Dr Tomasz Wojdyński
Harol Hryc
Faculty of Management Finance and Computer Science
The School of Banking and Management in Krakow

PROPOSAL FOR COMPUTER SYSTEM PROTECTION MEASURES FOLLOWNG THE EXPLOITATION OF LINUX SOFTWARE HOLES

Introduction

In the era of a widespread access to the Inernet and the incresing demand for services that are provided online, the amount of danger is growing. As a result, a computer system, which consists of four elements: hardware, an operational system, a utility program and users[1], and which provides services, has to confront the hazards. Media are full of news about the cases of website hacking, leaks of confidential information or servers that cannot be used due to unlawful attacks. The effects of such events may be disastrous. The revelation of company data may even result in bankruptcies (the theft of technologies or details of new products, etc.) or lawsuits of clients (when their data is leaked). Most commonly it is a human factor to be blamed for such events. However, the place where the mistake was made is crucial and consequently there may be various "culprits". They may be the users, who misused their passwords, the administrators who missed a potential security hole, the programmers who overlooked errors in the software they had written or finally, the hardware producers. Consequently one of the basic issues concerning the storage and processing of information in a computer system is the protection of data against unauthenticated access.[2]

The aim of the article is to present the most common holes in the Linux operating system security and the proposal for security solutions to protect a computer system against entering other resources than the compromised service.

The article begins with the presentation of the widespread security problems that present-day systems (not necessarily Linux) have to deal with. However, the concepts of protection against these threats are not goiong to be discussed. First, issues are discussed that concern passwords which frequently constitute the weakest link in system security. Then, the article

[1] A. Silberschatz, P.B. Galvin, *Podstawy systemów operacyjnych*, Wydawnictwo Naukowo-Techniczne, Warszawa 2002, p. 3.
[2] Op cit. A. Silberschatz, P.B. Galvin, *Podstawy …*, p. 422.

present issues related to backdoors and other holes that were written on purpose to enable the access of unauthorized persons to the system being attacked. Next, the article discusses errors in applications that were not made on purpose and which can be used to enter a system by an unauthorized person. Finally, the article discusses safeguards that – in the cases of a compromise of the previously discussed holes which may be used - will prevent or hinder the access to other files of the system by an attacker.

1. Presentation of selected common threats

The increasingly wide use of computers and a decreasing knowledge level of their users results in the increase in number of the threats. The dangers usually involve unauthenticated attempts to enter systems with the aim to take control over them and to steal various kinds of information in order to use them directly by people who ordered the illegal access or to publish the information aiming at the discreditation of people/institutions. The offences in question end up in ransomes when in return for money the compromised data is not revealed or - quite the opposite – the data that has been blocked/encrypted is regained to its lawful owners.

1.1. Attacks involving attempts to gain a password

All user accounts should have passwords. Passwords play a key role in the overall system security.[3] It is a frequent case that passwords with a low level of complexity that can be somehow deciphered or gained constitute weak points.

A repeated use of the same passwords for different services may be a potential source of a leak that facilitates the access to the password. The solution is to use unique passwords everywhere. However, this may cause problems with remembering them as the existing trend is that the scale of the use of passwords is increasing. This issue can be currently solved by the application of password managers (a kind of databases that store passwords which are encrypted by one, possible strong, main password). One should also remember not to take down passwords anywhere and – if it was necessary to do so – to destroy the carrier with the password when not needed anymore. You should never throw it away to a dustbin as you never know if there is not anybody willing to look through the rubbish to find the necessary information.

Password cracking is another way of gaining an unauthorized access and the speed of the proces depends on the structure of the password itself and the system that is hacked. In the cases of breaking the hash of a password that was accessed through e.g. eavesdropping the

[3] E. Frisch, *Unix – Administracja system – second edition,* Oficyna Wydawnicza READ ME, Łódź 1997, p. 156.

communication between a client and a server, the time mecessary to break it depends on the password structure, the hash calculating algorithm and the calculation capacity of the device used for cracking. The basic method of protection in both cases mentioned above is to use strong passwords. When selecting a password, it should be remembered that the password:

- should not include commonly known words – it is practically certain that a hacker will use a dictionary;
- consists of as many characters of different types as possible – this is the protection against brute-force attacks and the like (where all character combinations are tested);
- is of "adequate" length.

Thus, a good password is a sequence of randomly-selecteed charactes like)}s)#]rjbHd5; _gmi7$D/:7n or hkQJ9=kK4Z. The length should depend on the number of times allowed for login attempts with an incorrect password. To make it clear, let's look at the following example. Let's assume that a system allows for one login attempt per second and the password consists of capital and small ASCII characters (2 * 26), digits (10), special characters (33) and the password is changed once a year. Theoretically, the number of cracking attempts will amount to 31536000 times (60 seconds x 60 minutes x 24 hours x 365 days). Consequently, in this case a 4-character password (81450625 possible combinations) should be considered adequate. However, this is only theory. Practically, a correct password can be found quickly depending on the accepted hacking strategy; so it seems advisable to lengthen the password by 2 additional characters (which will result in the total of $7*10^{11}$ possible combinations). If a service uses mechanisms in which gaining the password's hash is extremely difficult, such assumption regarding the length may be considered sensible. (e.g. the combination of the SSH protocol and the authentication by Kerberos). However, when there is a possibility of a password hash leak (for instance, an insecured website with its own login system), in the case of popular hash generating algorithms, getting a password is a question of seconds either with the use of rainbow tables (previously calculated password hashes) or – if this cannot be done as the password was salted (a known string of characters was added to a normal password) - the hash must be cracked. The examples of speeds (number of attempts per second) for common hashes with the application of Hashcat v3.00-beta-145-g069634a on a computer with eight Nvidia GTX 1080 graphic cards (graphic cards suit such type of tasks as they were designed to calculate concurrently significant amount of independent data – every opearation of hash calculation is independent and consequently such operations can be conducted simultaneously):

- MD5 200.3 GH/s;
- SHA1 68771.0 MH/s;
- SHA256 23012.1 MH/s;
- Whirlpool 2027.7 MH/s;
- phpass (used in such frameworks as Joomla or Wordpress) 55130.8 kH/s;
- WPA/WPA2 3177.6 kH/s.[4]

The analysis of the above values and the comparison with a 6 character password whose hash can be calculated by the MD5 allgorithm lead to a frightening conclusion. The password is going to be cracked in up to 3 seconds! However, if the same password were applied to secure a WiFi network (WPA2), the cracking would last up to 64 hours. These values assume that the person cracking the hash knows its length. Otherwise, the number of all combinations is given by $\sum_{i=1}^{n} a^i$, where *n* is the length of the password and *a* is the number of characters from the set of characters that can be used (letters, digits, special characters)

In order to decrease the risk of cracking a password by the system, the waiting time should be lengthened after an unsuccessful login attempt or a mechanism should be applied that monitors login attempts in logs. Moreover, rules should be added to reject such process when an accepted login limit is exceeded in a given time period (e.g. 20 unsucessful login attempts in an hour result in the rejection of a connection with the attacking address for, say, 3 hours). As regards applications, they have to be updated frequently so that the discovered holes are promptly repaired. What is more, the programmers of such applications should pay attention to password management in application hashes which have to be salted and calculated by an algorithm with a high computational complexity (e.g. SHA3 or Whirlpool). The best solution is always to use an authentication system in which passwords are not stored in the same servers as the application and they do not send hashes through the network (e.g. kerberos). As regards the user, it should be remembered that one password should not be used to many servers; it should not be put down on paper (password manager should be used if there are problems with remembering the passwords) and dictionary passwords (well known words or sentences) should not be applied.

