Security-Enhanced Linux

Peter Loscocco
Information Assurance Research Group
National Security Agency
loscocco@tycho.nsa.gov

Outline

Importance of secure operating systems

Security-enhanced Linux

Related work

Conclusions

Importance of Secure OS

Growing need for security

Flawed assumption of security

OS is correct level to provide security

Key feature: Mandatory Access Control (MAC)

- Access to objects controlled by policy administrator
- Users/processes may not change access policy
- All accesses are mediated w.r.t. the policy

Mandatory Security: Key Gains

Provides critical support for application security

- Protects against tampering with secured application
- Protects against bypass of secured application
- Enables assured pipelines

Provides strong separation of applications

- Permits safe execution of untrustworthy applications
- Limits scope of potential damage due to penetration of applications
- Functional uses: isolated testing environments or insulated development environments

Protects information from

- Legitimate users with limited authorization
- Authorized users unwittingly using malicious applications

Why not just DAC?

Decisions are only based on user identity and ownership

Each user has complete discretion over his objects

Only two major categories of users: user and superuser

Many system services and privileged programs must run as superuser

No protection against malicious software

Traditional approach to MAC

Enforce system-wide security policy

- Based on confidentiality or integrity attributes of subjects and objects
- Can support many different categories of users
- Can confine malicious code

Too limiting for general solution

- Tightly coupled to Multilevel security policy
- Assumes hierarchical relationship in labeling
- Ignores least privilege and separation of duty
- No binding between security attributes of subjects and their executables (limits protection from executing malicious code)
- Requires trusted subjects

Main Design Goals

Secure architecture is driving concern

Flexibility of policy and mechanism

Separation of policy from enforcement

Policy Flexibility

Capable of Supporting wide variety of security policies

- Separation Policies
  - Enforcing Legal restrictions on data
  - Establishing well-defined user roles
  - Restrictions to classified/compartmented data
- Containment Policies
  - Restricting web server access to authorized data
  - Minimizing damage from viruses and other malicious code
- Integrity Policies
  - Protecting applications from modification
  - Preventing unauthorized modification of databases
- Invocation Policies
  - Guaranteeing that data is processed as required
  - Enforcing encryption policies

Type Enforcement

Access matrix defining permission between domains and types

Advantages

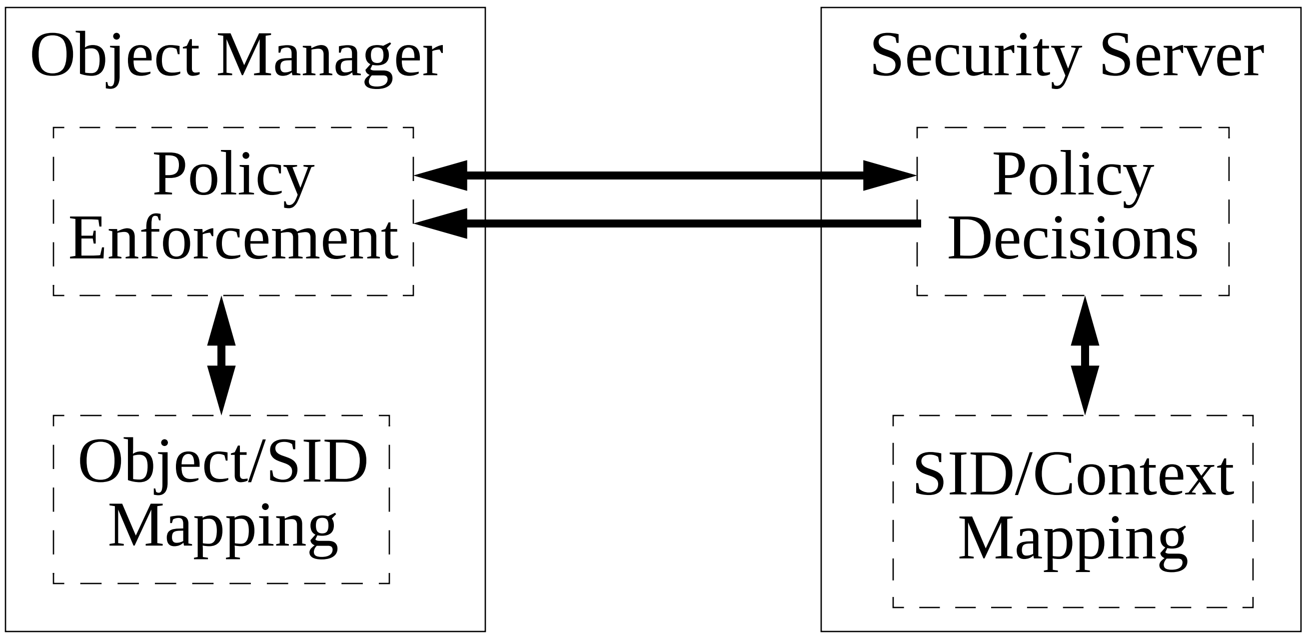
- Separates enforcement from policy
- No assumptions in labels
- Supports many policies
- No need for trusted subjects
- Controls entry into domains via program types
- Controls execution of program types by domains
- Enables assured pipelines

