DOI: 10.2507/33rd.daaam.proceedings.xxx

LINUX SECURITY IN PHYSICAL, VIRTUAL, AND CLOUD ENVIRONMENTS

Vedran Dakic, Karlo Jakobovic & Luka Zgrablic

This Publication has to be referred to as: Dakic, V[edran]; Jakobovic, K[arlo] & Zgrablic, L[uka] (2022). Linux security in physical, virtual and cloud environments, Proceedings of the 33rd DAAAM International Symposium, pp.xxxx-xxxx, B. Katalinic (Ed.), Published by DAAAM International, ISBN 978-3-902734-xx-x, ISSN 1726-9679, Vienna, Austria
DOI: 10.2507/33rd.daaam.proceedings.xxx

Abstract

This paper will cover some of the available Linux methodologies and tools that can be used to enhance Linux security. It gives the reader some orientation in the world of Linux security. We will define what are the components of security and risk management, followed by device encryption and decryption using LUKS (Linux Unified Key Setup) and NBDE (Network Bound Disk Encryption) and restricting USB access by using USBGuard. The paper then shifts its focus to authentication methods utilizing PAM (Pluggable Authentication Modules), followed by firewalld, system auditing and monitoring file system changes with Audit and AIDE. We also go through ACL and SELinux concepts, ending with resource usage management with cgroups and OpenSCAP for compliance management. In the last part, we talk about an overall approach to putting all of these principles into operation, significantly impacting security footprint of our Linux-based resources. We do all this with the purpose being to have all of the necessary details for top-to-bottom security approach, as well to discuss future frameworks and standards development.

Keywords: NBDE; PAM; SELinux; OpenSCAP; ACL

1. Introduction

In this day and age, security is more important than ever before. By having in mind that most of the servers used in production are Linux/Unix[1], their security becomes the imperative of OS security[2]. This paper will provide a comprehensive overview of the necessary security components of Linux OS hardening. First, we will dive into the definitions of risk and security with their respective components. After explaining these concepts, we will transition to device encryption and decryption by using LUKS and NBDE. Then, we will discuss restricting USB access to Linux/Unix servers by using USBGuard. Authentication methods for users and applications are explained by using the Linux native PAM system. When it comes to auditing and monitoring changes, Audit toolkit for system auditing and AIDE for tracking file system changes are the norm. With that, let's start our discussion with core concepts – security and risk management.

2. Managing security and risk

A risk is any event that presents the possibility of loss, whether it be financial or productivity loss or delays to services. Continuous risk management is a process of taking a proactive approach to discovering potential risks, assessing facts, and taking action based on the facts to resolve those risks. Monitoring risk indicators and the actions taken to resolve them is important to succeed in risk management. Communicating to stakeholders what those risk indicators are, the action taken to resolve them, and the results, all promote confidence and stability in the process.

The following diagram illustrates a process for continuously managing security risks by maintaining constant attention to potential security vulnerabilities and taking a proactive approach to maintaining a secure computing environment:

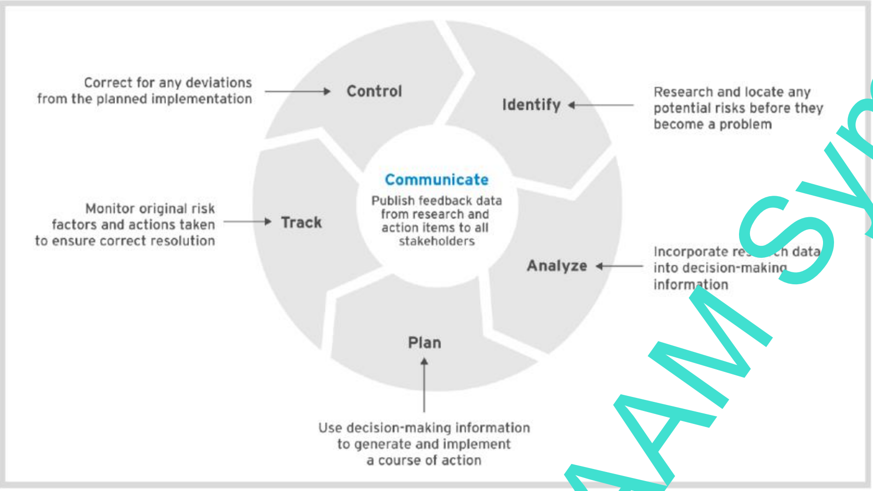
Correct for any deviations from the planned implementation
Control
Identify
Research and locate any potential risks before they become a problem
Communicate
Publish feedback data from research and action items to all stakeholders
Monitor original risk factors and actions taken to ensure correct resolution
Track
Analyze
Incorporate research data into decision-making information
Plan
Use decision-making information to generate and implement a course of action

Fig. 1. Continuous risk management life cycle [3]

Managing security in our computing environments is a collection of continuous activities. Red Hat offers solutions that enable security in each phase of the continuous security life cycle. Security must be both proactive and reactive. It must be considered at every stage of our application and infrastructure life cycle. To do this effectively, we need to integrate security experts into our application, deployment, and infrastructure teams. The following diagram illustrates a continuous security life cycle that incorporates the risk management life cycle:

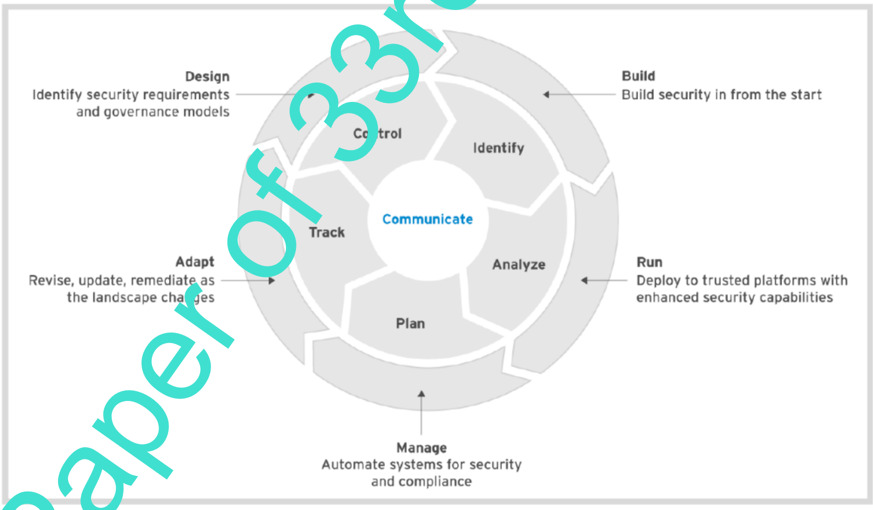
Design
Identify security requirements and governance models
Build
Build security in from the start
Control
Identify
Communicate
Track
Analyze
Plan
Adapt
Revise, update, remediate as the landscape changes
Run
Deploy to trusted platforms with enhanced security capabilities
Manage
Automate systems for security and compliance

Fig. 2. Continuous security and risk management life cycle [3]

When using Linux, that will surely mean using a layered approach, as there are many technologies involved, and none of them is top-to-bottom. Let's discuss some of them.

3. Protecting data with LUKS and NBDE

The risk of a system's physical compromise puts sensitive data in its storage devices at risk of compromise. This is particularly true for mobile systems such as laptops or removable media, but servers may also have security requirements that the data they store is protected at rest. Encrypting this data helps mitigate the risk of its exposure if the system is lost. Red Hat Enterprise Linux supports block device encryption with Linux Unified Key Setup (LUKS) technology. Encrypting a block device [4] (such as a disk partition or LVM physical volume) is easiest at installation time, but LUKS can also be configured after installation. However, setting up LUKS encryption requires that the file systems on the device be reformatted [5]. We can use the *cryptsetup luksDump* command post-installation to verify the encryption information for an encrypted device. This command displays information such as the LUKS header information[6], and the key slots (each of which may contain a valid passphrase) in use by the LUKS-encrypted device. The default encryption cipher is *aes-xts-plain64* [7].

One of the pitfalls of LUKS encryption is that we have to enter a passphrase manually to decrypt a disk at boot time. With a large number of servers, this requirement becomes a problem. Network-bound Disk Encryption (NBDE) automates the decryption of those encrypted disks without manually entering any passphrase at boot time in a secure way, by ensuring certain criteria are met. We can use NBDE to automatically decrypt LUKS storage devices containing root and non-root file systems. We must edit the */etc/crypttab* file to specify the encrypted block devices to open (decrypted and mapped) at boot time. This file contains an entry for each encrypted block device. We can use */etc/crypttab* on its own to configure LUKS to decrypt devices at boot time, but this requires either those passwords to be entered manually on the console or that they are stored in plain text on a decrypted device on the system. This is inadequate for use cases where servers need to boot unattended and still keep the decryption password secure [8].

NBDE provides a mechanism to automatically decrypt LUKS devices at boot time in a secure manner. This mechanism is based on two key components. Clevis runs on the system and decrypts devices and defines the policy whose conditions must be met for the data to be decrypted. Tang [9] is a very simple server that Clevis clients contact to determine if they are booting with access to a secure network and therefore decrypt their LUKS devices. We can use this to stop the automatic decryption of a device that has been removed from the data center [10].

The architecture of NBDE uses two key components: the Clevis framework, and the Tang server. The Clevis framework is a pluggable framework that supports NBDE on the client side and automates the unlocking of LUKS-encrypted block devices. The Tang server supports NBDE on the server side and makes data available on a system when the system can reach the Tang server over a specific secure network. The Clevis framework supports plug-ins to interact with Tang servers (PINs). Both the Clevis framework and the Tang server use the JOSE framework as their back end for encryption and decryption.

