CPSC 525: Security Architecture

Is this Enough?

Michael E. Locasto

Department of Computer Science
UofC
CPSC 525/625

March 13, 2013
Agenda

Learning Objectives for Today
Security Architecture Principles
Case Study
How do we evaluate the utility of a particular TCB?
Economy of mechanism (complexity kills)
Fail-safe defaults (fail closed, not fail open)
Complete mediation (identify all control/entry/measurement points)
Open Design (no security through obscurity)
Separation of privilege (map functionality to a disjoint set of roles)
Least privilege (what power do you need for Task T?)
Least-common mechanism (minimize “size” of TCB in terms of
common surface/shared surface)
Psychological acceptability (usable security, intuitive, people-centric
model)
Example of Privilege Separation

OpenSSH
Case Study

What is the totality of security mechanisms in the Linux kernel?
How does this compare, for example, to OpenBSD?

- in-kernel crypto libraries
- authentication: login, users, groups (PAM)
- file system permissions
- gcc/propolice patch, stack canary
- ASLR
- noexec / nx
- segments (what segments?)
- page permissions
- firewalls (iptables)
- application-level security policies (Java 2 P and P)
- SELinux
- security modules
- PAX/grsecurity
Linux Kernel File System Permission Check

vfs_read()
→ rw_verify_area()
→ → security_file_permission()
→ → → file_permission()
→ → → → inode_permission()
→ → → → → generic_permission()
```c
/**
* generic_permission - check for access rights on a Posix-like filesystem
* @inode:      inode to check access rights for
* @mask:       right to check for (%MAY_READ, %MAY_WRITE, %MAY_EXEC)
* @check_acl:  optional callback to check for Posix ACLs
* Used to check for read/write/execute permissions on a file.
* We use "fsuid" for this, letting us set arbitrary permissions
* for filesystem access without changing the "normal" uids which
* are used for other things..
*/
int generic_permission(struct inode *inode, int mask,
                       int (*check_acl)(struct inode *inode, int mask))
{
    int ret;

    /*
    * Do the basic POSIX ACL permission checks.
    */
    ret = acl_permission_check(inode, mask, check_acl);
    if (ret != -EACCES)
        return ret;

    /*
    * Read/write DACs are always overridable.
    * Executable DACs are overridable if at least one exec bit is set.
    */
    if (!((mask & MAY_EXEC) || execute_ok(inode))
        if (capable(CAP_DAC_OVERRIDE))
            return 0;

    /*
    * Searching includes executable on directories, else just read.
    */
    mask &= MAY_READ | MAY_WRITE | MAY_EXEC;
    if (mask == MAY_READ || (S_ISDIR(inode->i_mode) && !(mask & MAY_WRITE)))
        if (capable(CAP_DAC_READ_SEARCH))
            return 0;

    return -EACCES;
}
```
This is an essay by one of the three people on the planet who can write secure code from scratch. Keep an eye out for contrary opinions on the utility of privilege separation.

Some Thoughts on Security After Ten Years of qmail 1.0, DJB, CSAW 2007

http://cr.yp.to/qmail/qmailsec-20071101.pdf