Using RBAC to Enforce the Principle of Least Privilege in Industrial Remote Maintenance Sessions

Alexander Kern and Reiner Anderl
Department of Computer Integrated Systems (DiK)
Technical University of Darmstadt
Darmstadt, Germany
Email: {kern,anderl}@dik.tu-darmstadt.de

***Abstract*—In recent years, digitalization is having a great impact on industry. Especially the rising degree of cross-linked machines is resulting in new business models and great economic advantages. One example is remote maintenance. It leads to less downtime and gained efficiency due to a quicker response time of highly qualified maintenance technicians. However, connecting industrial machines to WANs opens up new attack surfaces. In particular, machines with long lifetimes are often severely outdated since updates that might cause downtime stand in direct conflict to the central objective of availability. In case of remote maintenance, it additionally requires great trust in the external maintenance technician not to use the remote connection for wrong doing. On the market multiple solutions for securing remote maintenance sessions exist. However, these solutions mainly focus on network security and disregard system security entirely. Security doctrines such as the principle of least privilege should be used to enhance the system security.**

This paper focuses on system security of industrial machines and proposes the use of role based access control to confine users and attackers alike. This way, consequences of security breaches and wrong doing can be minimized. The scientific contribution is the development, implementation and assessment of a concept for the usage of RBAC on system level to solve current system security issues, with the focus on remote maintenance sessions.

***Index Terms*—access control, RBAC, system security, industrial remote maintenance, principle of least privilege**

I. Introduction

With the market is putting pressure on companies, in recent years, major investments have been made worldwide to modernize production systems and develop new technologies to ensure future economic competitiveness. Especially, the idea to connect physical systems with computing and communication infrastructure gained momentum [20]. In case of a standard machine maintenance, there exists an area of conflict between the qualification, cost and response time of a maintenance technician. With the possibility of remote maintenance this conflict can be reduced noticeably. Traveling time and cost become obsolete. Consequently, technicians have more time to actively perform maintenance tasks, resulting in a higher capacity utilization. This gained efficiency results in cheaper and faster maintenance by highly qualified technicians. Additionally, the machines suffer shorter downtime, adding yet another economic advantage.

These upsides are great, but so are the potential downsides. New vulnerabilities for Information Technology (IT) systems are being discovered every day and Information Security is needed to protect against them [22]. Information Security also called InfoSec is the practice to monitor and prevent any kind of threats such as unauthorized access, use, disclosure, disruption, modification, inspection, recording or destruction of digital or non-digital information. This paper focuses on the protection of digital information only. InfoSec centers around the preservation of confidentiality, integrity and availability (CIA) of information. Additional objectives are the authenticity, accountability, non-repudiation and reliability of information [17, 7]. It can be split into computer security, network security and system security. Computer Security is ensuring that the actual physical hardware of the resource is secure [5]. Network or Cyber Security describes the policies and procedures applied to monitor and prevent unauthorized access, misuse, exploitation or modification of a resource from the network the resource is connected to. System Security describes the policies and procedures applied to monitor and prevent unauthorized access, misuse, exploitation or modification of a resource within the resource itself.

While network security is very similar for both office systems and industrial systems and can be maintained quite well, system security differs a lot between the two. For office systems system security is mainly focused on confidentiality and integrity. The systems' availability is not as critical and if an update has to be installed, individual computers are expendable. Oftentimes, office computers will update over night when they are not in use. This way, the security can be maintained quite well without having an impact on productivity. For production systems the system security often lacks behind which leads to major risks for the company. Especially, connecting legacy systems to the Internet and allowing connections to the system, which is a basic requirement for remote maintenance, leads to new attack scenarios against companies. Vulnerabilities in single machines can be very valuable for an attacker. Once in the production network, broadening rights and spreading onto other machines becomes much simpler. As a result, enormous damage can be caused.

The lacking system security has two main reasons. The first reason is, as already mentioned, the very different focus of IT security requirements. The most important requirement for production systems is safety. It ensures that neither humans or the environment suffer any harm. Any kind of security violation, both intentional and unintentional may potentially harm the plant safety. The second most important requirement is the availability of the systems. Systems have to be very robust since any kind of downtime is expensive. Unfortunately, these requirements contradict most every standard IT security administration process such as updates and software patches, where a reboot of the system is needed or unstable behavior might be caused.

The second reason is that the lifetime of production systems is much longer than the one of software systems [28]. Many security methods, algorithms and systems that seemed sufficiently secure a couple of years ago are known to be insecure today. One particular design principle often used in legacy systems is the "Security by Obscurity" principle. It aims at providing security using proprietary protocols and systems under the assumption that no attacker can reverse engineer the system due to its complexity. Without being connected to any kind of public wide area network (WAN), the main "security" goal was to protect the system from unintentional mistakes much rather than intentional attacks and thus did not require as much security. Nevertheless, for todays interconnected production systems, security plays a major role and securing both old systems and designing new ones securely, poses a substantial challenge. Most legacy systems use outdated operating systems and software where support has been discontinued for years. Even if wanted, it would not be possible to fully update and patch them. The problem is, that these machines often have considerable value and cannot easily be replaced. Since it is not possible to completely prevent vulnerabilities from being exploited, a method to minimize the damage of an exploitation is needed.

