrwx for user, group and others is not enough
- Groups can be created by administrators only
- Modern Unix supports ACL
  - commands getfacl, setfacl

Short format of ACL:

u::rw-,u:lisa:rw-,g::r--,g:toolies:rw-,
m::r--,o::r--,g:toolies:rw,u:lisa:rw,u::wr,
g::r,o::r,m::r

Long format of ACL:

```
user::rw-
user:lisa:rw-          #effective:r--
group::r--
group:toolies:rw-      #effective:r--
mask::r--
other::r--
```

ACL library functions

```
POSIX 1003.1e FUNCTIONS BY AVAILABILITY
  The first group of functions is supported on most systems with POSIX-like access control lists,
  while the second group is supported on fewer systems.  For applications that will be ported the
  second group is best avoided.

  acl_delete_def_file(3), acl_dup(3), acl_free(3), acl_from_text(3), acl_get_fd(3),
  acl_get_file(3), acl_init(3), acl_set_fd(3), acl_set_file(3), acl_to_text(3), acl_valid(3)

  acl_add_perm(3), acl_calc_mask(3), acl_clear_perms(3), acl_copy_entry(3), acl_copy_ext(3),
  acl_copy_int(3), acl_create_entry(3), acl_delete_entry(3), acl_delete_perm(3), acl_get_entry(3),
  acl_get_permset(3), acl_get_qualifier(3), acl_get_tag_type(3), acl_set_permset(3),
  acl_set_qualifier(3), acl_set_tag_type(3), acl_size(3)

LINUX EXTENSIONS
  These non-portable extensions are available on Linux systems.

  acl_check(3), acl_cmp(3), acl_entries(3), acl_equiv_mode(3), acl_error(3), acl_extended_fd(3),
  acl_extended_file(3), acl_extended_file_nofollow(3), acl_from_mode(3), acl_get_perm(3),
  acl_to_any_text(3)
```

Filesystem specific attributes

- EXT2/EXT3/EXT4
  - lsattr / chattr
    - append only (a), compressed (c), no dump (d), extent format (e), immutable (i), data journalling (j), secure deletion (s), no tail-merging (t), undeletable (u), no atime updates (A), no copy on write (C), synchronous directory updates (D), synchronous updates (S), and top of directory hierarchy (T).
    - huge file (h), compression error (E), indexed directory (I), compression raw access (X), and compressed dirty file (Z)

POSIX capabilities in Unix systems

- Limiting root privileges
- In Linux kernel since 2.2
  - for processes, not for files

> As of Linux 2.2, the power of the superuser (root) has been partitioned into a set of discrete capabilities. Each thread has a set of effective capabilities identifying which capabilities (if any) it may currently exercise. Each thread also has a set of inheritable capabilities that may be passed through an execve(2) call, and a set of permitted capabilities that it can make effective or inheritable.
>
> Source: man capget(2)

Linux capabilities

CAP_AUDIT_CONTROL (since Linux 2.6.11)
CAP_AUDIT_WRITE (since Linux 2.6.11)
CAP_CHOWN
CAP_DAC_OVERRIDE
CAP_DAC_READ_SEARCH
CAP_FOWNER
CAP_FSETID
CAP_IPC_LOCK
CAP_IPC_OWNER
CAP_KILL
CAP_LEASE
CAP_LINUX_IMMUTABLE
CAP_MKNOD
CAP_NET_ADMIN
CAP_NET_BIND_SERVICE
CAP_NET_BROADCAST
CAP_NET_RAW
CAP_SETGID
CAP_SETPCAP
CAP_SETUID
CAP_SYS_ADMIN
CAP_SYS_BOOT
CAP_SYS_CHROOT
CAP_SYS_MODULE
CAP_SYS_NICE
CAP_SYS_PACCT
CAP_SYS_PTRACE
CAP_SYS_RAWIO
CAP_SYS_RESOURCE
CAP_SYS_TIME
CAP_SYS_TTY_CONFIG

Library functions supporting capabilities

- Library functions
  - cap_set_proc(3), cap_get_proc(3), capsetp(3), capgetp(3), cap_clear(3), cap_copy_ext(3), cap_from_text(3), cap_get_file(3), cap_init(3)
- System calls
  - capget(2), capset(2)

Access control under Unix - umask

- umask(2) – setting the umask for a process
- umask(1) - bash built-in commands

umask [-p] [-S] [mode]

The user file-creation mask is set to mode. If mode begins with a digit, it is interpreted as an octal number; otherwise it is interpreted as a symbolic mode mask similar to that accepted by chmod(1). If mode is omitted, the current value of the mask is printed. The -S option causes the mask to be printed in symbolic form; the default output is an octal number. If the -p option is supplied, and mode is omitted, the output is in a form that may be reused as input. The return status is 0 if the mode was successfully changed or if no mode argument was supplied, and false otherwise.

Source: man umask

access(2)

int access(const char *pathname, int mode);

access() checks whether the calling process can access the file path- name. If pathname is a symbolic link, it is dereferenced.

The mode specifies the accessibility check(s) to be performed, and is either the value F_OK, or a mask consisting of the bitwise OR of one or more of R_OK, W_OK, and X_OK. F_OK tests for the existence of the file. R_OK, W_OK, and X_OK test whether the file exists and grants read, write, and execute permissions, respectively.

The check is done using the calling process's real UID and GID, rather than the effective IDs as is done when actually attempting an operation (e.g., open(2)) on the file. This allows set-user-ID programs to easily determine the invoking user's authority.

If the calling process is privileged (i.e., its real UID is zero), then an X_OK check is successful for a regular file if execute permission is enabled for any of the file owner, group, or other.

Source: man access

access(2)

Warning: Using access() to check if a user is authorized to, for example, open a file before actually doing so using open(2) creates a security hole, because the user might exploit the short time interval between checking and opening the file to manipulate it. For this reason, the use of this system call should be avoided.

