CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY

# ASSIGNMENT OF BACHELOR'S THESIS

| | |
|---|---|
| **Title:** | Performance Analysis of Linux Operating System with Mandatory Access Control Kernel Modules |
| **Student:** | Jakub Rumančík |
| **Supervisor:** | Ing. Jan Žďárek, Ph.D. |
| **Study Programme:** | Informatics |
| **Study Branch:** | Information Technology |
| **Department:** | Department of Computer Systems |
| **Validity:** | until the end of winter semester 2016/17 |

## Instructions

Get acquainted with SELinux technology, its architecture and setup in CentOS (Red Hat Enterprise Linux).
Work up tools and techniques of software and operating system performance analysis.
Propose measurements and their evaluation for typical setup and usage variants of a system, e.g., web server, file server, workstation.
Select and prepare benchmarking environment and draw up performance optimizations of the system with SELinux for each variant.
Benchmark all variants in three setups: SELinux off, on and on with previously proposed optimizations.
Evaluate the results and discuss conceivable improvements.

## References

1. Frank Mayer, Karl MacMillan, David Caplan. SELinux by Example: Using Security Enhanced Linux. 2006
2. Abraham Silberschatz, Peter B. Galvin, Greg Gagne. Operating System Concepts, 8th Edition. 2008
3. the U.S. National Security Agency. Security-Enhanced Linux, URL: http://www.nsa.gov/research/selinux/
4. McAllister, Radvan, Walsh, Grift, Paris, Morris. Security-Enhanced Linux. URL: http://docs.fedoraproject.org/en-US/Fedora/13/html/Security-Enhanced_Linux/
5. Daniel Walsh, Dan Walsh's Blog, URL: http://danwalsh.livejournal.com/

L.S.

| | |
|---|---|
| Ing. Tomáš Zahradnický, Ph.D. | prof.Ing. Pavel Tvrdík, CSc. |
| Head of the department | Dean |

Prague February 16, 2015

Czech Technical University in Prague
Faculty of Information Technology
Department of Computer Systems

Bachelor's thesis

# Performance Analysis of Linux Operating System with Mandatory Access Control Kernel Modules

***Jakub Rumančík***

Supervisor: Ing. Jan Žďárek, Ph.D.

12th May 2015

# Acknowledgements

Foremost, I would like to thank to the supervisor of this thesis Ing. Jan Žďárek Ph.D., for the guidance and valuable advice. I would also like to thank my family and friends for the support during my study.

# Declaration

## Citation of this thesis

Rumančík, Jakub. *Performance Analysis of Linux Operating System with Mandatory Access Control Kernel Modules.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2015.

# Abstrakt

Táto práca sa venuje analýze výkonnosti operačného systému Linux, ktorý obsahuje rozšírenie jadra o modul mandatórneho riadenia prístupu. Zoznamuje s technológiou SELinux a jej implementáciou v operačnom systéme CentOS. Navrhuje scenáre nasadenia technológie SELinux v rôznych prostrediach ako sú webserver, fileserver a pracovná stanica. V práci sú ďalej navrhnuté optimalizácie technológie SELinux na zlepšenie výkonnosti. V navrhnutom testovacom prostredí je otestovaná výkonnosť a vyhodnotené výsledky sa nachádzajú v závere práce.

**Klíčová slova** počítačová bezpečnosť, mandatórne riadenie prístupu, SELinux, CentOS, výkonnosť, analýza výkonnosti, benchmark

# Abstract

In this thesis we analyse performance of Linux operating system with mandatory access control modules in the kernel. It acquaints with the SELinux technology and its implementation in CentOS operating system. It proposes possible scenarios for using SELinux like webserver, fileserver or a workstation. Optimizations are drawn up for increasing performance of system that is using SELinux. Performance is tested in previously designed testing environment and results are evaluated.

**Keywords** computer security, mandatory access control, SELinux, CentOS, performance, performance analysis, benchmark

# Contents

# List of Figures

# Introduction

Security is one of the vital parts of performance of the operating system. There are various approaches of enforcing this security. One of the implementation of mandatory access control approach, SELinux, is examined in this thesis. Being default security module in the Linux distributions CentOS, Fedora or Red Hat Enterprise Linux, SELinux has proven it is a security feature that has found its way to widely used operating systems.

Goal of this work is to test SELinux in typical system setup like webserver, fileserver or a workstation. We prepare a testing environment and analyze the performance of these scenarios, determine the impact of SELinux and find possible bottlenecks of the systems.

This paper is divided into three chapters. In state-of-the-art we discuss access control in general, SELinux architecture and setup in CentOS as well as performance analysis methods. In second chapter testing environment and setup of the tested systems is proposed. We choose the tools for measuring the performance and draw up possible optimizations of tested setups. In the last chapter testing itself is performed and impact on performance is analyzed.

Chapter 1

# State-of-the-art

## 1.1 Importance of security in operating systems

The very reason for developing security of computer systems is the inevitability of software failure. According to [2] it is a serious issue:

“Public awareness of the need for security in computing systems is growing as critical services are becoming increasingly dependent on interconnected computing systems.”

The need to increase security comes from many sides. Security is the primary concern of businesses which want to use the Internet for commerce and maintaining business relationships[3].

Operating systems play crucial role in supporting the security at higher levels. This is well understood for at least twenty five years[4][5].

“The necessity of operating system security to overall system security is undeniable; the underlying operating system is responsible for protecting application-space mechanisms against tampering, bypassing, and spoofing attacks. If it fails to meet this responsibility, system-wide vulnerabilities will result.”[2]

## 1.2 Reference monitor concept

Reference monitor concept is a concept that is widely respected as the theoretical concept of access control. As the picture shows, it consists of several separated entities, that each represent particular resource in the operating system.

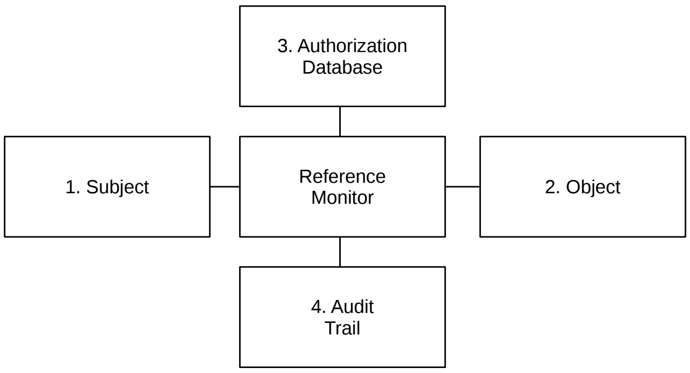

Figure 1.1: Reference Monitor Concept

1. **Subject**
   Entity that represents processes and users. Subject is actively seeking access to objects and data stored within.
2. **Object**
   Entity that represents files and directories. Objects contains data we are trying to protect.
3. **Authorization database**
   Set of rules that define access permissions between subjects and objects.
4. **Audit trail**
   Entity that represent system logs. Reports all relevant events and stores them according to the configuration.

In the middle of all this, there is reference monitor. Entity that enforces access control. For it to be effective it needs to meet some conditions. According to [1] the fundamental design goals of an implementation of the reference monitor are that it be:

- Tamper-proof (cannot be maliciously changed or modified)
- Nonbypassable (subjects cannot avoid the access control decisions)
- Verifiable (it is correct and implementation of the security policy can be demonstrated)

Today, every modern operating system implements some sort of reference monitor.

## 1.3 DAC vs. MAC

DAC (Discretionary Access Control) is a form of access control that is most common today. Basic principle is that there is an owner for every object, that decides what access permissions should that object have. Problem with this principle is, that mostly it is not the user that decides. To enforce access control with DAC, user uses software, that he did not create, therefore does not know for sure whether it is working properly. Flawless software can then easily penetrate given systems' security.

MAC (Mandatory Access Control) is a security concept that tries to address and solve the issue with DAC. Access to the object is not decided by owner, but it is tested against the set of predefined rules. These rules, we can also call it policy, are aimed to permit access only to the objects that subject needs to fulfill its task. We will demonstrate this principle later on, when we discuss SELinux implementation of MAC.

## 1.4 SELinux

SELinux is a kernel security module that implements MAC on Linux. It is implemented via Linux Security Module Framework and uses Type Enforcement method to implement MAC concept. Furthermore, it gives administrators various tools and other opportunities, to work with security policy and adapt SELinux to the need of particular system.

### 1.4.1 Architecture

#### 1.4.1.1 LSM Framework

LSM (Linux Security Modules) is a framework that allows integration of various security modules into Linux kernel. Following the reference monitor concept it provides a set of hooks that are placed after the standard Linux DAC permissions but before the access to the actual resource. SELinux is therefore loaded as an LSM module to the kernel and is consulted and has to allow every access attempt.

As the picture shows, the consequences of LSM Hook being placed last before accessing the inode of the resource, are that when DAC check does not allow access, SELinux is not even consulted for access decision. This should not hurt realization of SELinux access control, as SELinux policies tend to be more restrictive than DAC checks.

Important fact that we will use in performance analysis is that all the security checks are done in kernel space.

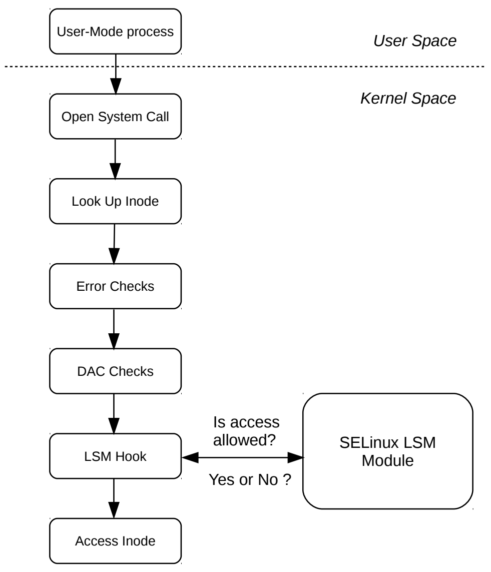

Figure 1.2: Position of LSM Hook [1]

#### 1.4.1.2 SELinux LSM module

As mentioned before, SELinux is loaded into kernel as a LSM module. According to [1], the SELinux kernel architecture reflects the Flask architecture, which was designed for microkernel environment. The Flask architecture has three primary components: security server, object managers, and the access vector cache.

Security server is in charge of decision making based on the policy. Object managers enforce decisions of the security server on the resources they manage. Access vector cache is a memory of previous decisions of the security server. The aim is performance improvement. The question, whether it is sufficient, will be attempted to answer further on.

Userspace object managers are not part of the LSM Module. They are separate from the kernel, although they work in similar way. According to [1], the main difference is absence of LSM hooks, which is a kernel concept, in userspace object managers. Userspace object managers have internal in-

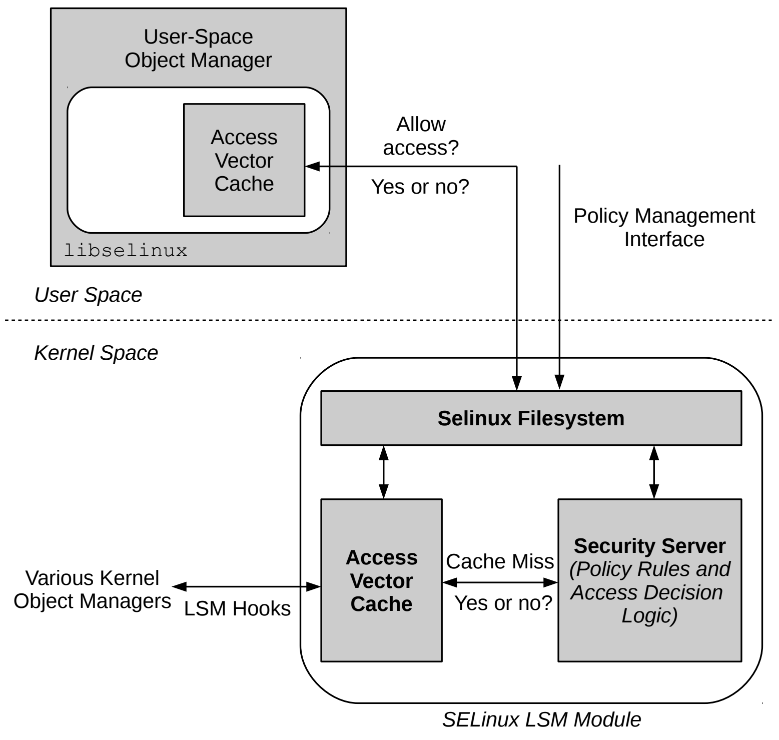

Figure 1.3: SELinux Architecture [1]

terfaces with its access vector cache inside `libselinux` instead. This cache handles misses and queries kernel for the userspace object manager.

