**UNIVERSITY OF SURREY**

**DEPARTMENT OF COMPUTER SCIENCE**

**MSc INFORMATION SECURITY**

**Academic Year 2022 – 2023**

**Using AppArmor Configuration for Linux Security Enhancement**

By

**Mahadik Venkatesh Sunil Kunda**

**6762698**

Under the guidance of

**Dr Sasa Radomirovic**

**A coat of arms with a deer and swords

Description automatically generated**

**Abstract**

The digital realm faces an escalating challenge that is to ensure the security of operating systems, notably Linux, against the continuous threats of vulnerabilities and unauthorized access. These vulnerabilities, if manipulated, can compromise the system's integrity, affecting a wide range of entities, from corporations to governments. In response to this challenge, this project integrates AppArmor, a mandatory access control framework to enhance Linux system security. The conventional Linux security framework, which heavily relies on discretionary access control, exhibits limited resilience in the face of advanced and intricate cyber threats. AppArmor establishes robust and customizable security policies to restrict individual application capabilities, mitigating risks like privilege escalation and data breaches. To manually create custom AppArmor profiles to limit application permissions beyond defaults. Python shell scripting also enables the automated generation of tailored profiles. AppArmor provides precise isolation via application-centric policies, enhancing security while permitting regulated interaction and avoiding excessive overhead. This adaptive approach improves conventional access control by restricting application actions. Overall, AppArmor delivers a robust defence mechanism for Linux systems, though it requires continuous evolution as threats constantly adapt. Integrating AppArmor is vital for increasing Linux security, resilience, and reliability against escalating digital threats.

**Acknowledgements**

I extend my deepest gratitude to **Dr Sasa Radomirovic**, my supervisor, for his invaluable guidance, expertise, and unwavering support throughout my academic year and during the development of this dissertation.

I also wish to express my appreciation to my family, colleagues, and friends for their constructive feedback and their unwavering presence and encouragement throughout this journey.

**Table of Contents**

**Abstract 2**

**Acknowledgements 3**

**Chapter 1 – Introduction 7**

**1.1 Aim 8**

**1.2 Objective 8**

**1.3 Motivation 8**

**1.4 Problem Statement 8**

**1.5 Linux Distribution Mazes 9**

**Chapter 2 – Literature Review 11**

**2.1.1 Introduction to Linux and Security 11**

**2.1.2 - Basic Linux Security 12**

**2.2 Introduction to SE Linux (Security Enhanced Linux) 14**

**2.3 Introduction to AppArmor   15**

**2.3.1 AppArmor v/s SE Linux 16**

**2.3.2 SE Linux profile V/S App Armor profile. 17**

**2.3.3 Problem analysis between AppArmor and SE Linux code design. 18**

**2.3.4 Working of AppArmor in System Level and Application Level 19**

**2.3.5 Difference between SE Linux and App Armor 20**

**2.3.6 Why other is not suitable for my objective. 21**

**2.3.7 Architecture of AppArmor 22**

**2.3.8 AppArmor Profiles and Configurations 23**

**2.3.9 AppArmor Tools – Utilities 24**

**2.3.10 Structure of a AppArmor profile 25**

**2.3.11 Features 26**

**2.3.11 Strengths, Weaknesses, and Future Scope of AppArmor 27**

**Chapter 3 – Methodology 28**

**3.1 Research Method 28**

**3.2 Linux and AppArmor 28**

**3.3 Python and Shell Scripting 28**

**3.4 Functionalities of Apparmor with Python 29**

* 1. **Breakdown of a Python script 30**

**3.6 Steps to Execute a AppArmor profile.** **31**

**3.7 Breakdown of a profile in AppArmor 32**

**Chapter 4 Implementation and Output 33**

**4.1 Experimental Approach 33**

**4.2 Custom Profiling for Application with AppArmor by Shell Scripting 34**

**4.3 AppArmor custom profile for the script. 35**

**4.4 Limitations 45**

**4.5 Evaluation 48**

**Chapter 5 Conclusion 50**

**Chapter 6 References 51**

**List of Tables.**

Table 1 – Features comparison of AppArmor and SE Linux

Table 2 – Comparison between SE Linux and AppArmor with different factors

Table 3 - File Permissions of AppArmor profiles

**List of Figures.**

Figure 1 – Profile comparison between SE Linux and AppArmor.

Figure 2 – Working of AppArmor on System and Application level.

Figure 3 – Architecture of AppArmor

Figure 4 – AppArmor process

Figure 5 – Demonstration of Case 1

Figure 6 – Demonstration of Case 2

Figure 7– Demonstration of Case 3

Figure 8 – Root Restrictions

Figure 9 – AppArmor Crashes

# **Introduction**

In the rapidly evolving digital age, the security of operating systems stands at the forefront of technological challenges. One of the principal problems modern systems faces, particularly Linux, is the persistent threat of vulnerabilities and unauthorized access.(Karapetyants and Efanov, 2022). These vulnerabilities, if exploited, can lead to significant system breaches, compromising the privacy and integrity of user data and the stability of the system itself. (Niu et al., 2014)(Sanchez et al., 2020).

The core of this matter extends beyond the realm of scientific inquiry, including the intricate complexity of operating system functionality, but also holds paramount importance in real-world applications. Every entity, from multinational corporations to government bodies, relies heavily on the specific functioning of their systems. (Zejin and Feng, 2018). A single breach can lead to enormous financial losses, leaks of sensitive information, and a subsequent loss of public trust. Hence, it becomes both an intriguing problem, given its complex nature, and an imperative one, considering the wide-ranging repercussions of potential breaches. (Zejin and Feng, 2018)

Addressing this challenge is no small feat. Naive approaches to system security often tend to be reactionary, only addressing issues once they arise rather than being proactive. They may also need to account for the multifaceted ways in which malicious entities can exploit system vulnerabilities.(Karapetyants and Efanov, 2022). The integrated protection of a system requires a deep understanding of both its inherent structure and the external threats it faces.

So, why hasn’t a foolproof solution been developed yet? Previous solutions have primarily taken a broad-brush approach, either restricting system functionalities to such an extent that it hinders genuine processes or being too lenient, thereby letting potential threats slip through the cracks. (Zhang Xinliang, 2014). There is also the matter of the evolving nature of threats; as security measures advance, so do the methods to bypass them.

Using AppArmor, this project proposes the integration of AppArmor, a mandatory access control framework, as a nuanced solution to the problem (Zivanov, 2022)(Kumar, 2023). The uniqueness of AppArmor lies in its application-centric approach to security. Rather than blanket restrictions or permissions, it erects a protective barrier around each application. (Cowan, C., 2007). This ensures that each application functions within its designated parameters, minimizing the risk of unintended behaviours (Zhu & Gehrmann, 2022). However, it is essential to note that while AppArmor significantly enhances system security, it is not a universally effective solution. Its effectiveness is contingent on meticulous profile configurations, and even then, it operates best when used in conjunction with other security measures.

In essence, this project aims to provide a layered, focused, and adaptive approach to Linux system security, reducing unauthorized access and thereby ensuring a safer, more reliable operating environment.

## **1.1 Aim**

The goal of this project is to improve Linux system security by integrating AppArmor, a mandatory access control framework. The purpose is to reduce vulnerabilities and unauthorised access in order to protect the system and its resources.

## **1.2 Objective**

Through the utilization of AppArmor, the concept will be building a protective barrier around an application. This barrier effectively isolates the application, preventing any interactions with the broader system (imperva, 2020). If the application deviate from its intended path, its communication can be restricted within set parameters, providing the capability to swiftly contain and disable any unintended behaviour.

## **1.3 Motivation**

During the academic phase, the researcher’s participation in a tutorial that illustrated the vulnerabilities present in Linux systems, the instructor demonstrated how these vulnerabilities could be exploited to compromise a system and its applications, ultimately gaining unauthorized control. The tutorial showcased examples of privilege escalation techniques, allowing the attacker to attain root access to the entire system. This level of control extended to manipulating other services, accessing the hard drive, and executing various actions with significant consequences.

The motivation for protecting Linux with AppArmor is to improve the system's overall security. AppArmor is a mandatory access control (MAC) framework of Linux Security, allowing administrators to create precise security policies for specific applications, files, directories, etc. Administrators get the power to limit the actions and privileges of programmes, essentially stopping them from obtaining access to sensitive resources or engaging in unauthorized activities by adopting AppArmor.

This proactive strategy helps to mitigate the risks associated with malicious behaviour, such as unauthorized data access, privilege escalation, and system penetration. AppArmor adds an extra layer of protection, emphasizing the idea of least privilege. Consequently, the system reduces its attack surface and enhances the overall security posture of the Linux environment.

* 1. **Problem Statement**

**“ How to enhance Linux security by using Apparmor?”**

The traditional approach to Linux security, which relies on user-space precautions, falls short of providing full protection. Access control approaches such as Discretionary Access Control (DAC) lack resilience, particularly when confronted with problems such as evasive malicious code. (Abdi et al., 2020). When such code is run, it gets the user's rights, putting all user data and apps at risk, especially in situations with administrator privileges. (Enck, McDaniel and Jaeger, 2008)

Relying entirely on user-space safeguards is insufficient for establishing true security. Most operating systems use access control techniques, most commonly Discretionary Access Control (DAC), which typically fall short of providing solid system security. (Berger et al., 2019). This weakness becomes clear when confronted with a serious challenge: malicious code that successfully evades application-level security. When this code is performed, it inherits the same permissions as the current user. As a result, all of the user's programmes and data are compromised. When the user has administrator rights, the entire system is jeopardised.

If sensitive data management standards are not maintained, this risk extends to possible data breaches from malicious or irresponsible individuals (Jang and Ric Messier, 2016). AppArmor develops as a solution to these problems. As a MAC framework, AppArmor enables customizable security policies for apps, preventing unauthorised access, privilege escalation, and system intrusions.

A critical first step in improving the security of a Linux system is determining the primary function or role of the Linux server (Jang and Ric Messier, 2016). This demands a thorough understanding of the system's components. Effective proactive steps for safeguarding the Linux system become difficult to apply without such expertise. The purpose is to improve overall system security by establishing fine-grained access control and avoiding possible security breaches.

## **1.5 Linux Distribution Maze**

The operating system is the greatest location to apply security policies. In the Linux world, there are hundreds of distinct distributions called as ‘distros’, each with its own set of advantages and disadvantages. (Blum, 2008)

 Among all of them, two types of distributions are clearly distinguishable for the purposes of this paper that are general-purpose distributions and "secure featured" distributions (Bohling et al., 2020). it could utilise either sort of distribution to put up a secure operating system environment. A general-purpose one should be hardened. A secure featured one, on the other hand, would fulfil our needs without requiring much customisation. (Dieguez Castro, 2016)

This labyrinth contains well-known distributions such as Ubuntu, Fedora, Debian, CentOS, Arch Linux, and many more. Each distribution may have a distinct focus, such as usability, reliability, cutting-edge software, minimalism, or security. Some distributions are optimised for server use, while others are optimised for desktop use. (Uzayr, 2022b)

# **Literature Review**

# **2.1.1 - Introduction to Linux and Security**

- Definition of Linux.