Downside is complexity of access matrix

The Flask Security Architecture

- Cleanly separates policy from enforcement
- Well-defined policy interfaces
- Support for policy changes
- Allows users to express policies naturally
- Fine-grained controls over kernel services
- Caching to minimize performance overhead
- Transparent to applications and users

Encapsulation of Policy



Policy Decisions

Labeling Decisions: Obtaining a label for a new subject or Object

Access Decisions: Determining whether a service on an object should be granted to a subject

Polyinstantiation Decisions: Determining where to redirect a process when accessing a polyinstantiated object

Permissions and Object Classes

Permissions are defined on objects and grouped together into object classes

Examples

- process: execute, fork, transition, sigchld, sigkill, sigstop, signal, ptrace, getsched, setsched, getsession, getpgid, setpgid, getcap, setcap, entrypoint
- file: poll, ioctl, read, write, create, getattr, setattr, lock, relabelfrom, relabelto, transition, append, access, unlink, link, rename, execute
- dir: file class perms + add_name, remove_name, reparent, search, mounton, mountassociate
- security: compute_av, notify_perm, transition_sid, member_sid, sid_to_context, context_to_sid, load_policy, get_sids, register_avc, change_sid
- Other classes include: socket, filesystem, fd, and capability

Security Server Interface

Object Labeling

- Request SID to label a new object

```
int security_transition_sid(
    security_id_t     ssid,       /* IN */
    security_id_t     tsid,       /* IN */
    security_class_t  tclass,     /* IN */
    security_id_t     *out_sid);  /* OUT */
```

- Example of usage for new file label in fs/namei.c:vfs_create

```
error = security_transition_sid(current->sid,
                                dir->i_sid,
                                SECCLASS_FILE,
                                &sid);
```

Security Server Interface (cont.)

Access Decisions

- Request Access Vector for a given object class/permissions

```
int security_compute_av(
    security_id_t      ssid,       /* IN */
    security_id_t      tsid,       /* IN */
    security_class_t   tclass,     /* IN */
    access_vector_t    requested,  /* IN */
    access_vector_t    *allowed,   /* OUT */
    access_vector_t    *decided,   /* OUT */
    __u32              *seqno);    /* OUT*/
```

- Ignores access vectors for auditing and requests of notifications of completed operations
- returns 0 unless error

Security Server Interface (cont.)

Access Vector Cache (AVC)

- security_compute_av() called indirectly through AVC

```
extern inline int avc_has_perm_ref_audit(
    security_id_t       ssid,        /* IN */
    security_id_t       tsid,        /* IN */
    security_class_t    tclass,      /* IN */
    access_vector_t     requested,   /* IN */
    avc_entry_ref_t     *aeref,      /* IN */
    avc_audit_data_t    *auditdata);/* IN */
```

- aeref is attempt to point directly to cache entry. If invalid then security_compute_av() is called
- Returns 0, -EACCES or an appropriate code if error occurs

File permissions check shortcuts

- inline int dentry_mac_permission(struct dentry *d, access_vector_t av)
- Usage: err = dentry_mac_permission(nd->dentry, DIR_SEARCH);
  /* from fs/namei.c:path_walk)

Permission Checking Examples

unlink() from fs/namei.c: vfs_unlink()

```
error = dentry_mac_permission(dentry, FILE__UNLINK);
if (error)
    return error;
```

- Additional directory-based checks in fs/namei.c:may_delete()
  - search and remove_name permissions