On top of protecting against attackers, a method to protect the know-how stored on the machine is needed. Today, the maintenance of machines is often done by the machine's manufacturer whose service technicians have full access to the system (root). However, the business model of maintenance by external technicians is becoming more popular. These external technicians also use the same service account with full system access. Sensitive data can easily be stolen or the machine can be manipulated. This is quite problematic as of today, but will be even more so in the future. Visionaries say that future industrial machines will be multi-vendor products with the capability to upgrade its functionalities by adding new hardware modules and new software applications. This development is advanced through the standardization of both hardware interfaces [31] and software components [30], [21]. If both hardware and software components have multiple different manufacturers there will not be one single remote maintenance technician anymore. Consequently, a service technician must not have full access, but must be confined within the system.

The remainder of this paper is organized as follows. In section II, the problem is described in the context of industrial remote maintenance and a research question is raised. Section III gives a brief overview about the state of the art on RBAC, discusses its usage in other contexts and outlines the concept as well as tools needed for the implementation. Section IV explains how the concept is implemented. In section V the implementation is tested and the results are discussed. Furthermore, limitations and possible improvements are considered. Last, in section VI a conclusion is drawn.

II. Problem Statement and Research Question

Until recently, the focus for remote maintenance systems was on remote desktop applications for office IT only. However, with increasingly complex production systems and their connection to public networks, industrial remote maintenance is gaining importance. While there do not exist any norms or standards for industrial remote maintenance yet, the Federal Office for Information Security in Germany (BSI) released recommendations for such systems [11, 9, 10]. They suggest, depending on the size of the enterprise, a set of seven mandatory rules for securing remote maintenance sessions [9]:

1) Remote maintenance sessions should be initiated in-house.
2) A remote maintenance connection should be fully encrypted.
3) A remote maintenance service technician has to be authenticated prior to the connection establishment.
4) The system being maintained should be isolated from the rest of the production network.
5) To establish a remote maintenance session the modifications of the remaining network should be kept to a minimum.
6) The remote maintenance session should be logged.
7) The permissions granted to the service technician should be kept to the required minimum (principle of least privilege).

These rules can be subdivided into two categories. One concerning the network security and the other the system security. Most commercial solutions for industrial remote maintenance use modems or gateways to establish a secure connection into the production network, complying only with the network security requirements. The system security requirements, such as end-to-end encryption between devices, session-logging and user confinement, are mostly ignored. Especially BSI requirement number 7, "The permissions granted for the service technician should be kept to a minimum (principle of least privilege)", regarding user confinement is not implemented by any solution on the market at this point. They all rely on the existing access control mechanisms implemented on the systems. Access control in this sense, describes the selective restriction of access to information on a resource (authorization). An example would be if a user is allowed to read/write/execute a certain file or folder.

For most computer systems the standard authorization system is a Discretionary Access Control (DAC) system. In DAC systems, access decisions are based on permissions assigned to the identity of a subject (authorization-based). Permissions for a specific object are assigned or rather passed on by a subject with permissions for that object. In linux systems for example, the DAC system uses the chmod (change mode) system call to adjust access permissions of system objects such as files and directories. Each system object is assigned independent access rights for user (owner and usually creator of system object), group (users belonging to the group associated with the file) and others (users who are neither the owner nor member of the associated group). The access rights define if a certain user can read (read file or view directory), write (change/delete file or add/delete directory's contents) and execute (run/launch a file or search/access directory and subdirectories) the respective system object.

However, there are several disadvantages associated with DAC. Since any user who owns a system object can give globally valid permissions for it to any other user, it is difficult to ensure consistency across the system. This makes security policies very difficult to maintain and an audit of security principles becomes virtually impossible in systems with multiple users. Another problem inherent to DAC is called the Safety Problem [15]. Because of a lack of constrains in copy privileges, the information flow can not be controlled. Once access is given to a certain object, the user can copy its information without any limitations. These problems expose the system to vulnerabilities such as trojan horses and other malicious code that is exploiting authorizations. Once a vulnerability gains access to the privileged system, the entire system is compromised.

To perform a remote maintenance task, a service technician needs to be granted access to a variety of system objects, some of which may require administrative rights. In a DAC system, this requirement can only be met by a user account with full administrative rights. This contradicts the requirement that the permissions granted to a service technician are to be kept to the minimum necessary for the maintenance task.

This raises the research question of how users can be efficiently confined during industrial maintenance sessions enforcing the principle of least privilege?