### 1.4.2 Type Enforcement

Type enforcement is the very implementation of access control policy in SELinux. It uses certain type assigned to all the resources, both subject and objects and defines rules of interaction between them.

#### 1.4.2.1 Security context

Security context is the piece of information that SELinux adds to resources in addition to DAC permissions, id of processes and so on. It is usually displayed as a triplet in the following format:

```
user_u:role_r:type_t
```

Three parts of the string are all defined in security policy. Every valid security context has these three parts: valid user, valid role and a valid type defined in the policy.

#### 1.4.2.2 Rules

SELinux requires all of the access to be granted explicitly. Therefore, we need rules that will define this access permission. Most used SELinux rule is `allow` rule. According to [1], `allow` rule has four elements:

- **Source type**
  Type of a process attempting access
- **Target type**
  Type of an object being accessed by the process
- **Object class**
  Class of an object that the specified access is permitted to
- **Permissions**
  Kind of access that the source type is allowed to the target type for previously defined object classes.

We will demonstrate how does `allow` rule work on simple example from [1]. User `joe` wants to change his password. After login he is granted `user_t` type. When he tries to invoke `passwd` program which has type `passwd_exec_t`, the rule

```
allow user_t passwd_exec_t : file {getattr execute};
```

grants him execute permission on the file. After running the program, `/usr/bin/passwd` file tries to start a new process. Next rule

```
allow passwd_t passwd_exec_t : file entrypoint;
```

grants the passwd program to create process by executing the file. Assuming that there is another rule

```
allow passwd_t shadow_t : file write;
```

we need to change our security context accordingly where the last rule,

```
allow user_t passwd_t : process transition;
```

grants permission for `user_t` to change its security type to `passwd_t`, using as

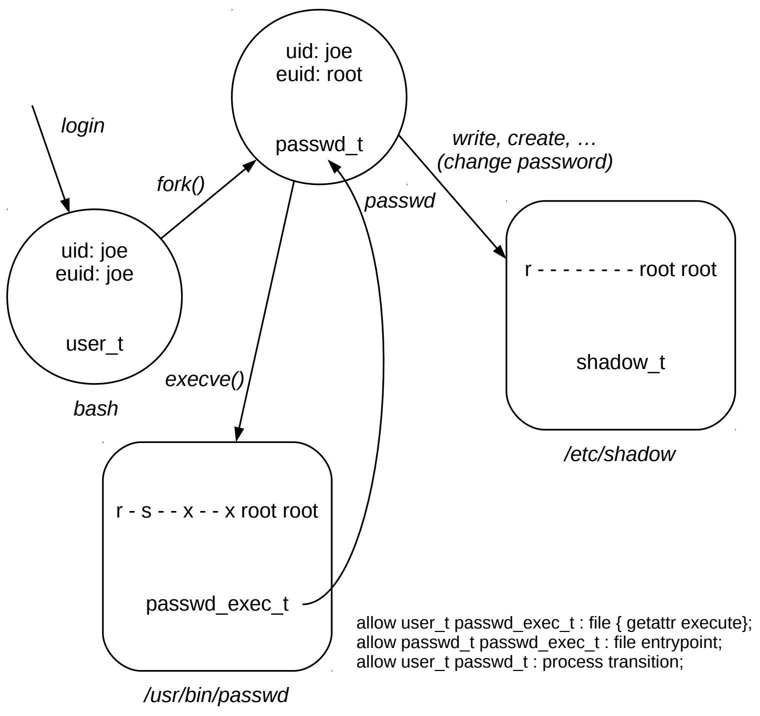
uid: joe
euid: root
passwd_t
login
fork()
write, create, ...
(change password)
passwd
uid: joe
euid: joe
user_t
bash
execve()
r - - - - - - - - root root
shadow_t
/etc/shadow
r - s - - x - - x root root
passwd_exec_t
/usr/bin/passwd
allow user_t passwd_exec_t : file { getattr execute};
allow passwd_t passwd_exec_t : file entrypoint;
allow user_t passwd_t : process transition;

Figure 1.4: Example Of Type Enforcement [1]

an entrypoint `passwd_exec_t`. Now `passwd` process is running with `passwd_t` type instead of `user_t` and is able to write to `/etc/shadow` file.

`allow` rule is one of the few rules that are needed to build a sufficient security policy. However, exploring all the rules and syntax of the policy language is not the goal of this work, and one can find more information on this matter on `selinuxproject.org`.

### 1.4.3 Booleans

Another important part for system administration of Linux system with SELinux are SELinux booleans. Most of SELinux aware applications have defined several booleans that can alter the policy decisions without the need to change the policy itself. Best way of administering these booleans is using SELinux utilities `getsebool` and `setsebool`. You can list all available booleans for a certain service by using following command:

```
getsebool -a | grep <service-name>
```

We will use these tools further on when configuring test environment.

### 1.4.4 SELinux setup in CentOS - targeted policy

Default SELinux policy in CentOS is targeted policy. The name for this policy is derived from 'targeting' critical processes in the system that it protects, while others, not so crucial for the system, are not protected. In this section we also discuss applications that we are going to use further on in testing.

#### 1.4.4.1 Confined and unconfined processes

Processes in CentOS, and Red Hat Enterprise Linux are divided into two main domains: *confined* and *unconfined*. The main difference is in the approach of the policy. *Confined* services are protected fully, while *unconfined* processes, although still protected by SELinux, have rules that allow almost all access. Therefore security of *unconfined* processes is mainly dependent on DAC.

According to [6], almost every service that listens on a network, such as `sshd` or `httpd`, is *confined* in CentOS. Also, most processes that run as the root user and perform tasks for users, such as the previously mentioned `passwd` utility, are *confined*. *Confined* process runs under its own type, such as the `httpd` process is using `httpd_t` domain.

If a *confined* process is compromised, attacker's access to resources and the damage they can do is limited.

According to [6], *unconfined* processes run in *unconfined* domains, for example, system processes started by `init` run in the *unconfined* `initrc_t` domain, *unconfined* kernel processes run in the `kernel_t` domain, and *unconfined* Linux users run in the `unconfined_t` domain.

#### 1.4.4.2 Apache HTTP server [7] and SELinux

Apache is by default running as a *confined* service under `httpd_t` type. Another kind of files labeled with `httpd_sys_content_t` type are only readable by `httpd`. According to [6] "booleans must be enabled to allow certain behavior, such as allowing scripts network access, allowing `httpd` access to NFS and CIFS volumes, and `httpd` being allowed to execute Common Gateway Interface (CGI) scripts." Also listening on a port other than 80, 443, 488, 8008, 8009, or 8443 must be allowed by command `semanage port`.

#### 1.4.4.3 FTP and SELinux

The FTP daemon that runs *confined* is `vsftpd` [8] by default. According to [6], when an authenticated user logs in via FTP, they cannot read from or write to files in their home directories: SELinux prevents `vsftpd` from accessing user home directories by default. Also, by default, `vsftpd` does not have access to NFS or CIFS volumes, and anonymous users do not have write access, even if such write access is configured in the `/etc/vsftpd/vsftpd.conf` file. These permissions can be managed with enabling particular booleans.

#### 1.4.4.4 MariaDB [9] and SELinux

Default MySQL database has changed to MariaDB in latest Fedora [10] and CentOS releases. MariaDB is drop-in replacement for MySQL and is a community developed fork of MySQL database project, independent on Oracle. It is a SELinux-aware application.

MariaDB runs as *confined* in SELinux by default. Features configurable by booleans are `httpd` network connections and `ftp` daemons access to the database.

## 1.5 Software and system performance analysis

### 1.5.1 Stress testing

Stress testing according to [11], "is a generic term used to describe the process of putting a system through exertion or stress. . . Stress testing is typically used to benchmark a systems performance to determine a systems upper performance limits."

### 1.5.2 Load testing

Load testing will be the main part of our performance analysis. According to [12], "load testing is the process of executing a concurrent user load and/or a system load onto a system, at incremented concurrency ramp-up rates, to determine its stability, performance and integrity. During load testing system components are monitored and correlated with transaction response times. At a point where a system becomes unstable due to erratic or dramatically extended response times the system will have reached its benchmark. At this point an analysis should take place to identify the bottlenecks and the tuning required to resolve them."

Chapter 2

# Analysis and design

## 2.1 What to look for?

Determining influence of a certain piece of software on the system is a task of determining what system resources are going to be influenced by this piece of software, and determining their performance.

In our case (SELinux) we have a kernel module that provides additional access control. Therefore we will look for changes in load of the CPU, namely kernel space CPU load.

## 2.2 Monitoring system performance

During tests we will use a monitoring system to take snapshots of resources used, mainly CPU, memory and disk. These data will be used to determine the impact of the SELinux on the performance.

As a monitoring system we will use `nmon` [13]. `nmon` is easy-to-use performance monitor. Simply running the following command:

```
nmon -fT -s 5 -c 2880 -m $HOME/nmon-data
```

will start a process that will be monitoring the system while we will generate traffic to get some useful data. As for explanation of the command:

| | |
|---|---|
| `-f` | - starts `nmon` in data-collect mode (instead of interactive) |
| `-T` | - include top processes in the output |
| `-s ⟨seconds⟩` | - # of seconds between snapshots of the system |
| `-c ⟨number⟩` | - # of snapshots before `nmon` ends |
| `-m ⟨directory⟩` | - directory that `nmon` will save file to |

The `-T` option is set because it might be interesting to know whether some processes resource consumption rises more than others, and therefore are more problematic under SELinux enabled system.

The `-c` option is not so important, as `nmon` can be killed using following command, mentioned in `nmon` documentation,

```
kill -USR2 <nmon-pid>
```

after finishing the traffic generating script and data-collecting (here, `nmon` is set to run for four hours).

For visualizing the result we will use `nmon-visualizer` [14].

## 2.3 General design

### 2.3.1 LVM

LVM should be present in our server installation. As [15] says, LVM includes all of the support for handling read/write operations on physical volumes (hard disks, RAID-Systems, magneto optical, etc., multiple devices (MD), see `mdadd`(8) or even loop devices, see `losetup`(8)po), creating volume groups (kind of virtual disks) from one or more physical volumes and creating one or more logical volumes (kind of logical partitions) in volume groups.

LVM is included in the default installation of CentOS.

### 2.3.2 Filesystem

For choice of filesystem we chose three different types that will be test.

- **ext2**
  Used as a standard for benchmarking. Unlike other two, it does not have journal.
- **ext4**
  Most popular filesystem on Linux today.
- **xfs**
  Red Hat recommended and default filesystem for RHEL and CentOS.

### 2.3.3 OS

For OS CentOS 7 [16] will be used. For server installation 'Minimal install' software selection will be used and for workstation 'GNOME Desktop' software selection with additional 'Development tools' package will be used.

## 2.4 Webserver

For a first scenario for our performance analysis we will configure and test a webserver. This webserver should be capable of serving requests for static `.html` pages, as well as dynamic PHP content with some possible communication with database.

### 2.4.1 Configuration

In configuring our webserver widely known and used open source software will be used. It is based on so called LAMP Stack. LAMP is an abbreviation of Linux, Apache, MySQL, PHP. It is an open source Web development platform that uses Linux as the operating system, Apache as the Web server, MySQL as the relational database management system and PHP as the object-oriented scripting language.

For obvious reasons we will use Linux distribution that natively supports SELinux - CentOS, in minimal installation image, that installs only necessary packages for functioning server and no GUI.

As a webserver we will use Apache HTTP webserver, which is widely used open source webserver. To install this webserver we need package httpd.

Default MySQL database has changed to MariaDB in latest Fedora and CentOS releases. It is a drop-in replacement for MySQL and is a community developed fork of MySQL database project, independent on Oracle.

To install PHP we use `php` package. Also, we need additional `php-mysql` package to provide functionality for `php` to use the database.

These packages are essentially all we need for a functioning webserver. For the webserver to work properly we need to make some configuration changes in some of the packages that are discussed later in Realisation chapter.

### 2.4.2 Testing

Our goal is to test every part of the set up configuration. Using `ab` [17] tool, we will stress test static content, dynamic content and dynamic content connecting to the database. After this, using `siege` [18] tool, we will load test webserver by accessing all of these pages randomly and adding some false pages that are not present on the server, expecting ‘404 Page not found’ response.