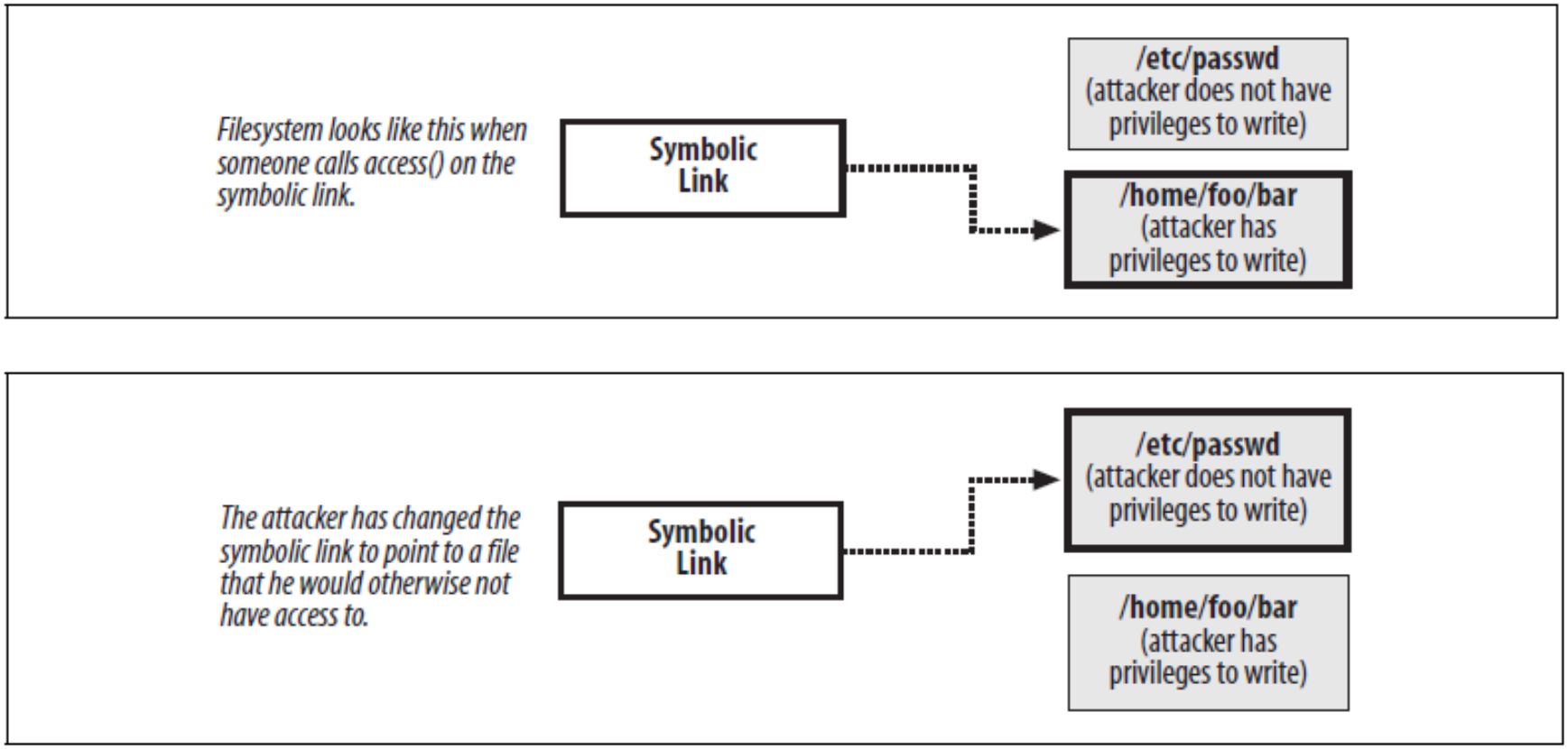
Source: man access

Links

- Hard links
  - link(2) or ln(1) creates a new link to an existing file.
  - This new name may be used exactly as the old one for any operation; both names refer to the same file (and so have the same permissions and ownership) and it is impossible to tell which name was the "original". [man link]

- Soft (symbolic) links
  - symlink(2) or ln(1) creates a symbolic link
  - Symbolic links are interpreted at run time as if the contents of the link had been substituted into the path being followed to find a file or directory. Symbolic links may contain .. path components, which (if used at the start of the link) refer to the parent directories of that in which the link resides. A symbolic link (also known as a soft link) may point to an existing file or to a nonexistent one; the latter case is known as a dangling link. The permissions of a symbolic link are irrelevant; the ownership is ignored when following the link, but is checked when removal or renaming of the link is requested and the link is in a directory with the sticky bit set. [man symlink]
  - Symbolic links often used for attacks
    - Time of check vs. time of use

TOCTOU – Time of check vs. time of use



Source: Viega, Messier: Programming cookbook for C and C++

Watch out for symbolic links

Functions that follow symbolic links automatically open, read, or write to the file whose path name is in the symbolic link file rather than the symbolic link file itself. Your application receives no notification that a symbolic link was followed; to your application, it appears as if the file addressed is the one that was used.

An attacker can use a symbolic link, for example, to cause your application to write the contents intended for a temporary file to a critical system file instead, thus corrupting the system. Alternatively, the attacker can capture data you are writing or can substitute the attacker's data for your own when you read the temporary file.

In general, you should avoid functions, such as chown and stat, that follow symbolic links. As with hard links, your program should evaluate whether a symbolic link is acceptable, and if not, should handle the situation gracefully.

Source: https://developer.apple.com/library/mac/documentation/security/conceptual/SecureCodingGuide/Articles/RaceConditions.html

Watch out for hard links

Functions that follow symbolic links automatically open, read, or write to the file whose path name is in the symbolic link file rather than the symbolic link file itself. Your application receives no notification that a symbolic link was followed; to your application, it appears as if the file addressed is the one that was used.

An attacker can use a symbolic link, for example, to cause your application to write the contents intended for a temporary file to a critical system file instead, thus corrupting the system. Alternatively, the attacker can capture data you are writing or can substitute the attacker's data for your own when you read the temporary file.

In general, you should avoid functions, such as chown and stat, that follow symbolic links. As with hard links, your program should evaluate whether a symbolic link is acceptable, and if not, should handle the situation gracefully.

Source: https://developer.apple.com/library/mac/documentation/security/conceptual/SecureCodingGuide/Articles/RaceConditions.html

Credentials of a task in Linux (cred.h in kernel)

```
struct cred {
        atomic_t        usage;
#ifdef CONFIG_DEBUG_CREDENTIALS
        atomic_t        subscribers;    /* number of processes subscribed */
        void            *put_addr;
        unsigned        magic;
#define CRED_MAGIC      0x43736564
#define CRED_MAGIC_DEAD 0x44656144
#endif
        kuid_t          uid;            /* real UID of the task */
        kgid_t          gid;            /* real GID of the task */
        kuid_t          suid;           /* saved UID of the task */
        kgid_t          sgid;           /* saved GID of the task */
        kuid_t          euid;           /* effective UID of the task */
        kgid_t          egid;           /* effective GID of the task */
        kuid_t          fsuid;          /* UID for VFS ops */
        kgid_t          fsgid;          /* GID for VFS ops */
        unsigned        securebits;     /* SUID-less security management */
        kernel_cap_t    cap_inheritable; /* caps our children can inherit */
        kernel_cap_t    cap_permitted;  /* caps we're permitted */
        kernel_cap_t    cap_effective;  /* caps we can actually use */
        kernel_cap_t    cap_bset;       /* capability bounding set */
#ifdef CONFIG_KEYS
        unsigned char   jit_keyring;    /* default keyring to attach requested
                                         * keys to */
        struct key __rcu *session_keyring; /* keyring inherited over fork */
        struct key      *process_keyring; /* keyring private to this process */
        struct key      *thread_keyring; /* keyring private to this thread */
        struct key      *request_key_auth; /* assumed request_key authority */
#endif
#ifdef CONFIG_SECURITY
        void            *security;      /* subjective LSM security */
#endif
        struct user_struct *user;       /* real user ID subscription */
        struct user_namespace *user_ns; /* user_ns the caps and keyrings are relative to. */
        struct group_info *group_info;  /* supplementary groups for euid/fsgid */
        struct rcu_head rcu;            /* RCU deletion hook */
};
```

Credentials of a task in Linux

- uid, gid
  - real values (who clicked)
- euid, egid
  - Effective values (used in permission checks)
- suid,sgid
  - saved values
  - you can temporarily drop privileges and restore them later
- fsuid, fsgid
  - used in filesystem permission checks

Example: FreeBSD-SA-13:06.mmap

- Privilege escalation via mmap

The FreeBSD virtual memory system allows files to be memory-mapped. All or parts of a file can be made available to a process via its address space. The process can then access the file using memory operations rather than filesystem I/O calls.

The ptrace(2) system call provides tracing and debugging facilities by allowing one process (the tracing process) to watch and control another (the traced process).

II. Problem Description

Due to insufficient permission checks in the virtual memory system, a tracing process (such as a debugger) may be able to modify portions of the traced process's address space to which the traced process itself does not have write access.