Process to socket check from net/ipv4/af_inet:inet_bind()

```
lock_sock(sk);
ret = avc_has_perm_ref(current->sid, sk->sid, sk->sclass,
                       SOCKET__BIND, &sk->avcr);
release_sock(sk);
if (ret)
    return ret;
```

Permission Checking Examples

open() from fs/namei.c: open_namei()

```
/* Checks for existing file */
if (flag & FMODE_READ) {
    error = dentry_mac_permission(dentry, FILE__READ);
    if (error)
        goto exit;
}
if (flag & FMODE_WRITE) {
    if (flag & O_APPEND) {
        error = dentry_mac_permission(dentry, FILE__APPEND);
        if (error)
            goto exit;
    } else {
        error = dentry_mac_permission(dentry, FILE__WRITE);
        if (error)
            goto exit;
    }
}
```

Permission Checking Examples

execve() from fs/exec.c: prepare_binprm()

```
if (!bprm->sid) {
    retval = security_transition_sid(current->sid, inode->i_sid,
                        SECCLASS_PROCESS, &bprm->sid);
    if (retval) return retval; }
if (current->sid !=bprm->sid && !bprm->sh_bang) {
    retval = AVC_HAS_PERM_AUDIT(current->sid, bprm->sid,
                        PROCESS, TRANSITION, &ad);
    if (retval) return retval;
    retval = process_file_mac_permission(bprm->sid, bprm->file,
                        PROCESS_ENTRYPOINT);
    if (retval) return retval; }
retval = process_file_mac_permission(bprm->sid, bprm->file,
                        PROCESS_EXECUTE);
if (retval) return retval;
```

Also checks file:execute, fd:inherit and process:ptrace and wakes parent for process:sigchld

Example Security Server

Implements combination of Identity-Based Access Control, Role-Based Access Control, Type Enforcement, and optional Multilevel Security

Labeling, access, and polyinstantiation decisions defined through set of configuration files

Security objectives of example policy include:

- Protection of kernel integrity and initialization
- Protection of system software and configuration integrity
- Protection of system administrator role and domain
- Confinement of damage caused by exploitation of flaws by limiting privileges
- Protection against privileged processes executing malicious code

Security Policy Example

Example from TE policy to allow sys adm to run insmod

```
allow sysadm_t insmod_exec_t:file x_file_perms;
allow sysadm_t insmod_t:process transition;
allow insmod_t insmod_exec_t:process {entrypoint execute};
allow insmod_t sysadm_t fd:inherit_fd_perms;
allow insmod_t self:capability sys_module;
allow insmod_t sysadm_t:process process sigchld;
```

Compatibility

SELinux controls transparent to applications and users

- Default behavior allows existing interface to be unchanged
- Access failures return normal error codes
  - e.g. EACCES, EPERM, ECONNREFUSED, ECONNRESET
  - few exceptions: e.g. read and write
- Extended API for security-aware applications
  - Specifying SIDs: e.g. execve, open, and socket
  - Getting/setting security information
  - Conversion between SIDs and Security Contexts
- SS interface for user-space object managers
  - Use controlled by policy
  - Enables refinement of kernel policy
  - Application AVC library easily produced

Code Maintainability

Existing Functionality

- Well-contained checking
  - Similar in complexity to existing DAC checks
  - Collocated w/existing DAC checks where possible
  - AVC managed automatically
- Security Server encapsulates security policy
  - Implemented policy may be changed w/o effecting rest of kernel
  - Changes to interface or existing permisions affect kernel
  - Changes to SS or permission set can affect existing policies
  - Changes to internal interfaces can affect SS or AVC

New Functionality requires:

- Definition of permissions for new functions that need control
- Updating distribution security policies for new permissions

Development mode useful for testing kernel functionality and writing policies

Performance Testing

Set up

- Benchmarks run on 333MHz Pentium II w/128M RAM and 64M swap
- lmbench net tests used 166MHz Pentium w/128M RAM and 64M swap for server programs
- Tests configurations
  - Base - unmodified Linux 2.4.2
  - SELinux - example Security Server w/policy

No attempt was made to optimize for performance

- Numbers should be treated as upper bounds

Performance Testing

Macrobenchmark results

- Compilation of 2.4.2 kernel w/default options

Time	Base	Selinux	%
elapsed	11:13.60	11:15.18	0
system	49.32	50.92	3
user	600.86	601.03	0

Performance Testing (cont.)