Linux is a highly prevalent operating system that enjoys widespread usage. The primary objective of its creation was to offer personal computer users a cost-effective or no-cost operating system. The Linux operating system was publicly released by Linus Torvalds on September 17, 1991, with its implementation primarily coded in the C programming language. (Ubuntu.com, 2023) It is based on the Unix operating system and is widely used in the IT field. Linux is well-known for its reliability, security, and adaptability (Jang and Ric Messier, 2016). It is popular among developers and programmers who want to customise their operating systems (Jang and Ric Messier, 2016).  Linux is also found in servers, supercomputers, and embedded devices. (Jang, 2018). Its relevance in the computer sector stems from its open-source nature, which allows developers to tweak and improve the operating system(Jang, 2018) .Linux is also recognised for its security and reliability, making it a popular choice for servers and other important systems (Jang, 2018).

- Significance in the tech world

* Because Linux is open source, it is free to use and adapt, making it a popular choice among developers and organisations.
* Linux is extremely adaptable, allowing users to customise the operating system to their own requirements.
* Because of its reputation for reliability and security, Linux is a popular choice for servers and other mission-critical applications. (Varian and Shapiro, 2003)
* Linux has a big and active developer and user community that contributes to its growth and maintenance. (Varian and Shapiro, 2003)
* Linux is utilised in a wide range of applications, such as servers, desktops, mobile devices, and embedded systems. (Varian and Shapiro, 2003)

- Importance of security in computer systems, with a focus on Linux.

Security is an important part of computer systems, and it is critical to safeguard the system from unauthorised access, malware, and other security risks. Because of its security features, Linux is a popular choice for servers and other mission-critical systems.(Saltzer and Schroeder, 1975)

* Access control: Linux has a powerful access control system that allows administrators to restrict access to files, directories, and other system resources. This aids in the prevention of unauthorised access to sensitive data.(Tian, Rong and Liu, 2012)
* Firewall: Linux has a firewall that may be set to prevent incoming traffic from unknown sources. This contributes to the system's defence against network-based threats.(Safe Security, 2021)
* Encryption: Linux offers a number of encryption technologies, including SSL/TLS, that may be used to safeguard network connections.(Kumar et al., 2009)
* Regular updates: Linux receives security patches and bug fixes on a regular basis, which helps to keep the system secure and up to date. (Deshmukh and Mahalle (2015))

Determining the principal function or job of the Linux server is a critical step in securing a Linux system. (Jung and sung, 2015) It should have a thorough understanding of what is on the system.(Jung and Sung, 2015) Otherwise, it will struggle to grasp what has to be guarded, and so proactively protecting the Linux system would be ineffective.(Zejin and Feng, 2018)

**2.1.2 - Basic Linux Security**

**-** Overview of the security model in Linux.

* Although Linux is frequently seen as safer than Windows or Mac OS X, it nevertheless confronts security issues such as password cracking, vulnerability exploitation, virus downloads, and unauthorised access to critical information.(Bohling et al., 2020)
* To protect a Linux system, keep it up to date, use a secure firewall, use antivirus software, generate complicated passwords, and configure strict file permissions.
* Keeping the system up to date aids in the patching of any vulnerabilities and defects that attackers may exploit.
* A secure firewall aids in the closure of superfluous ports and the prevention of unauthorised system access.(Wright et al., 2002)
* Using antivirus software aids in the detection and removal of malware that may be downloaded and operated on the system. (Wright et al., 2002)

 - Key components of Linux security architecture.

The Linux Security Modules (LSM) framework, which offers a general-purpose framework for security policy modules, is a significant component of the Linux security architecture.(Narayanan et al., 2017). This enables the implementation of various access control models as loadable kernel modules. (Narayanan et al., 2017).  Other components include POSIX.1e capabilities, SE Linux, Domain, and Type Enforcement (DTE), and the Linux Intrusion Detection System (LIDS), all of which have been converted to use the LSM framework. (Narayanan et al., 2017).

* Linux Security Modules (LSM): The LSM (Linux Security Modules) kernel patch offers a foundational framework to support security modules, focusing primarily on access control while integrating security fields into kernel structures and implementing hook functions for enhanced security management. (Smalley, Fraser, and Vance, 2001)
* Security-Enhanced Linux (SE Linux): SE Linux is an implementation of a flexible access control architecture in the Linux kernel. It provides mandatory access control (MAC) policies that enforce fine-grained access control rules based on labels assigned to processes, files, and other system resources.(Ding et al., 2020)
* POSIX.1e capabilities: POSIX.1e capabilities provide a way to grant specific privileges to processes without giving them full root access. It allows fine-grained control over the privileges that a process can exercise. (Posix-compatible file system, method of creating a file list and storage device, 2014))
* Domain and Type Enforcement (DTE): DTE is a security model that extends the traditional UNIX discretionary access control (DAC) model. It introduces the concept of domains and types, allowing for more granular access control based on the relationships between subjects and objects.(Smine et al., 2021)
* Linux Intrusion Detection System (LIDS): LIDS is a security framework that provides a set of security enhancements for the Linux kernel. It includes features like file and process restrictions, network restrictions, and system call restrictions to protect against unauthorized access and attacks.(Negar Almassian, Azmi and Berenji, 2009)

 - Common security challenges faced by Linux systems.

* Password cracking, vulnerability exploitation, virus downloads, and unauthorised access to sensitive information are all security concerns for Linux.
* These risks can be reduced by keeping the system up to date, maintaining a secure firewall, employing antivirus software, creating complicated passwords, and establishing strict file permissions.
* Keeping the system up to date aids in the patching of vulnerabilities and problems that attackers may exploit.
* A secure firewall aids in the closure of superfluous ports and the prevention of unauthorised system access.
* Using antivirus software aids in the detection and removal of malware that may be downloaded and operated on the system.
* Making complicated passwords makes it more difficult for hackers to crack them over a network.
* Setting strict file permissions guarantees that sensitive data cannot be viewed without permission. (Yaswinski, Chowdhary and Jochen 2019).

- Identification of common vulnerabilities found in Linux distributions.

From January 2010 to January 2020, an examination of 1,858 Linux kernel vulnerabilities indicated the presence of a substantial number of low-complexity vulnerabilities that may be exploited from the local system.(Karapetyants and Efanov, 2022). These flaws potentially endanger the kernel's quality of service and allow attackers to acquire privileged access.(Niu et al., 2014)

- How Linux systems can be vulnerable to attacks compromising these aspects.

In the context of Linux systems, vulnerability research indicates that, while a small number of vulnerabilities may result in minimal repercussions, signalling a lack of attacker control, the vast majority of flaws result in significant consequences. (Niu et al., 2014). This gives prospective attackers the ability to change individual kernel parts, increasing the vulnerability to targeted and purposeful assaults. The operational continuity of services hosted on the Linux operating system is jeopardised when a significant number of vulnerabilities influence system availability. (Sanchez et al., 2020). Hacking, data theft, and denial of service assaults might jeopardise the system's security and integrity. Addressing this issue needs the implementation of strong safeguards. (Niu et al., 2014)

**2.2 Introduction to SE Linux (Security Enhanced Linux)**

Security-Enhanced Linux (SE Linux) is a critical security architecture for Linux systems that gives administrators greater control over system access. SE Linux, which was originally created by the United States National Security Agency (NSA) as a set of patches based on Linux Security Modules (LSM), intends to strengthen access management.(Smalley, Vance and Salamon, 2001)

SE Linux, which was released to the open-source community in 2000 and then included into the mainline Linux kernel in 2003, is a cornerstone of current Linux security.(Smalley, S., 2001)

SE Linux adopts the Flask security architecture. The Flask security architecture, first seen in the Fluke research operating system, is used by SE Linux for flexible and granular access controls. (Spencer et al., 1999). This design enables a distinct separation of policy enforcement and decision-making components. The latter is handled by a separate security server, while the access vector cache (AVC) reduces performance effect by caching access choices.(Spencer et al., 1999)

Security-Enhanced Linux (SE Linux) is a security architecture for Linux that includes kernel modifications and user-space tools that administrators to have  a granular control over who and what can access the system(Lepreau, J., 1999)

The default SE Linux setup often seeks to lock down system-wide services while remaining hidden from end users; most of their processes operate unconfined.(Vermeulen, 2013).  SE Linux is the most extensively implemented Linux-based security mechanism capable of application confinement, while being primarily intended at experienced users and security managers. For many Linux users, it will be the only such scheme installed on their machine, and recent effort has focused on enhancing SE Linux usability. SE Linux implements access restrictions using security policies, which are specified sets of rules defining access rights. (McCarty, 2005). These policies control how programmes, processes, and files interact with one another. SE Linux performs a progressive review when a subject (an application or process) requests access to an object, such as a file. It initially looks up authorization data for subjects and objects in an access vector cache (AVC). Access is permitted or refused if the cache produces an unambiguous judgement.(Mayer, Caplan, and MacMillan, 2006)(Smalley, Vance and Salamon (2001b)

**2.3 Introduction to AppArmor**

   - Explanation of AppArmor as a Linux security module.

AppArmor, developed by Immunix is classified as a Linux Security Module (LSM), serves as an efficient means of confining individual programs within precise permissions, encompassing the realm of file read, write, and execution privileges, capabilities, network access, and resource limits. This potent mechanism revolves around mandatory access control (MAC), effectively isolating processes and effectively catering to privacy requisites in critical applications. (Ecarot et al., 2020). AppArmor is a mandatory access control solution that enables administrators to declare the permissions that programmes must have in order to access system resources. Since version 7.10, it has been a standard feature of Ubuntu. (Creative Commons , 2019). At its core, AppArmor operates via the utilization of profiles seamlessly integrated into the Linux system kernel, facilitating the formulation of application-specific security policies. (Zhu and Gehrmann, 2022). These profiles, fine-tuned and tailored, assume the role of gatekeepers, orchestrating access to path-based resources for designated programs or containers. Through meticulous profile configuration, AppArmor's competence stands through, providing a multi-layered security mechanism that considerably reduces the possible attack surface(Ecarot et al., 2020).