[4] https://gist.github.com/epixoip/a83d38f412b4737e99bbef804a270c40 - (Accessed: 6.11.2016)

1.2. Backdoors

Backdoor is a program or a part of a program written to enable an unauthorized access to a system by a third party. In the case of Linux, worms of this type most frequently spread through the attempts of cracking SSH passwords[5]. Another way is to download a crafted system installation image from an installed backdoor. One more possibility is to enable a crafted executable file that installs a malicious code. It also happens that a backdoor is appended to some programs. An example of such a case is the ProFTPD incident when a distribution server was accessed and the attackers modified the application source code. As a result, ProFTPD 1.3.3c allowed for root access without login[6]. Another interesting case given in Silberschatz A., Galvin P.B., Operating system concepts…. is the situation when a cunning trap entry is installed in a complier. The complier can generate a typical object code and a trap entry irrespectively of the source code that it compiles. Such action is particularly treacherous as browsing through the source program does not reveal any problems. The information is included only in the source code of the compiler.[7]

The basic way to be secure against the above dangers is to avoid installing software from uncertain sources and to use strong passwords. Moreover, one should block the opportunity of remote loging in as root. Unfortunately, the only way to defend against the above case of attack on a software distribution server is to use internal defence that is developed by the distributor. If a given application is to provide services online, it is beneficial to separate the application from the system by, for instance, the containerization. If it cannot be done, it is recommended to apply MAC (Mandatory Access Control). However, the use of chroot as the mechanism of system protection is not recommended[8]. In the cases when an application has to be enabled from root level, the safest solution is to enable it on separate hardware on which only the services of this application are provided. One should remember about it and follow this rule as "unlike some other operating systems, the superuser has all privileges all the time, access to all files, commands, etc. Therefore, it is entirely too easy for a superuser to crash the system …".[9]

[5] Sample program: http://vms.drweb-av.pl/virus/?_is=1&i=4323517 – Accessed: 6.11.2016
[6] https://www.aldeid.com/wiki/Exploits/proftpd-1.3.3c-backdoor – Accessed: 6.11.2016
[7] Op.cit. Silberschatz A., Galvin P.B., *Podstawy …*, p. 746.
[8] https://access.redhat.com/blogs/766093/posts/1975883 – Accessed: 6.11.2016
[9] Op.Cit. E. Frisch, *Unix …*, p. 6.

1.3. Errors in applications

According to Steve McConnell „average experience is about 1-25 errors per 1000 lines of code"[10]. When compared to the number of code lines in common programs, the number of potential errors is rather high. For instance, the http Pache server includes over 1.8 million of code lines[11]. Thus, with the above assumption, the program may have approx. 1 800 errors. Probably, some of them were rectified but it is difficult – if not impossible – to eliminate all errors in such a big project. Some of the errors may facilitate an unauthenticated access to information. What is more, they may offer the opportunity to conduct various operations with a privilege level of a program with holes, including shellcode starting.

The most common attact on applications is the buffer overflow. It accounted for about 20% of all attacks in 2005[12] and despite the fact that such attack is increasingly less effective, it is important to understand its idea when one wants to explain another, rather dangerous, type of attack against which there is no effective protection. The concept of the attack consists firstly in analyzing the program whether all possible data loading entries are protected against the excessive amount of data. If an excessive amount of data is entered, there is a possibility that other varaiables will be overwritten in the program, including the return address function. Having recognized such an error, it may be used to execute the code that was entered by the attacker. Usually, its aim is to start a shellcode. At present, this type of attack is not so easy to execute due to setting up all program sections where data can be turned to nonexecutable ones while the section with the executable code is set up in the read-only mode. The implementation of this mechanism depends on the processor (NX bit) and in the older processors without this functionality its emulation is necessary.

The so called *return-to-libc* attack is much more difficult to execute as it bypasses the NX bit, which makes it also difficult to protect. This type of attack was precisely described in Defeating Solaris/SPARC non-executable stack protection[13] in 1999. J.McDonald published a code that executes such attack in the Solaris 2.6 system for SPARC architecture. Moreover, in 2007 a post was published that includes other remarkable cases of exploits on the same architecture - from simple attacts with this technique to some hybrid techniques of injecting a malicious code to the executable memory segments of the attacked program[14]. The attack

[10] S. McConnell, *Code complete, 2nd edition*, Microsoft Press, Redmond 2004, ch. 22 p. 25.
[11] https://www.openhub.net/p/apache - Accessed: 6.11.2016
[12] J. C. Foster, *Buffer Overflow Attacks: Detect, Exploit, Prevent,* Syngress Publishing, Rockland 2005, p. 20.
[13] J. McDonald, *Defeating Solaris/SPARC non-executable stack protection*. Bugtraq, Mar. 1999, Online: http://seclists.org/bugtraq/1999/Mar/4.
[14] M. Ivaldi, *Re: Older SPARC return-into-libc exploits. Penetration Testing*, Aug. 2007.

consists in the use of the existing executable code, most commonly of the standard language C library (hence the name *return-to-libc*) as almost all programs use it. The point is to find the address of the function in question (e.g. system () which is used to execute the command that is its argument) and the argument - most commonly a /bin/sh character sequence. One of the ways to hinder the attack is to use the *Address space layout randomization* in which every time the program is enabled, the function addresses and other important structures together with the functions from libraries that are used, are randomly changed, which makes it difficult to hit resources that are necessary to execute the attack.

The basic protection against such attacks is to write a secure code, which means that the person who is responsible for security is mainly the application programmer. It is his/her duty to check whether the data entered is correct and in sufficient amount. Moreover Valgrind tools can be used to support debugging. As regards the administrator of the protected system where the application is to operate, techniques can be used that enable the development of the code or randomization of the addresses of program resurces. This, however will not solve all the problems.