III. Impact

This error can be exploited to allow unauthorized modification of an arbitrary file to which the attacker has read access, but not write access. Depending on the file and the nature of the modifications, this can result in privilege escalation.

Example: FreeBSD-SA-12:04.sysret

- Privilege escalation when returning from kernel

I. Background

The FreeBSD operating system implements a rings model of security, where privileged operations are done in the kernel, and most applications request access to these operations by making a system call, which puts the CPU into the required privilege level and passes control to the kernel.

II. Problem Description

FreeBSD/amd64 runs on CPUs from different vendors. Due to varying behaviour of CPUs in 64 bit mode a sanity check of the kernel may be insufficient when returning from a system call.

III. Impact

Successful exploitation of the problem can lead to local kernel privilege escalation, kernel data corruption and/or crash.

To exploit this vulnerability, an attacker must be able to run code with user privileges on the target system.

Example: FreeBSD-SA-12:04.sysret

```
Index: sys/amd64/amd64/trap.c
===================================================================
--- sys/amd64/amd64/trap.c.orig
+++ sys/amd64/amd64/trap.c          (working copy)
@@ -972,4 +972,21 @@
                        syscallname(td->td_proc, sa.code)));

                syscallret(td, error, &sa);
+
+               /*
+                * If the user-supplied value of %rip is not a canonical
+                * address, then some CPUs will trigger a ring 0 #GP during
+                * the sysret instruction.  However, the fault handler would
+                * execute with the user's %gs and %rsp in ring 0 which would
+                * not be safe.  Instead, preemptively kill the thread with a
+                * SIGBUS.
+                */
+               if (td->td_frame->tf_rip >= VM_MAXUSER_ADDRESS) {
+                       ksiginfo_init_trap(&ksi);
+                       ksi.ksi_signo = SIGBUS;
+                       ksi.ksi_code = BUS_OBJERR;
+                       ksi.ksi_trapno = T_PROTFLT;
+                       ksi.ksi_addr = (void *)td->td_frame->tf_rip;
+                       trapsignal(td, &ksi);
+               }
 }
```

Example: FreeBSD-SA-09:17.freebsd-update

- Inappropriate directory permissions in freebsd-update(8)

I. Background

The freebsd-update(8) utility is used to fetch, install, and rollback updates to the FreeBSD base system, and also to upgrade from one FreeBSD release to another.

II. Problem Description

When downloading updates to FreeBSD via 'freebsd-update fetch' or 'freebsd-update upgrade', the freebsd-update(8) utility copies currently installed files into its working directory (/var/db/freebsd-update by default) both for the purpose of merging changes to configuration files and in order to be able to roll back installed updates.

The default working directory used by freebsd-update(8) is normally created during the installation of FreeBSD with permissions which allow all local users to see its contents, and freebsd-update(8) does not take any steps to restrict access to files stored in said directory.

III. Impact

A local user can read files which have been updated by freebsd-update(8), even if those files have permissions which would normally not allow users to read them. In particular, on systems which have been upgraded using 'freebsd-update upgrade', local users can read freebsd-update's backed-up **copy of the master password file**.

Example: FreeBSD-SA-09:10.ipv6

- Missing permission check on SIOCSIFINFO_IN6 ioctl

I. Background

IPv6 is a new Internet Protocol, designed to replace (and avoid many of the problems with) the current Internet Protocol (version 4). Many properties of the FreeBSD IPv6 network stack can be configured via the ioctl(2) interface.

II. Problem Description

The SIOCSIFINFO_IN6 ioctl is missing a necessary permissions check.

III. Impact

Local users, including non-root users and users inside jails, can set some IPv6 interface properties. These include changing the link MTU and disabling interfaces entirely. Note that this affects IPv6 only; IPv4 functionality cannot be affected by exploiting this vulnerability.

UNIX: chroot

- chroot(1) - run command or interactive shell with special root directory
- chroot(2) changes the root directory of the calling process to that specified in path. This directory will be used for pathnames beginning with /. The root directory is inherited by all children of the calling process.
- May be OK for:
  - Testing and development
  - Compatibility (a special set of libraries/other files)
  - Recovery (booting from CD)
- It is not OK:
  - to defend against intentional tampering by privileged (root) users (see e.g. http://www.bpfh.net/simes/computing/chroot-break.html)

See wikipedia on chroot

Chroot examples

- The Postfix mail transfer agent operates as a pipeline of individually chrooted helper programs.
- Many FTP servers for POSIX systems use the chroot mechanism to sandbox untrusted FTP clients. This may be done by forking a process to handle an incoming connection, then chrooting the child (to avoid having to populate the chroot with libraries required for program startup).
- If privilege separation is enabled, the OpenSSH daemon will chroot an unprivileged helper process into an empty directory to handle pre-authentication network traffic for each client.

See wikipedia on chroot

FreeBSD jail

The jail(2) system call allows a system administrator to lock a process and all of its descendants inside an **environment with a very limited ability to affect the system outside** that environment, even for processes with superuser privileges. It is an extension of, but far more powerful than, the traditional UNIX chroot(2) system call.

By design, neither the chroot(2) nor the jail(2) system call modify existing open file descriptors of the calling process, in order to allow programmers to make fine grained access control and privilege separation.

The jail(8) utility creates a new jail or modifies an existing jail, optionally imprisoning the current process (and future descendants) inside it.