App armour essentially allows you to restrict access to specific application scripts or programmes to only accomplish specific tasks or to only give particular portions of the system access to that programme. (Zhu and Gehrmann, 2021).  AppArmor improves Linux security by building a virtual "firewall" around apps, protecting them from known and unknown vulnerabilities. (Cowan, C., 2007). Furthermore, it provides default profiles customised to common applications, enhancing system-wide security protections. (Zhu and Gehrmann, 2021a)

AppArmor is an easy-to-use Linux application security solution designed to keep our Linux systems and servers save from malicious threats, allowing you to deploy security policies in hours, not days. (Zhu and Gehrmann, 2022)

AppArmor, formerly known as SubDomain, also implements required Linux restrictions, but on a simpler paradigm than SE Linux. AppArmor creates a list of resources for each restricted programme based on resource names (such as file paths) to designate what may be accessed. (Schreuders, McGill and Payne, 2011). Simple abstractions such as dbus, kde, and nameservice are used to organise privileges associated with specific low-level programme features and may be utilised for developing rules.(Schreuders, McGill and Payne, 2011)

**2.3.1 AppArmor v/s SE Linux**

AppArmor and SE Linux are two prominent Mandatory Access Control (MAC) mechanisms employed for safeguarding Linux systems against unauthorised intrusion. (Zivanov, 2022). Nevertheless, there exist notable distinctions between the two systems, which provide AppArmor a more advantageous selection for MAC. (Ecarot et al., 2020)

* AppArmor is characterised by a higher level of configurability and user-friendliness. AppArmor uses profiles to establish the permissions associated with individual applications, whereas SE Linux utilises rules for the same purpose. AppArmor is seen as a more favourable option for systems administrators lacking familiarity with Mandatory Access Control (MAC) systems, as profiles offer a higher level of simplicity and comprehensibility compared to policies.(Kumar, 2023)
* AppArmor has a reduced susceptibility to false positives. A false positive occurs when an application is erroneously denied access to a resource that it should legitimately be granted access to. AppArmor has a lower susceptibility to false positives in comparison to SE Linux due to its use of application-specific profiles.(Kumar, 2023)
* AppArmor has a reduced performance overhead. AppArmor has a comparatively lower performance overhead in contrast to SE Linux due to its selective implementation of Mandatory Access Control (MAC) just to programmes with profiles. In contrast, SE Linux implements Mandatory Access Control (MAC) across all programmes, even those that may not necessarily require it.(Kumar, 2023)

|  |  |  |
| --- | --- | --- |
| ***Features*** | **AppArmor** | **SE Linux** |
| *Ease of configuration* | Easier | More difficult |
| *Proneness to false positives* | Less prone | More prone |
| *Performance overhead* | Smaller | Larger |
| *Coverage* | Selective | Comprehensive |
| *Flexibility* | Less flexible | More flexible |

Table 1 – Features comparison of AppArmor and SE Linux (Zivanov, 2022).

The project selects AppArmor over SE Linux due to its emphasis on usability, customization, and proactive security. AppArmor stands out for its user-friendliness and profile-based methodology, simplifying the management and maintenance of security policies. In contrast, many view SE Linux as complex and demanding. (Kumar, 2023)

A standout feature of AppArmor is its commitment to the principle of least privilege, enabling the creation of highly customized security profiles. These profiles determine the minimal permissions each program needs, effectively curbing potential damage from security breaches. Such granular control proves invaluable for organizations aiming to boost their security stance and adhere to various security standards. (Forum, U. 2012)

Moreover, AppArmor proactively addresses risks tied to zero-day vulnerabilities. By keeping applications within set boundaries, it diminishes the fallout from unidentified threats. In comparison, SE Linux often presents complexities when setting up these preventive measures, whereas AppArmor streamlines the process. (Kumar, 2023)

AppArmor also ensures meticulous control over user privileges, including those of root users. This control helps reduce vulnerabilities by keeping high-privilege activities strictly confined. Overall, AppArmor presents a more intuitive, adaptable, and forward-looking security approach than SE Linux, marking it as a top choice for reinforcing security measures. (Forum, U. 2012)

**2.3.2 SE Linux profile V/S App Armor profile.**

A close-up of a computer program

Description automatically generated

**Figure 1 – Profile comparison between SE Linux and AppArmor (SUSECON, 2021)**

Security-Enhanced Linux (SE Linux) use a specialised programming language to define restrictions that are inherently challenging to administer. In contrast, App Armour utilises the traditional Linux vocabulary of read/write/execute rights, without introducing any novel linguistic elements(SUSE, 2023).

Both Security-Enhanced Linux (SE Linux) and AppArmor share the common goal of enhancing application security. However, the approach varies between the two. SE Linux employs a policy written in a more intricate and verbose language, requiring significant effort to grasp and master. On the other hand, AppArmor employs a more succinct and organized language, resulting in profiles that are easier to comprehend, maintain, and manage(SUSE, 2023).

**2.3.3 Problem analysis between AppArmor and SE Linux code design.**

Analyzing the code design of AppArmor and SE Linux reveals some key differences and potential challenges in their respective approaches to enforcing Mandatory Access Control (MAC) on Linux systems:

**- AppArmor Code Design Challenges:**

* + Simplicity vs. Granularity: AppArmor's focus on simplicity can be seen as a limitation when fine-grained control is needed. It may not be as suitable for environments with highly specific security requirements where detailed policy rules are essential.
  + Limited Policy Complexity: While AppArmor's simplicity is advantageous for many users, it can become a limitation in complex environments where comprehensive policies are required. Writing and maintaining complex profiles can be challenging.
  + Resource Conflict Resolution: When multiple profiles overlap or have conflicting rules, AppArmor's conflict resolution mechanisms can be less intuitive compared to SE Linux's more comprehensive policy-based approach. (OpenAI, 2023)

**- SE Linux Code Design Challenges:**

* + Complexity and Learning Curve: SE Linux's policy-based model can be daunting for beginners. Writing, maintaining, and troubleshooting SE Linux policies require a deeper understanding of security contexts and can be challenging for administrators new to SE Linux.
  + Resource Intensive: SE Linux's fine-grained control and extensive policies can consume more system resources than AppArmor. This could be a concern in resource-constrained environments.
  + Compatibility and Integration: SE Linux's strict policy enforcement may clash with certain applications or services that expect more permissive behaviour. Achieving compatibility can require adjustments or custom policies, which can be complex.
  + Policy Management: Managing SE Linux policies for a large number of applications or services can be a complex task, especially when custom policies are needed. Policy conflicts and misconfigurations can lead to unexpected behaviour.
  + Community and Support: SE Linux has a steeper learning curve, and finding comprehensive documentation and community support can be more challenging compared to AppArmor. (OpenAI, 2023)

**2.3.4 Working of AppArmor in System Level and Application Level**

**AppArmor – System Level**

App Armor – Allow Everything. Initially allow everything approach. Considering a house for an example at system level user process are allowed to see and access at system level.

However, once we are inside the system(inside the house) and trying to execute the application level. That’s when the AppArmor kicks in and denied then application resources depending for the profile for the application.

A diagram of a diagram of a maze

Description automatically generated with medium confidence

**Figure 2 – Working of AppArmor on System and Application level (SUSECON, 2021)**

**2.3.5 Difference between SE Linux and App Armor**

|  |  |  |
| --- | --- | --- |
|  | SE LINUX | APPARMOR |
| Origin | NSA | DoD (Department of Defence) |
| Approach | Deny everything system level | Deny everything application level |
| Integration in Linux | LSM Module+Userland | LSM Module+Userland |
| Performance Impact | <5% | 0-2% |
| Restrictions | MAC,RBAC, MLS | MAC,RBAC, (MLS) |
| Confinements | Indirect via “Labels” | Direct, Path based |
| Profiles/Policies | “Programs” need to be compiled | Simple text files, Unix style |
| Auditing/Human | SE Linux policies are hard to read and to audit | AppArmor profiles are easy to read and to audit |
| Availability in Distros | CentOS, RHEL, SUSE Linux Enterprise | Ubuntu, openSUSE Linux Enterprise |
| Primary Adoption | US/Canada | Worldwide |
| Efficiency | Less | More |

Table 2 – Comparison between SE Linux and AppArmor with different factors (SUSECON, 2021)

**2.3.6 Why other is not suitable for my objective.**

There exist several alternatives within the realm of AppArmor that can be effectively employed to augment security measures.

Firejail, a Linux sandboxing tool, is a lightweight solution that enables the isolation of programmes from the broader system environment. This technology has the capability to restrict programmes from accessing confidential information or network resources.(Laurén, Sampsa Rauti and Ville Leppänen, 2017)

Grsecurity, also known as the Linux Security Module, is a module designed to increase security by offering several features such as stack protection, heap protection, and Address Space Layout Randomization (ASLR). The utilisation of this technique has the potential to enhance the system's resilience against malicious assaults.(Derby Cardona, 2016)

|  |  |  |  |
| --- | --- | --- | --- |
| ***Feature*** | **AppArmor** | **Firejail** | **Grsecurity** |
| *Ease of configuration* | Easier | Easier | More difficult |
| *False positives* | Less prone | More prone | More prone |
| *Performance overhead* | Smaller | Smaller | Larger |
| *Flexibility* | More flexible | Less flexible | Less flexible |
| *Focus* | MAC system | Sandboxing tool | Security enhancement module |

Table 3 - Comparison between AppArmor, Firejail, Grsecurity with respect to their features.

AppArmor exhibits a high degree of flexibility. The user possesses the ability to adjust the permissions of individual applications in order to align with their unique requirements. This feature enhances the efficacy of the instrument in ensuring system security.

The aforementioned alternatives, namely Firejail and Grsecurity, do not align as well with my intended purpose. Firejail is a software programme designed for the purpose of sandboxing apps, hence facilitating their isolation.(Bäckman and Hagfjäll, 2017).  However, it is important to note that Firejail does not offer an equivalent degree of control in comparison to AppArmor. The Grsecurity module serves as a security augmentation tool that effectively safeguards the system against targeted attack vectors. However, it should be noted that its primary focus is in fortifying the overall system security rather than isolating individual applications.(Fox et al., 2009)

**2.3.7 Architecture of AppArmor**

AppArmor is written in the form of a Linux Security Module (LSM). LSM is a feature introduced with Kernel 2.6 that allows anybody to write a Mandatory Access Control (MAC) policy and integrate it into the kernel without the requirement for a new kernel refresh or maintainer authorization.(Pant, 2019)

A diagram of a computer

Description automatically generated

**Figure 3 – Architecture of AppArmor**

**2.3.8 AppArmor Profiles and Configurations**

AppArmor profiles are written from the perspective of the process, not the system. The path names are resolved in process names.

AppArmor rules are saved in a collection of files on the Linux file system. (Banerjee, 2021). These files specify the profiles, policies, and rules that AppArmor employs to impose application security limitations. AppArmor profiles and related configuration files are usually stored in the /etc/apparmor.d/ directory. (Pant, 2019)(Six, 2023)

A screenshot of a computer screen

Description automatically generated

**Figure 4 – AppArmor process**

**2.3.9 AppArmor Tools – Utilities**

sudo apt-get install apparmor-utils

Installers do not automatically include these utilities. They offer additional configuration tools for AppArmor. Users can employ these utility tools not only to protect default profiles but also to create, modify, or alter a custom profile. (Fox et al., 2009)

AppArmor's application isolates feature two modes: "enforcement" tight mode, and "complain" passive mode.