1.4. Protection against the access to data in the case of a service compromise

When an attacker successfully takes control over a service (e.g. calling a shellcode by a return-to-libc attack), he/she has the access to any data that a particular service uses. For instance, if a WWW server is subject to attack and the https protocol is used, the attacker can gain access to encryption certificates and nothing will stop him as he has the same privileges as the service he has compromised.

It is possible to have protection against the attacker's access to the sections of the system that are not necessary for the program to function. A fairly effective way is to prevent a service from the execution of any programs; consequently, the possibility to start a shellcode in a compromised application is blocked and the attacker has a limited room for manoeuvre.

2. Practical application of security solutions after application compromise

The next chapter deals with the system configuration after an attacker compromised an application. It will include a suggestion for a change from a standard DAC access control model to MAC or RBAC. It will also discuss the installation and configuration processes of adequate packages.

2.1. Preliminary system configuration

Three distibutions, CentOS, Debian and Gentoo, have been selected for the considerations on system protection against the exploitation of the known security holes in Linux systems.

For the CentOS system, system instalation from image *CentOS-7-x86_64-Minimal-1511* was taken. During the installation Default security profile was selected.

For the Debian system, the system instalation was made from image *debian-8.6.0-amd64-netinst.* On the list *Software selection* only box *Basic system tools* was selected while all other ones were unticked.

The instalation of the Gentoo system was conducted with the use of image *install-amd64-minimal-20160929* and the system image *stage3-amd64-20160929.* The system installation process will be discussed in more detail as there are various configuration possibilities in the course of the process that may have an impact on the final result of achieving a secure configuration. Let's start with the preparation of the Portage system to install the software as the initially installed system does not include a repository; thus

```
livecd gentoo_root # emerge –sync
```

should be downloaded.

Then a change of the system profile is required. A list of the available profiles can be obtained by entering:

```
livecd / # eselect profile list
Available profile symlink targets:
[1]     default/linux/amd64/13.0
[2]     default/linux/amd64/13.0/selinux
[3]     default/linux/amd64/13.0/desktop
[4]     default/linux/amd64/13.0/desktop/gnome
[5]     default/linux/amd64/13.0/desktop/gnome/systemd
[6]     default/linux/amd64/13.0/desktop/kde
[7]     default/linux/amd64/13.0/desktop/kde/systemd
[8]     default/linux/amd64/13.0/desktop/plasma
[9]     default/linux/amd64/13.0/desktop/plasma/systemd
[10]    default/linux/amd64/13.0/developer
[11]    default/linux/amd64/13.0/no-multilib *
[12]    default/linux/amd64/13.0/systemd
[13]    default/linux/amd64/13.0/x32
[14]    hardened/linux/amd64
[15]    hardened/linux/amd64/selinux
[16]    hardened/linux/amd64/no-multilib
[17]    hardened/linux/amd64/no-multilib/selinux
[18]    hardened/linux/amd64/x32
```

```
[19]     hardened/linux/musl/amd64
[20]     hardened/linux/musl/amd64/x32
[21]     default/linux/uclibc/amd64
[22]     hardened/linux/uclibc/amd64
```

```
livecd / #
```

As it can be seen, the choice is wide. For the sake of the article the selected option is 16 – hardened without the possibility to enable 32-bit applications and without SELinux whose functions will be illustrated by the CentOS distribution. The aim of the procedure is to present solutions alternative to other distributions.

```
livecd / # eselect profile set 16
livecd / #
```

The change of the profile on *hardened* introduces the modifications of the options to some programs. Firstly, the compiler has to be recompiled so that it can build the software with the enabled mechanisms that improve the security and the binutils package.

```
livecd / # emerge gcc
livecd / # emerge binutils
```

A version with patches added to incrase security will be used as the core:

```
livecd / # emerge hardened-sources
```

On the day when this text was written, the current stable version of the core was 4.4.8-r1. Let's go on to the core configuration. The core sources are installed by default in catalogue /usr/src/.

```
livecd / # cd /usr/src/linux-4.4.8-hardened-r1/
```

Now, let's discuss the configuration process; however, the focus will be directed to the configuration of security options.

```
livecd linux-4.4.8-hardened-r1 # make menuconfig
```

First, standard mechanisms will be selected that are included in the basic core version and that increase the security:

```
General setup  --->
Stack Protector buffer overflow detection (None)  --->
```

Change None to Strong.

```
[ ] Disable heap randomization
```

Make sure that the option is disabled.

```
Processor type and features  --->
[ ] Intel MPX (Memory Protection Extensions)
```

If the system is going to operate on MPX processors (at the moment of writing the article, the only MPX processors are based on Intel's Skylake architecture), the following option should be selected:

```
vsyscall table for legacy applications (Emulate)  --->
```

Change Emulate to None.

```
[*] Allow userspace to modify the LDT by default
```

This position should be disabled despite the fact that this may cause problems for such programs as Dosemu or Wine; however, it will increase significantly the system security.

The options below concern only the cores with the grsecurity patch (as the hardened-sources) and the configuration is in:

```
Security options  --->Grsecurity  --->[ ] Grsecurity
```

After selecting **[ ] Grsecurity** menu will show:

```
[*] Grsecurity
      Configuration Method (Custom)  --->
      Customize Configuration  --->
```

In order to simplify the selection of configuration options (the choice is fairly extensive), it is recommended to change **Configuration Method** from *Custom* to *Automatic.* As a result, additional options will appear:

```
[*] Grsecurity
      Configuration Method (Automatic)  --->
      Usage Type (Server)  --->
      Virtualization Type (None)  --->
      Required Priorities (Performance)  --->
      Default Special Groups  --->
      Customize Configuration  --->
      Usage Type (Server)  --->
```

The final option should be left as Server unless the system is to function as user's end system.

```
Virtualization Type (None)  --->
```

If the system is to operate in a physical facility and it is not a virtual host, this option should stay *None* .However, if it is to a be a virtualization host, it should be changed to *Host*.

In the case in question the system is the guest so the type will be changed to *Guest*. When option other than *None* is selected, the following options will appear:

```
Virtualization Hardware (EPT/RVI Processor Support)  --->
```

This one should have the default value (EPT/RVI Processor Support) if the virtualization extension is enabled in the processor where the system operates: in the case of the Intel processors – EPT (Extended Page Tables), in the case of AMD – RVI (Rapid Virtualization Indexing. The first processors that had such opportunities were Intel Nehalem and AMD Barcelona architectures. One can check whether the processor of the computer system supports these extensions by means of the command : *grep " ept\|rvi" /proc/cpuinfo*. If anything shows on the screen, it means the support for the extensions; if not, the following option should be selected:

```
First-gen/No Hardware Virtualization
Virtualization Software (Xen)  --->
```

The next additional option that will appear if we select above other setting than *None* is the selection of the virtualization software. For the needs of the article, VMWare will be chosen:

```
Required Priorities (Performance)  --->
```

The aim of this configuration is to achieve the highet possible security level of the system being installed. That is why the option *Security* rather than *Performance* should be selected. However, it should be remembered that the patch authors state that in the worst case this choice may result in the decrease of the effectiveness by as much as 20%.