### 2.4.3 Performance

Webserver in general serves, by far, the most amount of requests (access to files or resources) out of all the tested scenarios. It is generating new files for every request asking for dynamic content, and that is why we expect SELinux to have significant impact on performance.

## 2.5 Fileserver

Second scenario for testing is a fileserver. The fileserver should be capable of serving FTP requests even on anonymous basis, and provide secure file transfer for authenticated users.

### 2.5.1 Configuration

For a functioning fileserver we need two basic services. Plain FTP used mainly for anonymous downloads and uploads and secure file transfer.

For plain FTP we will use `vsftpd` package. For SFTP we will use SSH server. SSH server offers a lot of funcionalities for administrators, but also sftp service for secure file transfer.

### 2.5.2 Testing

Fileserver will be tested for non-crypted FTP transfers and SFTP transfers. Using `wget` [19] and `wput` [20] tools we will test download and upload with plain FTP. Using `scp` [21] tool we will test SFTP file transfers.

### 2.5.3 Performance

Fileserver does not serve as many requests as webserver does. It is used for transferring mainly larger files, and so access control is used much less than it is on the webserver. Due to this fact, we expect less performance impact of SELinux, and it may appear for this impact to be insignificant.

## 2.6 Workstation

The last scenario for testing is a workstation. We expect workstation to be used for different tasks than previous two scenarios. In our scenario we expect workstation to be used for software development and that will determine tasks to be tested.

### 2.6.1 Configuration

Workstation should be configured using the standard default 'GNOME Desktop' installation of CentOS. For additional software we choose Development Tools. Nothing else should be installed further, as this image has all the needed packages for functioning workstation, like GUI, file manager, network support, web browser, basic media support etc.

### 2.6.2 Testing

To test performance workstation benchmarks will be used for generating traffic. Selected benchmarks will test workstations' performance in compiling, compressing and aging the filesystem by simulating some of the disk IO common operations.

### 2.6.3 Performance

Out of the three proposed scenarios workstation is the least loaded with requests. We expect for SELinux to have insignificant impact on performance. Workstation is usually used by one user at a time, who is incapable of generating enough access control requests for it to have any effect.

## 2.7 Testing environment

Environment for testing the performance of server using SELinux consists of two computers directly connected via crossover cable in a private network. One of the machines is the tested server itself, while other is a client where the traffic is generated and all the testing is led from.

### 2.7.1 Server

Here is list of server hardware and software parameters. Note that some parameters of the server might change due to the nature of the test. Server is using LVM and filesystems vary through ext2, ext4 and xfs. Also, the mode of SELinux will change through tests.

#### 2.7.1.1 Hardware

| | |
|---|---|
| Processor: | Intel Core 2 6300 @ 1.86GHz (2 Cores) |
| Motherboard: | LENOVO |
| Chipset: | Intel 82Q963/Q965 + ICH8/R |
| Memory: | 1024MB |
| Disk: | 80GB Western Digital WD800JD-08MS |
| Graphics: | Intel 82Q963/Q965 IGP |
| Network: | Broadcom NetXtreme BCM5755 Gigabit PCI |

#### 2.7.1.2 OS and additional packages

| | |
|---|---|
| Linux kernel: | 3.10.0-123.20.1.el7.x86_64 |
| OS distribution: | CentOS 7 Minimal installation |
| Additional packages: | httpd |
| | php |
| | mariadb |
| | vsftpd |

### 2.7.2 Network

Testing through network can cause defective results, because network speed tends to be a bottleneck of servers' performance. The goal for the network is to be as small and as fast as possible, but trying to preserve equal conditions for all the tests.

Loopback interface is unsuitable to generate traffic during tests. Loopback generates traffic on the machine itself, therefore is not slowed by network architecture, but is influenced by the SELinux mode causing conditions to be different when testing SELinux on and off.

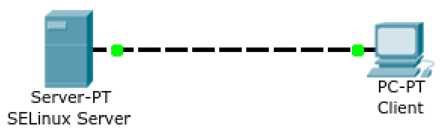

Figure 2.1: Simple Network

Traffic will be generated using a small private network consisting of two directly connected computers via Gigabit Ethernet, one of those being the tested server, other acts as gateway to WAN (this is being disconnected during testing). All requests (http, ftp) are coming from the gateway machine to minimize the possibility of network being a bottleneck of the testing.

### 2.7.3 Client

Here is list of client hardware and software parameters. Unlike server, client needs to be configured exactly the same way for all the test to ensure equal conditions during tests.

#### 2.7.3.1 Hardware

| | |
|---|---|
| Processor: | Intel 2140 @ 1.60GHz (2 Cores) |
| Motherboard: | ASUS P5L8L |
| Chipset: | Intel 82945G/GZ/P/PL + ICH7 |
| Memory: | 2 x 1024 MB SDRAM-667MHz |
| Disk: | 160GB Seagate ST3160813AS |
| Graphics: | Intel 82945G/GZ IGP |
| Network: | Realtek RTL8111/8168/8411 |

#### 2.7.3.2 OS and additional packages

| | |
|---|---|
| Linux kernel: | 3.10.0-123.20.1.el7.x86_64 |
| OS distribution: | CentOS 7 Minimal installation |
| Additional packages: | httpd-utils |
| | siege |
| | wget |
| | wput |

## 2.8 Data gathering scripts

These scripts are run from client machine and their function is to perform the whole testing. Main structure is the same in all of the scripts.

- Turn on `nmon` on the server (using `ssh`)
- Hit the server with requests
- Turn off `nmon` on the server
- Save the results
- Parse the data for further use

### 2.8.1 Web stress test

This script has four modes which it can stress test the webserver. First is *html*, requesting static `.html` website. Second is *php*, which is sending requests for dynamic `.php` page. *mysql* mode requests dynamic `.php` page that works with the database. Finally *false* mode requests page that is not on the server, thus expecting failed request. Other configurable parts of the script are concurrency of the requests and number of concurrent request waves to hit the webserver.

From this test we get time taken for tests, requests per second and transfer rate for each concurrency level to compare. Also, we get time in which certain amount of requests is served.

### 2.8.2 Web random test

This script tests web server under given concurrency of requests for a given time, hitting webserver with requests for pages listed in `/etc/siege/urls.txt` file, choosing randomly which page to request. File contains 4 types of records

- Static `.html` pages
- Dynamic content in form of `.php` pages
- `.php` pages communicating with the database
- False pages not present on the server

This test also gives us transaction performed, transactions per second and throughput values.

### 2.8.3 FTP and SFTP tests

These script starts given number of downloads and uploads using `wget`/`wput` programs using FTP, or `scp` for secure file transfer. Besides `nmon` file we will use download/upload time of files and throughput to compare.

### 2.8.4 Workstation benchmarks

We expect workstation to be used for software development and we need to choose benchmarks accordingly to this expectation. For testing we will use `phoronix-test-suite` [22] benchmarking utility. There will be two benchmarks we will be performing:

- Timed code compilation
  - Build apache
  - Build mplayer
  - Build php
  - Build linux kernel
- Compile bench

The ‘Timed code compilation’ test suite consists of four separate tests that are compiling various software and measuring time needed. Compile bench, as [23] says, “tries to age a filesystem by simulating some of the disk IO common in creating, compiling, patching, stating and reading kernel trees. It indirectly measures how well filesystems can maintain directory locality as the disk fills up and directories age.”

## 2.9 Optimizations

### 2.9.1 Webserver and Fileserver

Important files for both webserver and fileserver are stored within a particular directory in the system. For webserver it is `/var/www/html` and for fileserver `/var/ftp/pub`. Repeated access to these directories, that occurs during testing, may be well cached in AVC (Access Vector Cache).

Optimization we propose is to disable SELinux control over particular files in the mentioned directories and use SELinux only to control access to the directory itself. To do this we must change security context type to *unconfined* in these files. Decrease in amount of SELinux access control decisions and therefore better performance are expected.

Drawback of this optimization is leaving content of the files and their security to traditional DAC without additional SELinux protection.

### 2.9.2 Workstation

Optimizing a workstation is a different task than optimizing one of the servers. There is no service that is tested exclusively, like `http` or `ftp`. Optimization would have to be system-wide and would need to work with the policy file itself.

Taking in account SELinux is expected to have little or no impact on performance in this scenario, small changes of a few services would not be observable and system-wide optimizations of the whole policy file is beyond the limits of this work. It is expected that there would be no space left for optimization, so optimizations for workstation are not proposed.

CHAPTER 3

# Realisation

## 3.1 Client

For testing servers, it is necessary to have a client that will remain the same during all the tests, and will hit servers with requests.

### 3.1.1 Configuration

For configuring a client we need to do some necessary steps.

1. Install OS
2. Configure networking
3. Download necessary packages

After installing CentOS 'GNOME Desktop' installation, configuration of network interface is needed. This interface is directly connected to the server with a static IP (`192.168.0.1/24`), and is used as a gateway to the server. We can do this using Network Manager, since we have GUI installed.

Furthermore, we must download `httpd-utils` package, `siege` package and `wput` package, that we will need for the testing. Packages `scp` and `wget` are already installed within the default OS installation.

## 3.2 Webserver

### 3.2.1 Configuration

Before running tests webserver must be configured. Setting up a webserver can be divided into several steps that are needed to be performed for a properly running webserver.

1. Install OS
2. Configure networking
3. Download necessary packages
4. Generate websites
5. Enable necessary services
6. Set up database

Now we are going to take a look at these steps closer.

#### 3.2.1.1 Install OS

For OS installation we will use standard CentOS 7 `.iso` file, downloadable from CentOS website. It provides Anaconda walkthrough installation. In software selection we choose 'Minimal install' and automatic partitioning. From this point installation is automatic.

#### 3.2.1.2 Configure networking

In minimal install software selection the networking is disabled. In order to get network running we need to make some changes to configuration file of the network interface. Automatically generated interface configuration file looks like this:

```
HWADDR=00:16:41:36:20:24
TYPE=Ethernet
BOOTPROTO=dhcp
DEFROUTE=yes
PEERDNS=yes
PEERROUTES=yes
IPV4_FAILURE_FATAL=no
IPV6INIT=yes
IPV6_AUTOCONF=yes
IPV6_DEFROUTE=yes
IPV6_PEERDNS=yes
IPV6_PEERROUTES=yes
IPV6_FAILURE_FATAL=no
NAME=enp3s0
UUID=0842275e-8a44-431b-9634-a6bd2192e964
ONBOOT=no
```

As we do not have running DHCP server we must change `BOOTPROTO` option to 'static' and also define IP address, netmask, default gateway and DNS servers. We also change `ONBOOT` option to 'yes' for interface to be available and running on boot, and set `NM_CONTROLLED` to 'no' to deny NetworkManager access to this interface. Finally, the configuration file looks like this:

```
HWADDR=00:16:41:36:20:24
TYPE=Ethernet
BOOTPROTO=static
IPADDR=192.168.0.100
NETMASK=255.255.255.0
GATEWAY=192.168.0.1
NM_CONTROLLED=no
DNS1=8.8.8.8
DNS2=8.8.4.4
DEFROUTE=yes
PEERDNS=yes
PEERROUTES=yes
IPV4_FAILURE_FATAL=no
IPV6INIT=yes
IPV6_AUTOCONF=yes
IPV6_DEFROUTE=yes
IPV6_PEERDNS=yes
IPV6_PEERROUTES=yes
IPV6_FAILURE_FATAL=no
NAME=enp3s0
UUID=0842275e-8a44-431b-9634-a6bd2192e964
ONBOOT=yes
```

#### 3.2.1.3 Download necessary packages

Now that we have access to the Internet, necessary packages for webserver can be downloaded.

- `httpd`
- `php`
- `php-mysql`
- `mariadb-server`

These packages allow us to set up LAMP stack. We also need additional packages for system administration and performance monitoring.

- `policycoreutils-python`

- http://pkgs.repoforge.org/rpmforge-release/rpmforge-release-0.5.3-1.el7.rf.x86_64.rpm
- nmon

After installing these packages we have downloaded everything we need.

#### 3.2.1.4 Generate websites

We need to generate a large amount of websites, mainly for the Internet simulation testing. We use simple bash scripts for this task. This is a script for generating .php pages.

```
#!/bin/bash
for i in ‘seq 1000‘
do
echo "<html>
<body>
<?php
echo \"<h1>This is a test PHP page $i</h1>\";
echo microtime ( ) ;
?>
</body>
</html>" > testpage${i}.php
done
```

In similar manner we generate .html pages and .php pages communicating with database. These scripts are available in directory src/scripts on the CD.

