

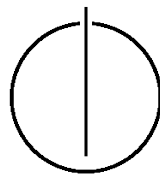
FAKULTÄT FÜR INFORMATIK

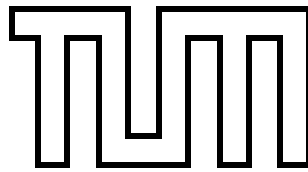
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's thesis in computer science

**Privileged User Password Management in
Shared Environments**

Manuel Söhner





FAKULTÄT FÜR INFORMATIK

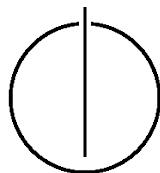
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's thesis in computer science

Privileged User Password Management in Shared
Environments

Privileged User Password Management in verteilten
Umgebungen

Author: Manuel Söhner
Supervisor: Prof. Dr.-Ing. Georg Carle
Advisors: Benjamin Hof, M.Sc
Daniela Pöhn, Felix von Eye (LRZ)
Dr.-Ing. Ingo Pansa (iC Consult GmbH)
Submission date: April 15, 2014



I assure the single handed composition of this master's thesis only supported by declared resources.

Munich, April 15, 2014

Manuel Söhner

Acknowledgments

I would like to thank the advisors Benjamin Hof from the Chair for Network Architectures and Services, Daniela Pöhn and Felix von Eye from the Leibniz Supercomputing Centre, and Ingo Pansa from iC Consult for their good advice, support, and feedback during the writing of this thesis.

I would also like to thank Prof. Dr.-Ing. Georg Carle for supervising this thesis and for the advice after the first presentation.

Abstract

Administrators often require access to privileged and shared accounts to manage systems, services, and applications. Subsequently, the unlimited power gives them unrestricted access to sensitive data and to important components of the network infrastructure.

In order to prevent from insider and outsider threats, the privileged user accounts need to be securely shared between authorized administrators and the access to these accounts needs to be monitored and audited.

This thesis analysis the threats that arise from privileged user accounts in shared environments and brings up countermeasures that need to be taken into account in a privileged user password management solution.

Besides the security aspects, generic as well as specific requirements of two organizations are presented. For that purpose, the status quo of the current password management as well as use cases at the Leibniz Supercomputing Centre (LRZ) and iC Consult GmbH, which is a system integrator specialized in identity and access management, were analyzed.

Based on that requirements catalog, three software products are evaluated and compared to each other. Moreover, the suitability of these products for the purpose of the LRZ and iC Consult is examined.

The thesis then proposes a generic architecture that addresses the management of privileged user passwords as well as the fine-grained control of access rights that administrators require to perform actions as privileged user. After that, a demonstrator illustrates some of the use cases that have been brought up during the requirements research by implementing the essential parts of the proposed architecture.

In conclusion, the thesis shows that the management of privileged user passwords not only requires a centralized component that securely controls which administrator is authorized to access what privileged user passwords. Moreover, it is necessary to consider special use cases that normal password managers do not have to consider, for example, an emergency access that allows administrators to use a privileged account which they usually are not authorized to use. Additionally, the auditing of shared privileged accounts needs to be addressed and the unrestricted access rights need to be taken into account.

Contents

1	Introduction	1
1.1	Problem Statement	1
1.2	Identity and Access Management	2
1.3	Definitions and Notation	3
1.3.1	Terms and Definitions	3
1.3.2	Notation for XML Namespaces	5
1.4	Approach and Structure of This Thesis	6
2	Related Work	9
2.1	eXtensible Access Control Markup Language	9
2.2	Distributed Enforcement of XACML Policies With pam_xacml	15
2.3	Security Assertion Markup Language	16
2.4	OpenID	17
2.5	OAuth	18
2.6	Shibboleth	19
3	Threat Model	21
3.1	Basic Terms	21
3.2	Scenarios Concerning Privileged User Password Management	23
3.3	Threat Targets, Threats and Attacks	25
3.3.1	Hardware	26
3.3.2	Network	27
3.3.3	Account	28
4	Requirements	33
4.1	Operational and Business Requirements	33
4.1.1	Security Regulations	34
4.1.2	Information Distribution	35
4.1.3	Separation of Duties	36
4.1.4	Auditing and Reporting	36
4.2	Practicability Requirements	36
4.3	Non-Functional Requirements	37
4.3.1	Availability	37
4.3.2	Systems Integration	38
4.3.3	Security	38
4.4	Use Cases and Requirements at the Leibniz Supercomputing Centre	39
4.4.1	Infrastructure of the LRZ	39
4.4.2	Status Quo and Requirements	39

