

# Security-Enhanced Linux

*(A brief introduction.)*

Stefan Schulze Frielinghaus  
[stefan@sf-net.com](mailto:stefan@sf-net.com)

June 27, 2007

# Agenda

## 1. Introduction

- ▶ What is SELinux
- ▶ Differences to other security models

## 2. Concepts

- ▶ Type Enforcement
- ▶ Role-Based Access Control (RBAC)
- ▶ Multi-Level Security (MLS) / Multi-Category Security (MCS)
- ▶ Putting all together

## 3. Modular policy

- ▶ Targeted
- ▶ Strict
- ▶ Booleans

▶ Standard SELinux users / roles / etc.

4. Preview

5. Appendix

▶ Literature

# Chapter 1

## Introduction

# What is SELinux

Security-enhanced Linux (SELinux) was originally developed as a research prototype of the Linux® kernel and a number of utilities with enhanced security functionality designed to demonstrate the value of mandatory access controls to the Linux community and how such controls could be added to Linux. Today SELinux is integrated into the mainline Linux 2.6 kernel series and several Linux distributions. The Security-enhanced Linux kernel contains new architectural components originally developed to improve the security of the **Flask operating system**. These architectural components provide general support for the enforcement of many kinds of **mandatory access control** policies, including those based on the concepts of **Type Enforcement®**, **Role-based Access Control**, and **Multi-level Security**.<sup>1</sup>

---

<sup>1</sup><http://www.nsa.gov/selinux/info/faq.cfm#I1>

# What is SELinux

And in plain?

- ▶ SELinux is based on the concepts of the Flux Advanced Security Kernel (Flask). The fundamentals were developed in the early 90th by the National Security Agency (NSA), Secure Computing Corporation (SCC) and the University of Utah.

# What is SELinux

- ▶ The system mainly implements the mandatory access control. This is called **type enforcement** (TE) under SELinux terminology which tries to break down the access of programs into different security levels. The goal is to achieve a flexible and secure system where it is possible to run every program under a different security context.

# What is SELinux

- ▶ Another important SELinux terminology
  - Subject: Specifies a process
  - Object: Specifies a file

# Differences to other security models

## Discretionary Access Control (DAC)

- ▶ Most common operating systems implement by default the DAC model (e.g. Linux, Windows, \*BSD).
- ▶ This model allows access to a resource based on the username and/or groupname.

# Differences to other security models

- ▶ The DAC model doesn't distinguish between commands the user executes.

For example, if the user has the rights to execute the utilities “*screen*” and “*ls*” the Linux operating system wouldn't distinguish between them. After execution the commands would run under the calling user context.

# Differences to other security models

## Type Enforcement

Every program could run under another security context.

This means:

- ▶ Subjects and objects could be distinct between themselves.
- ▶ It's possible to break down the access into different levels.

# Differences to other security models

For example, consider the access to a file. Linux for itself has three possibilities to limit the access:

read, write and execute (for user, group and world)

SELinux divides this to the following permissions: append, create, execute, getattr, ioctl, link, lock, mounon, quotaon, read, relabelfrom, relabelto, rename, setattr, swapon, unlink, write

# Differences to other security models

Remember the example with the commands “*screen*” and “*ls*”. Now we will run *screen* on a Linux machine without SELinux enabled:

```
# ps auxww | grep screen
stefan 5640 0.0 0.1 27552 884 p4 S+ 13:05 0:00 screen
```

The output looks like we had expected.

# Differences to other security models

And now with SELinux enabled:

```
# ps auxww | grep screen
staff_u:staff_r:staff_screen_t:s0 stefan  9459  0.0  0.
0    3820  1004 pts/0    S+   13:18   0:00 screen
```

## Differences to other security models

The interesting difference is that *screen* runs under a special context **staff\_u:staff\_r:staff\_screen\_t:s0**. So therefore it would be possible to write a policy allowing operations on resources especially for the program *screen*.

# Chapter 2

## Concepts

# Type Enforcement



Type: game\_t



Type: firefox\_t



Type: palm\_sync\_t



Type: music\_player\_t

Type: your\_tool\_t



# Type Enforcement

Access is granted for a subject and an object by four elements:

- ▶ source type
- ▶ destination type
- ▶ object type
- ▶ permissions

Example:

```
allow user_t gpg_exec_t:file { execute getattr read }
```

# Type Enforcement

Consider the example with GnuPG. GPG saves its configuration by default in *\$HOME/.gnupg*.