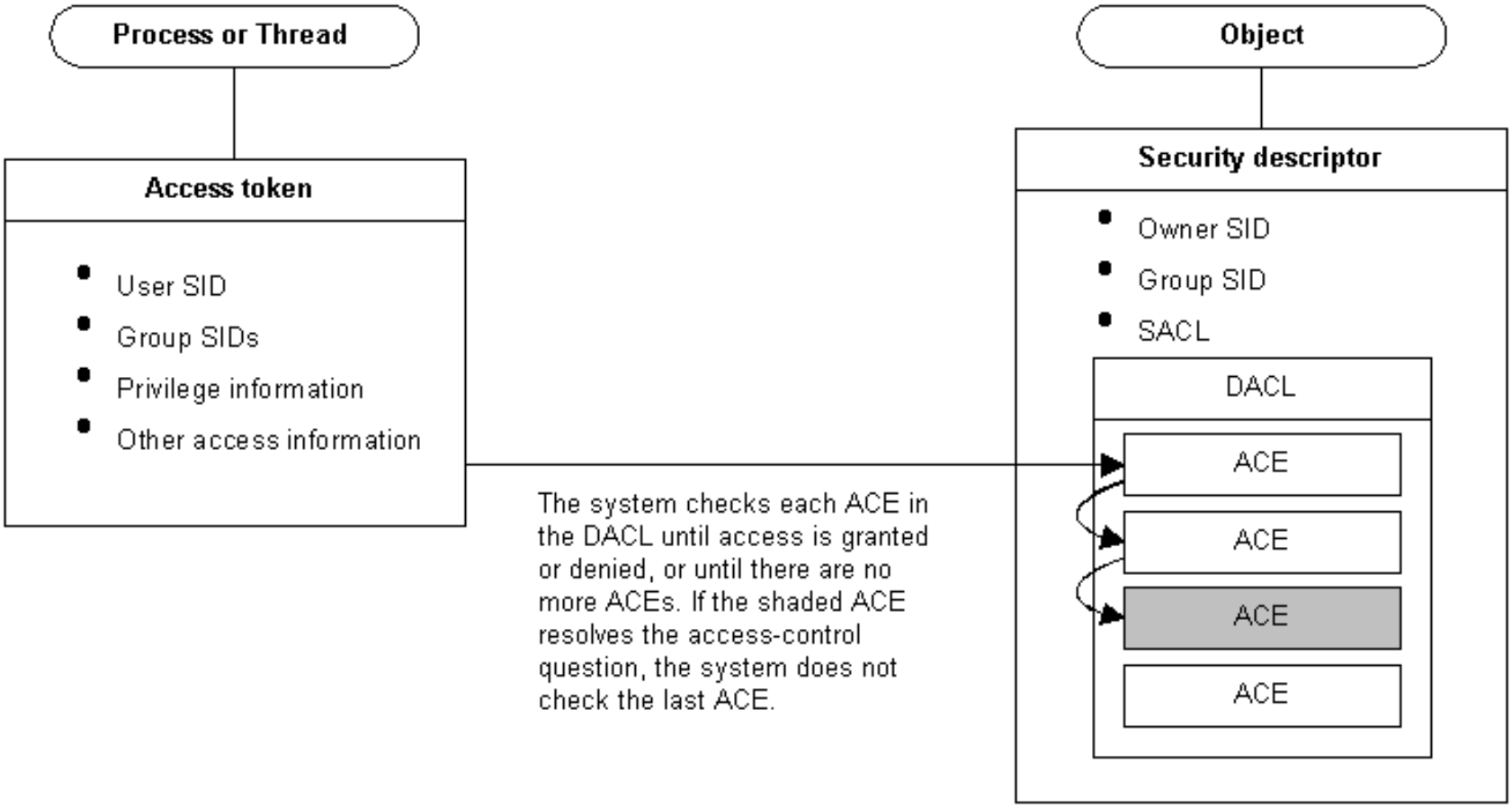
Access control in MS Windows

- There are two basic components of the access control model:
  - **Access tokens**, which contain information about a logged-on user
  - **Security descriptors**, which contain the security information that protects a securable object

Note: Windows 95, Windows 98, Windows Me and Windows CE do not support ACLs.

Source: MSDN

Access control in MS Windows



Source: MSDN

Access tokens

When a user logs on, the system authenticates the user's account name and password. If the logon is successful, the system creates an **access token**. Every process executed on behalf of this user will have a copy of this access token. The access token contains security identifiers that identify the user's account and any group accounts to which the user belongs. The token also contains a list of the privileges held by the user or the user's groups. The system uses this token to identify the associated user when a process tries to access a securable object or perform a system administration task that requires privileges.

Access token

- The information in a token includes the identity and privileges of the user account associated with the process or thread.

- Access tokens contain the following information:
  - The security identifier (SID) for the user's account
  - SIDs for the groups of which the user is a member
  - A logon SID that identifies the current logon session
  - A list of the privileges held by either the user or the user's groups
  - An owner SID
  - The SID for the primary group
  - The default DACL that the system uses when the user creates a securable object without specifying a security descriptor
  - The source of the access token
  - Whether the token is a primary or impersonation token
  - An optional list of restricting SIDs
  - Current impersonation levels
  - Other statistics

Source: MSDN

Security descriptor

When a securable object is created, the system assigns it a **security descriptor** that contains security information specified by its creator, or default security information if none is specified. Applications can use functions to retrieve and set the security information for an existing object. A security descriptor identifies the object's owner and can also contain the following access control lists:

- A discretionary access control list (DACL) that identifies the users and groups **allowed or denied access** to the object
- A system access control list (SACL) that controls how the system **audits** attempts to access the object

An ACL contains a list of access control entries (ACEs). Each ACE specifies a set of access rights and contains a SID that identifies a trustee for whom the rights are allowed, denied, or audited. A trustee can be a user account, group account, or logon session.

Source: MSDN

Security descriptors

- A security descriptor contains the security information associated with a securable object. A security descriptor consists of a SECURITY_DESCRIPTOR structure and its associated security information. A security descriptor can include the following security information:
  - Security identifiers (SIDs) for the owner and primary group of an object.
  - A DACL that specifies the access rights allowed or denied to particular users or groups.
  - A SACL that specifies the types of access attempts that generate audit records for the object.
  - A set of control bits that qualify the meaning of a security descriptor or its individual members.

Source: MSDN

SID = Security identifier

- A **security identifier (SID)** is a unique value of variable length used to identify a trustee. Each account has a unique SID issued by an authority, such as a Windows domain controller, and stored in a security database. Each time a user logs on, the system retrieves the user's SID from the database and places it in the user's access token.

- In addition to the uniquely-created, domain-specific SIDs assigned to specific users and groups, there are well-known SIDs that identify generic groups and generic users. For example, the well-known SIDs, Everyone and World, identify a group that includes all users.

- A SID represents a user, a group or a computer.

Source: MSDN

Examples of SID

Name	SID
Administrator	S-1-5-21-576009780-3087749231-2261803321-500
Guest	S-1-5-21-576009780-3087749231-2261803321-501
Host	S-1-5-21-576009780-3087749231-2261803321-1003
UpdatusUser	S-1-5-21-576009780-3087749231-2261803321-1002
zriha	S-1-5-21-576009780-3087749231-2261803321-1001
Administrator	S-1-5-21-1123561945-448539723-1801674531-500
ASPNET	S-1-5-21-1123561945-448539723-1801674531-1004
Guest	S-1-5-21-1123561945-448539723-1801674531-501
HelpAssistant	S-1-5-21-1123561945-448539723-1801674531-1000
SQLDebugger	S-1-5-21-1123561945-448539723-1801674531-1007
SUPPORT_388945a0	S-1-5-21-1123561945-448539723-1801674531-1002

Functions on SIDs

Function	Description
AllocateAndInitializeSid	Allocates and initializes a SID with the specified number of subauthorities.
ConvertSidToStringSid	Converts a SID to a string format suitable for display, storage, or transport.
ConvertStringSidToSid	Converts a string-format SID to a valid, functional SID.
CopySid	Copies a source SID to a buffer.
EqualPrefixSid	Tests two SID prefix values for equality. A SID prefix is the entire SID except for the last subauthority value.
EqualSid	Tests two SIDs for equality. They must match exactly to be considered equal.
FreeSid	Frees a previously allocated SID by using the **AllocateAndInitializeSid** function.
GetLengthSid	Retrieves the length of a SID.
GetSidIdentifierAuthority	Retrieves a pointer to a SID's identifier authority.
GetSidLengthRequired	Retrieves the size of the buffer required to store a SID with a specified number of subauthorities.
GetSidSubAuthority	Retrieves a pointer to a specified subauthority in a SID.
GetSidSubAuthorityCount	Retrieves the number of subauthorities in a SID.
InitializeSid	Initializes a **SID** structure.
IsValidSid	Tests the validity of a SID by verifying that the revision number is within a known range and that the number of subauthorities is less than the maximum.
LookupAccountName	Retrieves the SID corresponding to a specified account name.
LookupAccountSid	Retrieves the account name corresponding to a specified SID.

Source: MSDN

Access control lists

- An access control list (ACL) is a list of access control entries (ACE). Each ACE in an ACL identifies a trustee and specifies the access rights allowed, denied, or audited for that trustee. The security descriptor for a securable object can contain two types of ACLs: a DACL and a SACL.
- A **discretionary access control list** (DACL) identifies the trustees that are allowed or denied access to a securable object. The system checks the ACEs **in sequence** until it finds one or more ACEs that allow all the requested access rights, or until any of the requested access rights are denied.
- A **system access control list** (SACL) enables administrators to log attempts to access a secured object. Each ACE specifies the types of access attempts by a specified trustee that cause the system to generate a record in the security event log.