Optionally, SELinux can be disabled as it will not be used in this configuration (RBAC will be used instead):

```
Security options  --->[*] NSA SELinux Support
```

Now, the core can be compiled and installed:

```
livecd linux-4.4.8-hardened-r1 # make -j8
```

If the system has the numer of threads other than 8, change make option to –jX, where X = numer of threads. After the compilation, the core and its modules should be installed:

```
livecd linux-4.4.8-hardened-r1 # make install
livecd linux-4.4.8-hardened-r1 # make modules_install
```

Subsequently, the recompilation of the glibc library is made:

```
livecd / # emerge glibc
```

and the recompilation of the whole system:

```
livecd / # emerge --update –oneshot --emptytree --deep --with-bdeps\=y --newuse --
changed-use @system @world
```

Finally, the instalation of the bootloader:

```
livecd / # emerge grub
livecd / # grub-install /dev/sda
```

and the generation of the bootloader configurartion"

```
livecd / # grub-mkconfig -o /boot/grub/grub.cfg
```

After making sure that the root password is adequate, the rebooting of the system is done:

```
livecd ~ # reboot
```

And it should be ready for further configuration procedures.

2.2. Protection against the acces to data in the case of service compromise

The presented below solutions can be treated as „measures after", i.e. the ones that will only prevent the access to data that is normally not used by the application.

2.2.1. SELinux

Security-Enhanced Linux is a Mandatory Access Control system implementation (MAC), which was initiated by NSA and now it is developed mainly by Redhat. As a result, Linux branch distributons contain the best SELinux with well defined policies to practically every package that is included in the standard repository. Thus, in order to present SELinux, CentOS will be used where it is operating by dfault and is correctly configured.

This section of the article will also generally discuss SELinux and present how it can be used. The development of policies is carried out mainly by distribution creators and the issue – due to its complexity – will be skipped.

SELinux operates mainly on contexts which are stored in security labels and on policies that define privileges between subjects labeled with such contexts. For instance, a policy allows a certain process to read and execute a file labeled with context "A" and to record and read file labeled with context "B". However, the access to all other files labeled with other contexts is denied. In order to obtain information about a context, option *–Z* should be added to command *ls* . A sample context is given below:

```
[root@CentOS7 ~]# ls -Z /bin/bash
-rwxr-xr-x. root root system_u:object_r:shell_exec_t:s0 /bin/bash
[root@CentOS7 ~]#
```

Apart from the rights to the file and the information about the owner, the following character sequence appeared: *system_u:object_r:shell_exec_t:s0* . The sequence is divided by colons into the following fields:

- ***system_u*** — defines SELinux user
- ***object_r*** — defines SELinux function
- ***shell_exec_t*** –defines SELinux type (in the case of a process, it is refered to as the process domain)
- ***s0*** — defines sensitivity

In most cases, type/domain is the most significant field as over 99% of the rules operate on the relations between types/domains, not speaking of the remaining fields.[15] In this case, the bash shellcode has a special type that is assigned to executable files. This is done on purpose in order to prevent from enabling shellcode programs when the control over program is taken over by an attacker (the aim of the attack is usually to enable the shellcode on the target system).

Apart from the contexts that are attributed to the files on the disc, there is also the user's context. In order to check a user that is currently logged in, one has to:

```
[root@CentOS7 ~]# id -Z
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
[root@CentOS7 ~]#
```

In this case the user is in the domain *unconfined* – a special domain that allows practically anything. Usually it means that SELinux has active *targeted* policy (the names may vary depending on the distribution). The aim of the policy is to protect the system against background processes (hence the name – the policy is targeted at particular programs'services). Some more information can be gained through the *sestatus* tool:

```
[root@CentOS7 ~]# sestatus
SELinux status:                 enabled
SELinuxfs mount:                /sys/fs/selinux
SELinux root directory:         /etc/selinux
Loaded policy name:             targeted
Current mode:                   enforcing
Mode from config file:          enforcing
Policy MLS status:              enabled
Policy deny_unknown status:     allowed
```

[15] S. Vermeulen, *SELinux System Administration*, Packt Publishing 2013, p. 26.

```
Max kernel policy version:      28
[root@CentOS7 ~]#
```

From the information one can learn that SELinux operates with *targeted* policy. In order to gain the information on the domains in which system processes operate, command *ps* with option *Z* can be used. To make it more visible, a different way of listing the processes is demonstrated:

```
[root@CentOS7 ~]# ps -o user,pid,ucomm,context -A | grep -v kernel_t
USER        PID COMMAND         CONTEXT
root          1 systemd         system_u:system_r:init_t:s0
root        740 systemd-journal system_u:system_r:syslogd_t:s0
root        759 lvmetad         system_u:system_r:lvm_t:s0
root        875 auditd          system_u:system_r:auditd_t:s0
root        895 systemd-logind  system_u:system_r:systemd_logind_t:s0
root        896 rsyslogd        system_u:system_r:syslogd_t:s0
root        897 NetworkManager  system_u:system_r:NetworkManager_t:s0
root        898 vmtoolsd        system_u:system_r:vmtools_t:s0
dbus        899 dbus-daemon     system_u:system_r:system_dbusd_t:s0-s0:c0.c1023
root        905 irqbalance      system_u:system_r:irqbalance_t:s0
root        929 wpa_supplicant  system_u:system_r:NetworkManager_t:s0
polkitd     930 polkitd         system_u:system_r:policykit_t:s0
root       1141 sshd            system_u:system_r:sshd_t:s0-s0:c0.c1023
root       1142 tuned           system_u:system_r:tuned_t:s0
root       1220 master          system_u:system_r:postfix_master_t:s0
postfix    1222 qmgr            system_u:system_r:postfix_qmgr_t:s0
root       1279 agetty          system_u:system_r:getty_t:s0-s0:c0.c1023
root       1703 crond           system_u:system_r:crond_t:s0-s0:c0.c1023
root       1738 systemd-udevd   system_u:system_r:udev_t:s0-s0:c0.c1023
root      12549 dhclient        system_u:system_r:dhcpc_t:s0
root      12579 sshd            unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
root      12583 bash            unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
postfix   14331 pickup          system_u:system_r:postfix_pickup_t:s0
root      14403 ps              unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
root      14404 grep            unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
[root@CentOS7 ~]#
```

In the case of all the proceses the above command shows the name of the user whose rights are assigned to the process, the process identifier and the process domain. It should be pointed out that only the user's programs operate in the unconfined domain while all the other operate in domains that are developed for them individually.