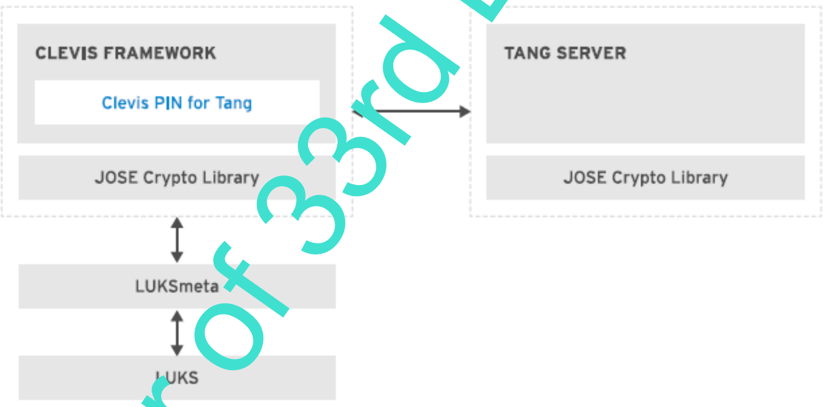

Fig. 3. NBDE architecture with Clevis and Tang [8]

Clevis gets a list of the keys for each Tang server and uses one of those keys to generate an encryption key. The Clevis PIN then uses the encryption key to encrypt the data. The encryption generates some state information, which is stored in the LUKS header. That state information is accessible with the *luksmeta show* command [11].

4. Restricting USB device access

USBGuard is a software framework that protects our systems against rogue USB devices by implementing basic whitelisting and blacklisting capabilities based on device attributes. The Linux kernel's USB device authorization system is used to enforce the settings configured by USBGuard. This allows us to control access to USB devices. For example, we can define what kind of USB devices are authorized and how a USB device may interact with our system. The USBGuard framework provides the following components:

- The daemon component has an inter-process communication (IPC) interface for dynamic interaction and policy enforcement
- The command-line interface used to interact with a running USBGuard instance

- The rule language used to write USB device authorization policies
- The C++ API used to interact with the daemon component, implemented in a shared library

The *usbguard* daemon determines whether or not to authorize a USB device based on a policy defined by a set of rules[12]. When a USB device is inserted into the system, the daemon scans the existing rules sequentially. When a matching rule is found[13], it either authorizes (allows), does not authorize (blocks), or removes (rejects) the device, based on the rule target. If no matching rule is found, the decision is based on an implicit default target. This implicit default is to block the device until a decision is made by the user [14].

5. Controlling authentication with Pluggable Authentication Modules (PAM)

The Pluggable Authentication Modules system, or PAM, provides a generic way for applications to implement support for authentication and authorization. Initially, an application that wanted to authenticate users used the local */etc/passwd* and */etc/shadow* files. Eventually, other authentication mechanisms were invented. For example, our company might decide to implement Kerberos, in which case the application must verify the correctness of the passwords against the KDC (*Key Distribution Center*). Alternatively, we may choose to deploy an LDAP (*Lightweight Directory Access Protocol*) infrastructure, and this time the application must contact the LDAP server for authentication. This means that every time a new authentication method came up, applications had to be rewritten to support it. Each application was handled separately, there was a lot of duplication of effort, and many different implementations had to be audited for correctness.

PAM provides a generic way for applications to implement support for authentication and authorization. A PAM-enabled application calls the PAM library, *libpam*, to perform all authentication tasks on its behalf and return a pass or fail response to the application. The PAM modules implement different authentication methods [15]. For example, the *pam_krb5* module performs authentication against Kerberos. The *pam_ldap* module authenticates against an LDAP server. The *pam_unix* module uses the standard local files, */etc/passwd* and */etc/shadow*. The PAM-enabled applications can directly use these authentication methods without having to be modified or recompiled [16].

The majority of the PAM configuration files are in */etc/pam.d/*, although some modules use additional configuration files in */etc/security*. PAM stores its binary modules responsible for evaluating criteria in the */usr/lib64/security* directory. Each PAM-enabled application has its own PAM configuration file in */etc/pam.d/*, and generally this file has the same name as the application: */etc/pam.d/login* for the login program, or */etc/pam.d/sshd* for the SSH daemon etc. If the service file for an application is missing, PAM uses */etc/pam.d/other* by default [17].

These configuration files contain rules that specify the modules to call for authentication and authorization. This way administrators can configure different authentication methods per application. In practice, we usually want all of our applications to use the same authentication method. To achieve that, most of the configuration files use the include directives to include the *password-auth* or *system-auth* files, also in the */etc/pam.d/* directory.

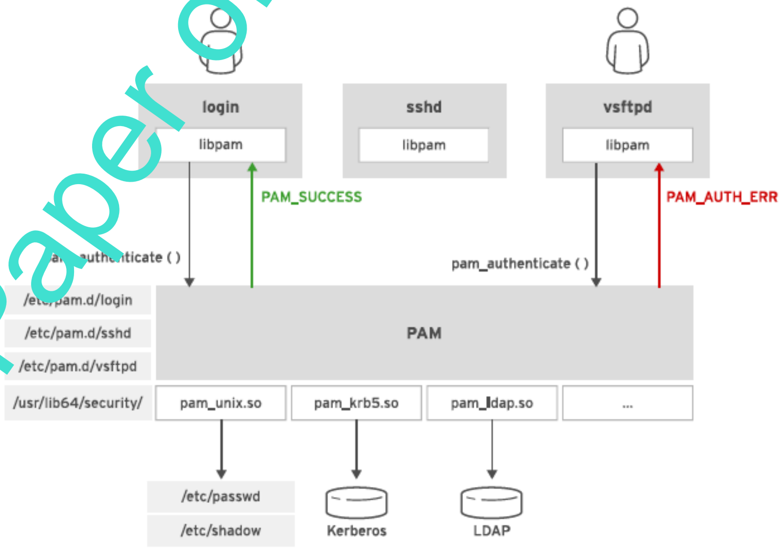

Fig. 4. PAM authentication [18]

In this figure, the login program contacts PAM for authentication. PAM reads the */etc/pam.d/login* configuration file to retrieve the list of modules to use for authentication. PAM calls the modules, stored in /usr/lib64/security/, and these modules perform the authentication. PAM returns to the login program a successful authentication status. The *vsftpd* FTP server also uses PAM to authenticate FTP users, but in the figure, the user provides incorrect credentials, and PAM returns a failure code to vsftpd [19].