#### 3.2.1.5 Enable necessary services

When everything is ready we must enable all the services we need and also adjust firewall for our new services using following commands:

```
systemctl enable httpd
systemctl enable mariadb-server
firewall-cmd --permanent --add-service=http
```

After reboot we have a running webserver capable of serving requests during testing.

#### 3.2.1.6 Set up database

Using CLI tool mysql we set up database. After connecting to the database as root using `mysql -u root -p` we invoke following commands:

```
CREATE USER 'tester_db'@'localhost' IDENTIFIED BY 'retset';
CREATE DATABASE IF NOT EXISTS test_db;
```

We have created an empty database and a user that will be used to connect the database. Using following simple shell command

```
mysql test_db < db_dump.sql
```

we fill up the database using `db_dump.sql` script and it is ready to use for testing.

### 3.2.2 Testing

During webserver testing, mainly its ability to satisfy a number of concurrent requests is being tested. Webserver uses an amount of small files to generate dynamic content. This might cause a performance decrease when enabling SELinux in enforcing mode, because every new file created or changed has to be permitted access by SELinux in addition to standard UNIX access control. For testing a webserver performance we will perform four tests.

#### 3.2.2.1 Tests with ab

We will test three various pages with `ab` tool. Number of concurrent requests will be rising over time to get data on the various workloads server might be facing, from very little to near-stress test amount. Example script to test html static page (scripts for testing *php* and *mysql* will be performed in similar manner. Both are available in `src/scripts` directory on CD):

```
#!/bin/bash
for i in 1 2 4 8 16 32 64 128
do
  ab -q -c $i -n $((i*50000)) -e html$i.csv $HOSTNAME/testpage.html
done
```

The `-q` option suppresses output to `stderr`, `-c` sets amount of concurrent requests, `-n` number of times requests should be sent and `-e` option generates a `.csv` file with output values of `ab`.

From the `ab` output throughput and time that it took to answer to all the requests can be measured. Furthermore CPU, memory and filesystem I/O performance can be measured with `nmon`. This is because the throughput or time might not change much, but it might take up much more system resources for the same task.

#### 3.2.2.2 HTML static page

This test hits the following static html page with a number of concurrent requests.

```
<html>
<body>
<h1> This is a test HTML page </h1>
</body>
</html>
```

#### 3.2.2.3 PHP dynamic page

This test is similar to the previous one, except it hits PHP dynamically generated pages. These will have greater impact on the filesystem, as a large number of small files are created during testing. Creation and changes made to these files can take up more time on SELinux enforcing system due to the need to perform additional access control.

```
<html>
<body>
<?php
echo "<h1>This is a test PHP page</h1>";
echo microtime();
?>
</body>
</html>
```

`microtime`() function is called to prevent as much as possible cached page optimization, so server needs to generate a new page, and, therefore a new file for each request.

#### 3.2.2.4 PHP+MySQL dynamic page test

Performing the same test as before, just with a PHP dynamically generated pages and MySQL queries. It is the most complicated test and should take up the most time. We just adjust php script in the previous page a little to communicate with the database each time a request is made.

```
<html>
<body>
<h1>This a test MySQL page <h1>
<?php
// Connecting, selecting database
$link = mysql_connect('localhost', 'tester_db', 'retset')
    or die('Error : ' . mysql_error());
```

```
echo 'Connected successfully';
mysql_select_db('test_db');

// Performing SQL query
$query = 'SELECT * FROM spell_pet_auras';
$result = mysql_query($query) or die('Failed: ' . mysql_error());

// Printing results
echo "<table>\n";
while ($line = mysql_fetch_array($result, MYSQL_ASSOC)) {
    echo "\t<tr>\n";
    foreach ($line as $col_value) {
        echo "\t\t<td>$col_value</td>\n";
    }
    echo "\t</tr>\n";
}
echo "</table>\n";

// Free result
mysql_free_result($result);

// Closing connection
mysql_close($link);
?>
</body>
</html>
```

#### 3.2.2.5 Real workload simulation with Siege

This test aims to simulate real workload using Siege tool. This test will be more random than `ab` tests before, so for it to have some value we need to run it long enough to be statistically more accurate.

```
siege -c 500 -t 2H -d 2 -i
```

This siege test is hitting webserver with 500 concurrent requests (`-c` option) for 12 hours (`-t` option). Every further concurrent request has random delay (`-d` option) from 1 to 2 seconds (aim is to stagger requests in time) and requests are send to random pages (`-i` option) included in the `/etc/siege/urls.txt` file.

As mentioned earlier, due to this randomness, the test needs to be performed for a longer period of time, here for two hours. In real traffic data-collection, time needed to collect data is much longer. It is mainly because of seasonal peaks of load. In this simulation, we don't have seasonal peaks, so two hours should be enough to make a reasonable simulation.

### 3.2.3 Results

Results of the webserver are divided into four parts. First three are stress tests of a single page, which differs in content. Last is load test that is simulating real world load of the server. Complete results are available in `res/web` directory on the CD. On the graphs enforcing SELinux is in red and orange lines, while disabled SELinux is in blue and purple lines. As the tests with increasing concurrencies were run sequentially, graphs contain all the concurrencies together.

#### 3.2.3.1 Testing static HTML page

First test that was running was hitting a static page. We chose exponentially rising concurrency from 1 to 128 each concurrency level consisted of 50 000 waves of concurrent requests. Results were following:

| Time in s | Disabled | Enforcing | Difference |
|---|---|---|---|
| ext2 | 2573.02 | 2500.95 | -2.8% |
| ext4 | 2482.31 | 2515.21 | 1.3% |
| xfs | 2435.77 | 2452.38 | 0.7% |

As we can see, results of the test are ambiguous. Considering the nature of the test, this is not entirely unexpected. Test was hitting single static website, which can be well cached in SELinux's AVC. When we look at CPU statistics in figures 3.1, 3.2 and 3.3, CPU load in kernel space was higher in Enforcing mode, as expected.

On ext2 filesystem we measured 4.5% more CPU load within kernel space with enforcing SELinux, on ext4 it was 3% and on XFS 1.8%. Even when the test was ambiguous, the presence of SELinux was observable.

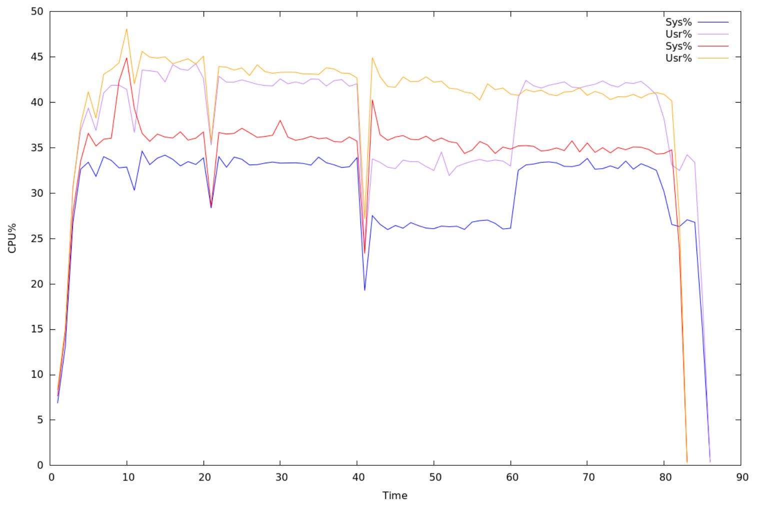

Figure 3.1: CPU load, HTML test, ext2

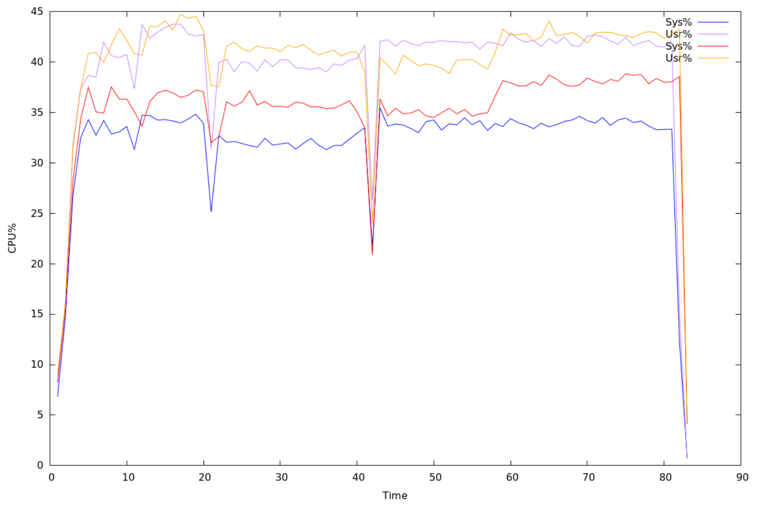

Figure 3.2: CPU load, HTML test, ext4

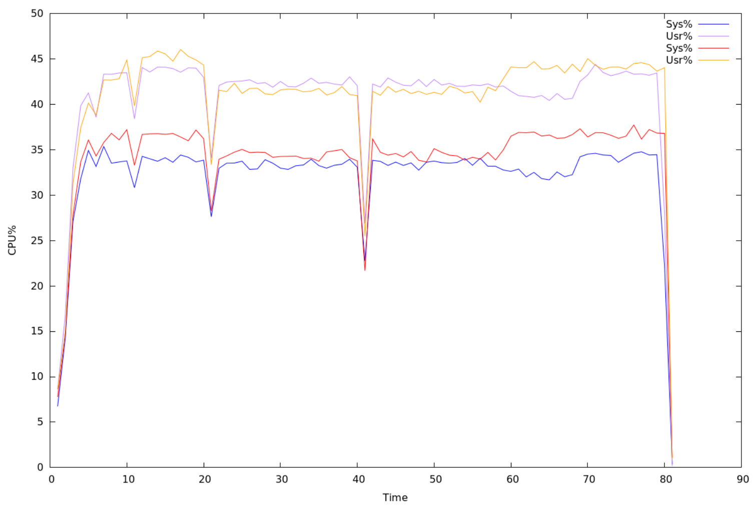

Figure 3.3: CPU load, HTML test, XFS

#### 3.2.3.2 Testing dynamic PHP page

Second test was hitting a PHP page that generated dynamic content. As well as the first test, there was exponentially rising concurrency from 1 to 128. Each concurrency level consisted of 50 000 waves of concurrent requests. Results were following:

| Time in s | Disabled | Enforcing | Difference |
|---|---|---|---|
| ext2 | 4305.27 | 4449 | 3.2% |
| ext4 | 4319.68 | 4482.65 | 3.6% |
| xfs | 4306.81 | 4400.47 | 2.1% |

This tests' results were more definite than previous result. In this test new dynamic page was generated for every request, which resulted in higher SELinux impact on the results.

CPU load in kernel space did not differ as much as it did in the previous test. It also decreased 5% in average, which can be explained by disk playing bigger role in this test. Generating PHP pages takes up more time on disk (disk was busy at rate around 5%), than it loads the CPU. Still, difference between enforcing and disabled SELinux was 1.6% on ext2, 1.8% on ext4 and 0.9% on XFS.