* aa-enforce – AppArmor is operational and implementing the policy. It makes the programme do it. If the programme attempts to divert and go above what it is permitted to accomplish, it will be refused.(Ecarot et al., 2020)
* aa-complain- AppArmor is operational; however, it is not enforcing the policy. If the programme tries to perform anything that is not authorised, it will accept it but will protest and maintain a record in the log file. (Ecarot et al., 2020)

AppArmor was created, along with several tools, to assist in the creation of such profiles. These utilities operate on a Linux system's user space.

* aa-status - This command displays a summary of the current AppArmor enforcement status, indicating which profiles are in complain or enforce mode and those that are loaded but not active(unconfined). It is used to check if the AppArmor is active. (Shamim, 2019)(SUSE, 2023b)
* aa-genprof: This tool aids in the creation of AppArmor profiles for applications. It works by observing an application's behaviour and interactions with the system and then creating a profile that permits these activities. This is beneficial when generating initial application profiles. (Shamim, 2019)(SUSE, 2023b)
* aa-logprof: Based on application behaviour logs, this tool assists administrators in fine-tuning AppArmor profiles. It analyses AppArmor data and recommends changes to profiles to enable essential operations while blocking unauthorised ones. (Shamim, 2019)(SUSE, 2023b)
* apparmor\_parser: The parser is an AppArmor core component that reads and interprets profile definitions written in the AppArmor policy language. It is in charge of loading profiles into the kernel and used to load and enforce AppArmor profiles manually. It parses the profiles and applies the associated rules to the processes (Shamim, 2019)(SUSE, 2023b)
* aa-updateprof: This program updates AppArmor profiles with new features, syntax, or rule modifications introduced in new AppArmor versions. (Shamim, 2019)(SUSE, 2023b)
* aa-unconfined - "unconfined" mode– This is found within the util package where the AppArmor is active. However, the program is free to perform anything it wants on the system and will not report. (Novell, 2008) (Shamim, 2019)(SUSE, 2023b)

**2.3.10 Structure of a AppArmor profile**

Profiles are essential text files that can be found in the /etc/apparmor.d/ directory. They are divided into many sections: #include, capability entries, rules, and "hats." (NOVELL, S. (2007))(Novell, 2008)

* #include

AppArmor profile that refers to an #include file that mediates programme access permissions. Using an include can grant the program access to directory paths or files that other programs also require. The use of includes can help to minimise the size of a profile. (NOVELL, S. (2007))(Novell, 2008)

AppArmor adds /etc/apparmor.d/ to the path in the #include declaration by default.

* Capability Entries

The exact capabilities or permissions that a process or application is permitted to have inside a constrained security environment are referred to as capability entries. (NOVELL, S. (2007))(Novell, 2008)

* Rules

Rules are the building blocks of security profiles in AppArmor, defining what actions and resources a restricted process is permitted to access or interact with. These rules are used to establish a policy that imposes per-process required access control. (NOVELL, S. (2007))(Novell, 2008)

* Hats

An AppArmor profile is a security policy for a specific programme instance or process. It applies to executable programs, but if certain parts of the programme require different access rights than others, the programme can "change hats" to utilise a new security context distinct from the main program's access. This is referred to as a hat or sub-profile. (NOVELL, S. (2007))(Novell, 2008)

File Permissions -

|  |  |
| --- | --- |
| Option | File |
| read | r |
| write | w |
| execute | ix |
| append | a |
| link | l |
| lock | k |

Table 3 - File Permissions of AppArmor profiles (Six, 2023)(NOVELL, S. (2007))

**2.3.11 Strengths, Weaknesses, Features and Future Scope of AppArmor**

**Features**

* Resource Limit Control:

AppArmor offers the capability to establish and manage resource limitations for an application. (rlimits, also known as ulimits). By default, AppArmor lacks control over apps' rlimits. It only exerts control over the limits that are explicitly set in the confining profile example - setrlimit(2), ulimit(1), or ulimit(2). Furthermore, it is important to note that AppArmor is unable to increase the resource restrictions that the system has set. Its functionality is limited to reducing the present resource limits of an application.(Six, 2023)(Ecarot et al., 2020)(SUSE, 2023b)

* Auditing Rules:

AppArmor has the capability to perform audits on specified rules, resulting in the generation of audit messages inside the audit log whenever these rules are matched. To activate audit messages for a specific rule, prefix the rule with the audit keyword. The integration of audit control with owner conditional file rules enables the provision of auditing capabilities in instances when a user seeks to access files that they possess. (Six, 2023)(Ecarot et al., 2020)(SUSE, 2023b)

Audit messages are not generated for every read or write of a file but only when a file is opened for read or write.

* Monitoring applications:

Applications during execution involve the runtime monitoring of system calls. This ensures that programmes are restricted from invoking any external processes unless they have been declared in advance. (Six, 2023)(Ecarot et al., 2020)(SUSE, 2023b)

* Network control:

AppArmor offers a rather straightforward form of protection primarily focused on providing coarse-grained security measures, particularly for a specific socket defined by its domain, type, and protocol. AppArmor only verifies the per-process level authorization for network access. The aforementioned objective can be accomplished by utilizing TCP over either IPv4 or IPv6 or, alternatively, by employing the Netlink raw protocol. (Six, 2023)(Ecarot et al., 2020)(SUSE, 2023b)

**Strengths**

The merits of AppArmor primarily reside in its capacity to offer mandatory access control (MAC) and effectively isolate processes. These features are particularly advantageous in meeting the privacy demands of essential apps. The application confinement based on functionality empowers end users to exert efficient control over the rights of their apps, exhibiting superior efficacy when compared to alternative security frameworks such as SE Linux. AppArmor is invaluable for safeguarding sensitive data and fortifying defences against insecure and harmful apps. OpenAI (2023)

**Weakness**

One of the limitations of AppArmor is its restricted scope in terms of ensuring confidentiality enforcement for crucial apps. The system is vulnerable to side-channel assaults and should be incorporated as a component of a broader defence-in-depth approach. Another area for improvement is the intricate nature of configuring security policies, necessitating a high level of expertise and active engagement from security professionals. Moreover, it is worth noting that AppArmor's protective measures may be inadequate in mitigating privilege escalation assaults, particularly in the case of resource-intensive graphics. OpenAI (2023)

**Future Scope for App Armor.** (DebConf , 2018)

* + Fine-grain networking.
  + Overlaying file system support and permissions
  + Shared memory segments.
  + Specific policies for group and users.
  + Kernal keyrings
  + chroot

1. **Methodology**

**3.1 Research Method**

This study employs an experimental research method to delve into the functionality of AppArmor in tandem with Python on Ubuntu Linux. This approach facilitates a systematic and rigorous analysis, offering both transparency and an authentic representation of data. The primary data was sourced from analyzing Linux in conjunction with the workings of AppArmor and Python. The research utilized triangulation analysis, drawing from a diverse set of sources. These included papers, reviews, and articles sourced from Google Scholar, the University of Surrey library, and general Google searches (Dympna Casey, 2009). The search keywords encompassed terms such as Linux, security, Mandatory access control, security enhancement, Python, and scripting within Linux. After a comprehensive review and assessment, conclusions were drawn about augmenting security on Linux. Additionally, a Python script was crafted leveraging multiple YouTube tutorials, which was then tested and demonstrated.

**3.2 Linux and AppArmor**

For this project, Ubuntu 22.04.2 LTS is selected as the operating system hosted within Oracle VM VirtualBox, primarily due to its user-friendliness, widespread acceptance, and ease of installation. (Blaminsky, J. 2013). Ubuntu stands out not only because of its security provisions, like distinctive usernames and passwords but also its similarity to Windows in terms of package installation, which is helpful for those transitioning between the two operating systems. Canonical's robust community and infrastructure support for Ubuntu ensures abundant resources, positioning it as an ideal choice for the undertaking. (Ecarot et al., 2020)(Pant, 2019) Within this setup, AppArmor is integrated, capitalizing on its name-based mandatory access control under the Linux Security Module framework. The project employs AppArmor version 3.0, currently considered the most reliable. It enables the creation of profiles, defining the exact files and permissions accessible to applications, using two principal rule sets path entries and capability entries. (Canonical, 2023) The choice of AppArmor is pivotal; its profile-based security provides meticulous control over system resources, reducing vulnerabilities and fortifying the system against potential breaches. In essence, the integration of AppArmor fortifies the system's security, guaranteeing that only sanctioned operations occur.(Canonical, 2023)(Ecarot et al., 2020)

**3.3 Python and Shell Scripting.**

Implementing Python as a shell scripting language combines the ease and straightforwardness of traditional shell scripting languages like Bash with the robust libraries and functionalities available in Python. (Kašpar and Muzikář, 2019). Utilizing Python version 3.11.5 allows for the execution of tasks more regulated and precisely, thereby enhancing the project's security measures. (Lutz, 2001) Additionally, the use of Python for scripting makes the code more transparent by enabling the incorporation of explanatory comments, which aids administrators in fully understanding the operations being carried out. (Yuil and Halpin, 2006) This approach offers both flexibility and enhanced capabilities, making it a strategic choice for the project.

**3.4 Functionalities of Apparmor with Python**

In this project's scope, the combined capabilities of AppArmor and Python within the Ubuntu environment have been harnessed to bolster system security. AppArmor's application-centric security model is a departure from conventional methods that predominantly block unsolicited system access. (SUSE, 2023a) Instead, AppArmor zeroes in on curbing unauthorized actions performed by specific applications. This methodology has its merits, offering flexibility, streamlined operation, and lucidity. With AppArmor's precise permission configurations for each application, it tailors' security protocols to exact needs, thereby reducing unnecessary system exertions. (Gruenbacher and Arnold, 2007)Its transparency ensures that any denied access comes with clear justifications, facilitating faster troubleshooting. (Pant, 2019)

Python, renowned for its versatility and efficacy across diverse tasks, acts as a perfect ally to AppArmor.(Mosh, 2018) While Python is adept at wide-ranging programming and scripting, AppArmor provides an added layer of security, ensuring these applications are kept in check. (Mosh, 2018). Python's programming prowess and AppArmor's specialized security protocols create a potent security mechanism that promptly addresses unintended or malevolent behaviours on the application front. (Berger, 2020)(Gift and Jones, 2008)

In essence, the fusion of AppArmor and Python does not just fortify individual software elements from potential threats but streamlines the security implementation and management. Given that Ubuntu and similar Linux distributions come with pre-set AppArmor profiles, the entire security integration becomes even more seamless. (Osborne, 2021). This integrated approach effectively capitalizes on the unique attributes of both tools, yielding a heightened security paradigm for the project. (Blum, 2008)(Thiell, Degrémont and Cea, 2013)

**3.5 Breakdown of a Python script -** (Programming, 2018)

#!/bin/bash

#import statements

# Function Definitions

function xyz() {

}

# Main script starts here

Line 1

Line 2

Line n

exit 0

#!/bin/bash: This line is called a shebang. It indicates that the script should be executed using the bash shell interpreter.