The *System Security Services Daemon* (SSSD), introduced in Red Hat Enterprise Linux 6.0 allows user authentication against remote directories and authentication services such as LDAP, Kerberos, Active Directory, and Red Hat Identity Management. When a remote user authenticates on the local system, SSSD stores their credentials and authentication parameters in a local cache. This way, even if the remote authentication server is not reachable, the remote user can still log in to the local system. This feature is particularly useful for laptops disconnected from the company network but also helps reduce the load on remote authentication services [20].

6. Managing network security with firewall (firewalld, iptables, ufw, …)

Firewalls are an essential part of modern system security and Linux systems are no different. *Firewalld* is the default firewall in modern Red Hat-based Linux distributions. It allows users to shape the network traffic using the concept of zones and services. Zones are a set of rules that apply to a certain network interface [21]. For example, some of the default firewalld zones are trusted and external. As the names may suggest, trusted zone has a default set of rules that allow almost all traffic, while external zone won't allow any external traffic. Zones can be applied to an interface using firewall-cmd command.

Services are a set of rules that can be applied to a zone that would allow a certain network service to function properly. For example, service ssh would allow traffic on port 22 once applied to a zone. Their purpose is to simplify firewall configuration for the end user. Rather than manually allowing and configuring each port needed for a given network service, the concept of firewalld service allows the user to configure it using a single command [21]. Services can be configured for a given zone using the firewall-cmd command. Firewalld architecture is described in the following figure:

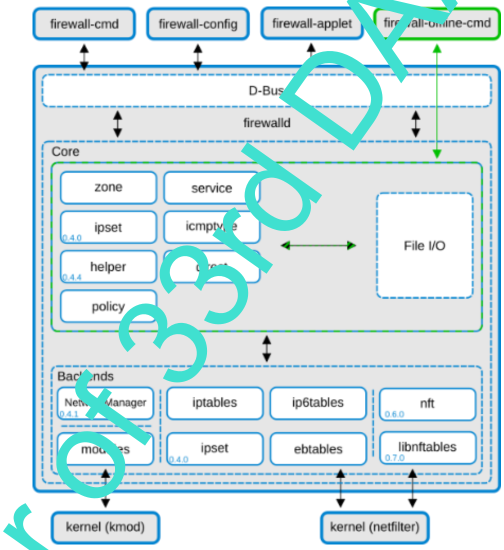

Fig. 5. Firewalld architecture [21]

Firewalld management is done using *firewall-cmd* command and the graphical counterpart *firewall-config*, which support a variety of options and configure the firewall while it's running, meaning there is no need to restart the machine or the service for the new rules to take effect. Rather than being a standalone firewall, firewalld is a frontend management tool for *nftables*, an IP packet filtering framework. Nftables was designed as a replacement for the widely used *iptables*. While their command and rule syntax differ, both use netfilter, the packet filtering component of the Linux kernel. Nftables provides a userspace configuration tool named *nft*. This tool is used by firewalld to configure nftables behind the scenes. All of the configuration done via firewalld is translated into nftables commands [22].

7. Recording system events with audit

The Linux Audit system is a mechanism in the kernel that provides a way to track security-related events and information on our systems. We define a set of rules that we load into the kernel to specify which events it should record in the audit log. The *auditd* system daemon writes the logged events to the local disk or forwards them to remote log servers. We can then use this information to discover and investigate the cause of violations or attempted violations of the system's security policy [23].

We can configure Audit to include or exclude events based on user identity, security contexts, or other labels. Some of the information that the Audit system can collect about events includes:

- Date and time, type, and outcome of an event
- Sensitivity labels of subjects and objects
- Association of an event with the identity of the user who triggered the event
- All modifications to the Audit configuration and attempts to access Audit log files
- All uses of authentication mechanisms, such as SSH, Kerberos, and others
- Changes to any trusted database, such as /etc/passwd
- Attempts to import or export information into or out of the system

We may need to use Audit system to meet the compliance requirements of particular security certifications. Audit has been evaluated by National Information Assurance Partnership (NIAP) and Best Security Industries (BSI). It has been certified to LSPP/CAPP/RSBAC/EAL4+ on RHEL5 and OSPP/EAL4+ on RHEL6. Audit is designed to meet or exceed the requirements of the CAPP, LSPP, RSBAC, NISPOM, FISMA, PCI-DSS, and STIG certifications or compliance guides [24].

Auditd is the user-space component of the Linux auditing subsystem. The system unit file that starts auditd normally loads any persistent audit rules when the daemon is started. After audit rules have been loaded into the kernel, the kernel uses them to record events of interest. The kernel sends information about those events to auditd. The main role of auditd is to collect the audit event messages recorded by the kernel and save them to a log file. When auditd is running, the log file configured for auditd (usually */var/log/audit/audit.log*) collects the audit messages sent by the kernel. If auditd is not running for any reason, *rsyslog* receives the kernel audit messages [25].