Source: MSDN

ACL – empty & NULL

- If the *DACL* belonging to an object's security descriptor is set to NULL, a **null DACL** is created. A null DACL grants full access to any user that requests it; normal security checking is not performed with respect to the object.
- A null DACL should not be confused with an empty DACL. An **empty DACL** is a properly allocated and initialized DACL containing no *ACEs*. An empty DACL grants no access to the object it is assigned to.
- BTW: NULL security descriptor will create a default security descriptor with a default DACL…

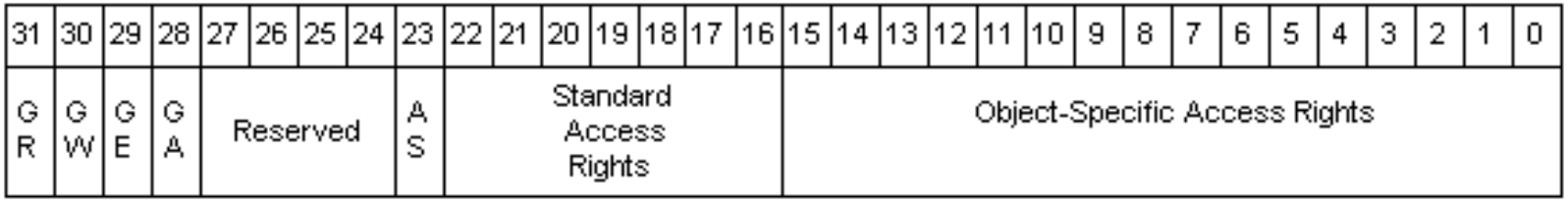
Source: MSDN

Access control entry

- An access control entry (ACE) is an element in an access control list (ACL). An ACL can have zero or more ACEs.
- There are six types of ACEs, three of which are supported by all securable objects. The other three types are object-specific ACEs supported by directory service objects.
- All types of ACEs contain the following access control information:
  - A security identifier (SID) that identifies the trustee to which the ACE applies.
  - An access mask that specifies the access rights controlled by the ACE.
  - A flag that indicates the type of ACE (allowed, denied, audit).
  - A set of bit flags that determine whether child containers or objects can inherit the ACE from the primary object to which the ACL is attached.

Source: MSDN

Windows Access Mask Format



GR --> Generic_Read
GW --> Generic_Write
GE --> Generic_Execute
GA --> Generic_ALL
AS --> Right to access SACL

Constant	Meaning
DELETE	The right to delete the object.
READ_CONTROL	The right to read the information in the object's *security descriptor*, not including the information in the SACL.
SYNCHRONIZE	The right to use the object for synchronization. This enables a thread to wait until the object is in the signaled *state*. Some object types do not support this access right.
WRITE_DAC	The right to modify the DACL in the object's security descriptor.
WRITE_OWNER	The right to change the owner in the object's security descriptor.

Source: MSDN

Dangerous ACE types

- Everyone (WRITE_DAC)
  - Right to modify the DACL
- Everyone (WRITE_OWNER)
  - Right to change the owner
- Everyone (FILE_ADD_FILE)
  - Right to add new files including executables

Securable objects

Object type	Security Descriptor Functions
Files or directories on an NTFS file system	GetNamedSecurityInfo, SetNamedSecurityInfo, GetSecurityInfo, SetSecurityInfo
Named pipes Anonymous pipes	GetSecurityInfo, SetSecurityInfo
Processes Threads	GetSecurityInfo, SetSecurityInfo
File-mapping objects	GetNamedSecurityInfo, SetNamedSecurityInfo, GetSecurityInfo, SetSecurityInfo
Access tokens	SetKernelObjectSecurity, GetKernelObjectSecurity
Window-management objects (window stations and desktops)	GetSecurityInfo, SetSecurityInfo
Registry keys	GetNamedSecurityInfo, SetNamedSecurityInfo, GetSecurityInfo, SetSecurityInfo
Windows services	GetNamedSecurityInfo, SetNamedSecurityInfo, GetSecurityInfo, SetSecurityInfo
Local or remote printers	GetNamedSecurityInfo, SetNamedSecurityInfo, GetSecurityInfo, SetSecurityInfo
Network shares	GetNamedSecurityInfo, SetNamedSecurityInfo, GetSecurityInfo, SetSecurityInfo
Interprocess synchronization objects (events, mutexes, semaphores, and waitable timers)	GetNamedSecurityInfo, SetNamedSecurityInfo, GetSecurityInfo, SetSecurityInfo
Job objects	GetNamedSecurityInfo, SetNamedSecurityInfo, GetSecurityInfo, SetSecurityInfo
Directory service objects	These objects are handled by Active Directory Objects. See Active Directory Service Interfaces.

Source: MSDN

ACE inheritance

- An object's ACL can contain ACEs that it inherited from its parent container. For example, a registry subkey can inherit ACEs from the key above it in the registry hierarchy. Likewise, a file in an NTFS file system can inherit ACEs from the directory that contains it.
- The ACE_HEADER structure of an ACE contains a set of inheritance flags that control ACE inheritance and the effect of an ACE on the object to which it is attached. The system interprets the inheritance flags and other inheritance information according to the rules of ACE inheritance.

