Meeting Critical Security Objectives with Security-Enhanced Linux

Peter A. Loscocco
Information Assurance Research Group
National Security Agency

Co-author: Stephen D. Smalley, NAI Labs
Presentation Outline

• Operating system security
• The Flask architecture
• Security-enhanced Linux
• Example security server
• Meeting critical security objectives
• Future Direction
The Need for Secure OS

- Increasing risk to valuable information
- Dependence on OS protection mechanisms
- Inadequacy of mainstream operating systems
- Key missing feature: Mandatory Access Control (MAC)
  - Administratively-set security policy
  - Control over all subjects and objects in system
  - Decisions based on all security-relevant information
Why is DAC inadequate?

- Decisions are only based on user identity and ownership
- No protection against malicious software
- Each user has complete discretion over his objects
- Only two major categories of users: superuser and other
- Many system services and privileged programs must run with coarse-grained privileges if not as superuser
What can MAC offer?

- Strong separation of security domains
- System and data integrity
- Ability to limit program privileges
- Protection against tamper and bypass
- Processing pipelines guarantees
- Authorization limits for legitimate users
MAC Implementation Issues

- Must overcome limitations of traditional implementations
  - More than just Multilevel Security
  - Address integrity, least privilege, separation of duty issues
  - Complete control using needed security relevant information
  - Control relationships between subjects and code

- Policy flexibility required
  - One size does not fit all!
  - Ability to change the model of security
  - Ability to express different policies within given model
  - Separation of policy from enforcement

- Maximize security transparency
Customize according to need

- **Separation policies**
  - Establishing Legal Restrictions on data
  - Restrictions to classified/compartmented data

- **Confinement policies**
  - Restricting web server access to authorized data
  - Minimizing damage from viruses and other malicious code

- **Integrity policies**
  - Protecting applications from modification
  - Preventing unauthorized modifications of databases

- **Invocation policies**
  - Guaranteeing that data is processed as required
  - Enforcing encryption policies
Security Solutions with Flexible MAC

- Confines malicious code
  - Can safely run code of uncertain pedigree
  - Constrains code inserted via buffer overflow attacks
  - Limits virus propagation

- Allows effective decomposition of root
  - Root no longer all powerful
  - Limits each root function to needed privilege
  - Eliminates most privilege elevation attacks

- Allows effective assignment of privilege
  - Servers need not run with complete access
  - Servers and needed resources can be isolated
  - Separate protections for system logs
Toward a New Form of MAC

- Research by NSA with help from SCC
- Generalized from prior Type Enforcement work
- Provide flexible support for security policies
- Cleanly separate policy from enforcement
- Address limitations of traditional MAC
- DTMach, DTOS, Flask
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.
The Flask Security Architecture

- Object Manager
  - Policy Enforcement
  - Object/SID Mapping

- Access Vector Cache
  - Subject, object, class, requested
  - Yes/no

- Security Server
  - Policy Decisions
  - SID/Context Mapping
  - Access vector
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.
Policy Changes

- Interfaces to AVC for policy changes
- Callbacks to Object Managers for retained permissions
- Sequence numbers to address interleaving
- Revalidation of permissions on use
**Controlled Services**

- Permissions are defined on objects and grouped together into object classes

- **Examples**
  - Process: code execution, transitions, entrypoints, signals, wait, ptrace, capabilities, etc.
  - File: fd inheritance and transfer, accesses to files, directories, file systems
  - Socket: accesses to sockets, messages, network interfaces, hosts
  - System V IPC: accesses to semaphores, message queues, shared memory
  - Security: accesses to security server services
Security Server Interface

- **Object Labeling**
  - Request SID to label a new object
    - `int security_transition_sid(ssid, tsid, tclass, *out_sid)`
  - Example of usage for new file label
    - `error = security_transition_sid(current->sid, dir->i_sid, FILE, &sid);`
Security Server Interface (cont.)

- **Access Decisions**
  - Request Access Vector for a given object class/permission
    - int security_compute_av(ssid, tsid, tclass, requested, *allowed, *decided, *seqno);
  - Ignores access vectors for auditing and requests of notifications of completed operations
Security Server Interface (cont.)