Under a standard Linux system every process running under the users context has potentially access to the private key stored in the GPG configuration directory.

# Type Enforcement

**GnuPG configfiles  
including private key**



**The Internet**



# Type Enforcement

On a standard system with DAC security Firefox is running with the privileges of user X and has therefore the rights to read the users X private key.

# Type Enforcement

On a SELinux system the files can't be read by Firefox until you explicitly allow it by the policy.

# Type Enforcement

Summary:

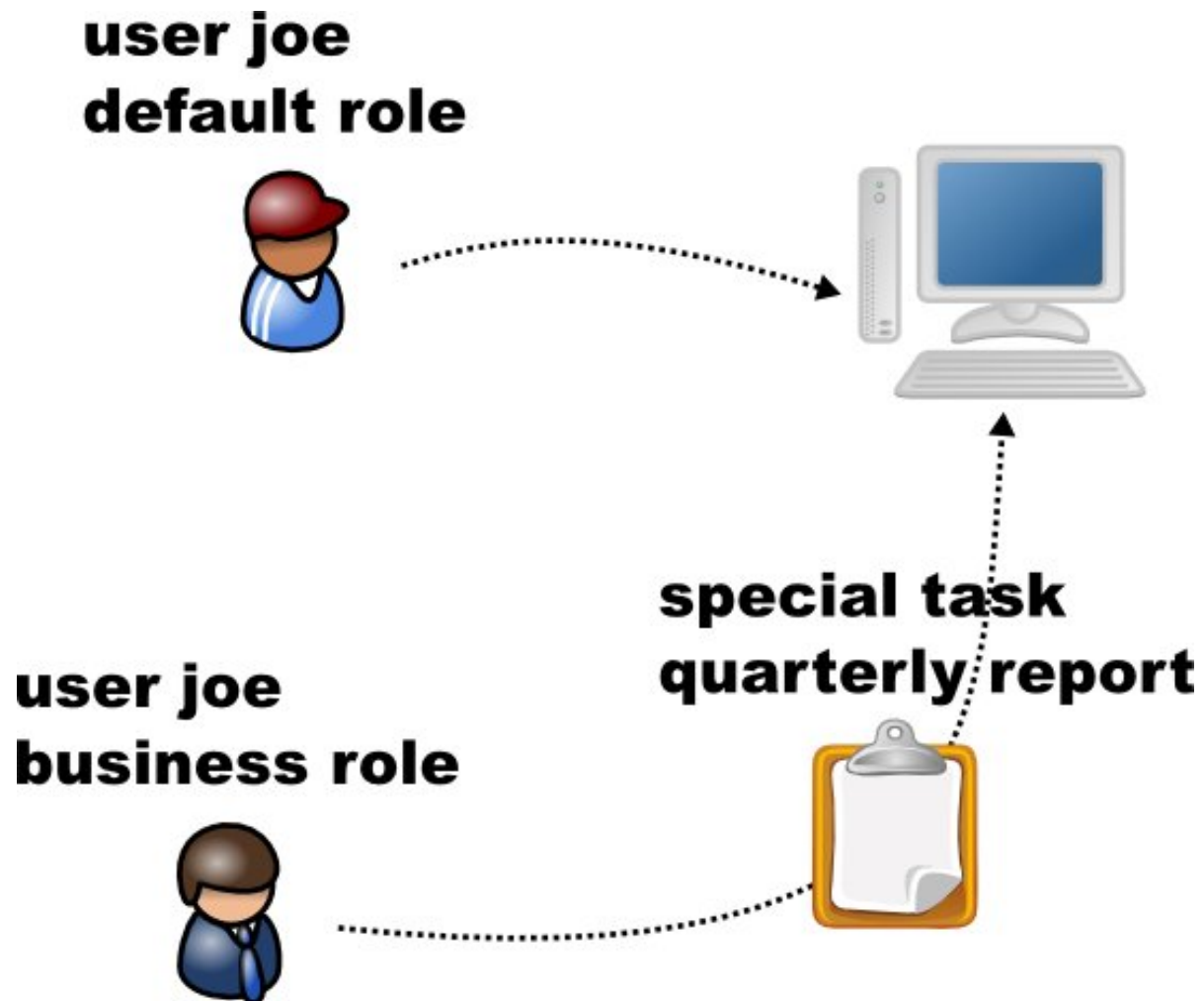
- ▶ There is no super user under SELinux.
- ▶ In theory every subject or object could be clearly differentiated.