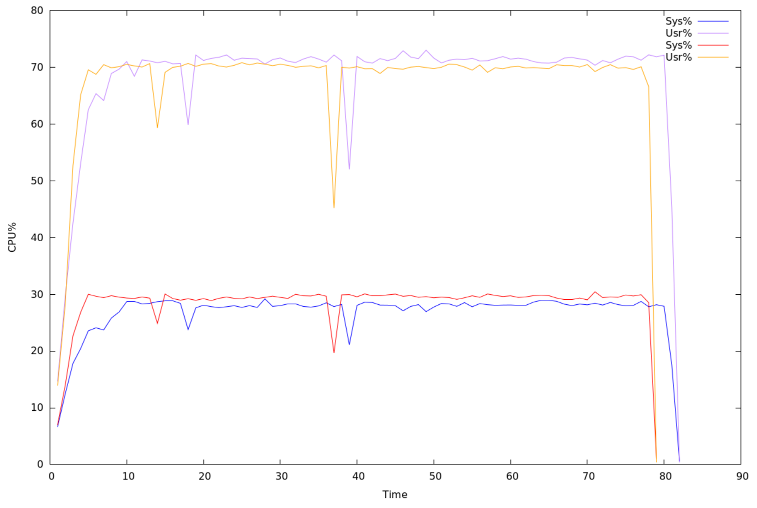

Figure 3.4: CPU load, PHP test, ext2

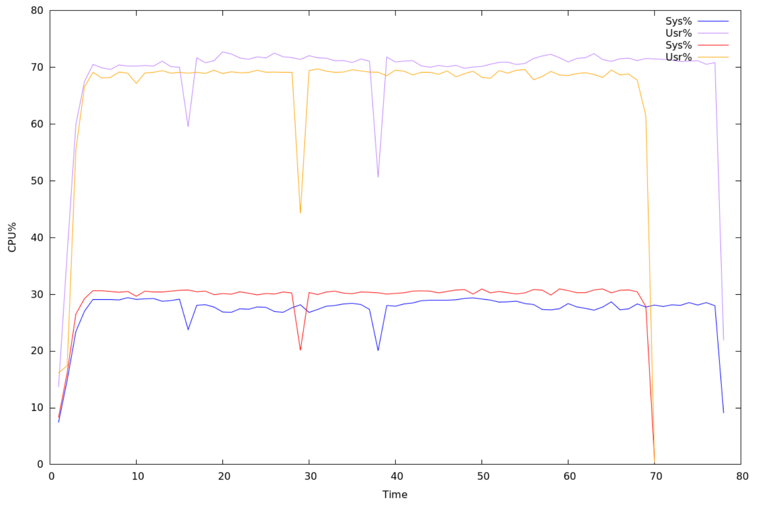

Figure 3.5: CPU load, PHP test, ext4

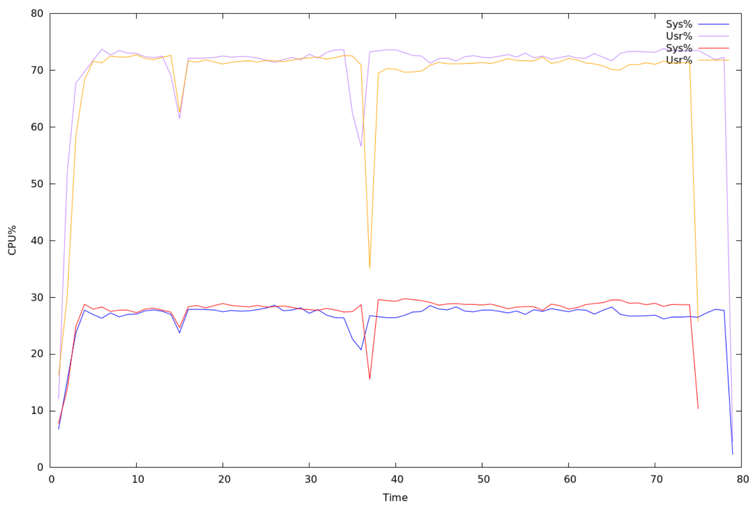

Figure 3.6: CPU load, PHP test, XFS

#### 3.2.3.3 Testing dynamic PHP page with MySQL queries

Third test was hitting a PHP page that queried database. As well as the first test, there was exponentially rising concurrency from 1 to 128. Each concurrency level consisted of 50 000 waves of concurrent requests. Results are presented in the following table:

| Time in s | Disabled | Enforcing | Difference |
|---|---|---|---|
| ext2 | 7411.52 | 7767.38 | 4.6% |
| ext4 | 7392.53 | 7828.81 | 5.6% |
| xfs | 7853.91 | 8177.09 | 3.9% |

The nature of this test is similar to the previous one, as it requests dynamic content, but in addition queries database. This was too much to handle for the server when the concurrency rose above 32, which can be seen on figures 3.7, 3.8 and 3.9. Load of the CPU in kernel space differed again in favor of disabled SELinux to enforcing by 2.8% on ext2, 2.9% on ext4 and 3.1% on XFS.

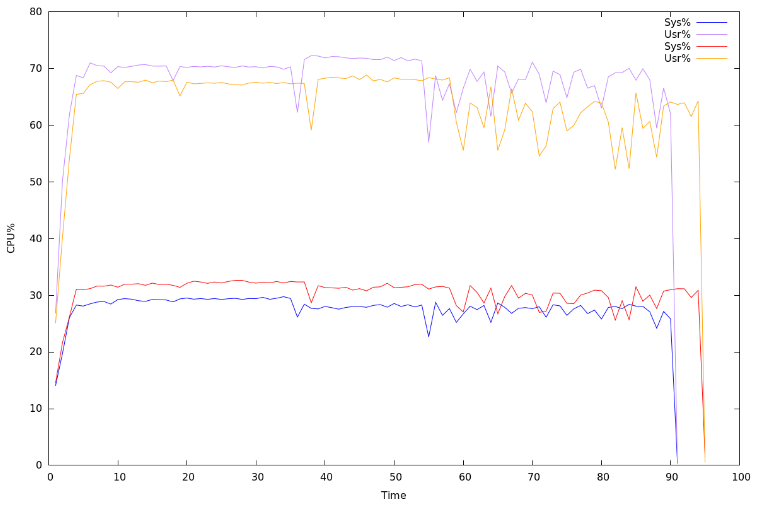

Figure 3.7: CPU load, MySQL test, ext2

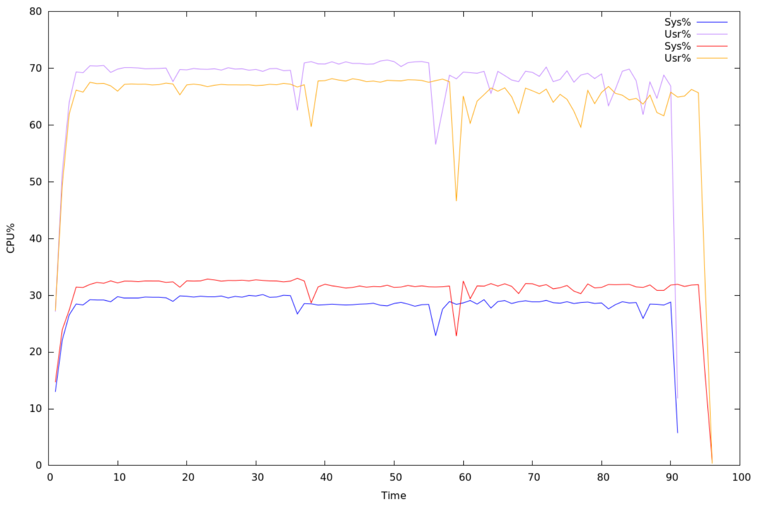

Figure 3.8: CPU load, MySQL test, ext4

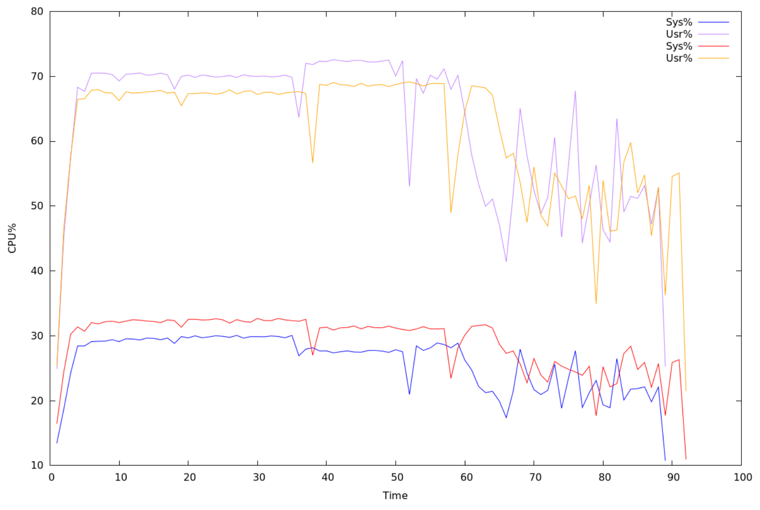

Figure 3.9: CPU load, MySQL test, XFS

#### 3.2.3.4 Random tests

Random tests each ran for two hours. These tests were randomly choosing from the list located in `/etc/siege/urls.txt`. This list contained 3000 different pages, evenly distributed between pages using HTML, PHP and PHP with MySQL queries. In addition, there were 1000 false pages that weren't located on the server. Goal of this approach was to simulate real world load of the webserver. This test was running in two modes that differ in concurrency of the requests. One was set 750 requests/second and the other one 1500 requests/second.

Concurrency: 750 requests/s

| Transaction/s | Disabled | Enforcing | Difference |
|---|---|---|---|
| ext2 | 747.73 | 747.87 | 0.0% |
| ext4 | 748.04 | 748.16 | 0.0% |
| xfs | 748.33 | 748.38 | 0.0% |

b However, looking at CPU load in figures 3.10, 3.11 and 3.12, we discover that kernel space is still more loaded under enforcing SELinux - 0.6% on ext2, 0.8% on ext4 and 0.9% on XFS.

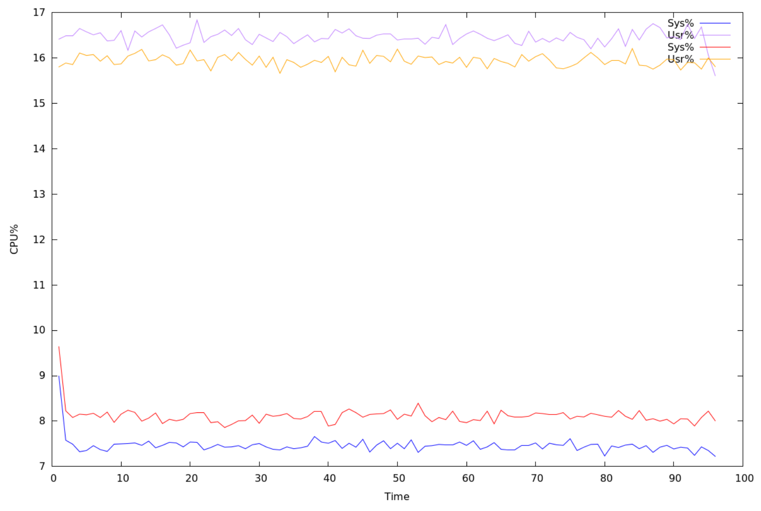

Figure 3.10: CPU load, random 750 test, ext2

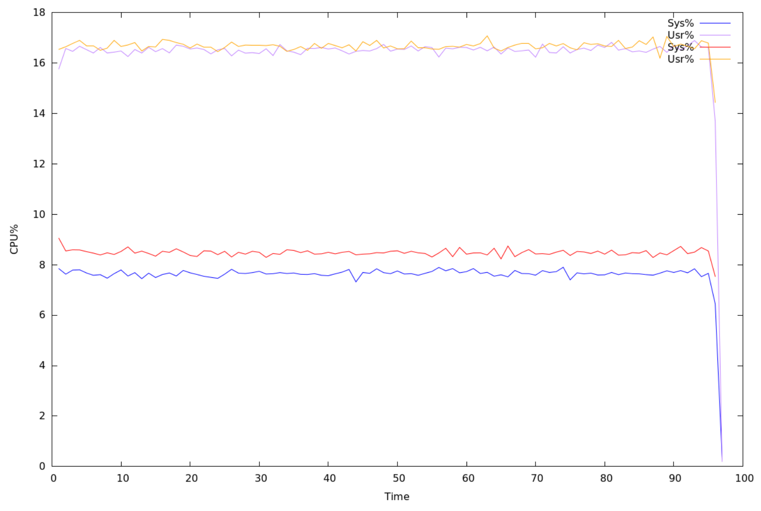

Figure 3.11: CPU load, random 750 test, ext4

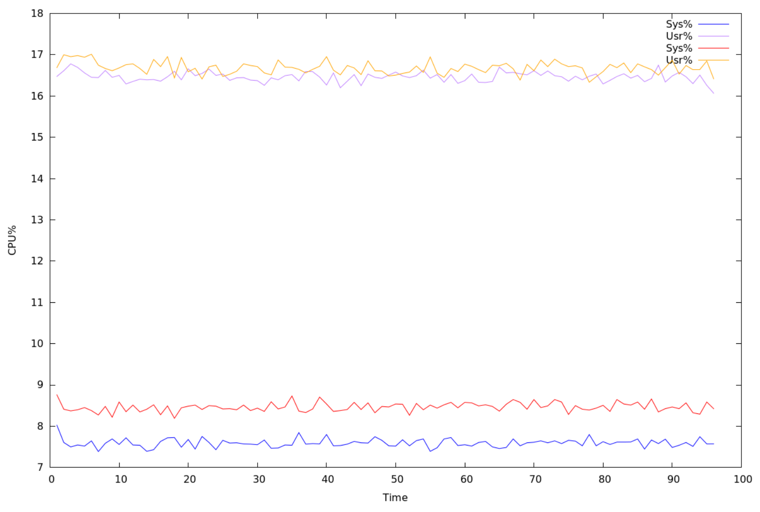

Figure 3.12: CPU load, random 750 test, XFS

Concurrency: 1500 requests/s

| Transaction/s | Disabled | Enforcing | Difference |
|---|---|---|---|
| ext2 | 1295.45 | 1300.08 | -0.3% |
| ext4 | 1310.92 | 1160.09 | 11.5% |
| xfs | 1305.64 | 1240.42 | 5.0% |

These results look rather chaotic. Results are very different for each filesystem tested. This may be caused by the randomness of the test.

