Security–Enhanced Linux

Implementation of a Formal Security Architecture

Achim D. Brucker

brucker@informatik.uni-freiburg.de

22. June 2001

Security–Enhanced Linux

Implementation of a Formal Security Architecture

Achim D. Brucker

brucker@informatik.uni-freiburg.de

22. June 2001

What's it all about?

End systems must be able to enforce the **separation of information** based on **confidentiality** and **integrity requirements** to provide system security.

What's it all about?

End systems must be able to enforce the **separation of information** based on **confidentiality** and **integrity requirements** to provide system security.

- Operating system security mechanism are the foundation for ensuring such separations.
- No existing mainstream operation systems supports critical security features.
- Application security mechanisms are vulnerable to tampering and bypassing.
- Malicious or flawed applications can easily cause failures in system security.

Note: Security-enhanced Linux is not an attempt to correct any flaws that may currently exist in Linux.

Overview

The formal Security Architecture: FLASK

Flask is an operating system security architecture that provides flexible support for security policies.

Achim D. Brucker 22nd June 2001 2

Overview

Achim D. Brucker 21st June 2001 0

The formal Security Architecture: FLASK

Flask is an operating system security architecture that provides flexible support for security policies.

Previous Next Start End Back FullScreen Quit

Implementation: SE–Linux

Security–enhanced Linux is a extension to the Linux operation system, that provides general support for the enforcement of many kinds of mandatory access control policies. SE–Linux is an implmentation of the FLASK architecture.

Achim D. Brucker 22nd June 2001 2

Overview

Philosophy and Future Development

The Open Source community and the NSA are integrating together a security layer into the Linux Kernel and pushing the further development of SE–Linux.

The Formal Security Architecture: Flask

Flask is an operating system security architecture that provides **flexible** support for security policies.

- Joined work from National Security Agency and Secure Computing Corporation (1992).
- First implementation using DTOS from SCC and Fluke from the University of Utah.
- On the operating system level: Flexible support for security policies.
- Using ideas of type enforcement to implement **mandatory access control (MAC)**.
- Specified using PVS, including proofs of the dynamic security policy.

Discretionary Access Control (DAC)

DAC is an access control mechanism that allows systems users to allow or disallow other users access to objects under their control.

- The "classical" Unix (POSIX.1) file access control is an implementation of DAC.
- Decisions are only based on user identity and ownership.
- Each user has complete discretion of his objects.

Discretionary Access Control (DAC)

DAC is an access control mechanism that allows systems users to allow or disallow other users access to objects under their control.

- The "classical" Unix (POSIX.1) file access control is an implementation of DAC.
- Decisions are only based on user identity and ownership.
- Each user has complete discretion of his objects.

- Only two major categories of users: users and superuser.
- Many programs and system services must run as superuser.
- No protection against malicious (flawed) software.

Mandatory Access Control (MAC)

MAC restricts access to objects based on the sensitivity of the information represented by the object and the formal authorization of subjects accessing information of such sensitivity.

- **Separation of Policies**, e.g. enforcing legal restriction on data
- **Containment of Policies**, e.g. minimizing damages from viruses or malicious code
- **Integrity of Policies**, e.g. protecting applications from modifications
- **Invocation of Policies**, e.g. enforcing encryption policies

Mandatory Access Control fulfills the need of (military) multi-level security!

Type Enforcement 1/2

Access matrix defining permissions between domains and types.

Traditional type enforcement policy:

- Each **subject** is labeled with a **domain**.
- Each **object** is labeled with a **type**.