It's not a security system based on vertical access. **It's a system based on horizontal secure access!**

# Role-Based Access Control (RBAC)

- ▶ Define roles and grant them special privileges
- ▶ Change the role while you are logged in

# Role-Based Access Control (RBAC)



# Multi { Level, Category } Security (MLS / MCS)

- ▶ Rather interesting for governments, military etc.
- ▶ **Security level** divided into two parts:
  - sensitivity
  - category
- ▶ “need to know” principle

# Multi { Level, Category } Security (MLS / MCS)

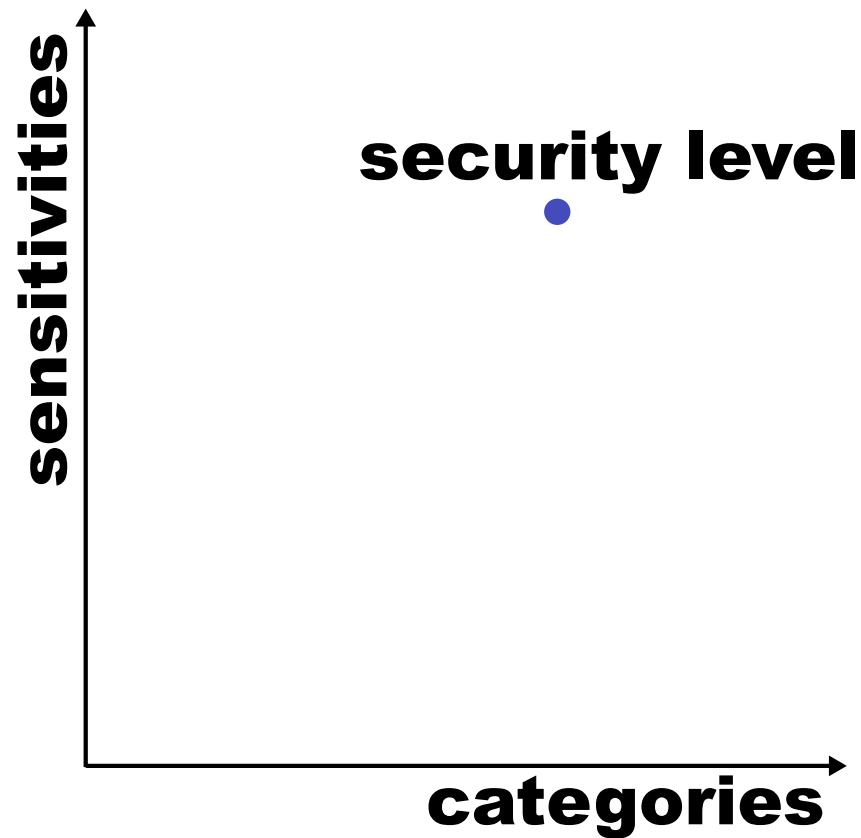
## Sensitivity example

unclassified ↔ restricted ↔ confidential ↔ secret ↔ top secret

## Category examples

students — professors — management

# Multi { Level, Category } Security (MLS / MCS)



Sensitivities: hierarchical

Categories: non hierarchical and unsorted like a set

# Multi { Level, Category } Security (MLS / MCS)

**Sensitivity:**

Users (subjects) can only **read files at or below** their own security level (**no read up**).

# Multi { Level, Category } Security (MLS / MCS)

**Sensitivity:**

Users (subjects) can only **create files at or above** their own security level (**no write down**).

# Multi { Level, Category } Security (MLS / MCS)

**Sensitivity:** A mathematical point of view.

If  $L(S)$  is the security clearance of subject  $S$  and  $L(O)$  is the security classification of object  $O$  then

$S$  can read  $O$  if and only if  $L(O) \leq L(S)$

$S$  can write  $O$  if and only if  $L(S) \leq L(O)$

# Multi { Level, Category } Security (MLS / MCS)

**Sensitivity + Category = Security Level:**

Consider the following categories:

{ STUDENTS, PROFESSORS, MANAGEMENT }

and the following sensitivities:

( UNCLASSIFIED, RESTRICTED, CONFIDENTIAL, SECRET, TOP SECRET )

**The security level (L, C) *dominates* the security level (L', C') if and only if  $L' \leq L$  and  $C' \subseteq C$ .**

Since Bell-LaPadula Model

# Multi { Level, Category } Security (MLS / MCS)

## Sensitivity + Category = Security Level: Example

Joe has the following security level ( SECRET, { STUDENTS, MANAGEMENT } ) and a file FileA with the level ( CONFIDENTIAL, { STUDENTS } ) and a file FileB with the level ( SECRET, { PROFESSORS, MANAGEMENT } ).