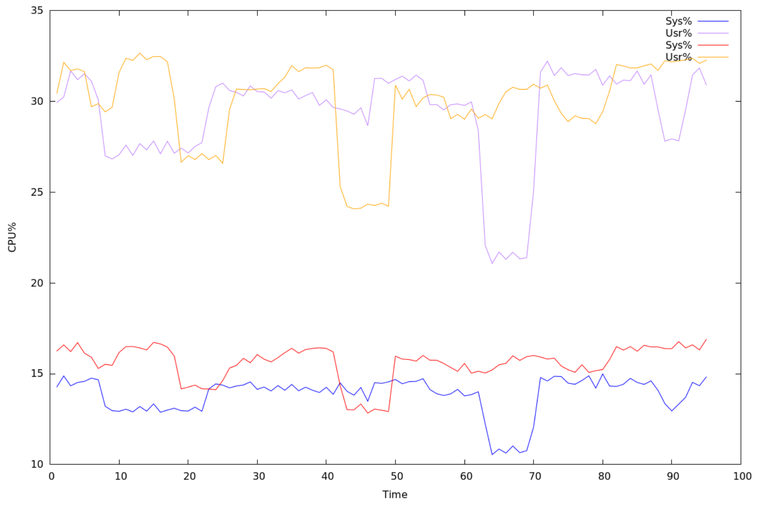

Figure 3.13: CPU load, random 1500 test, ext2

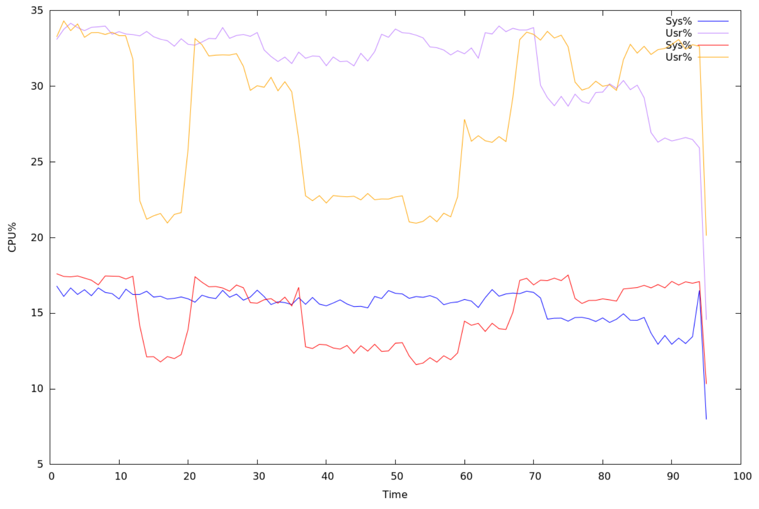

Figure 3.14: CPU load, random 1500 test, ext4

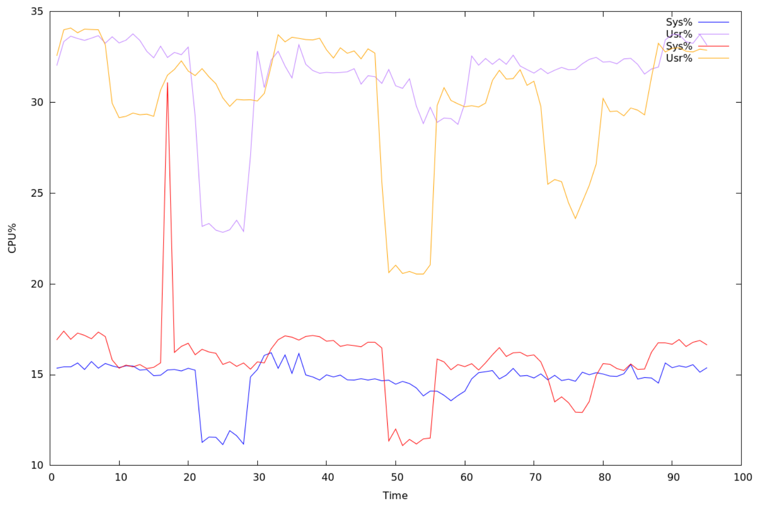

Figure 3.15: CPU load, random 1500 test, XFS

To get better result from this test, it was run again, this time for four hours. The results were following:

| Transaction/s | Disabled | Enforcing | Difference |
|---|---|---|---|
| ext2 | 1193.26 | 1195.80 | -0.2% |
| ext4 | 1231.35 | 1221.25 | 0.8% |
| xfs | 1231.55 | 1232.84 | -0.1% |

Running tests for longer period of time stabilized the measured values. Under this load, servers' ability to serve requests was not considerably influenced by SELinux. However, CPU load in kernel space was consistently higher in enforcing SELinux: by 1.3% on ext2, 1.1% on ext4 and 1.9% on XFS.

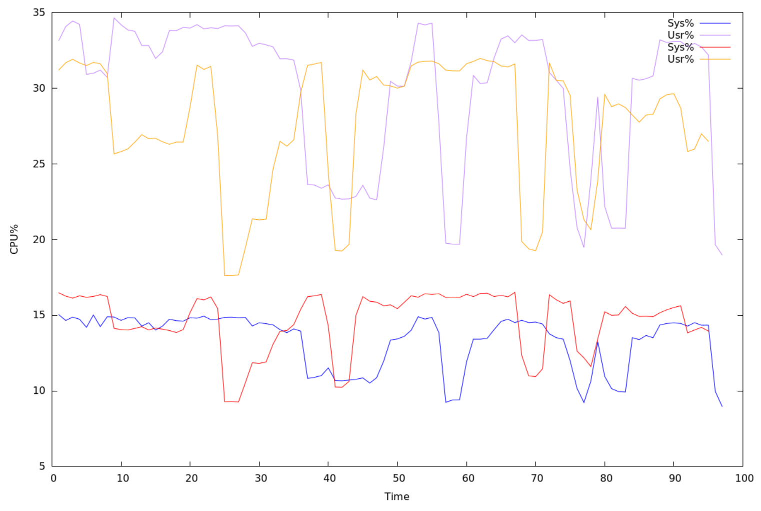

Figure 3.16: CPU load, random 1500 longer test, ext2

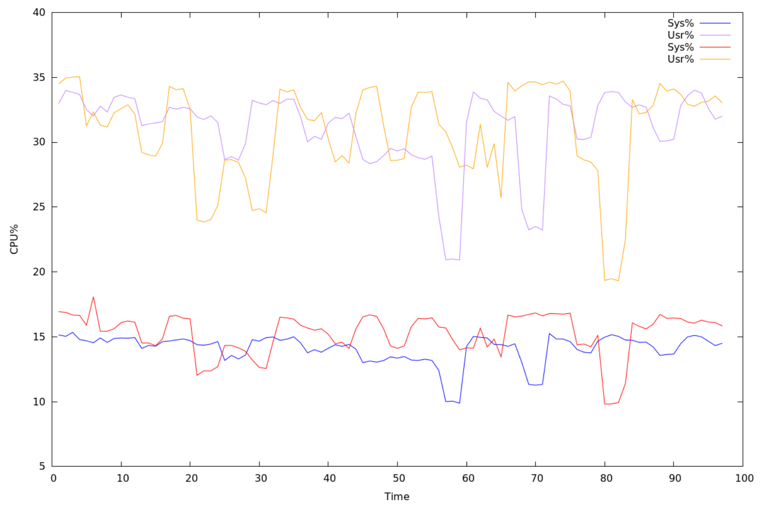

Figure 3.17: CPU load, random 1500 longer test, ext4

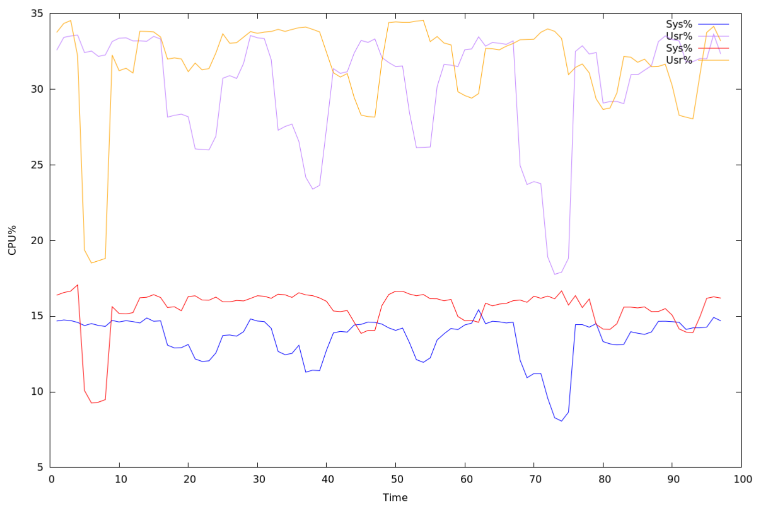

Figure 3.18: CPU load, random 1500 longer test, XFS

### 3.2.4 Optimizations

Previously proposed optimizations were tested using same set of tests that were used for testing disabled and enforcing SELinux.

#### 3.2.4.1 Additional configuration

Additional configuration was needed to prepare the environment. Namely the security context of the files in `/var/www/html` had to be changed to *unconfined*. `httpd` service, running as *confined* has special security context for files that should not be protected by SELinux: `httpd_unconfined_script_exec_t`. As the name suggests, this security context is supposed to be used for executable scripts, but we will use it also for static `.html` pages. Using the following commands:

```
semanage fcontext -a -t httpd_unconfined_script_exec_t \
"/var/www/html(/.*)?"
cd /var/www/
restorecon -R html/
```

we change the security context of the files and we can test this setup.

#### 3.2.4.2 Testing static HTML page

CPU kernel load was consistently lower on all filesystems, namely by 0.6% on ext2, 4.3% on ext4 and 0.9% on xfs. In spite of this fact, the results were following:

| Time in s | Optimized | Not Optimized | Difference |
|---|---|---|---|
| ext2 | 2422.18 | 2500.95 | 3.2% |
| ext4 | 2684.67 | 2515.21 | -6.7% |
| xfs | 2474.56 | 2452.38 | -0.9% |

#### 3.2.4.3 Testing dynamic PHP page

CPU kernel load was consistently lower, by 2.9% on ext2, 1.1% on ext4 and 2% on xfs. In spite of this fact, the results were following:

| Time in s | Optimized | Not Optimized | Difference |
|---|---|---|---|
| ext2 | 4762.57 | 4449 | -7% |
| ext4 | 4305.56 | 4482.65 | 4.1% |
| xfs | 4789.25 | 4400.47 | -8.8% |

#### 3.2.4.4 Testing dynamic PHP page with MySQL queries

In this test, the CPU load in kernel space was lower in optimized version on ext2 by 1.4% and on xfs by 1.2%. On ext4 the difference was less than 0.1%.

| Time in s | Optimized | Not optimized | Difference |
|---|---|---|---|
| ext2 | 8162.32 | 7767.38 | -5.1% |
| ext4 | 7555 | 7828.81 | 3.6% |
| xfs | 8464.68 | 8177.09 | -3.5% |

#### 3.2.4.5 Additional testing of dynamic content

In three tests above we got consistently lower CPU load in kernel space, which was expected. However, most of tasks did take more time in optimized version than in the default enforcing mode. For dynamic content results were as expected on ext4, but different on ext2 and xfs. We tried another test with various file sizes to determine whether these unexpected results were caused by filesystem. It was expected for the larger files to increase the difference between the two versions, while the smaller ones should have reduced it. However, results in the following table did not confirm this.

| Filesystem | Mode | Not Optimized | Optimized | Difference |
|---|---|---|---|---|
| ext2 | larger files | 23367.2 | 23413 | -1.96% |
| ext2 | original test | 7767.38 | 8162.32 | -5.08% |
| ext2 | smaller files | 7204.87 | 7131.59 | 1.03% |
| ext4 | larger files | 23757 | 23806.6 | -1.00% |
| ext4 | original test | 7828.81 | 7555 | 3.62% |
| ext4 | smaller files | 7169.76 | 7119.61 | 0.70% |
| xfs | larger files | 23022.3 | 23034.5 | -0.05% |
| xfs | original test | 8177.09 | 8464.68 | -3.52% |
| xfs | smaller files | 7639.36 | 7732.47 | -1.22% |

#### 3.2.4.6 Random tests

Results of random test with concurrency level 750 where as following:

| Transaction/s | Optimized | Not optimized | Difference |
|---|---|---|---|
| ext2 | 747.25 | 747.87 | -0.1% |
| ext4 | 748.04 | 748.16 | 0% |
| xfs | 747.75 | 748.38 | -0.1% |

As you can see, there is virtually no difference between the two SELinux modes in this test. 750 requests per second is too little load to make a reasonable difference, as it was the case in the previous test with disabled and enforcing modes.