Without any extra configuration, only a limited number of messages pass through the audit system, mainly authentication and authorization messages (users logging in, sudo being used, etc.) and SELinux messages. Administrators can add audit rules that control the audit system, watch files, or record information about any system call with the *auditctl* command [26].

8. Monitoring filesystem changes with Advanced Intrusion Detection Environment (AIDE)

On an operational server, it is normal for files to be added, removed, and modified on its file systems. However, unexpected changes to certain files, such as executable programs and configuration files, might indicate unauthorized modifications or other security issues. Therefore, it is important to monitor those files for changes to their content, permissions, or other characteristics. Red Hat Enterprise Linux provides Advanced Intrusion Detection Environment (AIDE), a user-space utility that can help with this monitoring. AIDE monitors files for a variety of changes including permission or ownership changes, timestamp changes (modification or access time stamps), or content changes.

The configuration file for AIDE is */etc/aide.conf*. This file controls which files AIDE monitors for changes, and what characteristics are monitored for each file. For example, by default AIDE monitors most files in the */etc* directory for permission changes only, but specific files are monitored more closely. As the security administrator, we might want to tune exactly what AIDE monitors for different parts of our computer's file system. Each line in the */etc/aide.conf* file is a directive. The file contains three types of lines: configuration lines, selection lines, and macro lines. Any line that starts with a hash (#) is a comment and has no effect [27].

The configuration lines adjust the configuration parameters of AIDE. These lines either tune the functional behavior of AIDE globally or set a group definition [28]. Group definitions are used by selection lines to specify what characteristics of a file AIDE should monitor when detecting file system changes [29]. The selection lines specify the files and directories that AIDE monitors, and the changes for which AIDE will watch. Selection lines can be regular, equals, or negative. A regular selection line is a regular expression matching the absolute path to a file or directory, followed by the name of a group definition. Files and directories matching the regular expression are added to the AIDE database, with checks performed as specified by the line's group definition. This effectively means that if the regular expression is /etc, then the regex will also recursively match all files and directories in the /etc directory. An equals selection line starts with an equals (=) sign followed by a regex and a group definition. AIDE records the files that match the regular expression, considering all the checks the line's group definition mentions. However, an equals selection line will only match the children of directories if the regex ends with a forward slash (/) character. The children of subdirectories will not be recursively matched. A negative selection line starts with a bang (!) character followed by a regular expression matching the absolute path to a file or directory. AIDE does not monitor files or directories that match a negative selection line. The macro lines set or clear variables that are useful for referring to lengthy URLs or file system paths in multiple occurrences throughout the AIDE configuration file [30].

9. Mitigating risks with Access Control Lists (ACL) and Security Enhanced Linux (SELinux)

Every file and directory on a Linux/Unix system has a set of permissions targeted at three categories of users: the user who owns the file/directory, the group of users who own the file/directory, and all other users on the system. The types of access that can be defined using these permissions are: read permission, write permission and execute permission. While this offers some amount of control over which user can access which object in the file system, it does not offer specific control over user permissions because it's limited to defining permissions for one user, one group, and all other users [31].

Access Control Lists (ACLs) are used to control user permissions in a more granular way. Using ACLs, each file or directory can have an unlimited number of permissions set on it, and each user on the system can have his own set of permission on a given file or directory. This allows users and administrators to strictly define user permissions, allowing for a greater level of security on the system and preventing unwanted access to specified filesystem objects [32].

Unlike regular filesystem permissions which are managed using ls, chown, and chmod commands, ACLs are managed using a different pair of commands: *getfacl* and *setfacl*. As their names imply, *getfacl* is used to read permissions currently set on a filesystem object, while *setfacl* is used to modify these permissions. For example, to give a user named "testuser" read and write permissions on the "*/testdir*" directory, "*setfacl -m testuser:rw- /testdir*" command would be used.

Security Enhanced Linux (SELinux) is an additional layer of system security. The primary goal of SELinux is to protect user data from system services that have been compromised. SELinux confines programs, mainly system services to the minimum amount of privilege they require to do their jobs successfully. For that purpose, a set of security rules determine which processes can access which files, directories, and ports. If there is no explicit rule for access, SELinux denies the operation. Every file, process, directory, and port has a particular security label called the SELinux context. A context is used by SELinux rules to control access. For files and directories, SELinux stores the contexts as file system extended attributes. For processes and ports, the kernel maintains the contexts in memory. Administrators can view the context of processes and files by adding the -Z option to the ps or ls commands. For ports, use the semanage port -l command.

In Red Hat Enterprise Linux, SELinux bases most of its rules on the third element: the context type. The context type for the web server is *httpd_t*. Context types for processes are also known as domains. The context type for files and directories in */var/www/html* is *httpd_sys_content_t*. The context type for web server ports is *http_port_t*.