Joe *dom* FileA as  $CONFIDENTIAL \leq SECRET$  and  $\{STUDENTS\} \subseteq \{STUDENTS, MANAGEMENT\}$

Joe  $\neg$ *dom* FileB as

$\{PROFESSORS, MANAGEMENT\} \not\subseteq \{STUDENTS, MANAGEMENT\}$

# Multi { Level, Category } Security (MLS / MCS)

and in SELinux terminology:

$SL_n \hat{=} \text{Security Level } n$

*dom*: (dominates)  $SL_1 \text{ dom } SL_2$  if the sensitivity of  $SL_1$  is *higher or equal to* the sensitivity of  $SL_2$ , *and* the categories of  $SL_1$  are a *superset* of the categories of  $SL_2$ .

# Multi { Level, Category } Security (MLS / MCS)

and in SELinux terminology:

$SL_n \hat{=} \text{Security Level } n$

*domby:* (dominated by)  $SL_1$  *domby*  $SL_2$  if the sensitivity of  $SL_1$  is *lower than or equal to* the sensitivity of  $SL_2$ , *and* the categories of  $SL_1$  are a *subset* of the categories of  $SL_2$ .

# Multi { Level, Category } Security (MLS / MCS)

and in SELinux terminology:

$SL_n \hat{=} \text{Security Level } n$

*eq:* (equals)  $SL_1 \text{ eq } SL_2$  if the sensitivity of  $SL_1$  and  $SL_2$  are equal, and the categories of  $SL_1$  and  $SL_2$  are the same set.

# Multi { Level, Category } Security (MLS / MCS)

and in SELinux terminology:

$SL_n \hat{=} \text{Security Level } n$

*incomp*: (*incomparable or noncomparable*)  $SL_1 \text{ incomp } SL_2$  if the categories of  $SL_1$  and  $SL_2$  cannot be compared (that is, neither is a subset of the other)

$\{ \text{STUDENTS} \} \cap \{ \text{PROFESSORS} \} = \{ \emptyset \}$

# Multi { Level, Category } Security (MLS / MCS)

and in SELinux terminology:

SL  $\hat{=}$  Security Level

This means if you want to:

- ▶ read an object your current SL must dominate the objects SL.
- ▶ write an object your current SL must be dominated by the objects SL.
- ▶ read and write an object your current SL must equals the objects SL.

# Multi { Level, Category } Security (MLS / MCS)

**MLS/BLP is not an integrity model,  
it's a confidentiality model!**

# Putting all together

Remember the statement at the beginning:

```
staff_u:staff_r:staff_screen_t:s0
```

# Putting all together

Remember the statement at the beginning:

```
staff_u:staff_r:staff_screen_t:s0
```

staff\_u: is an identifier to associate the Linux user with the SELinux user

# Putting all together

Remember the statement at the beginning:

```
staff_u:staff_r:staff_screen_t:s0
```

staff\_r: specifies the role → RBAC

# Putting all together

Remember the statement at the beginning:

```
staff_u:staff_r:staff_screen_t:s0
```

staff\_screen\_t: specifies the type → Type Enforcement

# Putting all together

Remember the statement at the beginning:

```
staff_u:staff_r:staff_screen_t:s0
```

s0: specifies the security level → MLS

# Chapter 3

## **Modular Policy**

# Targeted

- ▶ Everything is allowed by default.
- ▶ Default type: `unconfined_t`
  - Under targeted `unconfined_t` has full access.
- ▶ Only few daemons and commands are secured

# Strict

- ▶ Everything is denied by default unless a special security context is defined.
- ▶ Available modules  $\approx$  210 (refpolicy version 2007-04-17)
- ▶ Some important:
  - apache
  - samba
  - ISC bind, dhcpd
  - spamd
  - postfix
  - kerberos
  - postgresql
  - ...