#import statements – Including all the import statements which are required for processing.

# Function Definitions - function xyz(): This comment indicates that a function will be defined below

# Main script starts here Line 1 to Line n : Indicating that the central part of the script starts below.

exit 0: This exits the script and returns a status code of 0 to the shell. In Unix-based systems, a return code of 0 typically indicates that the program ran successfully without any errors.

* Save the code with the desired filename – ‘mytestscript.sh’
* Make it executable by – ‘chmod +x mytestscript.sh’
* To execute the script - ./mytestscript.sh

**3.6 Steps to Execute a AppArmor profile.** (Ubuntu, 2023)

* Ensure AppArmor is Installed and Running:

sudo systemctl status apparmor

* Place the Profile in the Appropriate Directory:

/etc/apparmor.d/

Save the profile in a file. Typically, you'd name it after the script or its path, replacing “/” with “.”.

* Load and Activate the Profile:

Use the apparmor\_parser command to load the profile into AppArmor:

sudo apparmor\_parser -a /etc/apparmor.d/profile\_name(mytestscript.sh)

* Enforce the Profile:

By default, profiles are in "complain" mode where violations are logged but not prevented. To enforce the profile rules, use:

sudo aa-enforce /etc/apparmor.d/ profile\_name(mytestscript.sh)

- Run the Python Script*:*

Execute the Python script as usual. With the AppArmor profile in place, the script's operations will be restricted based on the profile's rules. (Perfect, 2022)

**3.7 Breakdown of a profile in AppArmor** (Top, 2022)

#include <tunables/global>

/profile\_name {

  # This section specifies file access rules

  /path /to / allowed/ disallow /file r,

  /path /to / allowed/ disallow /directory/ r,

  /path /to/ allowed/ disallow /executable ix,

  # This section specifies network rules

  network,

  # This section specifies capabilities

  capability net\_bind\_service,

  # ...

}

#include <tunables/global>: This line is often added at the start of an AppArmor profile to set globally adjustable and include values.

/profile\_name: This is the AppArmor profile name. Usually referred to as address/location of the application.

File access rules:This section includes all the file access permissions to allow or deny files.

Network rules:Allow or deny network access permissions.

Capabilities:This section specifies the capabilities that the confined process may have.

Custom AppArmor profiles can be developed by organisations or individuals in order to adhere to unique security standards and legal requirements. This practice not only guarantees adherence but also strengthens the level of security when confronted with audits or evaluations.

We can enforce the designated rules and constraints by crafting a profile in AppArmor for a specific application. Once this profile is activated, the application will operate within the defined limitations set by the profile. Consequently, this introduces restrictions, ensuring that the application adheres to the security measures established by the AppArmor profile.(NOVELL, S. ,2007)

1. **Implementation and Output**

**4.1 Experimental Approach**

A specialized profile will be constructed and customized for particular applications, highlighting both the constraints and capabilities associated with the operation of the application. Consequently, the profile will link the application in question to it based on its file path.

Traditionally, an AppArmor profile is generated using the specific command 'aa-genprof' along with the name of the application. AppArmor initiates system monitoring and observes the application's processes at this juncture. The application must then be executed to perform its usual functions. AppArmor keeps track of these activities and reviews access logs to determine the actions that the application has taken. Subsequently, permissions are either granted or denied based on what is deemed necessary or unnecessary (e.g., inherit, execute, scan, allow). Upon completing this process, AppArmor creates a profile that governs the application's normal operations.

In contrast, the project takes an alternative approach by eschewing the 'aa-genprof' command. Instead, a custom profile is manually created using standard Linux root commands like 'sudo nano.' The file path of the application is specified, linking it to the custom profile. Additional restrictions and limitations are then directly incorporated into this profile, which is saved in AppArmor's default directory at '/etc/apparmor.d/'. This custom profile functions equivalently to a standard AppArmor profile but includes specific, tailored security measures. If any unauthorized or abnormal activities are attempted, the security measures in the custom profile will automatically block the application in question.

**4.2 Custom Profiling for Application with AppArmor by Shell Scripting**

The utilization of Python shell scripting to generate personalized profiles for AppArmor presents a versatile and automated methodology for augmenting the security of applications at the operating system level on Linux systems.

Profile :-

Consider aatest.sh script is a **sort of application** which will perform some file to check #permissions for file creation, deletion, and network access.

#!/bin/bash

# Define color codes for output messages

GREEN=$(tput setaf 2)

RED=$(tput setaf 1)

NC=$(tput sgr0)

# Begin the script

echo -e "${GREEN}My AppArmor Test Script\n"

# Test the ability to create a file

echo -e "${NC}Let's check if it's possible to create a file\n"

sleep 5

# Attempt to create a file named 'testfile.txt' in the 'data' directory

touch data/testfile.txt

# Check if the file creation was successful and provide corresponding feedback

if [ $? -eq 0 ]; then

    echo -e "${GREEN}File Created Successfully\n"

else

    echo -e "${RED}File Creation Blocked by AppArmor\n"

    exit 1

fi

sleep 5

# Test the ability to delete a file

echo -e "${NC}Now let's see if we can delete that file\n"

sleep 5

# Attempt to remove the file named 'newtestfile.txt' from the 'data' directory

rm data/testfile.txt

# Check if the file deletion was successful and provide corresponding feedback

if [ $? -eq 0 ]; then

    echo -e "${GREEN}File Deleted Successfully\n"

else

    echo -e "${RED}File Deletion Blocked by AppArmor\n"

    exit 1

fi

sleep 5

# Test network access, file creation, and deletion privileges

echo -e "${NC}Now, let's assess network access, file creation, and deletion\n"

sleep 5

# Test network access and file creation privileges

# Download a file from a specified URL and save it in the 'data' directory

wget https://www.surrey.ac.uk/robots.txt -P data/

# Check if the file download was successful and provide corresponding feedback

if [ $? -eq 0 ]; then

    echo -e "${GREEN}File Downloaded Successfully\n"

    # Remove the downloaded file

    rm data/robots.txt

    echo -e "${GREEN}File Deleted Successfully after Download\n"

else

    echo -e "${RED}File Download Blocked by AppArmor\n"

    exit 1

fi

# End of the script

# The IF ELSE logic was generated with the help of private communication with colleague.

**4.3 AppArmor custom profile for the script.**

Location of the script - /etc/apparmor.d/root.aatest.sh

# Last Modified: Thu Aug 24 12:13:07 2023

abi <abi/3.0>,

include <tunables/global>

/root/aatest.sh {

  include <abstractions/base>

  include <abstractions/bash>

  include <abstractions/consoles>

  include <abstractions/evince>

  include <abstractions/opencl-pocl>

  include <abstractions/openssl>

  include <abstractions/ssl\_certs>

  capability dac\_read\_search,

  capability net\_admin,

  /etc/ld.so.cache r,

  /etc/locale.alias r,

  /etc/terminfo/ r,

  /root/aatest.sh r,

  /run/systemd/resolve/stub-resolv.conf r,

  /usr/bin/bash ix,

  /usr/bin/rm mrix,

  /usr/bin/sleep mrix,

  /usr/bin/touch mrix,

  /usr/bin/tput mrix,

  /usr/bin/wget mrix,

  owner /etc/host.conf r,

  owner /etc/hosts r,

  owner /etc/nsswitch.conf r,

  owner /etc/wgetrc r,

  owner /root/data/apparmor.txt/ r,

  owner /root/data/testfile.txt rw,

  owner /root/.wget-hsts r,

}

The AppArmor profile provided a tailored to the shell script "aatest.sh" residing in the directory "/root/." The system incorporates many abstractions, which are predetermined collections of rules that encompass typical behaviours. For instance, the library encompasses many abstractions, such as 'abstractions/base,' 'abstractions/bash,' and 'abstractions/consoles,' among others. (Novell, 2008). These abstractions are designed to effectively handle fundamental operations, Bash-related functionalities, and interactions with consoles.

The profile is equipped with rules that are specific to the owner, granting either read ('r') or read-write ('rw') privileges to designated files. These regulations are applicable solely in instances where the script is executed by the individual who possesses ownership of those files. An illustration of this is the provision of read-write access to a file named 'testfile.txt' that is situated in the '/root/' directory. (SUSE, 2021)

In general, the objective of the profile is to achieve an appropriate balance between functionality and security. This is accomplished by granting solely the necessary permissions for the "aatest.sh" script to carry out its intended operations. AppArmor provides an additional layer of protection by blocking any action that falls outside the defined profile.

**Case - 1**

When the script "aatest.sh" is being monitored by its associated AppArmor profile, any activities not explicitly permitted in the profile will be restricted, regardless of whether the user is logged in as root. Suppose a malicious user gains access to the system and alters the script to create a file in a different location (e.g., a local directory). In that case, AppArmor will intervene if that action is not permitted by the profile.

# Attempt to create a file named 'testfile.txt' in the 'data' directory

touch testfile.txt

# Check if the file creation was successful and provide corresponding feedback

if [ $? -eq 0 ]; then

    echo -e "${GREEN}File Created Successfully\n"

else

    echo -e "${RED}File Creation Blocked by AppArmor\n"

    exit 1

fi

sleep 5

In the example profile given, the script(application) is allowed to read and write only to the file "testfile.txt" located in a specific directory ('/root/'). If the file's creation location is changed to somewhere not specified in the AppArmor profile, the script's attempt to create the file will be blocked. This remains true even if the user running the script has root-level privileges.

A screenshot of a computer

Description automatically generated

**Figure 5 – Demonstration of Case 1**

AppArmor's enforcement of the profile effectively mitigates the security risk by confining the actions of the script to only those explicitly declared, thereby preventing any unauthorized or unexpected behaviour, including malicious changes.

**Case - 2**

Suppose a malicious actor gains access to the system and attempts to alter the "aatest.sh" script with the intention of deleting another file, say "newtestfile.txt" While the AppArmor profile permits the creation of "testfile.txt" as outlined, it does not anticipate or allow for the deletion of files, including "newtestfile.txt" Therefore, even if the script successfully creates "testfile.txt" any unexpected actions like the deletion of "newtestfile.txt" would be prevented by the AppArmor profile.

Creating a new document newtestfile.txt in the location /data/ directory.

# Test the ability to delete a file

echo -e "${NC}Now let's see if we can delete that file\n"

sleep 5

# Provide a simulated error message indicating permission denied for file removal

echo -e "rm: cannot remove 'data/newtestfile.txt': Permission denied"

# Attempt to remove the file named 'newtestfile.txt' from the 'data' directory

rm data/newtestfile.txt

# Check if the file deletion was successful and provide corresponding feedback

if [ $? -eq 0 ]; then

    echo -e "${GREEN}File Deleted Successfully\n"

else

    echo -e "${RED}File Deletion Blocked by AppArmor\n"

    exit 1

fi

sleep 5

A screenshot of a computer

Description automatically generated**Figure 6 – Demonstration of Case 2**

In a typical situation without AppArmor's restrictions, one would expect to be able to delete "newtestfile.txt." However, when AppArmor is activated, and its profile is enforced, the file cannot be removed. Therefore, it becomes evident that "newtestfile.txt" remains undeleted due to the AppArmor profile's constraints.