III. Concept for using RBAC

A. Introduction to Role Based Access Control

Besides DAC, there exist multiple other access control mechanisms [8]. The two in practice most relevant types next to DAC, are Mandatory Access Control (MAC) [4], and Role-Based Access Control (RBAC) [13, 27]. In MAC, access decisions are based on policies controlled by a central security policy administration. Each subject and object is assigned security attributes that are compared when access is requested. DAC and MAC were first introduced in the early 1970's, while RBAC was first introduced by Ferraiolo et al. [13] and Sandhu et al. [27] about 20 years later to overcome some of the shortcomings of the DAC and MAC policy models. Quickly after its introduction, prototypical implementations followed [16]. Especially the use of RBAC in web-based server environments (e.g. corporate networks) [12] and HTTP environments [14, 25] quickly gained interest and sophisticated methods for role engineering were introduced [24, 6]. Today a vast variety of different applications an modifications exist and RBAC is broadly adapted in both research and industry. Nevertheless, almost no contributions are concerned with the use of RBAC to restrict access to industrial systems (e.g. for industrial remote maintenance), even though it is suited very well for this purpose.

In RBAC, access decisions are based on roles. Each role is allowed access to specific objects. Since one subject can be appointed multiple roles, this model scales very well with multiple users. RBAC itself is policy neutral and can be both implemented with the DAC or MAC paradigm [27, 26]. Besides resolving the issues mentioned for DAC, RBAC has some major advantages compared to DAC and MAC, supporting following security principles [19, 18, 2]:

- Least privilege: Only permissions required by the user to perform a given task are granted.
- Separation of duties: Mutually exclusive roles are needed to perform critical operations (enforcing four-eyes principle).
- Data abstraction: Roles combine specific system rights (read/write/execute) for certain system objects which is easier to understand.
- Authorization management: Straightforward to assign new roles and manage a user's field of responsibility.
- Hierarchical roles: Roles can include other roles and inherit the corresponding permissions.

Because of these advantages, this paper introduces the idea to use RBAC to restrict a user's access within an industrial control system for a remote maintenance session. The concept is developed on a linux system using frameworks, tools and features inherent to the system. These will be briefly explained in the following paragraphs.

B. Overview on the Linux Security Module

The Linux Security Module (LSM) is an access control framework in the linux kernel that enables different implementations of access control modules to be loaded as kernel modules [23]. This infrastructure leads to new security services for linux. Examples are AppArmor [3], SELinux [29], Smack and TOMOYO among others. The LSM tries to impose as few changes to the kernel as possible avoiding system call interpositions. Instead it uses upcalls to the kernel module, so called "hooks", whenever a user-level system call tries to access internal kernel objects. With the LSM integrated into the kernel, each system call has to pass three stages in the kernel space to achieve access to internal kernel objects. First an error check and a check if the process has the required capabilities is performed. Then, the DAC verifies that the user is allowed access. Finally, the LSM surveys the different security implementations, using its framework, if access is allowed. This process is shown in the segment "Kernel Space" in "Fig. 1".

C. Implementing RBAC with AppArmor

One very powerful, but yet simple implementation of such a security module is AppArmor. AppArmor is a pathname-

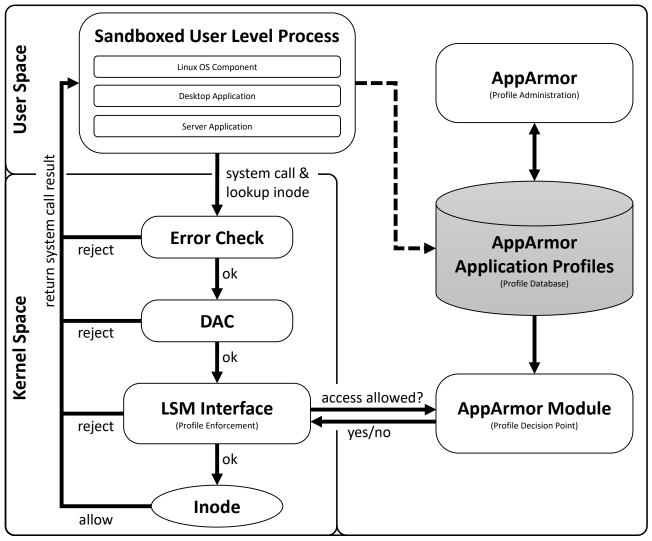

Fig. 1. AppArmor in combination with LSM.

based security system that enables MAC on linux systems. For each individual application a security profile defines which system resources are available to the application. The profiles use a "whitelist" approach to grant access to minimal system resources such as files and directories including the corresponding access rights (r - read, w - write, a - append, x - execute, k - lock and l - link) and POSIX capabilities needed by the application. The security of this proactive whitelist approach is based on the principle of least privilege and therefore does not depend on attack signature databases or else. Since AppArmor uses the pathname-based approach, it does not require any labeling or relabeling of the file system in contrast to the popular SELinux or similar implementations. Another advantage of this approach is the simplicity of pathnames, which make the policies easy to understand and audit. Because the profiles are loaded into the kernel and the enforcement is done on kernel level as well, AppArmor is able to protect the operating system and applications from both internal (programming error) and external (zero-day exploit) threats. How AppArmor in combination with the LSM works is shown in "Fig. 1".