- **Access Vector Cache (AVC)**
  - `security_compute_av()` called indirectly through AVC
    - `int avc_has_perm_ref(ssid, tsid, tclass, requested, *aeref, *auditdata)`
  - `aeref` is hint to cache entry. If invalid then `security_compute_av()` is called

- **File permission check shortcuts**
  - `int dentry_mac_permission(struct dentry *d, access_vector_t av)`
Permission Checking Examples

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

  ```c
  error = dentry_mac_permission(dentry, FILE_UNLINK);
  if (error)
      return error;
  ```

  - Additional directory-based checks for search and remove_name permissions

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

  ```c
  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

- `execve()` from fs/exec.c: `prepare_binprm()`
  ```
  if (!bprm->sid) {
    retval = security_transition_sid(current->sid, inode->i_sid,
                                       SECLASS_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, process:ptrace
API Enhancements

- Existing Linux API calls unchanged
- New API calls for security-aware applications: execve_secure, mkdir_secure, stat_secure, socket_secure, accept_secure, etc.
- New API calls for application policy enforcers: security_compute_av, security_transition_sid, etc.
Example Security Server

- Implements combination of Role-Based Access Control, Type Enforcement, optional Multi-Level Security.
- Labeling, access, and polyinstantiation decisions defined through set of configuration files.
- Example policy configuration provided.
Example Policy Configuration: TE Concepts

• Domains for processes, types for objects.
• Specifies allowable accesses by domains to types.
• Specifies allowable interactions among domains.
• Specifies allowable and automatic domain transitions.
• Specifies entrypoint and code execution restrictions for domains.
Type Enforcement: Domains

- init_t
  - init_exec_t
  - getty_t
    - getty_exec_t
  - initrc_t
    - initrc_exec_t
  - login_t
    - login_exec_t
  - user_t
    - shell_exec_t
  - user_netscape_t
    - netscape_exec_t
  - sendmail_t
    - sendmail_exec_t
  - klogd_t
    - klogd_exec_t
Type Enforcement: Types

/:
  root_t

dev:
  device_t
  kmem:
    memory_device_t

bin:
  bin_t
  login:
    login_exec_t

var:
  var_t
  log:
    var_log_t
  wtmp:
    wtmp_t
Sample TE Rules

allow sendmail_t smtp_port_t:tcp_socket name_bind;

type_transition getty_t login_exec_t:process local_login_t;
Example Policy Configuration: RBAC concepts

- Roles for processes
- Specifies domains that can be entered by each role
- Specifies roles that are authorized for each user
- Initial domain associated with each user role
- Role transitions are typically explicit, e.g. login or newrole
### Role-Based Access Control: Roles

<table>
<thead>
<tr>
<th>Role</th>
<th>Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>system_r</td>
<td>init_t, getty_t, klogd_t, sendmail_t</td>
</tr>
<tr>
<td>user_r</td>
<td>user_t, user_netscape_t, passwd_t</td>
</tr>
<tr>
<td>sysadm_r</td>
<td>sysadm_t, insmod_t, fsadm_t</td>
</tr>
</tbody>
</table>
Example Policy Configuration: Security Objectives

- Protect kernel integrity, including boot files, kernel modules, sysctl variables
- Protect integrity of system software, configuration files, and logs
- Protect administrator role and domain
- Confine system processes and privileged programs
- Protect against execution of malicious software
Limiting raw access to data

- Controlling `fsck` and related utilities

  ```
  allow fsadm_t fsadm_exec_t:process
    { entrypoint execute };
  allow fsadm_t fixed_disk_device_t:blk_file
    { read write };
  allow initrc_t fsadm_t:process transition;
  allow sysadm_t fsadm_t:process transition;
  ```
Limiting raw access to data

- Granting access to `klogd`

```plaintext
allow klogd_t klogd_exec_t:process
  { entrypoint execute };
allow klogd_t memory_device_t:chr_file
  { read write };
allow initrc_t klogd_t:process transition;
```
Kernel integrity protection

- Protecting `/boot` files

```c
allow initrc_t boot_t:dir
    { read search add_name remove_name };  
allow initrc_t boot_runtime_t:file
    { create write unlink };  
type_transition initrc_t boot_t:file boot_runtime_t;
```
Kernel integrity protection