Random tests with 1500 concurrency were run for 4 hours, because of the experience from the previous testing. Results were following:

| Transaction/s | Optimized | Not optimized | Difference |
|---|---|---|---|
| ext2 | 1258.94 | 1195.80 | 5.3% |
| ext4 | 1232.42 | 1221.25 | 0.9% |
| xfs | 1225.05 | 1232.84 | -0.6% |

These results shows that on xfs non-optimized version serves more requests per second. On ext2 and ext4 however, more requests are served in optimized version.

## 3.3 Fileserver

### 3.3.1 Configuration

Setting up a fileserver can be divided into several steps that are needed to be performed for a properly running fileserver.

1. Install OS
2. Configure networking
3. Download necessary packages
4. Configure vsftpd
5. Configure access rights and SELinux security contexts
6. Configure SELinux booleans
7. Generate files
8. Enable necessary services

Install OS and configure networking steps are same as mentioned in configuring webserver. Now we are going to take a look at rest of these steps closer.

#### 3.3.1.1 Download necessary packages

Now, that we have access to the Internet, we can download necessary packages for fileserver. In fact, we need only FTP client (in our case `vsftpd`), because we will test SFTP with Open-SSH server that is already part of minimal installation of CentOS. We also need additional packages for system administration and performance monitoring.

- `policycoreutils-python`
- `http://pkgs.repoforge.org/rpmforge-release/rpmforge-release-0.5.3-1.el7.rf.x86_64.rpm`
- `nmon`

#### 3.3.1.2 Configure vsftpd

In `/etc/vsftpd/vsftpd.conf` file we have to change upload policy of the `vsftpd`. We add following two lines to the config file:

```
anon_upload_enable=yes
no_anon_password=yes
```

Now anonymous upload is enabled without prompting for password, although there are still some changes needed for this feature to function properly.

#### 3.3.1.3 Configure access rights and SELinux security contexts

During installation of `vsftpd`, there is a new directory created, `/var/ftp`. This is the directory the anonymous FTP user is 'chrooted' to when he connects to the server. This directory is not writable (in terms of standard UNIX access control) and it needs to stay this way (security check of `vsftpd` will not let us perform FTP transfer with writable FTP root directory, unless forced to). There is another directory inside previous one: `/var/ftp/pub`. We give full access to all users on this directory, as we want to enable anonymous download and upload of files, from and to this directory.

This itself is still not enough for anonymous FTP upload to work. By default, `/var/ftp/pub` directory has security context `public_content_t`. This security context grants read permissions, but denies any write access to the directory. We need to change context of this directory and all the files and subdirectories created within it. Our target security context is `public_content_rw_t`, which grants users both read and write permissions. To do so we use following commands:

```
semanage fcontext -a -t public_content_rw_t "/var/ftp/pub(/.*)?"
cd /var/ftp/
restorecon -R pub/
```

After this change the security context of the file changes to `public_content_rw_t`.

#### 3.3.1.4 Configure SELinux booleans

After all of the previous configuration, FTP upload still exits with an access error. Last thing to examine when configuring fileserver is SELinux booleans. The boolean that we need to change is `ftpd_anon_write`, that is in the off state by default. It has to be changed using following command:

```
setsebool -P ftpd_anon_write on
```

Another boolean that would solve our problem is `ftpd_full_access`. Although this boolean solves the problem of anonymous upload, it disables SELinux for FTP completely and therefore our results would be useless.

#### 3.3.1.5 Generate files

For testing we need to generate enough files. We will generate these files randomly using `/dev/urandom` and `dd` tool. Files will have size of 1 GiB.

```
for i in ‘seq $NumOfFiles‘
do
dd if=/dev/urandom of=/var/ftp/pub/dfile$i bs=4M count=256
```

```
done
```

#### 3.3.1.6 Enable necessary services

When everything is ready we must enable all the services we need and also adjust firewall for our new services using following commands:

```
systemctl enable vsftpd
firewall-cmd --permanent --add-service=ftp
```

After reboot we have a running fileserver capable of serving requests during testing.

### 3.3.2 Testing

Testing a fileserver will consist of simultaneously downloading and uploading larger files, with simple `ftp`, but also secure `sftp` using `ssh`.

#### 3.3.2.1 Testing FTP with wget and wput

`wget` is a non-interactive network downloader and `wput` is `wget`-like FTP uploader. Using these two programs we will simulate FTP traffic on the fileserver. As the previous tests on the webserver with `ab`, this test should also start with a small load and go up to large number of concurrent downloads and uploads.

```
#!/bin/bash
for i in ‘seq $1‘
do
  wget -b ftp://$HOSTNAME/file$i
  DELAY=$RANDOM
  let "DELAY %= $RANGE"
  sleep $DELAY
done
```

This simple script will invoke a number of concurrent downloads (set as an argument of the script) using wget that is sent to background (`-b` option). `wget` will save output to the log file from which we will parse download time of each file or can compute average download time. All other important data will be collected with `nmon`. Same applies for `wput`.

Three commands after running `wget` are computing delay before the next download is initiated. This is done to scatter the downloads/uploads in time, simulating more real load.

#### 3.3.2.2 Testing SFTP with scp

This test will generate traffic using `sftp` protocol to download/upload files with `scp` program.

```
#!/bin/bash
for i in ‘seq $i‘
do
  scp -v up_file$i $HOSTNAME:uploaded_file$i 2> log.up$i &
  DELAY=$RANDOM
  let "DELAY %= $RANGE"
  sleep $DELAY
done
```

The `scp` program sends log data (`-v` option) to stderr, which will be redirected into a file, from which we can parse time needed to transfer file or throughput of the transaction and also compute average time of transaction. All other important data will be collected by `nmon`.

We will also try concurrent download/upload by invoking `scp` both ways at the same time. To solve the problem with interactive password authentication, we will use `sshpass` [24] program.

### 3.3.3 Results

Testing of fileserver consisted of two tests. First was testing plain `ftp` using `wget` and `wput` tools and second was testing `sftp` using `ssh`. Complete results are available in `res/file` directory on the CD. On the graphs enforcing SELinux is in red and orange lines, while disabled SELinux is in blue and purple lines.

#### 3.3.3.1 FTP test

We carried out concurrent download/upload of 25 files (20 download, 5 upload) that had size 1 GiB and we repeated each test three times. 10% of files during testing timeouted and were not completed. During all FTP tests as you can see in example figure 3.16, CPU load was never above 10% for a longer period of time.

Large number of timeouts indicates that there is a bottleneck in this configuration. It is not the CPU for obvious reasons. Other candidates, that are stressed during file transfer are network and disk. As you can see in example figure 3.17, network is not working at peak of its capabilities and network load is fluctuating. This is the case in all the results.

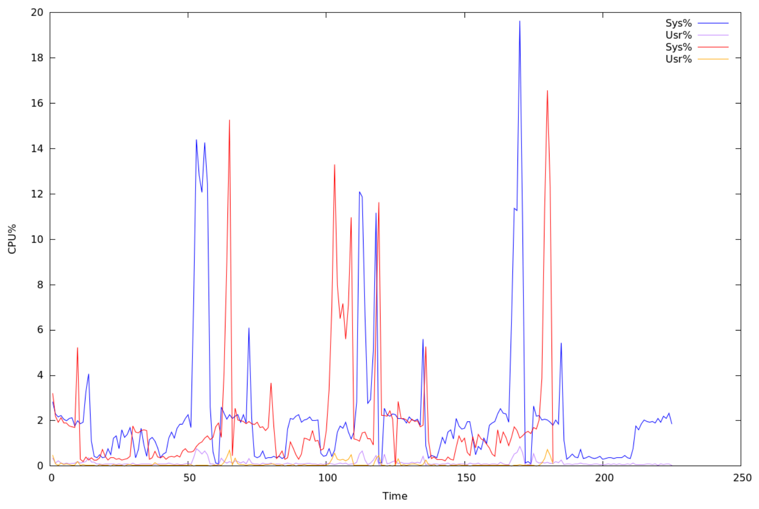

Figure 3.19: CPU load, ftp test, XFS

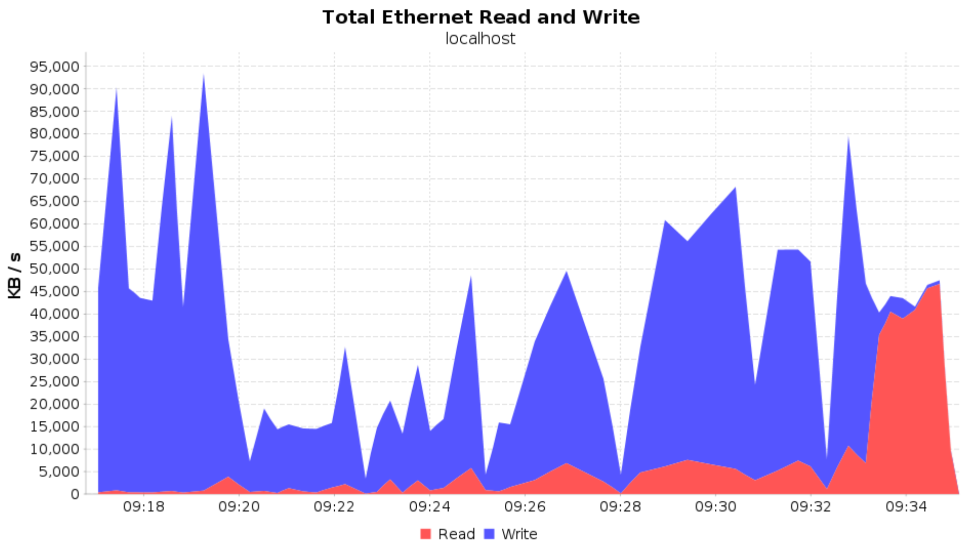

Figure 3.20: Network load, ftp test, ext2

After excluding network as a bottleneck, the next possible candidate is disk. Disk load differs depending on the filesystem. Results:

| Disk busy in % | Disabled | Enforcing |
|---|---|---|
| ext2 | 88.3 | 91.5 |
| ext4 | 70.6 | 77.6 |
| xfs | 96.5 | 98.9 |

Disk is clearly the bottleneck when using xfs. It has potential of being a bottleneck when using ext filesystems, as this test was aimed to be a load test. By increasing concurrent downloads/uploads we would be limited by disk on xfs server immediately and after rising the load also on other filesystems.

#### 3.3.3.2 SFTP test

We carried out concurrent download/upload of 25 files (20 download, 5 upload) that had size 1 GiB and we repeated each test three times. There were no timeouts, like during FTP test. Secure transfer did put more load on the CPU.

| CPU kernel load in % | Disabled | Enforcing | Difference |
|---|---|---|---|
| ext2 | 5.7 | 5.4 | -0.3 |
| ext4 | 8.8 | 9.3 | 0.5 |
| xfs | 5.6 | 6.1 | 0.5 |

Difference spotted in CPU kernel load is not sufficient to say that SELinux does have impact here. As seen in the example in figure 3.17, CPU is load is roughly the same.

We must still examine the possibility of disk being the bottleneck of this test. We get following results:

| Disk busy in % | Disabled | Enforcing |
|---|---|---|
| ext2 | 87.8 | 86.6 |
| ext4 | 83.5 | 81.9 |
| xfs | 94.4 | 95.3 |

We get similar results, but disk is by roughly 3-4% less busy than it was during FTP test. This may be caused by higher CPU load, in other words, CPU is more loaded and is processing data for longer period of time, therefore disk has the same work scattered throughout longer period of time. Increasing load by adding additional downloads/uploads would cause disk to be fully busy and limit us before SELinux would make a difference in the overall performance.