Even though, AppArmor is primarily designed to use MAC to restrict the capabilities of applications, it is possible to implement RBAC with mandatory policies to confine individual users. AppArmor profiles usually confine single applications. In order to confine a user, the login program has to be confined. To do so, the user login has to be linked to a user individual instance of a login service (e.g. login shell). With an AppArmor profile, this login service can then be restricted to only resources necessary for the assigned task. However, this architecture makes it difficult to have one role apply to multiple users or one user have multiple roles as typically seen for RBAC systems. As a solution, role specific rule sets have to be defined. Depending on the user's roles, these rule sets can then be included into the profile for the user individual login service. This setup is shown in "Fig. 2".

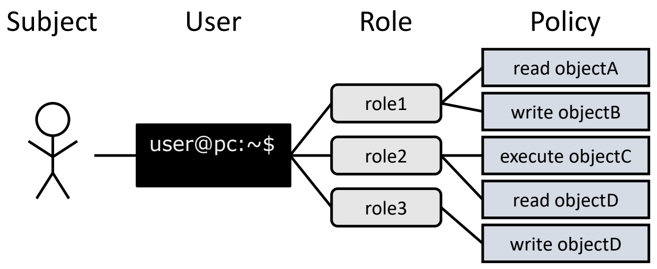

Fig. 2. Architecture of RBAC System.

With this architecture it is easily possible to assign a combination of roles to a specific user (e.g. maintenance technician) that are needed to fulfill a given task while conforming with the BSI requirement of least privilege.

IV. Implementation of the concept

The following section presents a prototypical implementation of the concept. Besides explaining the implementation of the architecture itself, an approach for the role assignment and automatic initiation of a maintenance session is introduced.

For the implementation a Raspberry Pi 3 model B+ is used to compare to the characteristics of a low-performance machine controller. The Pi runs Raspbian Stretch Lite, Kernel version 4.14. To use AppArmor it has to be installed and additionally enabled in the Kernel by recompiling it with the correct options enabled.

A new user account "service" is created for the service technician. The DAC rights should be extended to the maximum rights needed for any possible maintenance scenario. Since some maintenance tasks may require administrative rights, the user is therefore added to the sudoers group.

A special login shell is created (*sudo ln /bin/bash /usr/local/servicebash*) and referenced as such (*sudo echo /usr/local/servicebash >> /etc/shells*). After that it is specified as the standard login shell for the user service (*sudo chsh service*). Furthermore, the authentication method for ssh sessions is set to public-key authentication, so subjects can be uniquely identified. With this being setup, AppArmor can now be used to confine the servicebash and therefore the user service. To implement RBAC, role specific profiles are created. Since AppArmor is a pathname-based security system, rules can easily be created. The path to a specific system object is given followed by file access permissions (r, w, a, l, k, x). The role profiles are saved in the directory /etc/apparmor.d/roles. The typical structure of a role profile is as follows:

/etc/apparmor.d/roles/role1:

```
#include <abstractions/standard_resources>
/path/to/object1 rwalkx
/path/to/object2 rwx
```

Depending on the roles needed to work on a certain maintenance task, the role profiles are included in the application profile of the servicebash, as shown in the code-extract below:

/etc/apparmor.d/usr.bin.servicebash:

```
#include <tunables/global>
/usr/local/bin/servicebash {
  #include <role/role1>
  #include <role/role2>
  #include <role/role3>
  #include <role/role4>
}
```

After putting the servicebash profile in enforce mode (*sudo aa-enforce usr.local.bin.servicebash*) the program and therefore the user service is confined. The complete setup is shown in “Fig. 3”.

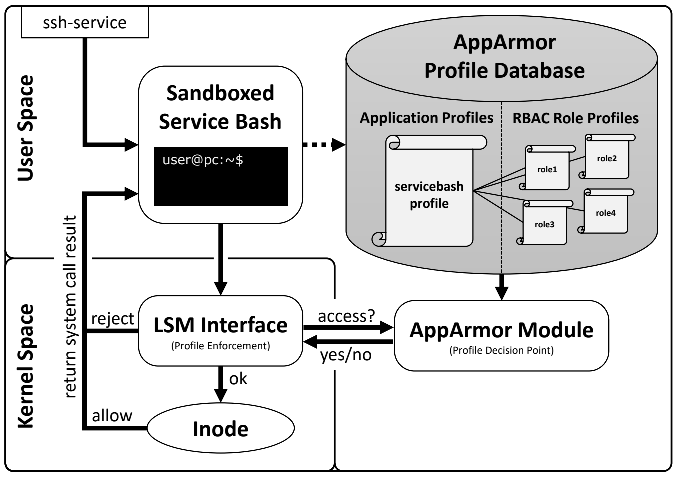

Fig. 3. Architecture of implementation.