Below, practical application of SELinux is presented together with a typical problem that may be encountered.

Sharing files by http; the system includes an http server (apache in this case) and the firewall is configured adequately (accepting connections on 80/tcp port). However, the server itself has a catalog with sites (DocumentRoot) that is changed from default /var/www/html to /mywebsite (the catalog is not developed yet). The user wishes to develop an adequate catalog, place a file there and to download it on a remote computer:

```
[root@CentOS7 ~]# mkdir /mywebsite
[root@CentOS7 ~]# cd /mywebsite/
[root@CentOS7 mywebsite]# echo "test strony" > index.html
[root@CentOS7 mywebsite]# ls -al
razem 8
drwxr-xr-x.  2 root root   23 12-18 01:44 .
dr-xr-xr-x. 18 root root 4096 12-18 01:43 ..
-rw-r--r--.  1 root root   12 12-18 01:44 index.html
[root@CentOS7 mywebsite]# systemctl start httpd
```

While in the remote computer:

```
user@client /tmp $ wget http://172.21.6.10/
--2016-12-18 01:50:53--  http://172.21.6.10/
Connecting with 172.21.6.10:80... connected.
HTTP requirement sent, waiting for reply... 403 Forbidden
2016-12-18 01:50:53 ERROR 403: Forbidden.
user@client /tmp $
```

For some reason the acces to the file is denied. When analyzing the logs, the following information can be found:

```
[root@CentOS7 ~]# tail -n 1 /var/log/httpd/error_log
[Sun Dec 18 01:50:53.062055 2016] [core:error] [pid 15345] (13)Permission denied:
[client172.21.6.33:41910] AH00035: access to /index.html denied (filesystem path
'/mywebsite
/index.html') because search permissions are missing on a component of the path
[root@CentOS7 ~]#
```

However, the file and catalog rights make it possible for anyone to read the file content and to enter the catalog. In this case *audit.log will* be helpful, where the information related with SELinux will be recorded:

```
[root@CentOS7 ~]# tail -n 2 /var/log/audit/audit.log
type=AVC msg=audit(1482022253.061:259): avc:  denied  { getattr } for  pid=15345
comm="httpd" path="/mywebsite/index.html" dev="dm-0" ino=50617843
scontext=system_u:system_r:httpd_t:s0 tcontext=unconfined_u:object_r:default_t:s0
tclass=file
type=SYSCALL msg=audit(1482022253.061:259): arch=c000003e syscall=6 success=no
exit=-13 a0=7efcbcff8118 a1=7ffc88838b90 a2=7ffc88838b90 a3=1 items=0 ppid=15343
pid=15345 auid=4294967295 uid=48 gid=48 euid=48 suid=48 fsuid=48 egid=48 sgid=48
fsgid=48 tty=(none) ses=4294967295 comm="httpd" exe="/usr/sbin/httpd"
subj=system_u:system_r:httpd_t:s0 key=(null)
[root@CentOS7 ~]#
```

AVC and SYSCALL – type entries usually appear together; however AVC type is more useful as it informs that operation *getattr* was blocked (before the file is read, its attributes must be read first; thus: getattr) for the process with identifier 15345 and called httpd for path /mywebsite/index.html on facility dm-0 (hard disc), whose i-node has identifier 50617843, where the context of the process intending to get the access to the file was system_u:system_r:httpd_t:s0, the resource context was unconfined_u:object_r:default_t:s0 and the resource was a file.

When there are suspicions that it is SELinux that blocks the access, it can be changed to permissive mode, in which any blocking of the access is disabled (however, the events are still logged):

```
[root@CentOS7 ~]# setenforce 0
[root@CentOS7 ~]# getenforce
Permissive
[root@CentOS7 ~]#
```

Command getenforce can be used to check the mode of SELinux. Then, once again one can make attempt to download file:

```
user@client /tmp $ wget http://172.21.6.10/
--2016-12-18 02:26:04--  http://172.21.6.10/
Connecting with 172.21.6.10:80... connected.
HTTP requirement  sent, waiting for reply... 200 OK
Length: 12 [text/html]
Entered to: `index.html'

index.html                    100%[==================================>]      12  --
.-KB/s    in 0s

2016-12-18 02:26:04 (1,31 MB/s) - entered `index.html' [12/12]
user@client /tmp $ cat index.html
site test
user@client /tmp $
```

The cause has been found; however leaving SELinux in a permissive mode is pointless, Setting an adequate contenx is a solution to the problem but it is not always known what context is allowed. This can be done with the application of the sesearch tool (it is not installed by default in CentOS7. However it can be installed by command yum install setools-console). The required enquiry template is as follows:

```
sesearch -s <source context> -c <target object class> -p <required operation> -Ad.[16]
```

As the source context, the type from scontext from AVC logs will be used, in this case *httpd_t*. Object class will be tclass, i.e. *file*. The required operation is not *getattr* a *read* as one might expect. The following operations can be usually executed on the file: *ioctl read write create getattr setattr lock append unlink link rename open execute execute_no_trans entrypoint.*

```
root@CentOS7 ~]# sesearch -s httpd_t -c file -p read -Ad | head -n 1
Found 220 semantic av rules:
[root@CentOS7 ~]#
```

The number of suitable 220 rules does not look optimistic. However, the number can be restricted – with the use of command grep one can get rid of entries with undesired operations (such as write or execute):

```
[root@CentOS7 ~]# sesearch -s httpd_t -c file -p read -Ad | grep -v execute | grep -v write |
wc -l
91
[root@CentOS7 ~]#
```

At the same time the word *content* appears in the names, which is suitable for the needs of file hosting:

```
[root@CentOS7 ~]# sesearch -s httpd_t -c file -p read -Ad | grep -v execute | grep -v write |
grep content | grep user
  allow httpd_t httpd_user_ra_content_t : file { ioctl read getattr lock open } ;
  allow httpd_t httpd_user_content_type : file { ioctl read getattr lock open } ;
[root@CentOS7 ~]#
```

There is one more problem with the list. It presents all the contexts, and there are different contexts for a process and a file. In this case, command seinfo -afile_type –x will be helpful as it shows all contexts that can be assigned to files. Thanks to the information gained from the previous command one can filter the undesired result with the use of command grep:

```
[root@CentOS7 ~]# seinfo -afile_type -x | wc -l
2881
[root@CentOS7 ~]# seinfo -afile_type -x | grep httpd | grep content | grep user
     httpd_user_content_t
     httpd_user_rw_content_t
     httpd_user_ra_content_t
[root@CentOS7 ~]#
```