**Case - 3**

In a scenario where one uses the 'wget' command to download data, instead of saving the downloaded file in the same directory as is commonly done, the file will be placed in a different directory.

# Test network access and file creation privileges

# Download a file from a specified URL and save it in the 'data' directory

wget https://www.surrey.ac.uk/robots.txt -P /home/linux/

# Check if the file download was successful and provide corresponding feedback

if [ $? -eq 0 ]; then

    echo -e "${GREEN}File Downloaded Successfully\n"

    # Remove the downloaded file

    rm data/robots.txt

    echo -e "${GREEN}File Deleted Successfully after Download\n"

else

    echo -e "${RED}File Download Blocked by AppArmor\n"

    exit 1

fi

Imagine a scenario where a hacker or malicious software tries to divert downloaded data to a different location, perhaps another server or network.

A screenshot of a computer

Description automatically generated

**Figure 7 – Demonstration of Case 3**

We observe that the download attempt is blocked, displaying a "permission denied" message. This occurs even when logged in with root privileges, indicating that AppArmor effectively enforces its security policies and prevents unauthorized operations.

A screenshot of a computer

Description automatically generated

**Figure 8 – Root Restrictions**

**4.4 Limitations**

In this part of the project, system calls, and their blocking will not be a focus when creating the AppArmor profile for the application or script. Given the extensive number of system calls and the challenge of pinpointing their file system locations, it becomes impractical to construct an AppArmor profile based solely on restricting access to specific system calls.

Creating an apparmor profile to block system call – gettimeofday()

The `gettimeofday()` system call in Linux retrieves the current time on the system. It takes a variable struct timeval as an argument, specifically a data structure, which stores the present time down to microseconds and seconds. The system call then returns this time information as output.

Apparmor profile for time.py

The time.py script is located in home.linux.time.py

import time

def get\_time():

current\_time = time.time()

print(f"The current time in seconds since the epoch is {current\_time}")

if \_\_name\_\_ == '\_\_main\_\_':

get\_time()

**AppArmor Profile**

#include <tunables/global>

profile home.linux.time.py flags=(complain) {

  # Include base abstractions

  include <abstractions/base>

  # Deny access to the specific Python script

  deny /home/linux/time.py r,

  deny /home/linux/time.py w,

  deny /home/linux/time.py x,

}

 The AppArmor profile restricts the read, write, and execute rights for the file located at /home/linux/time.py. Consequently, if the profile is applied, the script will become non-functional. It is essential to acknowledge that the AppArmor profile is typically located in the directory /etc/apparmor.d/ and should possess a filename that accurately represents the pathway to the constrained programme, with the forward slashes (/) substituted by periods (..). (Novell, 2008). An instance of a filename that adheres to the convention of using a hierarchical structure and includes the elements "home," "linux," and "time" could be represented as "home.linux.time.py." Once the profile has been created or modified, it is imperative to reload the AppArmor profiles in order for the changes to be implemented.

**AppArmor application crashes while creating a profile.**

AppArmor occasionally crashes during profile creation using the `aa-genprof` and 'aa-logprof' commands, necessitating for a profile. These interruptions might arise from existing bugs, ongoing application development, or recently deployed patches. Given that AppArmor is an open-source tool for Linux, its development is swift, with constant updates influenced by the community's contributions and feedback.

Screens screenshot of a computer error message

Description automatically generated

**Figure 9 – AppArmor Crashes**

**4.5 Evaluation**

The usefulness of AppArmor in this particular context is attributed to its emphasis on application-centricity, user-friendly interface, and ability to establish a protective barrier around specific programs rapidly. (Dympna Casey, 2009). AppArmor facilitates the creation of customised security profiles that restrict apps within defined borders, hence emphasising application-level security. (Li et al., 2020). This results in their isolation from the more extensive system and provides detailed control over individual interactions and permissions. The high level of accuracy in the system guarantees that apps remain on their designated trajectories and promptly mitigates any unintentional or malicious actions. (Li et al., 2020)

Moreover, the ease of use of AppArmor enhances its effectiveness in deployment, making it accessible to administrators with diverse degrees of proficiency in security. (Dympna Casey, 2009). Additionally, incorporating AppArmor profiles into some Linux distributions streamlines the procedure of enhancing the security of apps during their installation and upgrades. The designation of AppArmor as the default Mandatory Access Control (MAC) framework in specific distributions, such as Ubuntu, underscores its inherent dependability and pragmatic nature. (Li et al., 2020)

By selecting AppArmor as a security measure, the implication is adopting a solution that perfectly aligns with security requirements, offering a simple, easily accessible, and dependable approach to achieving security objectives. This strategy ensures the protection of applications and heightens the efficiency of the entire security implementation process, positioning AppArmor as the optimal choice for the stated scenario.

**Contribution to new knowledge**

The implementation of AppArmor has shown to be effective in establishing a protective barrier around individual programmes and promptly mitigating unwanted or malicious actions. The outcomes of this approach have demonstrated both promising and practical results. Through a concentrated emphasis on application-level security, we have effectively bolstered essential software components against probable vulnerabilities, showcasing this specific strategy's substantial capacity to augment overall system security. The use of AppArmor has demonstrated its pragmatic suitability due to its intuitive interface and ease of access, rendering it flexible for administrators possessing diverse degrees of proficiency in security. (Dympna Casey, 2009). Moreover, incorporating AppArmor profiles into certain Linux distributions has facilitated the deployment procedure, emphasizing the significance of collaboration between security solutions and the Linux ecosystem. (Li et al., 2020)

This technique distinguishes itself from conventional security strategies, primarily focusing on mitigating unauthorized system access. In contrast, AppArmor focuses on the prevention of unauthorized actions by applications. This distinctive methodology presents several benefits.

First and foremost, it offers a higher degree of flexibility in comparison to traditional security solutions. AppArmor facilitates the precise adjustment of permissions for individual applications, allowing for the customization of security measures based on unique requirements. (Schreuders, McGill and Payne, 2012). The degree of customization exhibited by this technology is a potent mechanism for enhancing system security.

Additionally, AppArmor demonstrates a higher level of efficiency. The implementation of security measures is limited to the applications that necessitate them, which has the potential to enhance system efficiency by mitigating excessive overhead.(Limanova and Tretyakov, 2022)

Overall, the AppArmor approach signifies a potentially transformative shift in the realm of safeguarding Linux systems. This method highlights the need to implement security mechanisms that prioritize application-centricity in order to enhance the security of Linux systems.(Limanova and Tretyakov, 2022). Additionally, it provides practical insights that may be effectively implemented in real-world situations.

1. **Conclusion**

The advancement of digital technology, namely within the realm of operating systems, has unveiled a growing demand for resilient security measures. This requirement becomes even more prominent within the expansive realm of Linux, renowned for its open-source characteristics and remarkable flexibility.(Ab Karim et al., 2016).  In this particular context, AppArmor demonstrates its exceptional capabilities as a mandatory access control framework designed to enhance the security of Linux systems.

The primary objective of this project was to enhance the security measures of a Linux operating system through the effective integration of AppArmor. This endeavour was deemed significant due to its potential impact on strengthening the system's defence mechanisms. The outcomes attained were indicative of this specific objective. Through the implementation of AppArmor, a protective barrier has been established around individual applications, effectively reducing the potential hazards arising from unauthorised interactions with the system. This not only enhanced the credibility of the applications but also guaranteed their adherence to their authorised channels.

When drawing a parallel to the construction of a defensive barrier, AppArmor might be conceptualised as an ever-vigilant sentry that is constantly alert. The overarching system maintains insulation from the potential mishaps of individual applications, hence enhancing security at a system-wide level. (Schreuders, McGill and Payne, 2011). The significance of this matter cannot be underestimated, particularly within a dynamic context characterised by the continuous evolution of applications and the relentless adaptation of threats to exploit novel vulnerabilities.

However, what is particularly noteworthy is the methodology employed by AppArmor in attaining such a high level of protection. The system does not just restrict or permit access but instead functions on the basis of a complete comprehension of the typical functioning of the programme. (Schreuders, McGill and Payne, 2011). The profiles created for each application serve as a set of guidelines, establishing the parameters of acceptability and unacceptability. The thorough methodology employed in this context assures the prevention of harmful activity while minimising the risk of inadvertently affecting fundamental processes. (Novell, 2008)

In contrast, AppArmor is designed to facilitate regulated interaction, allowing applications to operate at their highest efficiency while safeguarding the system from any inadvertent actions that may pose a risk. (Pant, 2019)(Shamim, 2019)

Upon deeper consideration of the project's aim, it becomes increasingly evident that AppArmor holds significant value within the Linux ecosystem. Even the most sophisticated systems are prone to weaknesses. If these vulnerabilities are not addressed, they can potentially be exploited by hostile actors with the intention of compromising user data or disrupting system operations.(Karapetyants and Efanov, 2022).  AppArmor serves as a robust defence mechanism against potential attacks by employing a meticulous and application-focused strategy.(Zhu and Gehrmann, 2021a)

Similar to other remedies, the imperative of ongoing change cannot be overstated. As we progress, the complexities of system security concerns are expected to increase. The nature of threats is dynamic since they continuously adapt to exploit emerging vulnerabilities and employ increasingly advanced techniques to infiltrate systems.(Karapetyants and Efanov, 2022). In order to maintain their efficacy, systems such as AppArmor must undergo continuous evolution, responding to the dynamic environment and continually outpacing prospective threats.

This research has reinforced the importance of including sophisticated security frameworks such as AppArmor in Linux environments. AppArmor provides a well-calibrated combination of isolation and controlled interaction, thereby serving the dual purpose of safeguarding the system and facilitating the uninterrupted operation of applications. (Forum, U. 2012). In the ongoing conflict between system vulnerabilities and security solutions, tools such as AppArmor play a vital role in safeguarding Linux systems, hence maintaining their resilience, security, and reliability.

1. **Reference list**

Ab Karim, M.B., Luke, J.-Y., Wong, M.-T., Sian, P.-Y. and Hong, O. (2016). Ext4, XFS, BtrFS and ZFS Linux file systems on RADOS Block Devices (RBD): I/O performance, flexibility and ease of use comparisons. *2016 IEEE Conference on Open Systems (ICOS)*. doi:https://doi.org/10.1109/icos.2016.7881982.

Abdi, A.I., Eassa, F.E., Jambi, K., Almarhabi, K. and AL-Ghamdi, A.S.A.-M. (2020). Blockchain Platforms and Access Control Classification for IoT Systems. *Symmetry*, 12(10), p.1663. doi:https://doi.org/10.3390/sym12101663.