The roles are highly depended on the task corresponding to the maintenance request and have to be assigned respectively. Because of the RBAC inherent data abstraction and the possibility to implement hierarchical role structures, roles can in advance be created for recurring or predefined tasks. Nevertheless, the manual creation and assignment of roles can be time-consuming, especially for non-standard tasks. For easier usage a python program is written to initiate a maintenance request and assign the correct roles to the servicebash profile. The program uses error messages to identify task fields for a maintenance technician and suggests suitable roles. In the implemented scenario, a lookup table is created that assigns specific roles to different error messages. This way, an employee who creates the service request only has to check and approve the request. Other approaches to both monitor the condition of the system to initiate a maintenance request and identify the correct roles for the session are also conceivable, but not subject of this paper.

In addition, an automatic mode for establishing a maintenance session that additionally identifies a suitable maintenance technician from a database according to the requirements and automatically triggers a remote maintenance case without human intervention was implemented and tested. A flow diagram of the program is shown in “Fig. 4”.

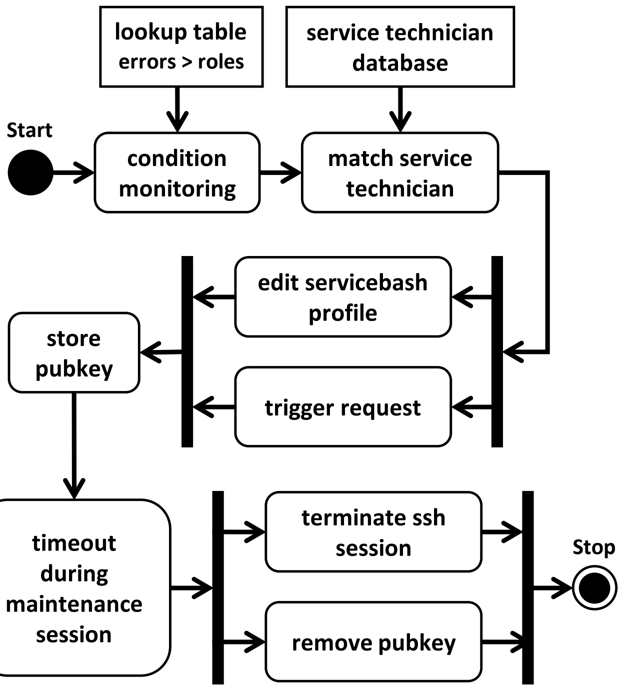

Fig. 4. Flow diagram of maintenance application.

Depending on the analysis of the error, different requirements regarding the subject (expertise, skill level, clearance and employer of the person performing the maintenance), roles (roles needed for the maintenance) and context (timeframe for the maintenance) are defined. According to the assigned task, the correct roles are included in the profile for the login shell. That followed the profile is reloaded and put into enforce mode. The requirements regarding the subject are compared to a local database of maintenance technicians and additionally written into a XML-File. In a more complex scenario, the XML-File could be transferred to a server to find a suitable maintenance technician instead of comparing locally. Once a maintenance technician is found, the corresponding X.509 public key certificate is stored to the service user’s authorized_keys directory. The subject’s attributes such as the expertise, skill level, clearance and employer could be integrated into the certificate as critical extensions. However, since these attributes often both change over time and are customer-specific this option is disregarded in this use-case. Next, the maintenance technician is notified about the task including connection details and the given time frame. The application starts the ssh service at the announced starting time and the maintenance technician can connect to the machine. This implementation uses SSH-2 with standard settings (AES 128), however other encryption algorithms such as 3DES, Blowfish, Twofish, CAST, IDEA, Arcfour or SEED and different key-length are supported as well. As soon as the end time is reached, the user is disconnected, the ssh service is disabled and the public key certificate is removed from the authorized_keys directory again. For monitoring purposes, all activities performed by the maintenance technician are logged with Linux' build-in audit system (auditd) [1]. A manipulation of the log files is not possible as long as the user is not given a role that can edit the log files or manipulate the auditd service.

V. Results and discussion

The implementation is tested and compared to the initial goals. Even though, the implemented use-case is limited to ssh as a login method, the same implementation can easily be extended to other login services such as gdm. Therefore, the concept is not only valid for remote maintenance, but also for the confinement of local users.

To validate the concept, the BSI requirement 7, that is not satisfied by any of the current solutions available is revisited. Using AppArmor, mandatory RBAC can be implement to confine the user's permissions to the required minimum and enforce the principle of least privilege. Additionally, the introduced implementation complies with requirements 2 and 6 that are mostly ignored by current solutions as well. Using an ssh-tunnel, the remote maintenance technician establishes a fully end-to-end encrypted tunnel. With the auditd service, it is possible to fully log the remote maintenance session.

A. Efficiency of the implementation

To evaluate the suggested solution further, the efficiency is tested. Efficiency in this manner is regarding the performance overhead introduced by the security measure. For this purpose, a tool called ssh-perf is used. The tool is specifically developed to measure the performance of ssh connections, such as raw connections per second, exec commands per second, SCP or SFTP upload and download speeds and maximum open connections.