[16] https://wiki.gentoo.org/wiki/SELinux/Tutorials/How_does_a_process_get_into_a_certain_context Accessed: 18.12.2016

From the above types, the most suitable one is httpd_user_content_t. Thus, one can try and change the catalog and file contexts into the above context and then it is enough to change SELinux into enforcing mode and then try again and download the file:

```
[root@CentOS7 ~]# ls -alZ /mywebsite/
drwxr-xr-x. root root unconfined_u:object_r:default_t:s0 .
dr-xr-xr-x. root root system_u:object_r:root_t:s0      ..
-rw-r--r--. root root unconfined_u:object_r:default_t:s0 index.html
[root@CentOS7 ~]# chcon -R -t httpd_user_content_t /mywebsite/
[root@CentOS7 ~]# ls -alZ /mywebsite/
drwxr-xr-x. root root unconfined_u:object_r:httpd_user_content_t:s0 .
dr-xr-xr-x. root root system_u:object_r:root_t:s0      ..
-rw-r--r--. root root unconfined_u:object_r:httpd_user_content_t:s0 index.html
[root@CentOS7 ~]# setenforce 1
[root@CentOS7 ~]# getenforce
Enforcing
[root@CentOS7 ~]#
```

Additionally, flag should be set in SELinux to ensure the access of process *httpd* to the files with context *httpd_user_content_t*. In order to list all flags, command *getsebool –a* should be used. The list of flags that concern process httpd is long so let's move on to the most interesting ones, i.e. the ones that include entries httpd and user:

```
[root@CentOS7 ~]# getsebool -a | grep httpd | grep user
httpd_read_user_content --> off
[root@CentOS7 ~]#
```

After it is enabled with command *setsebool –P*, the problem with the access denial is solved:

```
[root@CentOS7 ~]# setsebool -P httpd_read_user_content 1
[root@CentOS7 ~]# getsebool -a | grep httpd | grep user
httpd_read_user_content --> on
[root@CentOS7 ~]#
```

Now, the changes that were introduced can be tested:

```
user@client /tmp $ wget http://172.21.6.10/
--2016-12-18 03:20:05--  http://172.21.6.10/
Łączenie się z 172.21.6.10:80... połączono.
HTTP requirement sent, waiting for reply... 200 OK
Length: 12 [text/html]
Entry to: `index.html'

index.html                    100%[=================================>]     12  --
.-KB/s    in 0s

2016-12-18 03:20:05 (1,31 MB/s) - entered `index.html' [12/12]
user@client /tmp $
```

The problem has been solved. Now all newly created files in this catalog are going to inherit the context so there is no need to assign a proper context to them.

In fact, the above presentation is only an introduction to the use of SELinux. More information can be found in Vermeulen S., SELinux System Administration, Packt Publishing 2013.

2.2.2. RBAC

RBAC (*Role-based access control*) is a subsystem in *grsecurity* which plays the role of a mandatory access control system. RBAC is much simpler to use than RBAC. It does not need files with extended attributes (xattrs) and the access list is generated automatically by a self-learning mechanism. One of the few distributions that officially support this solution is Gento; and this is the reason why the demonstration will be conducted on RBAC (SELinux is also avaialable in this system).

RBAC is disabled by default and the admin tool is not installed. Thus, one has to install:

```
gentoo ~ # emerge gradm
```

After the installation, three passwords are required to be set up:

```
gentoo ~ # gradm -P
Setting up grsecurity RBAC password
Password:
Re-enter Password:
Password written to /etc/grsec/pw.

gentoo ~ # gradm -P admin
Setting up password for role admin
Password:
Re-enter Password:
Password written to /etc/grsec/pw.

gentoo ~ # gradm -P shutdown
Setting up password for role shutdown
Password:
Re-enter Password:
Password written to /etc/grsec/pw.
gentoo ~ #
```

Now, it is possible to check whether RBAC can be enabled by command:

```
gentoo ~ # gradm -E
Warning: permission for symlink /dev/cdrom in role default, subject / does not match that of its matching target object /dev.  Symlink is specified on line 332 of /etc/grsec/policy.
gentoo ~ #
```

The above warning can be ignored as it concerns a standard policy with which *gradm* in installed and which is going to be changed. The next step is to generate a policy by self-learning. However, RBAC should be disabled first:

```
        entoo ~ # gradm -D
Password:
        gentoo ~ #
```

The password is the main RBAC password (in the demonstration it is entered as the first *Setting up grsecurity RBAC password*). At this moment, it is recommended to install the target software that is used in the system. After the installation, the registrations of events in the system can be started:

```
gentoo ~ # gradm -F -L /etc/grsec/learning.log
```

Since that moment, standard application of the system can be started but one should restrain from such admin operations as software installation and configuration. However, when necessary, one can change to admin mode by using command:

```
gentoo ~ # gradm -a admin
Password:
gentoo ~ #
```

Then, the learning proces will be stopped. Having completed admin operations, the admin mode should be left in order to continue the learning process.

```
gentoo ~ # gradm -u
gentoo ~ #
```

It should be emphasized that every activity is logged in the file system. The Authors suggest refraining from executing operations on a significant number of files (e.g. browsing the catalog with command find/). In order to stop the learning process, RBAC (*gradm –D)* can be disabled or the system should be started again. When the learning process is over, the system will include a file that is indicated in the command that enables learning, in this case it is /etc/grsec/learning.log. A fragment of such a file is as follows:

```
default 68    1000   1000   /bin/cat    /    1    1    /lib64/ld-2.22.so     16    0.0.0.0
default 68    1000   1000   /bin/cat    /    1    1    /bin/cat    8    0.0.0.0
default 68    1000   1000   /bin/cat    /    1    1    /lib64/ld-2.22.so     8     0.0.0.0
default 68    1000   1000   /bin/cat    /    1    1    /etc/ld.so.cache     16    0.0.0.0
default 68    1000   1000   /bin/cat    /    1    1    /etc/ld.so.cache     17    0.0.0.0
default 68    1000   1000   /bin/cat    /    1    1    /lib64/libc-2.22.so   16    0.0.0.0
default 68    1000   1000   /bin/cat    /    1    1    /lib64/libc-2.22.so   17    0.0.0.0
default 68    1000   1000   /bin/cat    /    1    1    /lib64/libc-2.22.so   8     0.0.0.0
default 68     1000    1000    /bin/cat        /       1       1       /usr/lib64/locale/locale-
archive        16      0.0.0.0
default 68     1000    1000    /bin/cat        /       1       1       /usr/lib64/locale/locale-
archive        17      0.0.0.0
```