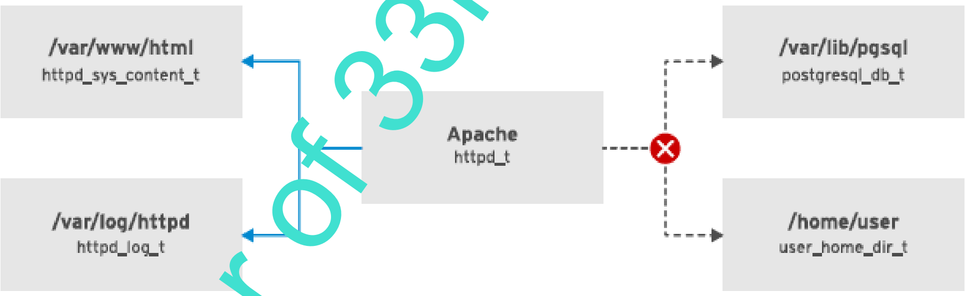

Fig. 6. Apache service with SELinux enabled [18]

The previous figure shows that an SELinux rule allows Apache, whose domain is *httpd_t*, to access files and directories with the *httpd_sys_content_t* context type. Another rule allows Apache to access its log files under /var/log/httpd. The rule allows the *httpd_t* domain to access files and directories with the *httpd_log_t* type.

There is no rule to allow the httpd_t domain to access files and directories with the user_home_dir_t type. Access is, therefore, denied [33]. If a malicious user manages to exploit a security flaw in Apache, they cannot access the contents of user home directories, even if the Linux access rights permit the access. This way, SELinux mitigates the risk while the security flaw is fixed [34].

10. Restricting resource usage via Control Groups (cgroups)

Control groups (*cgroups*) are a Linux kernel feature that allows users to group a set of processes on the system and allocate a certain amount of resources to them. Resources that can be allocated include CPU time, memory, network bandwidth, disk I/O etc. These groups are ordered hierarchically, meaning one group can be a subgroup of an already existing group, creating a tree. However, unlike with the regular Linux process model, in which only a single process tree exists, starting with the init process, multiple cgroup trees can exist. These cgroups are created and managed in the

/sys/fs/cgroup/ directory. Cgroups are a vital component of containers, which use cgroups in combination with Linux namespaces as a basis of their architecture [35].

Cgroup resources are managed by a kernel component named resource controller (also named control group subsystem). These resource controllers represent a resource in question. For example, a resource controller named memory takes care of memory allocation for a given cgroup, while blkio controller takes care of I/O access [36]. Cgroups can be created and managed by manually creating and editing files and directories in the aforementioned /sys/fs/cgroup/ directory.

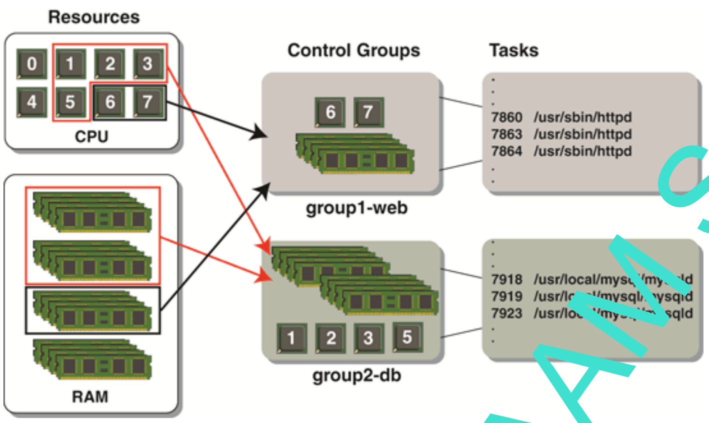

Fig. 7. Control group relationship with resources [37]

However, *systemd* can be used to create and manage cgroups in modern Linux systems. By default, every systemd service, scope, and slice (systemd units) map to objects in a cgroup tree, and new units can be created on the fly. Service units contain one or more processes that are started and stopped at the same time. These units are defined using a unit configuration file. In comparison, scope units do not create their processes, but manage those created externally. The third type of unit, slice, does not contain any processes on its own but is instead used to group services and scopes hierarchically [38].

To create a systemd unit, *systemd-run --unit=unit_name* command can be used to start command under a unit. This also creates a cgroup and starts that command within it. Furthermore, systemd provides a way to manage units/cgroups resources on the fly, using the systemctl command. For example, command *systemctl set-property unitname MemoryLimit=500M* sets a memory limit of 500 MiB for a unit named "*unitname*". These changes are persistent after reboots unless they're run with the --*runtime* parameter.

11. Managing compliance with OpenSCAP

Enterprise computing environments may consist of hundreds or thousands of interconnected computer systems, running numerous applications and services, and accessed by a large and diverse set of users and applications. To maintain control over the security of this vast environment, a standard way to scan systems for compliance with security policies is needed. The National Institute of Standards and Technology (NIST), in conjunction with other authorities, developed a standard compliance system called Security Content Automation Protocol (SCAP).

SCAP [39] can be described as a framework of security specifications. It supports automated configuration, vulnerability, and patch-checking measurement. OpenSCAP is an open-source project that develops tools for implementing and enforcing security policies using the SCAP standard. We need to perform a compliance audit to verify that a given object follows a rule in a compliance policy. Usually, the compliance policy varies across organizations. The needs and risk profiles of each organization are different and often require the ability to customize the compliance policy checklist. The OpenSCAP project provides several predefined and customizable compliance policies in SCAP format for use with OpenSCAP tools [40].