For the performance test, two identical virtual machines (A and B) are setup on a Sysgem Supermicro Server with 2 Haswell-Xeon E5-2643 v3 with12 physical cores, 128GB DDR4 RAM and 8 times 4TB Storage storage running a VMWare ESXi Server version 6.0.0, build 3620759. Both machines are allocated one virtual core with 3.4 GHz, 1 GB RAM, and run Ubuntu Server 16.04.4 LTS x86_64, Kernel Version 4.4.0-128. The machines are connected via an internal VM network to reduce any latency variations introduced through physical networks. The test consists of two runs. In the first run, a script running different test scenarios with ssh-perf is run on machine A. In the test, machine A connects to machine B and measures performance indicators that will be described in more detail later. In a second run, the exact same test is performed again. The only difference between both runs is that the first time, AppArmor is completely turned off and the second time, AppArmor is restricting access. For this second run, an AppArmor profile that allows access to all resources needed for the test is created with the aa-genprof tool. The tool is used to create a profile for each test. To create a similar structure as proposed in the concept, the rules common to all tests and the test individual rules are each put into separate files. Each file being the equivalent to a role. Then an AppArmor profile for the servicebash is created and the roles are included accordingly. Each test is run for one hour to reduce the influence of short term variabilities.

The test results are shown in "Tab. I". The first column on the left side of the table lists the abbreviations of the different tests carried out:

- raw connections per second: rcps
- executed commands per second: ecps
- sftp upload (put) performance (1GB file): sftpp
- sftp download (get) performance (1GB file): sftpg
- scp upload (put) performance (1GB file): scpp
- scp download (get) performance (1GB file): scpg

The remaining columns are split into groups of two. The first pair shows the successes per second (sps) and latency per second (lps) for the test run with AppArmor disabled (unconfined). The second pair shows the results for the test run with AppArmor enabled (confined). The last pair lists the performance difference between the two. The performance difference for successes per second is calculated by $spd = (1 - confined/unconfined) * 100$. The performance difference for latency per second is calculated by $lpd = (confined/unconfined - 1) * 100$. All results are rounded to two decimal places after doing the calculations.

TABLE I
Performance Results

test	unconfined		confined		difference	
name	sps	lps	sps	lps	spd	lpd
rcps	6.96	0.09	5.59	0.10	19.71%	17.49%
ecps	5.29	0.18	4.46	0.22	15.66%	19.32%
sftpp	0.11	8.73	0.11	8.71	-0.18%	-0.18%
sftpg	0.11	9.46	0.10	9.51	0.58%	0.55%
scpp	0.10	9.62	0.10	9.64	0.18%	0.18%
scpg	0.11	9.47	0.10	9.54	0.67%	0.68%

The first interesting observation to make is that the sftp upload performance (sftpp) seems to have improved about 0.18%. This improvement can only be explained by the testing procedure not being completely accurate. Therefore, the results should be treated as a tendency rather than absolute values. Having this in mind, the performance overhead introduced through the AppArmor RBAC implementation is almost negligible for both scp and sftp upload and download performance, being less than 1%. These small performance decreases are plausible, since the performance in these tests is mostly depended on the upload and download speeds. The authorization is only checked in the beginning of the file transfer. Once the connection is established AppArmor has no part in the rest of the test. For raw connections per second and executed commands per second, a performance decrease of maximum 20% can be observed. In these tests, AppArmor is involved for every new connection and every execution, which results in the performance decrease.

Even though, in the tests a performance decrease for connections per second and executed commands per second was identified, the relevance in a real world maintenance scenario is less significant. The latency difference for executed commands per second for example, is less than 40 milliseconds and therefore not noticeable for a service technician. Overall, the performance overhead introduced through the proposed RBAC implementation does not impair a maintenance session in any way and is well suited for enforcing the principle of least privilege in industrial remote maintenance sessions.

B. Limitations and possible solutions

However, RBAC still has some limitations. In the use-case, the maintenance session depends on requirements regarding the subject, role and context. For the implementation, three mechanisms are needed to archive these requirements. First a database is needed to select the correct subject and a script is needed to assign a service technician (subject) to a user. Then AppArmor is used to confine the subject depending on its specific roles. Last, a script to terminate the ssh connection is used to enforce the contextual requirement such as a time frame for the maintenance session. Furthermore, the concept of roles is very limited in itself. Roles have multiple dimensions. The roles in AppArmor define actions (read, write, execute, etc.) that can be performed on certain objects (resources). However, it is not possible to require multiple roles to access a certain resource (e.g. user can access system objectA if exhibiting roles role1 and role2). Instead, a new role has to be defined that includes both requirements and combines them into a new role (role3). As shown in this example, roles are either not very granular or very many roles are needed to describe more complicated scenarios, which makes the assignment of roles rather difficult.