- Controlling use of `insmod` program

```c
allow syssadm_t insmod_exec_t:file x_file_perms;
allow syssadm_t insmod_t:process transition;
allow insmod_t insmod_exec_t:process
    { entrypoint execute };
allow insmod_t syssadm_t:fd inherit_fd_perms;
allow insmod_t self:capability sys_module;
allow insmod_t syssadm_t:process sigchld;
```
System file integrity protection

- Separate types for system programs
  - e.g. bin_t, sbin_t
- Separate types for system configuration files
  - e.g. etc_t
- Separate type for shared libraries
  - e.g. shlib_t
- Separate types for system logs
  - e.g. wtmp_t
- Separate type for dynamic linker
  - e.g. ld_so_t
System file integrity protection

- **Granting sendmail accesses**
  - allow sendmail_t etc_aliases_t:file { read write };
  - allow sendmail_t etc_mail:dir
    - { read search add_name remove_name };
  - allow sendmail_t etc_mail_t:file
    - { create read write unlink };

- **Granting logfile accesses**
  - allow local_login_t wtmp_t:file { read write };
  - allow remote_login_t wtmp_t:file { read write };
  - allow utempter_t wtmp_t:file { read write };

Confining privileged processes

- excerpt for sendmail

```plaintext
allow sendmail_t smpt_port_t:tcp_socket name_bind;
allow sendmail_t mail_spool_t:dir
    { read search add_name remove_name };
allow sendmail_t mail_spool_t:file
    { create read write unlink };
allow sendmail_t mqueue_spool_t:dir
    { read search add_name remove_name };
allow sendmail_t mqueue_spool_t:file
    { create read write unlink };
```
Confining privileged processes

- excerpt for ftpd

allow ftpd_t wtmp_t:file append;
allow ftpd_t var_log_t:file append;
allow ftpd_t ls_exec_t:process execute;
Separating Processes

- Access across domains restricted to privilege processes
  - signals, ptrace, /proc
- Access to temporary files controlled
  
  ```
  allow user_t tmp_t:dir
  { read search add_name remove_name } ;
  allow user_t user_tmp_t:file
  { creat read write unlink };
  type_transition user_t tmp_t:file user_tmp_t;
  ```

- Similar controls for home directories and terminal devices
Administrator domain protection

- Controlling access to sysadm_t
  
  ```
  type_transition getty_t login_exec_t:process local_login_t;
  allow local_login_t sysadm_t:process transition;
  allow newrole_t sysadm_t:process transition;
  ```

- Execution limited to approved types
- Separation from other domains
Malicious software protection

- Example putting netscape in its own domain
  
  ```
  type_transition user_t netscape_exec_t:process user_netscape_t;
  allow user_t netscape_exec_t:process
  { entrypoint execute };
  allow user_netscape_t user_netscape_rw_t:file
  { read write create unlink };
  ```
Performance

- Initial performance measurements reported at 2001 Usenix Conference

- Benchmark Summary
  - Macrobenchmarks showed no measurable overhead
  - Microbenchmarks showed small fixed overhead proportional to complexity of permission checks
  - Should be treated as upper bound - no optimization done

- Ongoing performance work (IBM Watson)
  - Scalability and locking issues
Ongoing and future work

- Define generalized hooks for kernel (LSM Project)
- Integrate with IPSEC/IKE and extend to support packet labeling and policy-based protection.
- Implement labeling and controls for NFS.
- Implement complete polyinstantiation support.
- Develop policy specification and analysis tools
Linux Security Module Project

- Goal is to develop common set of kernel hooks to allow security LKMs to be defined
- Hosted by WireX
  - http://lsm.immunix.com/
  - linux-security-module@wirex.com
- Status
  - Patch to 2.4.6 kernel w/most hooks defined
  - Currently working on networking hooks
- SELinux LKM using LSM patch ready
  - Available at http://www.nsa.gov/selinux/ soon
Questions?

Available at: http://www.nsa.gov/selinux/

Mailing list: Send ‘subscribe selinux’ to majordomo@tycho.nsa.gov

email: loscocco@tycho.nsa.gov
        ssmalley@nai.com