The SCAP Security Guide is a collection of security policies for Linux systems, in the form of SCAP documents. It consists of rules with detailed descriptions and proven remediation scripts and Ansible playbooks. The SCAP Security Guide can be used with OpenSCAP tools to automate the auditing of a Linux system. For ease of use, all the available security policies are broken into profiles. A profile can be defined as a grouping of security settings that correlate to a known policy. The SCAP Security Guide provides profiles for verifying system compliance against standards established by governmental or other organizations. For example, the PCI-DSS profile tests compliance against the rules mandated by the Payment Card Industry Security Standards Council.

However, in the real-world situations vary. Most profiles in the SCAP Security Guide are meant as a catalogue, not a checklist, and satisfaction of every item may not be sensible or even possible in many operational scenarios. Our organization, auditors, or other stakeholders may not require our systems to comply with every item specified in a particular profile, but instead, require our systems to comply with a specific subset of the profile. Some rules may not apply in our environment, and others, not in the profile, may be required. With SCAP Workbench, we can create custom profiles. As a starting point, we select an existing profile that we adjust by selecting and clearing compliance rules. We then save our new profile in an XML tailoring file that we can copy to the systems to scan [41].

Having this in mind, we can see that SCAP is a tool that can be used for a variety of different scenarios – albeit at the expense of time. But that also leads us to a larger discussion that needs to be further researched in the future - for example, problems with the correct detection of the identity of someone who's attacking our infrastructure [42]. That's also a chance to expand our research into different frameworks, like GDPR. Although it could be improved [43], it's still a valid framework for working with private data, and all of these technologies that we mentioned in this paper are underlying to that in a technical sense. That also means introducing new frameworks to govern security infrastructure, and that can also lead to some changes in the way we approach network infrastructure design [44]. Putting all of this together will be an area of intensive research in the future, and, as time goes on, we expect new methodologies and standards to be developed.

12. Conclusion

We have shown that managing security in our computing environments is a collection of continuous activities and layering different technologies. LUKS and NBDE provide us with a solid framework for managing encryption and decryption of our block devices, paired with USBGuard for defending USB access to our systems. We are provided a generic authentication mechanism with PAM, it enables us to authenticate users and applications on our systems. With Audit, we can discover and investigate the cause of violations or attempted violations of the system's security policy. On an operating server, it is normal for files to be added, removed, and modified on its file systems. However, unexpected changes to certain files, such as executable programs and configuration files, might indicate unauthorized modifications or other security issues. Therefore, it is important to monitor those files for changes to their content, permissions, or other characteristics; AIDE can help us with these operations. In addition to standard POSIX permissions, we have introduced SELinux which confines programs, mainly system services to the minimum amount of privilege they require to do their jobs successfully. Lastly, we discussed OpenSCAP, the standardized compliance system. With OpenSCAP we can benchmark our system against predefined security policies and guidelines and then remediate the non-compliant systems. These tools and their overall methodology need to be a part of larger discussion and research that could lead to some new standard or framework being adopter for a more global approach to Linux security management. This is also going to be our future research topic.

13. References

[1] https://w3techs.com/technologies/details/os-unix, (2018). Q-Success Web-based Services, Accessed on: 2022-07-01

[2] https://w3techs.com/technologies/overview/operating_system, (2018). Q-Success Web-based Services, Accessed on: 2022-07-01

[3] https://docs.fedoraproject.org/en-US/quick-docs/getting-started-with-selinux, (2019). RedHat, Accessed on: 2022-08-15

[4] https://www.redhat.com/sysadmin/disk-encryption-luks, (2019). RedHat, Accessed on: 2022-07-01

[5] https://en.wikipedia.org/wiki/Linux_Unified_Key_Setup, (2022). Wikipedia, Accessed on: 2022-07-01

[6] Danut, A. & Simion E. (2018). Linux Unified Key Setup (LUKS) – The Good, the Bad, the Ugly, Proceedings of ECAI 2018 International Conference, Lasi, Romania, ISBN: 978-1-5386-4901-5, doi: 10.1109/ECAI.2018.8678978

[7] https://www.thegeeksearch.com/beginners-guide-to-luks-disk-encryption-in-linux, (2016). The Geek Search, Accessed on: 2022-07-02

[8] https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/security_hardening/configuring-automated-unlocking-of-encrypted-volumes-using-policy-based-decryption_security-hardening, (2019). RedHat, Accessed on: 2022-07-02

[9] https://www.redhat.com/en/blog/network-bound-disk-encryption-improvements-rhel-8, (2021). RedHat, Accessed on: 2022-07-02

[10] https://docs.openshift.com/container-platform/4.9/security/network_bound_disk_encryption/nbde-about-disk-encryption-technology.html, (2022). RedHat, Accessed on: 2022-07-02

[11] https://www.cybersecurity-insiders.com/network-bound-disk-encryption-in-red-hat-linux-7-2, (2017). Cybersecurity Insiders, Accessed on: 2022-07-02

[12] https://www.redhat.com/en/blog/built-protection-against-usb-security-attacks-usbguard, (2017). RedHat, Accessed on: 2022-07-01

[13] https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/security_guide/sec-using-usbguard, (2015). RedHat, Accessed on: 2022-07-01