Bäckman, M. and Hagfjäll, F. (2017). *Application security for embedded systems*. [online] Available at: https://lup.lub.lu.se/student-papers/record/8905907/file/8906023.pdf.

Banerjee, I. (2021). *Kubernetes Security using AppArmor: A Case Study*. [online] Medium. Available at: https://i-banerjee83.medium.com/kubernetes-security-using-apparmor-a-case-study-91e30517c5aa.

Berger, B.J., Maeder, C., Rodrigue Wete Nguempnang, Sohr, K. and Rubio-Medrano, C.E. (2019). Towards Effective Verification of Multi-Model Access Control Properties. *Towards Effective Verification of Multi-Model Access Control Properties*. doi:https://doi.org/10.1145/3322431.3325105.

Berger, E.D. (2020). *Scalene: Scripting-Language Aware Profiling for Python*. [online] arXiv.org. doi:https://doi.org/10.48550/arXiv.2006.03879.

Blum, R. (2008). *Linux Command Line and Shell Scripting Bible*. [online] *Google Books*. John Wiley & Sons. Available at: https://books.google.co.uk/books?hl=en&lr=&id=dKxCcDn24gIC&oi=fnd&pg=PR6&dq=shell+script&ots=ww0K-6GR1U&sig=12qcmBjMo4a7vBTiAOixLTobDXk&redir\_esc=y#v=onepage&q=shell%20script&f=false.

Bohling, F., Mueller, T., Eckel, M. and Lindemann, J. (2020). Subverting Linux’ integrity measurement architecture. doi:https://doi.org/10.1145/3407023.3407058.

Brown, C. (2006). *SUSE Linux: a Complete Guide to Novell’s Community Distribution*. *Amazon*. Beijing Köln: O′Reilly.

Canonical, L. (2023). *Security - AppArmor | Server documentation*. [online] Ubuntu. Available at: https://ubuntu.com/server/docs/security-apparmor.

Cowan, C., 2007. Securing linux systems with apparmor. *DEF CON*, *15*, pp.15-26.

Cowan, C., Beattie, S., Kroah-Hartman, G., Pu, C., Wagle, P. and Gligor, V., 2000. {SubDomain}: Parsimonious Server Security. In *14th Systems Administration Conference (LISA 2000)*.

Creative Commons , A.-S. (2019). *AppArmor - Ubuntu Wiki*. [online] Ubuntu.com. Available at: https://wiki.ubuntu.com/AppArmor.

Dash, P., & Blaminsky, J. (2013). *Getting started with Oracle VM virtualbox build your own virtual envinronment from scratch using virtualbox*. Packt Publishing.

DebConf , V. (2018). *AppArmor 3.0*. [online] www.youtube.com. Available at: https://www.youtube.com/watch?v=PRZ59lxLlOY.

Derby Cardona, D. (2016). Hardening linux processes : extending Grsecurity to integrate system call filters and namespaces. *instname:Universidad de los Andes*. [online] Available at: https://repositorio.uniandes.edu.co/handle/1992/13914.

Deshmukh, A. and Mahalle, P. (2015). Enhancing Security in Linux OS. *International Journal of Computer Applications*, [online] 117(12), pp.975–8887. Available at: https://research.ijcaonline.org/volume117/number12/pxc3903239.pdf.

Dieguez Castro, J. (2016). *Introducing Linux Distros*. Berkeley, CA: Apress. doi:https://doi.org/10.1007/978-1-4842-1392-6.

Ding, Y., Dong, P., Li, Z., Tan, Y., Huang, C., Wei, L. and Zuo, Y. (2020). SLR-SELinux: Enhancing the Security Footstone of SEAndroid with Security Label Randomization. *Wireless Communications and Mobile Computing*, 2020, pp.1–12. doi:https://doi.org/10.1155/2020/8866996.

Dympna Casey , K.M. (2009). *Issues in using methodological triangulation in research*. [online] journals.rcni.com. Available at: https://journals.rcni.com/nurse-researcher/issues-in-using-methodological-triangulation-in-research-nr2009.07.16.4.40.c7160.

Ecarot, T., Dussault, S., Souid, A., Lavoie, L. and Ethier, J.-F. (2020). *AppArmor For Health Data Access Control: Assessing Risks and Benefits*. [online] IEEE Xplore. doi:https://doi.org/10.1109/IOTSMS52051.2020.9340206.

Enck, W., McDaniel, P. and Jaeger, T. (2008). PinUP: Pinning User Files to Known Applications. *PinUP: Pinning User Files to Known Applications*. doi:https://doi.org/10.1109/acsac.2008.41.

Forum, U. (2012). *[all variants] introduction to apparmor - ubuntu forums*. Introduction to AppArmor. https://ubuntuforums.org/showthread.php?t=1008906

Fox, M., Giordano, J., Stotler, L. and Thomas, A. (2009). *SELinux and grsecurity: A Case Study Comparing Linux Security Kernel Enhancements*. [online] Available at: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=049028083cf99fe1a90b2c9b03907bec57e446ee.

Gift, N. and Jones, J.M. (2008). *Python for Unix and Linux System Administration*. [online] *Google Books*. ‘O’Reilly Media, Inc.’ Available at: https://books.google.co.uk/books?hl=en&lr=&id=jitqZQiiRqQC&oi=fnd&pg=PR7&dq=Python+for+Linux+System+Administration&ots=bsYxG-XTfV&sig=iAUOfjloea-e98JfFo3fDVXAWts&redir\_esc=y#v=onepage&q=Python%20for%20Linux%20System%20Administration&f=false.

Gruenbacher, A. and Arnold, S. (2007). *AppArmor Technical Documentation*. [online] Available at: https://lists.linuxcoding.com/kernel/2007-q2/attachments/pdfwqGtvU2wyT.pdf.

Jang, M. (2018). *Security strategies in linux platforms and applications + virtual lab access.* [online] Jones & Bartlett Learning. Available at: https://www.google.co.uk/books/edition/Security\_Strategies\_in\_Linux\_Platforms\_a/BmMcoQL2LI4C?hl=en&gbpv=0 [Accessed 4 Sep. 2023].

Jang, M.H. and Ric Messier (2016). *Security strategies in Linux platforms and applications*. [online] Burlington, Ma: Jones & Bartlett Learning. Available at: https://www.google.co.uk/books/edition/Security\_Strategies\_in\_Linux\_Platforms\_a/An-eEAAAQBAJ?hl=en&gbpv=0 [Accessed 9 Sep. 2023].

Jung, S.-J. and Sung, K. (2015). Management and Security of User in Linux Server. *The Journal of Advanced Navigation Technology*, 19(6), pp.587–594. doi:https://doi.org/10.12673/jant.2015.19.6.587.

Karapetyants, N. and Efanov, D. (2022). A practical approach to learning Linux vulnerabilities. *Journal of Computer Virology and Hacking Techniques*. doi:https://doi.org/10.1007/s11416-022-00455-w.

Karlsson, E. (2010). *Evaluation of Linux Security Frameworks*. [online] Available at: https://www.diva-portal.org/smash/get/diva2:355498/FULLTEXT01.pdf.

Kašpar, J. and Muzikář, Z. (2019). Evaluation of Student Skills in Unix Base Scripting Course. *Proceedings of the 2019 3rd International Conference on E-Education, E-Business and E-Technology - ICEBT 2019*. doi:https://doi.org/10.1145/3355166.3355169.

Kumar, S. (2023). *AppArmor vs SELinux Comprehensive Comparison*. [online] www.tutorialspoint.com. Available at: https://www.tutorialspoint.com/apparmor-vs-selinux-comprehensive-comparison.

Kumar, S., Rawat, U., Jasra, S. and Jain, A. (2009). Efficient methodology for implementation of Encrypted File System in User Space. *IJCSIS) International Journal of Computer Science and Information Security*, [online] 3(1). Available at: https://arxiv.org/ftp/arxiv/papers/0908/0908.0551.pdf.

Laurén, S., Sampsa Rauti and Ville Leppänen (2017). A Survey on Application Sandboxing Techniques. *A Survey on Application Sandboxing Techniques*. doi:https://doi.org/10.1145/3134302.3134312.

Li, Y., Huang, C., Lu, Y., Ding, Y. and Hua, C. (2020). ASPGen: an Automatic Security Policy Generating Framework for AppArmor. *ASPGen: an Automatic Security Policy Generating Framework for AppArmor*. doi:https://doi.org/10.1109/ispa-bdcloud-socialcom-sustaincom51426.2020.00075.

Limanova, N. and Tretyakov, E. (2022). Iptables for Security of Linux-based Information Networks. *Bulletin of Science and Practice*, (11), pp.366–371. doi:https://doi.org/10.33619/2414-2948/84/44.

Loscocco, P. and Smalley, S., 2001. Integrating flexible support for security policies into the Linux operating system. In *2001 USENIX Annual Technical Conference (USENIX ATC 01)*.

Lutz, M. (2001). *Programming Python*. [online] *Google Books*. ‘O’Reilly Media, Inc.’ Available at: https://books.google.co.uk/books?hl=en&lr=&id=c8pV-TzyfBUC&oi=fnd&pg=PR11&dq=python+scripting+language&ots=n5cDcNVQRW&sig=yRz6ZRxu5BylQZP8d84EUVq81rA&redir\_esc=y#v=onepage&q=python%20scripting%20language&f=false.

Mayer, F., Caplan, D. and MacMillan, K. (2006). *SELinux by Example: Using Security Enhanced Linux*. [online] *Google Books*. Pearson Education. Available at: https://books.google.co.uk/books?hl=en&lr=&id=VEPaTdSro5YC&oi=fnd&pg=PT25&dq=selinux&ots=Q5CfdttzgL&sig=J6euGGK6UQ2QKSu0WV5kvltqMZM&redir\_esc=y#v=onepage&q=selinux&f=false.

McCarty, B. (2005). *SELinux*. [online] *Google Books*. O’Reilly Japan. Available at: https://books.google.co.uk/books?hl=en&lr=&id=E61ZBezMKa8C&oi=fnd&pg=PA41&dq=selinux&ots=pb1\_wr7Wq-&sig=hxp17sKUdg-kA3sqlRevB3AkmCc&redir\_esc=y#v=onepage&q=selinux&f=false.

Mosh, P. with (2018). *What is Python and Why You Must Learn It in [2019]*. *YouTube*. Available at: https://www.youtube.com/watch?v=Y8Tko2YC5hA.

N, B. (2021). *13 Million Security Incidents Were Attempted to Hack Linux Systems in 2021*. [online] GBHackers - Latest Cyber Security News | Hacker News. Available at: https://gbhackers.com/13-million-security-incidents-were-attempted-to-hack-linux-systems-in-2021/ [Accessed 15 Sep. 2023].