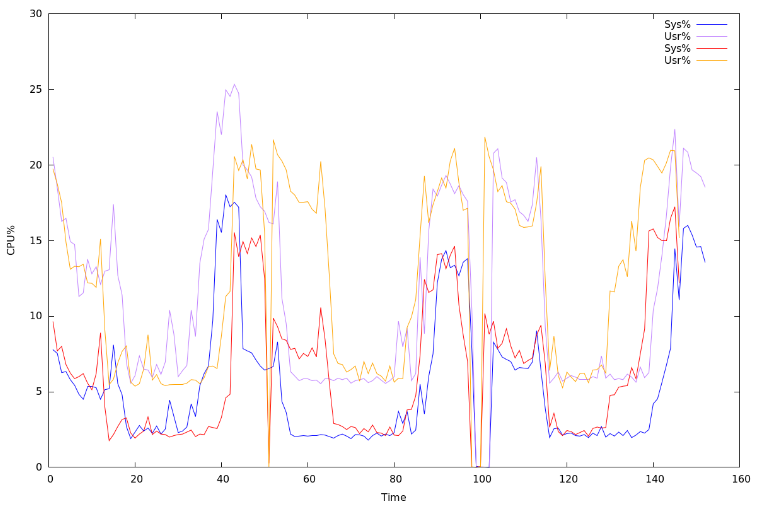

Figure 3.21: CPU load, ftp test, XFS

### 3.3.4 Optimizations

As was mentioned in the previous section, fileserver is limited by performance of the disk before SELinux makes any difference in the overall performance. Based on this observation, testing proposed optimizations is no longer reasonable, as the same results with disk as a bottleneck of the system would be measured.

## 3.4 Workstation

### 3.4.1 Configuration

Steps needed for workstation configuration:

1. Install OS
2. Configure networking
3. Download necessary packages
4. Install benchmarks

#### 3.4.1.1 Install OS

Configuring a workstation is less complicated than configuring one of the servers. It is mainly because we will not install 'Minimal installation' software like before, but 'GNOME Desktop' with 'Development tools' option of CentOS.

#### 3.4.1.2 Configure networking

We set up networking using NetworkManager with GUI. We use the same configuration as before. Running selected benchmarks requires internet connection, so client/firewall should be running same as it was the case when testing servers, but this time it is used only for connecting to the Internet and not for the testing itself.

#### 3.4.1.3 Download necessary packages

Packages we need to download:

- `policycoreutils-python`
- `http://pkgs.repoforge.org/rpmforge-release/rpmforge-release-0.5.3-1.el7.rf.x86_64.rpm`
- `nmon`
- `php-cli`
- `php-xml`
- `phoronix-test-suite` (needs to be downloaded and installed separately)

PHP packages are needed for Phoronix Test Suite to work properly.

#### 3.4.1.4 Install benchmarks

To install benchmarks we are going to perform, we invoke following commands:

```
phoronix-test-suite install pts/compilation
phoronix-test-suite install pts/compilebench
```

### 3.4.2 Testing

For testing workstation we have previously selected ‘Timed code compilation’ and ‘Compile bench’ benchmarks. These will be run using following command:

```
nmon -fT -s 5 -c 2880 -m $HOME/nmon-data && phoronix-test-suite benchmark
pts/compilation
nmon -fT -s 5 -c 2880 -m $HOME/nmon-data && phoronix-test-suite benchmark
pts/compilebench
```

During ‘Compile bench’, when asked what type of test do we want to perform, we choose option ‘4: Test all options’.

### 3.4.3 Results

Testing of workstation was divided into two parts. First was testing compilation times using ‘Timed Code Compilation’ benchmark. Second was using ‘Compilebench’ benchmark that works with the filesystem and simulates aging of the filesystem. Complete results are available in `res/work` directory on the CD.

#### 3.4.3.1 Timed Code Compilation

Test consisted of four smaller compilation tasks: Timed Apache Compilation 2.4.7, Timed Linux Kernel Compilation 3.18-rc6, Timed MPlayer Compilation 1.0-rc3 and Timed PHP Compilation 5.2.9. These tasks were invoked using `phoronix-test-suite`, which ran each test three times and drew average value.

| Time in s | ext2 | | ext4 | | xfs | |
|---|---|---|---|---|---|---|
| | Disabled | Enforcing | Disabled | Enforcing | Disabled | Enforcing |
| Apache | 127.73 | 127.68 | 126.43 | 128.66 | 126.81 | 128.51 |
| Linux Kernel | 683.13 | 691.87 | 685.69 | 699.41 | 682.93 | 694.90 |
| MPlayer | 252.99 | 265.23 | 252.79 | 256.23 | 252.17 | 266.92 |
| PHP | 155.26 | 166.58 | 156.53 | 157.31 | 156.37 | 167.24 |

These values show us, that enforcing SELinux did take more time compiling. We drew average CPU load from whole benchmark run and the results were as follows:

| CPU kernel load in % | Disabled | Enforcing | Difference |
|---|---|---|---|
| ext2 | 10.7 | 11.4 | 0.7 |
| ext4 | 10.7 | 11.3 | 0.6 |
| xfs | 11 | 11.6 | 0.6 |

#### 3.4.3.2 Compilebench

This benchmark was divided into three smaller tasks: Compile, Initial Create, Read Compiled Tree. It measured filesystems’ throughput in MB/s.

| Throughput in MB/s | ext2 | | ext4 | | xfs | |
|---|---|---|---|---|---|---|
| | Disabled | Enforcing | Disabled | Enforcing | Disabled | Enforcing |
| Compile | 21.81 | 21.72 | 30.42 | 29.67 | 25.18 | 22.52 |
| Initial Create | 21.13 | 21.43 | 26.86 | 26.04 | 15.43 | 13.46 |
| Read Tree | 10.06 | 10.73 | 16.49 | 16.47 | 15.65 | 13.43 |

In this test, we can see decrease of throughput after turning SELinux on into enforcing mode when using `xfs`, but we do not observe any change of throughput on `ext2` and `ext4` filesystems. CPU loads are as follows:

| CPU kernel load in % | Disabled | Enforcing | Difference |
|---|---|---|---|
| ext2 | 4.1 | 4.9 | 0.8 |
| ext4 | 5.3 | 6.2 | 0.9 |
| xfs | 4.9 | 5.3 | 0.4 |

# Conclusion

In this thesis we examined SELinux technology and its setup in CentOS operating system. We designed testing environment and configuration of three scenarios: webserver, fileserver and workstation. Using the previously created testing scripts we tested all three scenarios in three versions: SELinux off, SELinux on and SELinux on with previously drawn up optimizations.

Webserver was tested with stress tests of a single page and load test in form of random simulation. Stress tests showed, that SELinux does have observable impact on performance, however the load test did not show any significant impact. This means that presence of SELinux would cause performance issues only in traffic that would fully load the server. Optimizations for this version were tested and lowered the CPU load in kernel space. On the other hand, some of the tests took longer to complete, which makes these tests ambiguous. Additional tests did not identify the problem.

Fileserver was tested with concurrent downloads and uploads which did not load CPU as was expected. Disk was identified as the bottleneck of the filesystem. Presence of SELinux has no effect on performance of fileserver, as disk will limit fileservers' performance before SELinux. Optimizations for this scenario were not tested, because even the initial test was limited by the disk.

Workstation was tested using benchmarks simulating compiling and work with filesystem. This showed us that workstation cannot be loaded to such an extent that SELinux would have impact on performance. Optimizations for this scenario were not drawn up, because it was expected for the results to not have observable differences in performance.

# Bibliography

[1] Mayer, F.; MacMillan, K.; Caplan, D. *SELinux by Example: Using Security Enhanced Linux*. Prentice Hall, 2006, ISBN 0131963694.

[2] Loscocco, P. A.; Smalley, S. D.; Muckelbauer, P. A.; et al. The Inevitability of Failure: The Flawed Assumption of Security in Modern Computing Environments. 2002. Available from: `http://www.windowsecurity.com/whitepapers/misc/The_Inevitability_of_Failure_The_Flawed_Assumption_of_Security_in_Modern_Computing_Environments_.html`

[3] Garfinkel, S.; Spafford, G. *Web Security and Commerce (Nutshell Handbooks)*. O'Reilly Media, 1997, ISBN 1565922697.

[4] Anderson, J. Computer Security Technology Planning Study. 1972. Available from: `http://csrc.nist.gov/publications/history/ande72.pdf`

[5] Linden, T. Operating System Structures to Support Security and Reliable Software. Dec. 1976. Available from: `http://csrc.nist.gov/publications/history/lind76.pdf`

[6] Čapek, T.; Ančincová, B. Red Hat Enterprise Linux 7 SELinux User's and Administrator's Guide. 2015. Available from: `https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/pdf/SELinux_Users_and_Administrators_Guide/Red_Hat_Enterprise_Linux-7-SELinux_Users_and_Administrators_Guide-en-US.pdf`

[7] The Apache Software Foundation. httpd. [software]. Available from: `http://httpd.apache.org/`

[8] Evans, C. vsftpd. [software]. Available from: `https://security.appspot.com/vsftpd.html`

[9] MariaDB Foundation. mariadb-server. [software]. Available from: `https://mariadb.org/`

[10] Red Hat, Inc. Fedora. [software]. Available from: https://getfedora.org/

[11] What is Stress Testing? 2013. Available from: http://www.testingperformance.org/definitions/what-is-stress-testing

[12] What is Load Testing? 2013. Available from: http://www.testingperformance.org/definitions/what-is-load-testing

[13] IBM. Nigel's Performance Monitor. [software]. Available from: http://nmon.sourceforge.net/

[14] Presnall, H. Nmon Visualizer. [software]. Available from: http://nmonvisualizer.github.io/nmonvisualizer/

[15] Mauelshagen, H. LVM Readme. 2003. Available from: http://ftp.gwdg.de/pub/linux/misc/lvm/1.0/README

[16] The CentOS Project. CentOS 7. [software]. Available from: https://www.centos.org

[17] The Apache Software Foundation. ab. [software]. Available from: http://httpd.apache.org/docs/2.2/programs/ab.html

[18] Fulmer, J. Siege. [software]. Available from: https://www.joedog.org/siege-home/

[19] Free Software Foundation, Inc. wget. [software]. Available from: http://www.gnu.org/software/wget/

[20] Fritsch, H. wput. [software]. Available from: http://wput.sourceforge.net/

[21] OpenBSD. ssh. [software]. Available from: http://www.openssh.com/

[22] Phoronix Media. phoronix-test-suite. [software]. Available from: http://www.phoronix-test-suite.com/

[23] COMPILE BENCH. 2010. Available from: http://openbenchmarking.org/test/pts/compilebench

[24] sshpass. [software]. Available from: http://sourceforge.net/projects/sshpass/

APPENDIX A

# Acronyms

| | |
|---|---|
| **AVC** | Access Vector Cache |
| **CGI** | Common Gateway Interface |
| **CIFS** | Common Internet File System |
| **CLI** | Command Line Interface |
| **CPU** | Central Processing Unit |
| **DAC** | Discretionary Access Control |
| **DHCP** | Dynamic Host Configuration Protocol |
| **DNS** | Domain Name System |
| **FTP** | File Transfer Protocol |
| **GUI** | Graphical User Interface |
| **HTTP** | Hyper Text Transfer Protocol |
| **IO** | Input/Output |
| **LAMP** | Linux, Apache, MySQL, PHP |
| **LSM** | Linux Security Modules |
| **LVM** | Logical Volume Manager |
| **MAC** | Mandatory Access Control |
| **NFS** | Network File System |
| **RHEL** | Red Hat Enterprise Linux |
| **SFTP** | Secure File Transfer Protocol |
| **SSH** | Secure Shell |
| **WAN** | Wide Area Network |

# Appendix B

# Contents of enclosed CD

```
readme.txt .......................... the file with CD contents description
src ................................... the directory of source codes
  scripts ............................. implementation sources
  thesis ............ the directory of LaTeX source codes of the thesis
res ................................... the directory with results
  web ................................. complete results for webserver
  file ................................ complete results for fileserver
  work ................................ complete results for workstation
text .................................. the thesis text directory
  thesis.pdf .......................... the thesis text in PDF format
  thesis.ps ........................... the thesis text in PS format
  assignment.pdf ...................... the thesis assignment
```