Source: MSDN

ACE Inheritance – flags in ACE_HEADER

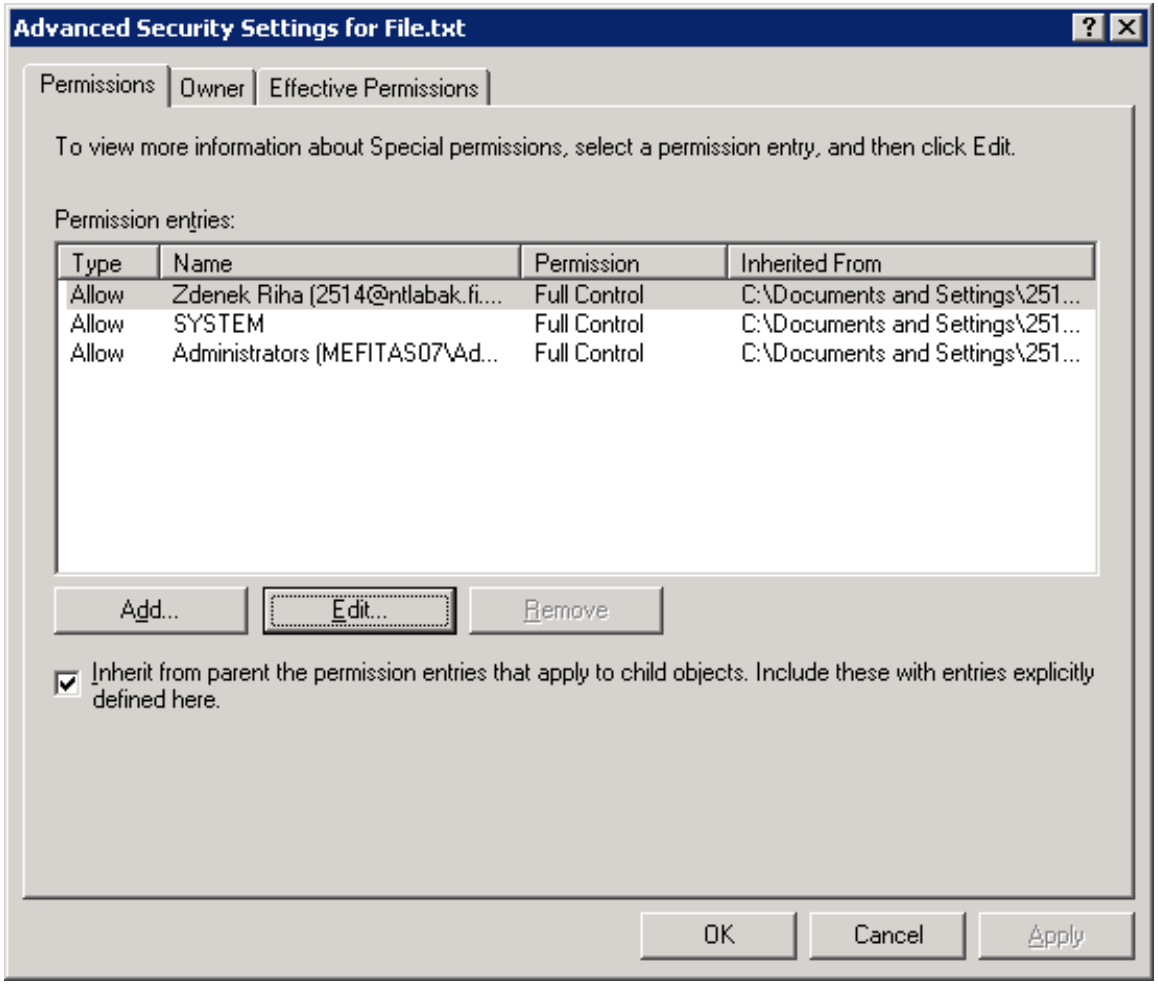
Value	Meaning
CONTAINER_INHERIT_ACE	Child objects that are containers, such as directories, inherit the ACE as an effective ACE. The inherited ACE is inheritable unless the NO_PROPAGATE_INHERIT_ACE bit flag is also set.
FAILED_ACCESS_ACE_FLAG	Used with system-audit ACEs in a SACL to generate audit messages for failed access attempts.
INHERIT_ONLY_ACE	Indicates an inherit-only ACE which does not control access to the object to which it is attached. If this flag is not set, the ACE is an effective ACE which controls access to the object to which it is attached. Both effective and inherit-only ACEs can be inherited depending on the state of the other inheritance flags.
INHERITED_ACE	Indicates that the ACE was inherited. The system sets this bit when it propagates an inherited ACE to a child object. **Windows NT:** Not supported.
NO_PROPAGATE_INHERIT_ACE	If the ACE is inherited by a child object, the system clears the OBJECT_INHERIT_ACE and CONTAINER_INHERIT_ACE flags in the inherited ACE. This prevents the ACE from being inherited by subsequent generations of objects.
OBJECT_INHERIT_ACE	Noncontainer child objects inherit the ACE as an effective ACE. For child objects that are containers, the ACE is inherited as an inherit-only ACE unless the NO_PROPAGATE_INHERIT_ACE bit flag is also set.
SUCCESSFUL_ACCESS_ACE_FLAG	Used with system-audit ACEs in a SACL to generate audit messages for successful access attempts.

ACE inheritance rules

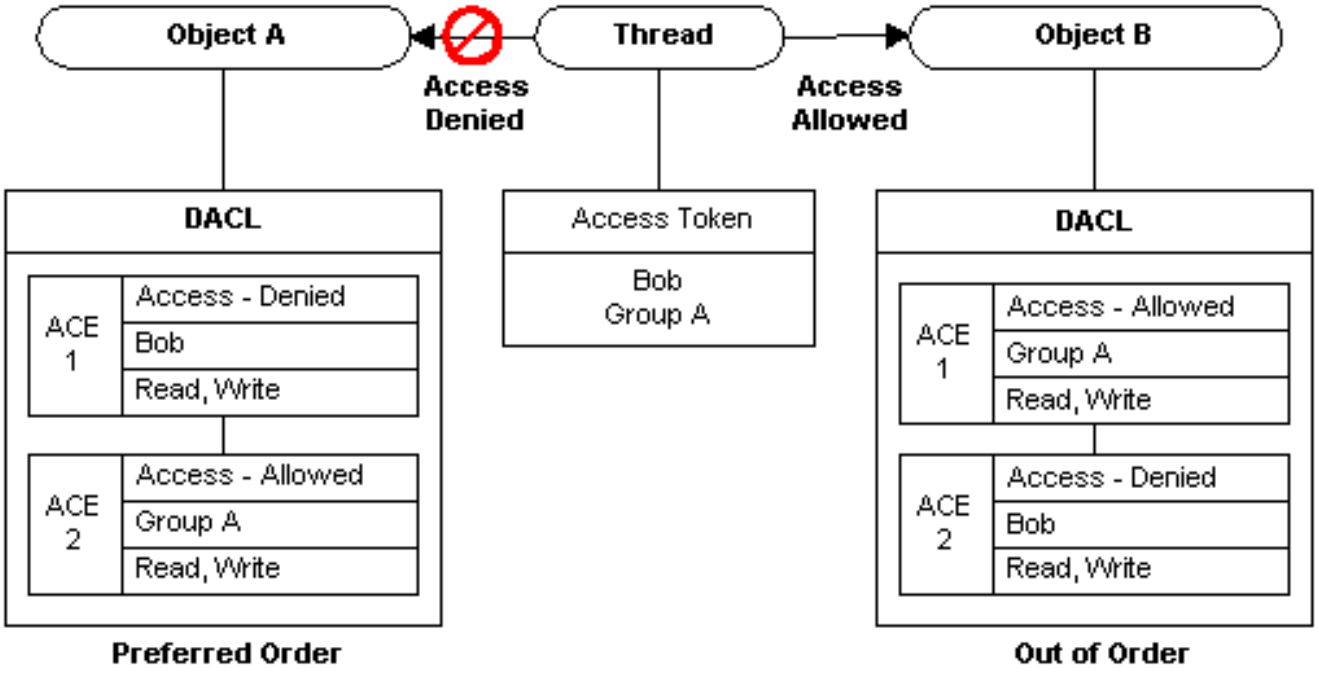
Parent ACE type	Effect on Child ACL
OBJECT_INHERIT_ACE only	**Noncontainer child objects:** Inherited as an effective ACE. **Container child objects:** Containers inherit an inherit-only ACE unless the NO_PROPAGATE_INHERIT_ACE bit flag is also set.
CONTAINER_INHERIT_ACE only	**Noncontainer child objects:** No effect on the child object. **Container child objects:** The child object inherits an effective ACE. The inherited ACE is inheritable unless the NO_PROPAGATE_INHERIT_ACE bit flag is also set.
CONTAINER_INHERIT_ACE and OBJECT_INHERIT_ACE	**Noncontainer child objects:** Inherited as an effective ACE. **Container child objects:** The child object inherits an effective ACE. The inherited ACE is inheritable unless the NO_PROPAGATE_INHERIT_ACE bit flag is also set.
No inheritance flags set	No effect on child container or noncontainer objects.