The Flask type enforcement policy:

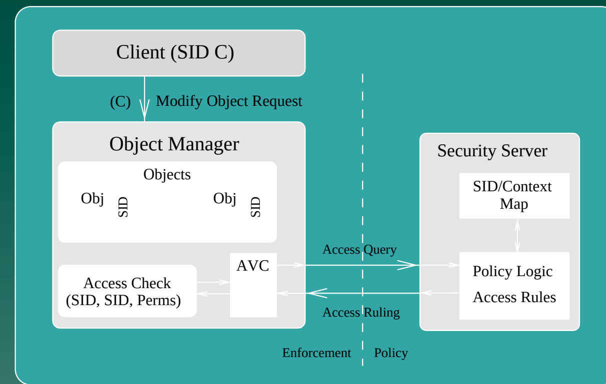
- Merges concept of domain and types.
- A "domain" is a type which can be associated with processes.
- Types can describe processes (subjects) and objects.
- Policy describes type hierarchy.

Type Enforcement 2/2

- Support for many policies.
- Separates enforcement from policy.
- No assumptions in labels.
- No need for trusted subjects.
- Control entry into domain via program types.
- Control execution of program types by domains.

Downside: Complexity of access matrix.

Requesting and caching of security decisions



Formal Specification: Excerpt of `THEORY AVC`

```
node allows(node, scls, ssid, tcls, tsid, perm, current time) : bool =
     sclass(node) = scls
     AND ssid(node) = ssid
     AND tclass(node) = tcls
     AND tsid(node) = tsid
     AND (expiration(node) = 0 OR expiration(node) ?= current time)
     AND member (perm, decided(node)) 30
     AND member (perm, allowed(node))
```


Formal Specification using PVS

- Formal “Top Level” specification.
- Formal specification of the “Security Policy Model”
- Formal proofs of dynamic security policy, e.g. security invariants hold in all system states.
- Formal proofs of fairness and liveness (security server), e.g. being deadlock free
- Implementation notes describing (informal) correspondence between specification and implementation.

Summary: The Flask Security Architecture

- Cleanly separates policy from enforcement.
- Well defined policy interfaces.
- Support for policy changes.
- Fine-grained control over kernel services.
- Caching to minimize performance overhead.
- Transparent to applications and users.

The Implementation

The NSA is now integrating the Flask architecture into the Linux operating system to transfer the technology to a larger developer and user community.

At time of writing, the implementation is available for the latest "user" kernel (2.4.3). Alternatively a version for kernel 2.2.19 is provided.

New Kernel functions and New or Changed Utilities

- Nearly 50 new system calls were added to the Linux kernel:
  `chsid`, `chsidfs`, `accept_secure`, `mkdir_secure`, `security_change_sid`, `send_secure`
  `security_load_policy`, `stat_secure`, `socket_secure`,...
- Around 15 user utilities were afected:
  `chcon`, `killall`, `ls`, `newrole`, `runas`, `tar`, `ps`, `mknod`, `mkfifo`,...