Microbenchmarks

- UnixBench 4.1.0

Microbenchmark	Base	SELinux	%
execl	403.8	383.3	5.3
file copy -4K	50652.0	49759.0	1.8
file copy -1K	41406.0	39566.0	4.7
file copy -256	23586.0	21485.0	9.8
pipe	161955.9	139475.2	16.1
pipe switching	78555.8	66805.4	17.6
process creation	2061.9	2022.7	1.9
system call	162049.4	162033.4	0.0
shell scripts (8)	91.0	87.7	3.8

- file copy in KB/sec, rest in loops/sec (minute for shell scripts)

Performance Testing (cont.)

- lmbench 2
  - simple file operations, process creation, and program execution(microseconds)

Microbenchmark	Base	SELinux	%
null I/O	1.45	1.93	33
stat	8.06	10.25	27
open/close	11.0	14	27
0K create	22	26	18
0K delete	1.72	1.90	10
fork	499.0	504.75	1
execve	2725.75	2816.5	3
sh	10K	11K	10

Performance Testing (cont.)

- lmbench 2
  - communication latency (microseconds)

Microbenchmark	Base	SELinux	%
pipe	12.5	14	12
AF_UNIX	20.6	24.6	19
UDP	309.75	355.60	14.8
RPC/UDP	441.25	519.20	17.7
TCP	389.00	425.00	9.25
RPC/TCP	667.25	725.80	8.77
TCP connect	674.50	737.80	9.38

  - no difference on bandwidth benchmarks

More on Performance

Security does not come for free!

Preliminary results w/o optimization not too bad

Areas for Improvement

- Optimization of AVC and SS
- Analysis of AVC refs
- Fine-grained locks on AVC

Related Work

Rule Set Based Access Control (RSBAC)

- Generalized Framework for Access Control for Linux
- Similarities
  - Separates policy from enforcement
  - Role Compatibility Modules close to SELinux TE module
- Distinctions
  - RSBAC lacks controls for each kernel subsystem
  - RSBAC lacks support for dynamic policy changes as GFAC doesn't address atomic policy changes
  - RSBAC relies on kernel-specific data structures - No policy-independent data types
  - RSBAC decisions not cleanly decoupled from kernel
  - RSBAC has no decision caching mechanism
  - RSBAC bases decisions on real uid and must control change
  - RSBAC lacks support for user-level object managers
  - RSBAC uses new utilities/syscalls for policy config rather than human readable config files

Related Work

Type Enforcement/Domain and type Enforcement

- Configurable Access Control Effort and DTE for Linux
- Similarities
  - Flask Architecture includes generalization of TE
  - DTE and SELinux both use policy language for expressing policy
- Distinctions
  - SELinux has greater encapsulation of labels and policy logic
  - SELinux has more support for dynamic policies
  - SELinux has finer-grained controls and explicit persistent labels
  - SELinux can support both TE or DTE in Security Server

Trusted BSD

- Adding security features including MAC
- SELinux more mature and more flexible
- TrustedBSD plans to migrate to more flexible approach

Related Work

Linux Intrusion Detection System (LIDS)

- Provides set of additional security features for Linux
  - Controls view, preventing process from being killed, security alerts, detecting port scans
- Administrative ACLs for files that ID subjects based on program

Medusa DS9

- Similar to SElinux at high level
  - AC architecture that separates policy from enforcement
- Very different in specifics
  - User space authorization server based on labeling with sets of virtual spaces to define view and accesses
  - Support for syscall interception and forcing execution of code provided by server

LOMAC

- Implements Low Watermark model in a loadable kernel module
  - Useful integrity protection w/compatibility for existing SW
  - SELinux should be able to implement LOMAC

Key Distinctions of SELinux

Comprehensive and flexible system with a well-defined security architecture validated by several prototypes

Provides

- Clean separation of policy and enforcement with well-defined policy interfaces
- Policy, policy language, and label format independence
- Support for policy changes
- Individual labels and controls for kernel objects and services
- Caching of access decisions for efficiency
- Fine-grained controls over file systems, directories, files, open file descriptions, sockets, messages, network interfaces, and capability use
- Transparency to security-unaware applications via default behavior

Assurance work for Flask Architecture

Conclusion

Mandatory Access Control needed for meaningful security

Flexibility and completeness is correct way to go

SELinux warrants a close look

Open Source community is in a position to take a good security architecture, improve its implementation and enable its wide distribution

Additional Features needed for security

- Trusted Path and Protected Path