# Booleans

- ▶ Easy adjustment of policy
  - No policy writing is needed

```
$ getsebool -a
```

```
...
```

```
httpd_enable_cgi --> off
```

```
...
```

```
user_ping --> off
```

```
...
```

# Standard SELinux users / roles / etc.

- ▶ Standard users:
  - sysadm
  - staff
  - user
- ▶ On systems with MLS enabled
  - *standard users*
  - secadm
  - auditadm

# Standard SELinux users / roles / etc.

Every SELinux user allocates a bunch of types and a role, e.g. the *staff*:

- ▶ Role: `staff_r`
- ▶ Types:
  - `staff_home_t` (files under `$HOME`)
  - `staff_home_dir_t` (the `$HOME` directory)

# Chapter 4

## Preview

# Preview

1. Access revocation
2. Trusted X
3. PostgreSQL
4. NFS / Samba
5. Policy Server
6. Benchmark
7. Rumours
8. Some figures for the end...

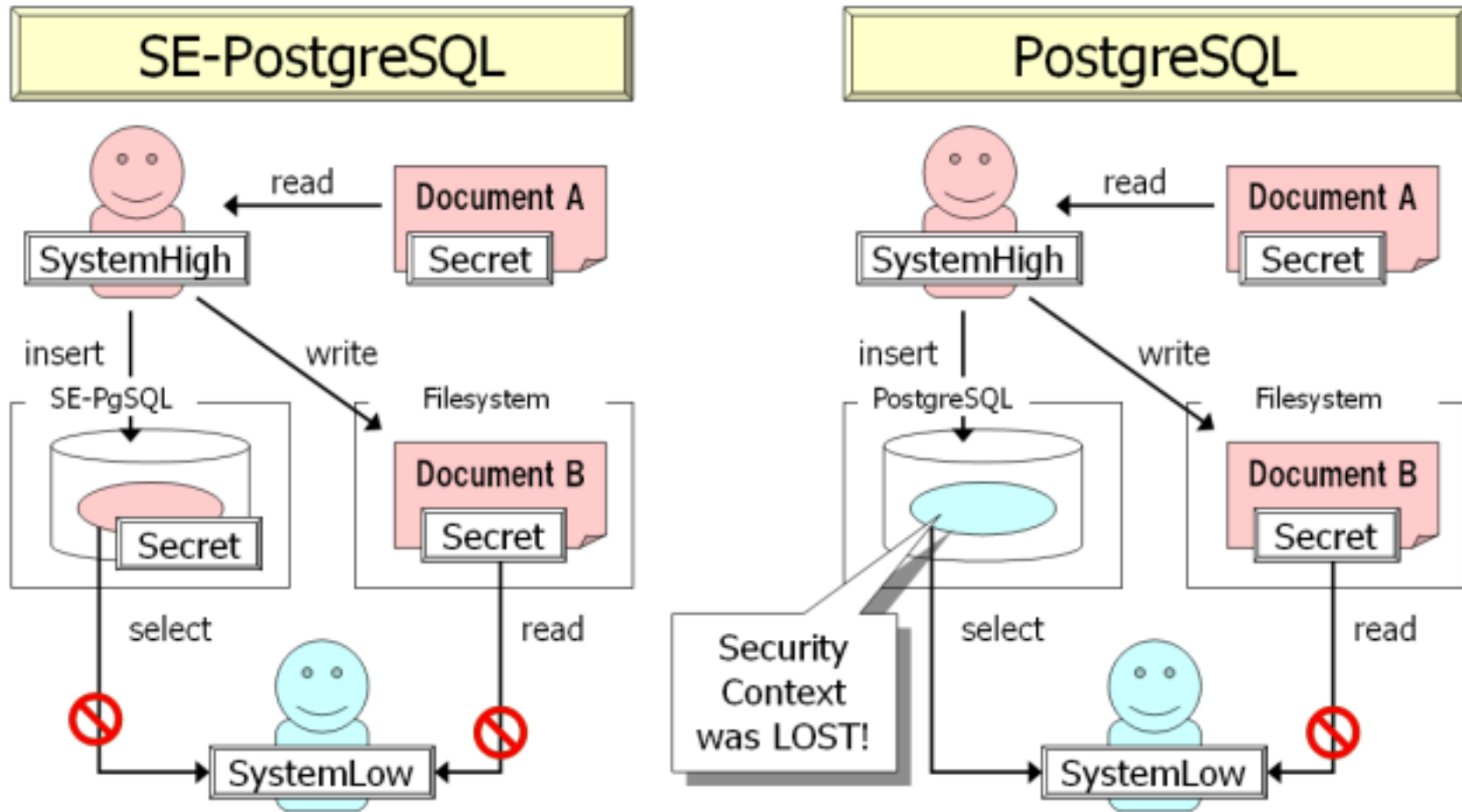
# Preview: Access revocation

- ▶ Available for most IPC resources
- ▶ Standard Linux proofs once
- ▶ SELinux tries to proof on access

# Preview: Trusted X

- ▶ Limit access to keyboard
- ▶ Limit access for screenshots
- ▶ Common: "Every" window in his own context
- ▶ ...