Security Policy Example

Example from Type Enforcement policy to allow system administrator 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;
```

Code Maintainability

- Existing Functionality:
  - Well contained checking, similar in complexity to existing DAC checks.
  - Security Server encapsulates security policy, e.g. changes do not affect the kernel.
- New Functionality requires:
  - Definitions of permissions for new functions that need control.
  - Updating distribution security policies for new permissions.

Compatibility

Security–Enhanced Linux controls transparent to applications and users.

- Default behavior allows existing interface to be unchanged.
- Access failures return normal error codes.
- Extended API for security–aware applications.
- Security server interface for user–space object managers.

Impact on Performance (excerpt)

- simple file operations

Benchmark	Base	SELinux	%
open/close	11.00	14.00	27.0
stat	8.06	10.25	27.0
fork	499.00	504.75	1.0
file copy (4k)	50652.00	49759.0	1.8

- communication latency

Benchmark	Base	SELinux	%
UDP	309.75	355.60	14.80
TCP	389.00	425.00	9.25
TCP connect	674.50	737.80	9.38
AF_UNIX	20.60	24.60	19.00

Security comes not for free!

Benefits of Using Security–Enhanced Linux

- Separation of information based on confidentiallity and integrity requirements.
- Protection against unauthorized modification or disclosure of data.
- Protection against tampering with the kernel or applications.
- Protection against bypassing application security mechanisms.
- Protection against the execution of untrustworthy programs.
- Protection against interference with other processes.
- Confinement of the potential damage caused by malicious or flawed programs.

Security–Enhanced Linux distitions

- Clean separation of policy and enforcement with well-defined policy interfaces.
- Support for policy changes.
- Caching for efficiency .
- Fine-grained controls over: file system, sockets, messages, network interfaces, capabilities.
- Transparency to security-unaware applications via default behavior.

Remaining (research) tasks

- Integrate IPSEC with network mandatory controls.
- Implement mandatory controls for NFS.
- Improve and simplify the policy configuration system.
- Complete the general purpose policy configuration.
- Perform functional and performance testing.
- Integrate existing publicly available file cryptography with file mandatory controls.
- Packages for several Linux distributions.

Related Work

- Rule Set Based Access Control (RSBAC)
- Type Enforcement (TE) *and* Domain and type Enforcement (TDE)
- Trusted BSD
- Linux Intrusion Detection System (LIDS)
- Medusa DS9
- LOMAC

NSA development meets Open Source

NSA plays the game of the Open Source community:

- Released under the GNU General Public Licence ☺.
- CVS based development hosted on sourceforge.
- Mailing lists for communication.
- NSA presented work on the Linux Kernel Summit (for kernel version 2.5).

Why was Linux chosen as the base platform?

Linux was chosen as the platform for the work because of its growing success and open development environment. Linux provides an excellent opportunity to demonstrate that this functionality can be successful in a **mainstream operating system** and, at the same time, contribute to the security of a widely used system. A Linux platform also offers an excellent opportunity for this work to receive the widest possible review and perhaps provide the foundation for additional security research by others.

Future Development: The NSA World

- NSA does further development (open positions!).
- NSA pays for NAI for further development ($1.2 million 2-year contract)
- NSA is trying to get Security–Enhanced Linux into the official kernel branch.
- No (official) statement concerning internal use.

Future Development: The Open Source World

- Common understanding that security (within the operating system) is important.
- Only simple security (DAC and capabilities) will be in the kernel by default.
- The major kernel release will support a "security layer" providing a easy way to plug any security mechanism into the kernel.
- The Linux Security Module (lsm) project was founded for defining the security layer (mainly driven by NSA and Immunix..

Personal Conclusion

My personal "killer application" combining formal methods, security, Linux and open source development.

- Fully formalized and powerful security architecture.
- Clean implementation of this architecture for Linux.
- Even if Security–Enhanced Linux is not included in the standard kernel, it has a great impact of security model support of future Linux kernels.

[1] NAI Labs Security-Enhanced Linux, June 2001. http://pgp.com/research/nailabs/secure-execution/secure-linux.asp.

[2] Rule Based Access Control, June 2001. http://csrc.nist.gov/rbac/.

[3] Assurance in the Fluke Microkernel Formal Top Level Specification. February 1999.

[4] Peter Loscocco. Security-Enhanced Linux, 2001. http://www.nsa.gov/selinux/docs.html. Presentation at the Linux Kernel 2.5 Summit.

[5] Peter Loscocco and Stephen Smalley. Integrating Flexible Support for Security Policies into the Linux Operating System. April 2001.

[6] Peter A. Loscocco, Stephen D. Smalley, Patrick A. Muckelbauer, Ruth C. Tayler, S. Jeff Turner, and John F. Farrell. The Inevitability of Failure: The Flawed Assumption of Security in Modern Computing Environments.

[7] Stephen Smalley and Timothy Fraser. A Security Policy Configuration for the Security-Enhanced Linux. April 2001.

[8] Ray Spencer, Stephen Smalley, Peter Loscocco, Mike Hilber, David Anderson, and Jay Lepreau. The Flask Security Architecture: Support for Diverse Security Policies. http://www.cs.utah.edu/flux/flask.