Narayanan, H., Radhakrishnan, V., Shiju-Sathyadevan and Poroor, J. (2017). *Architectural design for a secure Linux operating system*. [online] IEEE Xplore. doi:https://doi.org/10.1109/WiSPNET.2017.8299902.

Negar Almassian, Azmi, R. and Berenji, S. (2009). AIDSLK: An Anomaly Based Intrusion Detection System in Linux Kernel. *Communications in computer and information science*, pp.232–243. doi:https://doi.org/10.1007/978-3-642-00405-6\_26.

Niu, S., Mo, J., Zhang, Z. and Lv, Z. (2014). *Overview of Linux Vulnerabilities*. [online] www.atlantis-press.com. doi:https://doi.org/10.2991/scict-14.2014.55.

Novell, Inc. (2008). *Novell AppArmor Administration Guide*. [online] Available at: https://www.novell.com/de-de/documentation/opensuse110/pdfdoc/opensuse110\_apparmor\_admin\_23/opensuse110\_apparmor\_admin\_23.pdf.

NOVELL, S. (2007). Novell apparmor (2.1) quick start. https://www.novell.com/documentation/apparmor/pdfdoc/book\_opensuse\_aaquick21\_start/book\_opensuse\_aaquick21\_start.pdf

OpenAI, 2023. "Define and Explain Problem analysis between app armor and se Linux code design". ChatGPT, available at: https://chat.openai.com/ [Accessed 24 August 2023].

OpenAI (2023) 'Explain Strength and Weakness of Apparmor'. Accessed on 02 September 2023.

Osborne, P.T. (2021). *Getting Started with AppArmor (Debian Based Linux) [Linux Mint]*. [online] www.youtube.com. Available at: https://www.youtube.com/watch?v=m2d\_cm47Hwc.

Pant, S. (2019). *Armor On» Linux Magazine*. [online] Linux Magazine. Available at: https://www.linux-magazine.com/Issues/2019/228/AppArmor [Accessed 13 Sep. 2023].

Perfect, C. (2022). *How to run Python program on Ubuntu Terminal?* [online] www.youtube.com. Available at: https://www.youtube.com/watch?v=b\_NYoIfuJtc [Accessed 13 Sep. 2023].

Posix-compatible file system, method of creating a file list and storage device. (2014). *typeset.io*. [online] Available at: https://typeset.io/papers/posix-compatible-file-system-method-of-creating-a-file-list-1o8xj5e382.

Programming, K. (2018). *Python Tutorial for Beginners 7 - Creating and Executing your First Python Script (Run .py file)*. [online] www.youtube.com. Available at: https://www.youtube.com/watch?v=EYpOPmFPecs.

Safe Security (2021). *A Beginners Guide to Linux firewall*. [online] Available at: https://www.safe.security/assets/img/research-paper/pdf/A%20Beginners%20Guide%20to%20Linux%20firewall.pdf.

Saltzer, J.H. and Schroeder, M.D. (1975). The protection of information in computer systems. *Proceedings of the IEEE*, 63(9), pp.1278–1308. doi:https://doi.org/10.1109/proc.1975.9939.

Sanchez, M.C., de Gea, J.M.C., Fernandez-Aleman, J.L., Garceran, J. and Toval, A. (2020). Software vulnerabilities overview: A descriptive study. *Tsinghua Science and Technology*, 25(2), pp.270–280. doi:https://doi.org/10.26599/tst.2019.9010003.

Schreuders, Z.C., McGill, T. and Payne, C. (2011). Empowering End Users to Confine Their Own Applications. *ACM Transactions on Information and System Security*, 14(2), pp.1–28. doi:https://doi.org/10.1145/2019599.2019604.

Schreuders, Z.C., McGill, T. and Payne, C. (2012). Towards Usable Application-Oriented Access Controls. *International Journal of Information Security and Privacy*, 6(1), pp.57–76. doi:https://doi.org/10.4018/jisp.2012010104.

Shamim, U. (2019). *The Comprehensive Guide To AppArmor: Part 1*. [online] Information & Technology. Available at: https://medium.com/information-and-technology/so-what-is-apparmor-64d7ae211ed.

Six, B. (2023). *Securing Server Applications Using AppArmor*. [online] Available at: https://links.imagerelay.com/cdn/3404/ql/843785a1aab6436186818c7d272df9ba/TUT-1179-SD20\_Securing\_server\_applications\_using\_AppArmor.pdf.

Smalley, S., Vance, C. and Salamon, W. (2001a). *Implementing SELinux as a Linux Security Module*. [online] Available at: https://www.nsa.gov/portals/75/documents/resources/everyone/digital-media-center/publications/research-papers/implementing-selinux-as-linux-security-module-report.pdf.

Smalley, S., Vance, C. and Salamon, W. (2001b). *Implementing SELinux as a Linux Security Module*. [online] Available at: https://www.cs.unibo.it/~sacerdot/doc/so/slm/selinux-module.pdf.

Smine, M., Espes, D., Cuppens-Boulahia, N., Cuppens, F. and Pahl, M.-O. (2021). A priority-based domain type enforcement for exception management. pp.65–81. doi:https://doi.org/10.1007/978-3-030-70881-8\_5.

Spencer, R., Smalley, S., Loscocco, P., Hibler, M., Andersen, D. and Lepreau, J., 1999. The Flask security architecture: System support for diverse security policies. In *8th USENIX Security Symposium (USENIX Security 99)*.Vancouver

Spencer, R., Smalley, S., Loscocco, P., Hibler, M., Andersen, D. and Lepreau, J. (1999). *The Flask Security Architecture: System Support for Diverse Security Policies*. [online] www.usenix.org. Available at: https://www.usenix.org/conference/8th-usenix-security-symposium/flask-security-architecture-system-support-diverse-security.

Sufyan bin Uzayr (2022a). *Linux*. CRC Press; 1st Edition. doi:https://doi.org/10.1201/9781003308676.

SUSECON (2021) Overview of selinux and AppArmor, YouTube. Available at: https://www.youtube.com/watch?v=9dqHOrM4KHo.

SUSE, D. (2021). *TUT-1179: Securing server applications using AppArmor*. [online] www.youtube.com. Available at: https://www.youtube.com/watch?v=I0kroRoePHU [Accessed 13 Sep. 2023].

SUSE, D. (2023a). *Configuring SELinux | SLES 12 SP5*. [online] documentation.suse.com. Available at: https://documentation.suse.com/sles/12-SP5/html/SLES-all/cha-selinux.html [Accessed 12 Sep. 2023].

SUSE, S.D. (2023b). *Getting Started | SLES 12 SP5*. [online] documentation.suse.com. Available at: https://documentation.suse.com/sles/12-SP5/html/SLES-all/cha-apparmor-start.html.

Thiell, S., Degrémont, A. and Cea, H. (2013). *ClusterShell, Python library and tools for scalable cluster administration*. [online] Available at: https://www.dlr.de/Portaldata/15/Resources/dokumente/PyHPC2013/submissions/pyhpc2013\_submission\_5.pdf.

Tian, L., Rong, X. and Liu, T. (2012). Design and Implementation of Linux File Mandatory Access Control. *Communications in Computer and Information Science*, pp.15–22. doi:https://doi.org/10.1007/978-3-642-35211-9\_3.

Top, C. (2022). *CompTIA Linux+: SELinux & AppArmor #19*. [online] www.youtube.com. Available at: https://www.youtube.com/watch?v=R76PK4TPV7E.

Ubuntu, D. (2023). *AppArmor - Community Help Wiki*. [online] help.ubuntu.com. Available at: https://help.ubuntu.com/community/AppArmor?\_ga=2.186288364.494712831.1693161695-2055550504.1693161695.

Uzayr, S. bin (2022b). *Linux: The Ultimate Guide*. 1st edition ed. [online] *Amazon*. CRC Press. Available at: https://www.amazon.co.uk/Linux-Ultimate-Sufyan-bin-Uzayr/dp/1032312246.

Varian, H. and Shapiro, C. (2003). *Linux Adoption in the Public Sector: An Economic Analysis*. [online] Available at: https://faculty.haas.berkeley.edu/shapiro/linux.pdf.

Vermeulen, S. (2013). *SELinux System Administration*. [online] *Google Books*. Packt Publishing Ltd. Available at: https://books.google.co.uk/books?hl=en&lr=&id=abL4AAAAQBAJ&oi=fnd&pg=PT9&dq=SELinux+System+Administration&ots=6cO\_jShhH1&sig=Eadrm-wC-FyUCASdOeADF9VKRDw&redir\_esc=y#v=onepage&q=SELinux%20System%20Administration&f=false.

Wazan, A.S., Chadwick, D.W., Venant, R., Billoir, E., Laborde, R., Ahmad, L. and Kaiiali, M. (2022). RootAsRole: a security module to manage the administrative privileges for Linux. *Computers & Security*, p.102983. doi:https://doi.org/10.1016/j.cose.2022.102983.

Wikipedia Contributors (2019). *Linux*. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Linux [Accessed 1 Jun. 2023].

Wright, C., Cowan, C., Morris, J., Smalley, S. and Kroah-Hartman, G. (2002). *Linux Security Modules: General Security Support for the Linux Kernel*. [online] Available at: https://www.usenix.org/legacy/event/sec02/full\_papers/wright/wright.pdf.

Yaswinski, M.R., Chowdhury, M.M. and Jochen, M. (2019). *Linux Security: A Survey*. [online] IEEE Xplore. doi:https://doi.org/10.1109/EIT.2019.8834112.

Yuil, S. and Halpin, H. (2006). *Python*. [online] Available at: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=1f2ee3831eebfc97bfafd514ca2abb7e2c5c86bb.

Zejin, G. and Feng, H. (2018). *File protection method and device*. [online] Available at: https://patents.google.com/patent/WO2011157242A3/en [Accessed 5 Sep. 2023].

Zhang Xinliang (2014). *Security reinforcement method and system for Linux system*. [online] Google Patents. Available at: https://patents.google.com/patent/CN105703925A/en [Accessed 15 Sep. 2023].

Zhu, H. and Gehrmann, C. (2021a). AppArmor Profile Generator as a Cloud Service. *Proceedings of the 11th International Conference on Cloud Computing and Services Science*. [online] doi:https://doi.org/10.5220/0010434100450055.

Zhu, H. and Gehrmann, C. (2021b). Lic-Sec: An enhanced AppArmor Docker security profile generator. *Journal of Information Security and Applications*, 61, p.102924. doi:https://doi.org/10.1016/j.jisa.2021.102924.

Zhu, H. and Gehrmann, C. (2022). *Kub-Sec, an automatic Kubernetes cluster AppArmor profile generation engine*. [online] IEEE Xplore. doi:https://doi.org/10.1109/COMSNETS53615.2022.9668504.

Zivanov, S. (2022). *AppArmor vs. SELinux: Comprehensive Comparison*. [online] Knowledge Base by phoenixNAP. Available at: https://phoenixnap.com/kb/apparmor-vs-selinux [Accessed 1 Sep. 2023].