The presented fragment shows the use of command *cat* and it is visible that all files have been registered that are used by the programme. The next step of RBAC configuration is the generation of policy from the collected log. To do this the following command has to be executed:

```
        gentoo ~ # gradm -F -L /etc/grsec/learning.log -O /etc/grsec/learning.roles
        Beginning full learning 1st pass...done.
Beginning full learning role reduction...done.
Beginning full learning 2nd pass...done.
Beginning full learning subject reduction for user user...done.
Beginning full learning subject reduction for user root...done.
Beginning full learning subject reduction for user sshd...done.
Beginning full learning object reduction for subject /...done.
Beginning full learning object reduction for subject /bin/ifconfig...done.
Beginning full learning object reduction for subject /bin/ps...done.
Beginning full learning object reduction for subject /usr/bin/top...done.
Beginning full learning object reduction for subject /...done.
Beginning full learning object reduction for subject /bin/bash...done.
Beginning full learning object reduction for subject /bin/dmesg...done.
Beginning full learning object reduction for subject /bin/login...done.
Beginning full learning object reduction for subject /etc/init.d...done.
Beginning full learning object reduction for subject /sbin/agetty...done.
Beginning full learning object reduction for subject /sbin/init...done.
Beginning full learning object reduction for subject /sbin/openrc...done.
Beginning full learning object reduction for subject /sbin/shutdown...done.
Beginning full learning object reduction for subject /sbin/udevd...done.
Beginning full learning object reduction for subject /usr/sbin/sshd...done.
Full learning complete.
        gentoo ~ #
```

After log transformatiom, one can see that in the course of learning three users logged in: root, user and sshd (sshd is a special user for the needs of ssh) and every execution of the program has been analyzed. The output file is policy, which should be attached to the standard one in the file */etc/grsec/policy*. This can be done by adding a generated file with policy to the existing one or to append (preferably at the end) line:

```
include </etc/grsec/learning.policy>
```

Now, the correctness of the configuraiton can be checked:

```
        gentoo ~ # gradm -C
Duplicate variable "grsec_denied" defined on line 17 of /etc/grsec/learning.policy.
```

```
        gentoo ~ #
```

A variable duplicate error appeared. To solve the problem, one should comment on or delete the section that redefines the variable (the original variable is in the file *policy*). The line number indicates the end of the section that causes the conflict. Moreover, a similar error will involve two roles – *admin* and *shutdown*- which will have to be treated in an analogous way. In the case of the file that was generated by the author, 35 lines had to be deleted. Now, the configured RBAC can be enabled:

```
        gentoo ~ # gradm -E
```

The functioninng can be tested, for instance:

```
        gentoo ~ # cat /etc/grsec/policy
cat: /etc/grsec/policy: No such file or directory
        gentoo ~ #
```

In the system logs, the following information appeared:

```
[14417.259336] grsec: From 192.168.0.10: (root:U:/) denied access to hidden file
/etc/grsec/policy by /bin/cat[cat:2982] uid/euid:0/0 gid/egid:0/0, parent /bin/bash[bash:2849]
uid/euid:0/0 gid/egid:0/0
```

It happens occasionally that the learning proces does not exactly meet the administrator's expectations. Then, the file with policy has to be modified. The general form of the entries is as follows[17]:

```
role <role name> <role flags>
<extra role attributes>
subject / <subject flags> {
            / <object rights>
            <listed paths to files/catalogs> <object rights>
            < „Linux Capabilities" rights>
            < IP rights>
}
subject <path to optional subject> <subject flags> {
            / <object rights>
            <listed paths to files/catalogs> <object rights>
            …
}
role <role 2 name> <role flags>
…
```

It is crucial that proper order should be kept: when defining a particular role (key word ***role***), all lines that follow its definition refer to it to the end of the file with policy or to a

[17] https://en.wikibooks.org/wiki/Grsecurity/The_RBAC_System - Accessed: 9.01.2017

subsequent role definition. This works analogically with subjects (key word ***subject***). A minimal role definition should include the definition of role and subject “/”. In the template it is shown by underlining. A sample minimal role definition that allows for everything:

```
role admin sA
subject / rakv {
		/rwxcdmli
}
```

Below find the list of the most commonly used flags and rights:

Role name

This may be an existing user in the system, an existing group, key word *default* or a sequence of characters that are not the name of the user in the system or the name of the group in the system – then it would be the definition ofa special role (more information below). In the course of loging in, it is checked whether there is a role with the same name as login; if there is, the user is assigned an adequate role. Otherwise, there is an attempt to match the group of the user that is logging in to the group role. If such a role is found, the user will be assigned the group role. Otherwise, the user will be assigned the *default* role. One should also pay attention to the fact that if the user role includes an entry that denies a remote use of such role (such entry is usually created when the system starts learning), the user will be assigned the *default* role!

Role flags

- ***u*** – user role. The role name should be the login of the existing user in the system. This option is used to assign rights to a particular user.
- ***g*** – group role. The role name must be the existing group in the system. The application is similar to that of the user role but it influences all the users in a given group.
- ***s*** – special role. It exists only in RBAC; one can enable it by command *gradm -p <role name>*. It is usually used to increase privileges.
- ***l*** – the role where learning is enabled. In the course of learning, the logs of the flagged roles only will be collected and the policy will be generated.
- ***A*** – admin role. For these roles the restrictions concerning library and *ptrace* mechanism download will be lifted.
- ***G* –** enables the role to use command *gradm.*
- ***N* –** disables the authentication mechanism by the transition to this role. The change is executed by command *gradm -n <role name>*
- ***P* –** subsystem PAM is used for authenticaton.

Additional role attributes

- ***role transition*** – allows for the role to transition to other special roles that were listed, e.g. *role_transitions developers Staff*
- ***role_allow_ip*** – enables the authentication to a role from a given address/subnetwork in CIDR format. Address 0.0.0.0/32 disables remote authentication; e.g. *rule_allow_ip 192.168.1.0/24*
  - ***Subject flags***
- ***a*** **–** enables the process to access /dev/grsec
- ***h*** **–** hiding the proces.
- ***v*** – the proces can see hidden processes
- ***p*** **–** the proces is protected against killing. Only the processes from the same subject and the ones with flag *k* can kill it.
- ***k*** **–** the proces can kill hidden processes
- ***r*** **–** disable restrictions related to the use of *ptrace*
- ***o*** – overwrite inherited rights

Object rights

- ***r*** **–** object can be open in the read mode
- ***w*** **–** the object can be open in write and append modes
- ***a*** **–** the object can be open in append mode which results in the fact that the data existing in the object cannot be changed (application example: logs)
- ***x*** **–** the object cannot be enabled
- ***c*** – facilitates the creation of files/catalogs
- ***d*** **–** facilitates the deletion of files/catalogs
- ***h*** **–** the object is hidden
- ***m*** **–** the object can be assigned bits setuig/setgid
- ***i*** – create an object with rights inherited from the subject from which it was executed. This concerns executable files only
- ***l*** – facilitates the creation of hardlinks
- - (blank space) entering a path to the object without rights facilitates reading the object metadata (such as volume, creation date, rights, names). It does not grant any access to the object contents.