In recent years, research efforts are focused on an more general approach. A new access control paradigm called Attribute-Based Access Control (ABAC) is being developed to combine the advantages of MAC, DAC and RBAC, while resolving their limitations [19]. With ABAC Boolean operations can be used to define access policies. An example for such a policy would be: *"Allow access to all system objects with attribute A1 if subject has attributes S1 and S2 and action is read or write and context is between 9am and 3pm"*. In this example, all users exhibiting the attributes S1 and S2 are allowed to read and write any resources with the attribute A1 between 9am and 3pm. However, ABAC cannot be integrated into preexisting systems as easily, since tags for system objects are needed. There does not exist an implementation of ABAC on system level yet. The implementation of such system is a matter of future research.

VI. Conclusion

In this paper, RBAC is used to enhance the system security of industrial computers during remote maintenance sessions. First a concept for using RBAC to protect a system is presented. The concept is implemented on a linux system using LSM and AppArmor. This approach leads to significantly more system security than an ordinary DAC system as used in most systems. The greatest advantage of this access control system is the enforcement of good user behavior beyond the point of known security issues. RBAC confines the user to the least amount of access needed to perform a certain task (principle of least privilege) and does not depend on the updates of attack signature databases or else. The security does not necessarily weaken when new vulnerabilities are being discovered. Therefore, this security approach is particularly suitable for the use in long-lived systems as seen with industrial computers and machines.

Moreover, RBAC cannot only be used to confine users but also applications and other connected systems. It is a convenient addition to existing security systems and can oftentimes easily be integrated. For the design of new systems, it is an opportunity to facilitate long-term security as well.

In the discussion several shortcomings of RBAC were identified. Some of which could be overcome using ABAC. However, for preexisting systems ABAC is no valid addition since the file system needs to be adapted. RBAC benefits with its ease of integration in older systems. Nevertheless, using ABAC for enhanced system security will be focus of future research and seems to offer promising advantages.

Acknowledgment

This research was supported by the IUNO Project (National Reference Project on IT-Security in the Industry 4.0). Especially, working group three supplied input for important ideas and results in this paper.

References

[1] *auditd(8): Audit daemon - Linux man page*. URL: https://linux.die.net/man/8/auditd (visited on 04/16/2018).

[2] Ryan Ausanka-Crues. "Methods for Access Control: Advances and Limitations". In: *Harvey Mudd College* 301 (2001), pp. 20–25.

[3] Mick Bauer. "Paranoid Penguin: An Introduction to Novell AppArmor". In: *Linux J.* 2006.148 (Aug. 2006), pp. 1–13. ISSN: 1075-3583.

[4] D. Elliott Bell and Leonard J. La Padula. *Secure computer system: Unified exposition and multics interpretation*. MITRE CORP BEDFORD MA, 1976.

[5] Alan Calder and Steve G. Watkins. *A dictionary of Information Security Terms, Abbreviations and Acronyms*. Cambridgeshire, United Kingdom: IT Governance Publishing, 2007. ISBN: 978-1-905356-21-8. URL: www.itgovernance.co.uk.

[6] Dana Al Kukhun and Florence Sedes. "Adaptive Solutions for Access Control within Pervasive Healthcare Systems." In: *Lecture Notes in Computer Science*. Springer, Berlin, Heidelberg, 2002, pp. 42–53. ISBN: 978-3-540-69914-9.

[7] Georg Disterer. "ISO/IEC 27000, 27001 and 27002 for Information Security Management". In: *Journal of Information Security* 04.2 (2013), pp. 92–100. ISSN: 2153-1234, 2153-1242. DOI: 10.4236/jis.2013.42011.

[8] Claudia Eckert. *IT-Sicherheit: Konzepte - Verfahren - Protokolle*. 8. Aufl. München: De Gruyter, 2013. 1016 pp. ISBN: 978-3-486-73587-1.

[9] Federal Office for Information Security. *Grundregeln zur Absicherung von Fernwartungszugängen*. June 27, 2013. URL: https://www.allianz-fuer-cybersicherheit.de/ACS/DE/_/downloads/BSI-CS_054.pdf;jsessionid=CE2EAED31E61E2A7885A6206CF47773F.2_cid360?__blob=publicationFile&v=3 (visited on 08/25/2017).

[10] Federal Office for Information Security. *Industrial Control System Security - Top 10 Threats and Countermeasures 2016*. Jan. 8, 2016. URL: https://www.allianz-fuer-cybersicherheit.de/ACS/DE/_/downloads/BSI-CS_005E.pdf?__blob=publicationFile&v=3 (visited on 08/25/2017).

[11] Federal Office for Information Security. *Remote maintenance in industrial environments*. Jan. 12, 2015. URL: https://www.allianz-fuer-cybersicherheit.de/ACS/DE/_/downloads/BSI-CS_108E.pdf?__blob=publicationFile&v=2 (visited on 08/25/2017).

[12] David F. Ferraiolo, John F. Barkley, and D. Richard Kuhn. "A role-based access control model and reference implementation within a corporate intranet". In: *ACM Transactions on Information and System Security* 2.1 (Feb. 1, 1999), pp. 34–64. ISSN: 10949224. DOI: 10.1145/300830.300834.