Source: MSDN

ACE inheritance



The order of ACE does matter



Note: Keep the right order also when modifying the ACL …

Source: MSDN

Security Descriptor Definition Language (SDDL)

- To specify ACL you can use the SDDL
- The format is a null-terminated string with tokens to indicate each of the four main components of a security descriptor: owner (O:), primary group (G:), DACL (D:), and SACL (S:).
  - O:owner_sid
  - G:group_sid
  - D:dacl_flags(string_ace1)(string_ace2)... (string_acen)
  - S:sacl_flags(string_ace1)(string_ace2)... (string_acen)
- (A;;RPWPCCDCLCSWRCWDWOGA;;;S-1-0-0)

Source: MSDN

ACE string

- ace_type;ace_flags;rights;object_guid;inherit_object_guid;account_sid

ACE type string	Constant in Sddl.h	AceType value
"A"	SDDL_ACCESS_ALLOWED	ACCESS_ALLOWED_ACE_TYPE
"D"	SDDL_ACCESS_DENIED	ACCESS_DENIED_ACE_TYPE
"OA"	SDDL_OBJECT_ACCESS_ALLOWED	ACCESS_ALLOWED_OBJECT_ACE_TYPE
"OD"	SDDL_OBJECT_ACCESS_DENIED	ACCESS_DENIED_OBJECT_ACE_TYPE
"AU"	SDDL_AUDIT	SYSTEM_AUDIT_ACE_TYPE
"AL"	SDDL_ALARM	SYSTEM_ALARM_ACE_TYPE
"OU"	SDDL_OBJECT_AUDIT	SYSTEM_AUDIT_OBJECT_ACE_TYPE
"OL"	SDDL_OBJECT_ALARM	SYSTEM_ALARM_OBJECT_ACE_TYPE

ACE flags string	Constant in Sddl.h	AceFlag value
"CI"	SDDL_CONTAINER_INHERIT	CONTAINER_INHERIT_ACE
"OI"	SDDL_OBJECT_INHERIT	OBJECT_INHERIT_ACE
"NP"	SDDL_NO_PROPAGATE	NO_PROPAGATE_INHERIT_ACE
"IO"	SDDL_INHERIT_ONLY	INHERIT_ONLY_ACE
"ID"	SDDL_INHERITED	INHERITED_ACE
"SA"	SDDL_AUDIT_SUCCESS	SUCCESSFUL_ACCESS_ACE_FLAG
"FA"	SDDL_AUDIT_FAILURE	FAILED_ACCESS_ACE_FLAG

Access rights string	Constant in Sddl.h	Access right value
Generic access rights		
"GA"	SDDL_GENERIC_ALL	GENERIC_ALL
"GR"	SDDL_GENERIC_READ	GENERIC_READ
"GW"	SDDL_GENERIC_WRITE	GENERIC_WRITE
"GX"	SDDL_GENERIC_EXECUTE	GENERIC_EXECUTE
Standard access rights		
"RC"	SDDL_READ_CONTROL	READ_CONTROL
"SD"	SDDL_STANDARD_DELETE	DELETE
"WD"	SDDL_WRITE_DAC	WRITE_DAC
"WO"	SDDL_WRITE_OWNER	WRITE_OWNER
Directory service object access rights		
"RP"	SDDL_READ_PROPERTY	ADS_RIGHT_DS_READ_PROP
"WP"	SDDL_WRITE_PROPERTY	ADS_RIGHT_DS_WRITE_PROP
"CC"	SDDL_CREATE_CHILD	ADS_RIGHT_DS_CREATE_CHILD
"DC"	SDDL_DELETE_CHILD	ADS_RIGHT_DS_DELETE_CHILD
"LC"	SDDL_LIST_CHILDREN	ADS_RIGHT_DS_LIST
"SW"	SDDL_SELF_WRITE	ADS_RIGHT_DS_SELF
"LO"	SDDL_LIST_OBJECT	ADS_RIGHT_DS_LIST_OBJECT
"DT"	SDDL_DELETE_TREE	ADS_RIGHT_DS_DELETE_TREE
"CR"	SDDL_CONTROL_ACCESS	ADS_RIGHT_DS_CONTROL_ACCESS

Source: MSDN

Terminal server

- Many users can log in
- ACLs is even more important here

- In Unix it is usually expected the system is multiuser…

Example: MS01-003

Weak Permissions on Winsock Mutex Can Allow Service Failure

Like all other objects under Windows NT 4.0, mutexes - synchronization objects that govern access to resources - have permissions associated with them, that govern how they can be accessed. However, a particular mutex used to govern access to a networking resource has inappropriately loose permissions. This could enable an attacker who had the ability to run code on a local machine to monopolize the mutex, thereby preventing any other processes from using the resource that it controlled. This would have the effect of preventing the machine from participating in the network.

The attacker would require interactive logon access to the affected machine. This significantly limits the scope of the vulnerability because, if normal security recommendations have been followed, unprivileged users will not be granted interactive logon rights to critical machines like servers. Unprivileged users typically are granted interactive logon rights to workstations and terminal servers. However, a workstation would not be a tempting target for an attacker, because he could only use this vulnerability to deny service to himself. The machines most likely to be affected would be Terminal Servers.

Windows privileges

- User accounts can have privileges that allow or disallow certain privileged operations affecting an entire computer
  - E.g. ability to log on to a computer, to debug programs of other users, changing the system time, …
- SeBackupPrivilege
- SeDebugPrivilege
- SeLoadDriverPrivilege
- SeTakeOwnershipPrivilege

“whoami /all”

GROUP INFORMATION

Group Name	Type	SID	Attributes
Everyone	Well-known group	S-1-1-0	Mandatory group, Enabled by default, Enabled group
BUILTIN\Administrators	Alias	S-1-5-32-544	Group used for deny only
BUILTIN\Users	Alias	S-1-5-32-545	Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\REMOTE INTERACTIVE LOGON	Well-known group	S-1-5-14	Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\INTERACTIVE	Well-known group	S-1-5-4	Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\Authenticated Users	Well-known group	S-1-5-11	Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\This Organization	Well-known group	S-1-5-15	Mandatory group, Enabled by default, Enabled group
LOCAL	Well-known group	S-1-2-0	Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\NTLM Authentication	Well-known group	S-1-5-64-10	Mandatory group, Enabled by default, Enabled group
Mandatory Label\Střední povinná úroveň	Label	S-1-16-8192	Mandatory group, Enabled by default, Enabled group

PRIVILEGES INFORMATION

Privilege Name	Description	State
SeShutdownPrivilege	Vypnout systém	Disabled
SeChangeNotifyPrivilege	Nepoužívat kontrolu procházení	Enabled
SeUndockPrivilege	Vyjmout počítač z dokovací stanice	Disabled
SeIncreaseWorkingSetPrivilege	Zvýšit pracovní sadu procesu	Disabled
SeTimeZonePrivilege	Změnit časové pásmo	Disabled

Determine correct SIDs and privileges

1. Find you which resources you use
2. Find you which privileged API calls you use
3. Evaluate the account under which you require to run
4. List the SIDs and privileges of the account
5. Determine which SID and privileges you need to perform the application tasks
6. Adjust the token

Windows integrity levels

- Processes are assigned an integrity level (Vista+)
  - Low, medium, high, system
- The process manager assigns the mandatory policy options NO_READ_UP and NO_WRITE_UP to restrict lower-integrity processes from opening a higher-integrity process for either read or write access.
- Also affects access to registry, folders, COM, …

See: http://msdn.microsoft.com/en-us/library/bb625957.aspx

Privilege separation

privilege separation is a technique in which a program is divided into parts which are limited to the specific privileges they require in order to perform a specific task. This is used to mitigate the potential damage of a computer security attack.