# Preview: SE-PostgreSQL



<http://www.kaigai.gr.jp/index.php?sepgsql>

## Preview: NFS / Samba

- ▶ At the moment TE is not enforced if you leave *local*
- ▶ In comparison *Trusted Solaris* has extensions to support NFS with MAC.
- ▶ SELinux integration in NFS, Samba etc. in the near future? *Yes , No?*
  - SENFS<sup>2</sup>

---

<sup>2</sup><http://namei.org/selinux/nfs/senfs-requirements-draft-05.txt>

# Preview: Policy Server

- ▶ all or nothing permission to load/change the policy
  - devide policy into parts and manage by different individuals
- ▶ better integration into applications e.g. SE-PostgreSQL
- ▶ policy management infrastructure to handle policy local and networked

see <http://sepolicy-server.sourceforge.net/> for more details

# Preview: Rumours

- ▶ Java, Solaris with Type Enforcement<sup>3</sup>
- ▶ Guest policy<sup>4</sup>

---

<sup>3</sup>[http://www.gcn.com/print/26\\_12/44365-1.html](http://www.gcn.com/print/26_12/44365-1.html)

<sup>4</sup><http://danwalsh.livejournal.com/10461.html>

## Preview: Benchmark

Microbenchmark	Base	SELinux	Overhead
file copy 4KB	49.5	48.6	2%
file copy 1KB	40.4	38.6	5%
file copy 256KB	23.0	21.0	10%
pipe	6.17	7.17	16%
pipe switching	12.7	15.0	18%
process creation	48.5	494	2%
execl	2480	2610	5%
shell scripts (8)	659	684	4%

Measurements are in microseconds

Figures from <http://www.nsa.gov/selinux/papers/freenix01.pdf>

## Preview: Benchmark 2

Microbenchmark	Base	SELinux	Overhead
...	...	...	...
stat	8.06	10.3	28%
open/close	11.0	14.0	27%
0KB create	22.0	26.0	18%
0KB delete	1.72	1.90	10%
...	...	...	...

Measurements are in microseconds

Figures from <http://www.nsa.gov/selinux/papers/freenix01.pdf>

## Preview: Some figures for the end...

- ▶ Policy rules (source, non-MLS): 95.409
- ▶ Policy rules (binary, non-MLS): 166.581
- ▶ Policy rules (binary, MLS): 213.551

Figures via seinfo<sup>5</sup> and reference policy (20061212)

---

<sup>5</sup><http://oss.tresys.com/projects/setools>  
Stefan Schulze Frielinghaus

# Chapter 5

## Appendix

# Literature

- ▶ F. Mayer, K. MacMillan, D. Caplan: **SELinux by Example**, Prentice Hall International, 2006
- ▶ M. Bishop: **Computer Security. Art and Science**, Addison-Wesley, 2002
- ▶ D. Bell, L. LaPadula: **Secure Computer Systems: Unified Exposition and Multics Interpretation**, 1975
- ▶ <http://fedora.redhat.com/docs/selinux-faq/>
- ▶ <http://www.selinuxnews.org>
- ▶ <http://selinux-symposium.org>
- ▶ <http://www.redhat.com/magazine/017mar06/features/riskreport/>
- ▶ <http://securityblog.org/brindle/2006/08/24/the-future-of-selinux-or-how-we-are-going-to-take-over-the-world/>

# Literature

## Useful Blogs

- ▶ <http://securityblog.org/brindle/>
- ▶ <http://danwalsh.livejournal.com/>
- ▶ <http://james-morris.livejournal.com/>
- ▶ <http://beyondabstraction.net/>

# Questions



Feel free to contact me:

[stefan@sf-net.com](mailto:stefan@sf-net.com)

<http://www.seekline.net>