Linux Capabilities

The access to Capabilities can be granted or blocked by appending "+" or "-" in front of the name of a given Capability. Grsecurity introduces an additional item to the list of accessible Capabilities – *CAP_ALL* – which takes into consideration all Capabilities.

For instance, if there is a need to disable all Capabilities except for the possibility to change the file/catalog owner, in the defined place one should append:

```
-CAP_ALL
+CAP_CHOWN
```

The list of all Capabilities is available in *man capabilities* or in https://en.wikibooks.org/wiki/Grsecurity/Appendix/Capability_Names_and_Descriptions.

IP rights

There are two types of rights that are related to network connections: outgoing connection (key word ***connect***) and incoming connections (key word ***bind***). The template of such definitios is as follows:

```
connect/bind <!> <IP/host name>/<mask>:<port/port range> <types of network nests>
<protocols>
```

or

```
connect/bind disabled
```

The types of network nests are: *ip, dgram, stream, raw_sock, rdm, any_sock*. The protocols are as in file /etc/protocols or key word *any_proto* can be used. The required fields are only *<IP/host name>* and when the address is not necessary, 0.0.0.0. should be entered. One can also enter the name of the network interface to this field. It is also acceptable to specify vlan as *<interface name>#<vlan number>*

```
bind eth0#2:22 stream tcp
```

The subject may listen on vlan number 2 on interface eth0 on port 22 and answer tcp connections.

```
connect 0.0.0.0/0 1234-4321 dgram udp
```

The subject may send udp packages to any hosts on target ports from 1234 to 4321.

```
bind ! 12.34.0.0/16:80 stream tcp
```

The subject cannot receive tcp connections from subnetwork 12.34.0.0/16 on port 80.

Find below a sample definition of the role of a *user* with a learned behavior for program dmesg:

```
role user uG
role_transitions admin shutdown
role_allow_ip   192.168.0.0/16
role_allow_ip   172.16.0.0/12
role_allow_ip   10.0.0.0/8
subject / {
        /               r
        /opt            rx
        /home/user      rwxcd
        /mnt            rw
        /dev
        /dev/urandom    r
        /dev/random     r
        /dev/zero       rw
        /dev/input      rw
        /dev/psaux      rw
        /dev/null       rw
        /dev/tty?       Rw
        /dev/console    rw
        /dev/tty        rw
        /dev/pts        rw
        /dev/ptmx       rw
        /dev/mixer      rw
        /dev/cdrom      r
        /bin            rx
        /sbin           rx
        /lib            rx
        /lib32          rx
        /lib64          rx
        /usr            rx
        /usr/src        h
        /etc            rx
        /proc           rwx
        /proc/sys       r
        /sys            h
        /root           h
        /run            r
        /tmp            rwcd
        /var            rwxcd
        /var/tmp        rwcd
        /var/log

        -CAP_ALL
        +CAP_KILL
        +CAP_NET_RAW
        +CAP_NET_ADMIN
        +CAP_NET_RAW

        bind disabled
}
subject /bin/dmesg o {
        /                           h
        /bin                        h
        /bin/dmesg                  x
        /dev                        h
        /dev/kmsg                   r
        /etc                        h
        /etc/ld.so.cache            r
        /lib64                      h
        /lib64/ld-2.22.so           x
        /lib64/libc-2.22.so         rx
        /usr                        h
        /usr/lib64/gconv/gconv-modules.cache    r
        /usr/lib64/locale/locale-archive        r
        -CAP_ALL
        +CAP_SYSLOG
        bind    disabled
        connect disabled
}
```

The best final result is going to be reached when manual edition is combined with the result of the learning proces.

3. Conclusions

The article presents the sources of the dangers to the system including the ones that result from gaining the access to the system by unauthorized subjects. Such access may lead to the loss of data or reputation. Thus, they may result directly in economic losses or the loss of prestige which consequently may cause financial losses.

The Authors presented several basic and most common existing threats. The protection against them is not discussed here; the article presents ways to limit the effects of a successful attack. In such cases MAC in SELinux (by default in CentOS, Fedora or RedHat Enterprise Linux) or RBAC with *grsecurity* patch may disable unauthorized access to resources to which the access of the application was not necessary for it to operate correctly.

3.1. Bibliography

1. Foster J. C., *Buffer Overflow Attacks: Detect, Exploit, Prevent*, Syngress Publishing, Rockland 2005
2. Frisch E., *Unix – Administracja system – drugie wydanie*, Oficyna Wydawncza READ ME, Łódź 1997
3. McConnell S., *Code complete, 2nd edition*, Microsoft Press, Redmond 2004
4. Silberschatz A., Galvin P.B., *Podstawy systemów operacyjnych*, Wydawnictwo Naukowo-Techniczne, Warszawa 2002
5. Vermeulen S., *SELinux System Administration*, Packt Publishing 2013

3.2. Netography

1) https://www.theguardian.com/technology/2015/aug/10/carphone-warehouse-uk-data-watchdog-investigating-customer-hack - Accessed: 6.11.2016
2) https://gist.github.com/epixoip/a83d38f412b4737e99bbef804a270c40 - Accessed: 6.11.2016
3) http://vms.drweb-av.pl/virus/?_is=1&i=4323517 – Accessed: 6.11.2016
4) https://www.aldeid.com/wiki/Exploits/proftpd-1.3.3c-backdoor – Accessed: 6.11.2016
5) https://access.redhat.com/blogs/766093/posts/1975883 – Accessed: 6.11.2016
6) https://www.openhub.net/p/apache - Accessed: 6.11.2016
7) https://wiki.gentoo.org/wiki/SELinux/Tutorials/How_does_a_process_get_into_a_certain_context - Accessed: 18.12.2016
8) J. McDonald, *Defeating Solaris/SPARC non-executable stack protection*. Bugtraq, Mar. 1999, Online: http://seclists.org/bugtraq/1999/Mar/4.
9) M. Ivaldi, *Re: Older SPARC return-into-libc exploits. Penetration Testing*, Aug. 2007, Online:http://seclists.org/pen-test/2007/Aug/214
10) https://en.wikibooks.org/wiki/Grsecurity/The_RBAC_System

Abstract

The aim of the article is to present the most common holes in the Linux operating system security and the proposals of security solutions to protect computer systems against entering other resources than the compromised service.