See Wikipedia on privilege separation

- Example:
  - Low privileged client x high privileged server

Example: Apache web server privileges

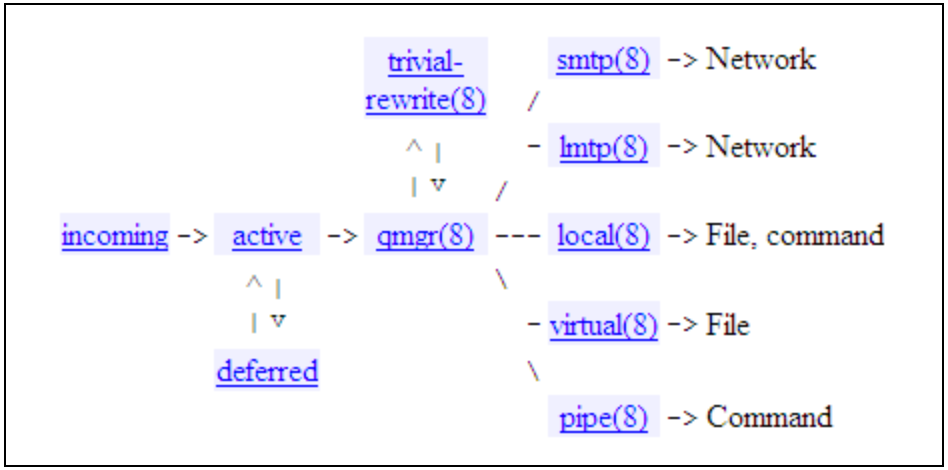
- If the Listen specified in the configuration file is default of 80 (or any other port below 1024), then it is necessary to have root privileges in order to start apache, so that it can bind to this privileged port. Once the server has started and performed a few preliminary activities such as opening its log files, it will launch several child processes which do the work of listening for and answering requests from clients. The main httpd process continues to run as the root user, but the child processes run as a less privileged user.
  - Unprivileged users often called apache, www, webuser, …
- Static pages: No need to be owned by apache user, RO access is enough (accessed with apache user rights)
- Dynamic pages: Running as apache user unless suEXEC is used, then “scripts” are running with user privileges (is part of user’s web page)

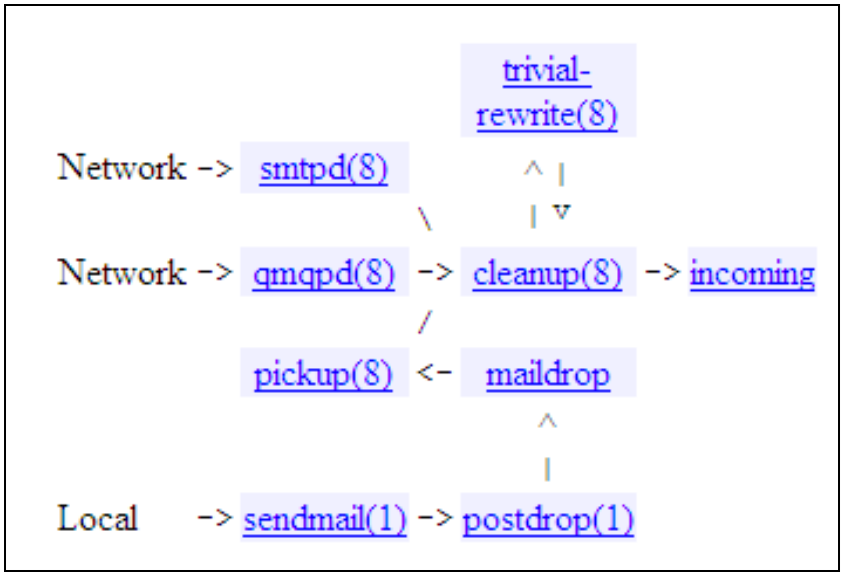
Apache processes - example

```
apache   12883  0.0  0.4  50964 13704 ?    S    Oct20   0:00 /usr/sbin/httpd -DFOREGROUND
apache   12884  0.0  0.4  50800 12924 ?    S    Oct20   0:00 /usr/sbin/httpd -DFOREGROUND
apache   12887  0.0  0.4  50916 13164 ?    S    Oct20   0:00 /usr/sbin/httpd -DFOREGROUND
apache   14436  0.0  0.4  50676 13056 ?    S    Oct20   0:00 /usr/sbin/httpd -DFOREGROUND
apache   15243  0.0  0.4  50776 12932 ?    S    Oct20   0:00 /usr/sbin/httpd -DFOREGROUND
root     15483  0.0  0.4  46448 12960 ?    Ss   Oct07   1:05 /usr/sbin/httpd -DFOREGROUND
apache   17822  0.0  0.4  51232 13476 ?    S    07:56   0:00 /usr/sbin/httpd -DFOREGROUND
apache   17825  0.0  0.3  50536 12364 ?    S    07:56   0:00 /usr/sbin/httpd -DFOREGROUND
apache   18284  0.0  0.2  46468  8216 ?    S    11:32   0:00 /usr/sbin/httpd -DFOREGROUND
apache   18523  0.0  0.2  46468  8200 ?    S    13:33   0:00 /usr/sbin/httpd -DFOREGROUND
apache   18527  0.0  0.2  46468  8168 ?    S    13:42   0:00 /usr/sbin/httpd -DFOREGROUND
```

Example: Delivering emails (postfix)

- Postfix is a email delivery program
  - A “secure” replacement of Sendmail
  - Principle of least priviledge





Postfix architecture (default master.cf file)

```
# ==========================================================================
# service type  private unpriv  chroot  wakeup  maxproc command + args
#               (yes)   (yes)   (yes)   (never) (100)
# ==========================================================================
smtp      inet  n       -       n       -       -       smtpd
smtps     inet  n       -       n       -       -       smtpd
628       inet  n       -       n       -       -       qmqpd
pickup    fifo  n       -       n       60      1       pickup
cleanup   unix  n       -       n       -       0       cleanup
qmgr      fifo  n       -       n       300     1       qmgr
tlsmgr    unix  -       -       n       1000    1       tlsmgr
rewrite   unix  -       -       n       -       -       trivial-rewrite
bounce    unix  -       -       n       -       0       bounce
defer     unix  -       -       n       -       0       bounce
trace     unix  -       -       n       -       0       bounce
verify    unix  -       -       n       -       1       verify
flush     unix  n       -       n       1000    0       flush
proxymap  unix  -       -       n       -       -       proxymap
smtp      unix  -       -       n       -       -       smtp
showq     unix  n       -       n       -       -       showq
error     unix  -       -       n       -       -       error
retry     unix  -       -       n       -       -       error
discard   unix  -       -       n       -       -       discard
local     unix  -       n       n       -       -       local
virtual   unix  -       n       n       -       -       virtual
lmtp      unix  -       -       n       -       -       lmtp
anvil     unix  -       -       n       -       1       anvil
Scache    unix  -       -       n       -       1       scache
```

Restricting access based on IP

- Based on IP
  - Specific IP (147.251.48.3)
  - Subnet (147.251.0.0/16)
- Based on hostname (depends on DNS)
  - Specific hostname (aisa.fi.muni.cz)
  - Group of hostnames (*.fi.muni.cz)
- Where to block
  - Firewall at the router
  - Firewall at the computer (personal firewall)
  - TCP Wrapper
  - Your application

Best practices

- Protect your resources from unauthorized access
  - Good ACLs are part of your “Defence in depth” strategy
- Defend against race condition when working with files
  - Time of check, Time of use (TOCTOU)
- Minimize the time a program runs with high privileges
- Protect sensitive data

Check error codes

- Permissions / access rights can be a reason why a file operation fails
- Check result codes!
- Example:
  - you can't remove a directory that has anything inside it. If a directory is in a location where other users have access to it, any attempt to remove the directory might fail because another process might add new files while you are removing the old ones…

Exercise

- Set ACL on files (Windows)
- Erasing files securely (Linux)