[14] https://www.systutorials.com/docs/linux/man/5-usbguard-rules.conf, (2017). Systutorials, Accessed on: 2022-07-1

[15] https://www.redhat.com/sysadmin/pluggable-authentication-modules-pam, (2020). RedHat, Accessed on: 2022-07-02

[16] https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/managing_smart_cards/pluggable_authentication_modules, (2012). RedHat, Accessed on: 2022-07-03

[17] https://www.ibm.com/docs/en/aix/7.2?topic=system-pluggable-authentication-modules, (2022). IBM, Accessed on: 2022-07-03

[18] Vazquez, A.; Glogowski,A.; Costea, V.; Quatremain, H.; Paul, S. & Karmakar, S. (2018): Red Hat Security: Linux in Physical, Virtual, and Cloud, Bonneville, S.; O'Brien, D.; Kenlon, S.; Sacco, D. & Charles, H., 1-534, Red Hat, Raleigh, NC27601, USA

[19] https://en.wikipedia.org/wiki/Pluggable_authentication_module, (2022). Wikipedia, Accessed on: 2022-07-03

[20] http://web.mit.edu/rhel-doc/5/RHEL-5-manual/Deployment_Guide-en-US/ch-pam.html, (2007). MIT, Accessed on: 2022-07-01

[21] https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html/configuring_firewalls_and_packet_filters/using-and-configuring-firewalld_firewall-packet-filters#when-to-use-firewalld-nftables-or-iptables_getting-started-with-firewalld, (2022). RedHat, Accessed on: 2022-07-02

[22] https://firewalld.org/2018/07/nftables-backend, (2018). FirewallD, Accessed on: 2022-07-02

[23] https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/security_guide/chap-system_auditing, (2011). RedHat, Accessed on: 2022-07-04

[24] https://www.redhat.com/sysadmin/configure-linux-auditing-auditd, (2021). RedHat, Accessed on: 2022-07-04

[25] https://linux.die.net/man/8/auditd, (2008). Die.net, Accessed on: 2022-07-04

[26] https://documentation.suse.com/sles/12-SP4/html/SLES-all/cha-audit-setup.html, (2009). SuSe, Accessed on: 2022-07-04

[27] https://en.wikipedia.org/wiki/Advanced_Intrusion_Detection_Environment, (2022). Wikipedia, Accessed on: 2022-07-04

[28] https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/security_guide/sec-using-aide, (2015). RedHat, Accessed on: 2022-07-03

[29] https://www.redhat.com/sysadmin/linux-security-aide, (2021). RedHat, Accessed on: 2022-07-03

[30] https://documentation.suse.com/sles/15-SP1/html/SLES-all/cha-aide.html, (2019). SuSe, Accessed on: 2022-07-03

[31] https://www.redhat.com/sysadmin/linux-access-control-lists, (2020). RedHat, Accessed on: 2022-07-02

[32] https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/system_administrators_guide/ch-access_control_lists, (2015). RedHat, Accessed on: 2022-07-05

[33] https://www.redhat.com/en/topics/linux/what-is-selinux, (2019). RedHat, Accessed on: 2022-07-06

[34] https://wiki.centos.org/HowTos/SELinux, (2020). RedHat, Accessed on: 2022-07-06

[35] https://opensource.com/article/21/8/container-linux-technology, (2021). RedHat, Accessed on: 2022-07-26

[36] https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html/managing_monitoring_and_updating_the_kernel/setting-limits-for-applications_managing-monitoring-and-updating-the-kernel#what-kernel-resource-controllers-are_setting-limits-for-applications, (2022). RedHat, Accessed on: 2022-07-26

[37] https://www.oracle.com/technical-resources/articles/linux/resource-controllers-linux.html, (2012). Oracle, Accessed on: 2022-08-01

[38] https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/resource_management_guide/sec-default_cgroup_hierarchies, (2015). RedHat, Accessed on: 2022-08-15

[39] https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/security_guide/chap-compliance_and_vulnerability_scanning, (2012). RedHat, Accessed on: 2022-08-15

[40] https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/security_guide/sect-using_openscap_to_remediate_the_system, (2012). RedHat, Accessed on: 2022-08-15

[41] https://www.redhat.com/en/blog/getting-started-red-hat-insights-and-openscap-compliance-reporting, (2021). RedHat, Accessed on: 2022-08-15

[42] Zaharova, A.; Elin, V. & Panfilov, P. (2018): Technological and legal issues of identifying a person on the internet to ensure information security, Proceedings of DAAAM., at Zadar, Croatia, ISSN 1726-9679, DOI: 10.2507/29th.daaam.proceedings.069

[43] Dakic, V. & Ribaric, S. (2020): Judicial and technical improvement of General Data Protection Regulation, Proceedings of DAAAM., at Mostar, Bosnia and Herzegovina, ISSN 1726-9679, DOI: 10.2507/31st.daaam.proceedings.025

[44] Reithner, I.; Papa, M.; Lueger, B.; Cato, M.; Hollerer, S. & Seemann, D. (2020): Development and implementation of a secure production network, Proceedings of DAAAM., at Mostar, Bosnia and Herzegovina, ISSN 1726-9679, DOI: 10.2507/31st.daaam.proceedings.102