[13] David F. Ferraiolo, Janet A. Cugini, and D. Richard Kuhn. "Role-Based Access Control (RBAC): Features and Motivations". In: *Proceedings of 11th annual computer security application conference*. 1995, pp. 241–248.

[14] Kurt Gutzmann. "Access control and session management in the HTTP environment". In: *IEEE Internet Computing* 5.1 (Feb. 2001), pp. 26–35. ISSN: 10897801. DOI: 10.1109/4236.895139.

[15] Michael A. Harrison and Walter L. Ruzzo. "Protection in Operating Systems". In: *Communications of the ACM* 19.8 (1976), pp. 461–471.

[16] J. Hoffman. "Implementing RBAC on a type enforced system". In: *Proceedings 13th Annual Computer Security Applications Conference*. 13th Annual Computer Security Applications Conference. San Diego, CA, USA: IEEE Comput. Soc, 1997, pp. 158–163. ISBN: 978-0-8186-8274-2. DOI: 10.1109/CSAC.1997.646185.

[17] *ISO 27000*. Oct. 2017.

[18] Bokefode Jayant.D et al. "Analysis of DAC MAC RBAC Access Control based Models for Security". In: *International Journal of Computer Applications* 104.5 (Oct. 18, 2014), pp. 6–13. ISSN: 09758887. DOI: 10.5120/18196-9115.

[19] Xin Jin, Ram Krishnan, and Ravi Sandhu. "A Unified Attribute-Based Access Control Model Covering DAC, MAC and RBAC". In: *Data and Applications Security and Privacy XXVI*. Ed. by Nora Cuppens-Boulahia, Frederic Cuppens, and Joaquin Garcia-Alfaro. Vol. 7371. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 41–55. ISBN: 978-3-642-31539-8 978-3-642-31540-4. DOI: 10.1007/978-3-642-31540-4_4.

[20] E. A. Lee. "Cyber Physical Systems: Design Challenges". In: *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*. 2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC). May 2008, pp. 363–369. DOI: 10.1109/ISORC.2008.25.

[21] Wolfgang Mahnke, Stefan-Helmut Leitner, and Matthias Damm. *OPC unified architecture*. OCLC: ocn268784080. Berlin: Springer, 2009. 339 pp. ISBN: 978-3-540-68898-3 978-3-540-68899-0.

[22] Gary McGraw and John Viega. "Building Secure Software - A Difficult But Critical Step in Protecting Your Business". In: *RTO/NATO Real-Time Intrusion Detection Symposium*. RTO/NATO Real-Time Intrusion Detection Symposium. Cigital, Inc., 2002.

[23] James Morris, Stephen Smalley, and Greg Kroah-Hartman. "Linux security modules: General security support for the linux kernel". In: *USENIX Security Symposium*. 2002.

[24] Gustaf Neumann and Mark Strembeck. "A Scenario-driven Role Engineering Process for Functional RBAC Roles". In: *Proceedings of the seventh ACM symposium on Access control models and technologies*. ACM. 2002, pp. 33–42.

[25] Gustaf Neumann and Mark Strembeck. "Design and Implementation of a Flexible RBAC-Service in an Object-Oriented Scripting Language". In: *Proceedings of the 8th ACM conference on Computer and Communications Security*. ACM. 2001, pp. 58–67.

[26] Sylvia Osborn, Ravi Sandhu, and Qamar Munawer. "Configuring role-based access control to enforce mandatory and discretionary access control policies". In: *ACM Transactions on Information and System Security* 3.2 (May 1, 2000), pp. 85–106. ISSN: 10949224. DOI: 10.1145/354876.354878.

[27] Ravi S. Sandhu et al. "Role-based access control models". In: *Computer* 29.2 (1996), pp. 38–47.

[28] Stefan Schweiger, ed. *Lebenszykluskosten optimieren: Paradigmenwechsel für Anbieter und Nutzer von Investitionsgütern*. 1. Aufl. OCLC: 269453821. Wiesbaden: Gabler, 2009. 188 pp. ISBN: 978-3-8349-0989-3.

[29] Stephen Smalley, Chris Vance, and Wayne Salamon. "Implementing SELinux as a Linux security module". In: *NAI Labs Report* 1.43 (2001), p. 139.

[30] Erdal Tantik and Reiner Anderl. "Integrated Data Model and Structure for the Asset Administration Shell in Industrie 4.0". In: *Procedia CIRP* 60 (2017), pp. 86–91. ISSN: 22128271. DOI: 10.1016/j.procir.2017.01.048.

[31] Stephan Weyer et al. "Towards Industry 4.0 - Standardization as the crucial challenge for highly modular, multi-vendor production systems". In: *IFAC-PapersOnLine* 48.3 (2015), pp. 579–584. ISSN: 24058963. DOI: 10.1016/j.ifacol.2015.06.143.