4.5	Use Cases and Requirements of iC Consult	42
4.5.1	Infrastructure of iC Consult	42
4.5.2	Status Quo and Requirements	42
4.6	Requirements Catalog	43
5	Analysis of Established Solutions	47
5.1	CA ControlMinder	47
5.1.1	System Architecture	48
5.1.2	Test Environment	50
5.1.3	Analysis Based on the Requirements Catalog	52
5.2	Netwrix Privileged Account Manager	55
5.2.1	System Architecture	55
5.2.2	Test Environment	57
5.2.3	Analysis Based on the Requirements Catalog	58
5.3	Soffid Identity and Access Management	61
5.3.1	System Architecture	62
5.3.2	Test Environment	63
5.3.3	Analysis Based on the Requirements Catalog	63
5.4	Summary of Software Analysis	67
6	Architecture	69
6.1	Components	70
6.1.1	Policy Administration Point	71
6.1.2	Policy Decision Point	71
6.1.3	Policy Enforcement Point	72
6.1.4	Password Manager	76
6.1.5	Audit Logging System	77
6.2	Security Considerations	78
6.2.1	Encryption of Credentials	78
6.2.2	Secure Audit Logs	79
6.3	Analysis Based on the Requirements Catalog	81
6.4	Summary	84
7	Specification and Implementation of a Demonstrator	89
7.1	Authorization for PAM-Enabled Applications	91
7.2	Policy Decision Point	92
7.3	Jump Server	93
7.4	Web Interface and Password Database	94
8	Evaluation	95
8.1	iC Consult Compared to the Leibniz Supercomputing Centre	95
8.2	Applicability Analysis for the Leibniz Supercomputing Centre	97
8.2.1	Applicability of the Architecture	97
8.2.2	Applicability of the Analyzed Software	97
8.3	Applicability Analysis for iC Consult	98
8.3.1	Applicability of the Architecture	98

8.3.2	Applicability of the Analyzed Software	98
9	Conclusion and Future Work	101
9.1	Summary	101
9.2	Future Work	102
9.2.1	Attribute-Based Encryption For Sensitive Data	103
9.2.2	Transformation of High-Level Policies to Low-Level Policies	104
9.2.3	Expansion to Other Operating Systems	104
9.2.4	Virtual Environments	105
	Bibliography	107

List of Figures

3.1	The Source of Greatest Risk to Sensitive Data	22
4.1	Organization Structure of the LRZ	40
5.1	Screenshot of PuTTY Demonstrating Non-Repudiation in CA ControlMinder	49
5.2	Screenshot of PuTTY Demonstrating Fine-Grained Access Control in CA ControlMinder	50
5.3	Netwrix Privileged Account Manager	56
5.4	Architecture of Soffid Identity and Access Management	62
5.5	Revealing a Password in the Soffid Self-Service Portal	63
6.1	IETF Policy Framework Architecture	70
6.2	Jump Server	75
6.3	Full Architecture	86
7.1	Architecture of the Demonstrator	89

List of Tables

2.1	Attributes in the XACML Request Message	10
2.2	Attributes in the XACML Response Message	11
4.1	Use Cases at iC Consult	43
5.1	Environment Specification for CA ControlMinder (Windows Server)	51
5.2	Environment Specification for CA ControlMinder (Linux Server)	51
5.3	Environment Specification for Windows Domain Controller	58
5.4	Environment Specification for Netwrix Privileged Account Manager	58
5.5	Environment Specification for Soffid Identity and Access Management	64
5.6	Summary of the Software Analysis Based on the Requirements Catalog	68
6.1	Summary of the Architecture Based on the Requirements Catalog	87
7.1	Environment Specification for the Policy Decision Point	90
7.2	Environment Specification for the Jump Server	90
7.3	Environment Specification for the Web Server	91
8.1	Requirements of the Leibniz Supercomputing Centre Compared to iC Consult	96

Listings

2.1	Example for a Separation of Duties in XACML	12
2.2	Excerpt of a Break-Glass Policy in XACML	13
2.3	Example for a Policy that Allows the Delegation of Rights	14
7.1	PAM Configuration for sudo	91
7.2	Example Request Message for sudo	92
7.3	Example Response Message That Permits the use of sudo	93
7.4	ProxyCommand in authorized_keys File	93
7.5	Command to Reach a Target System via the Jump Server	94

1 Introduction

The recent disclosure of global surveillance, which was uncovered by Edward Snowden, a former contractor of the U.S. National Security Agency (NSA), showed that the unrestricted access of administrators to sensitive data is a crucial aspect in computer networks.

Often, privileged user accounts are shared between different administrators and thereby the abuse by an insider is hard to impute to a specific person. In order to reduce the security risk for an organization, the access to privileged user passwords needs to be controlled and the privileged access rights need to be managed.

This chapter gives an overview on the problem that is addressed by this thesis and classifies the privileged user password management into the context of identity and access management.

1.1 Problem Statement

Many organizations employ at least one administrator that manages the IT infrastructure with its corresponding business-critical resources. In order to do their job, the administrators are equipped with rights that give them access to operating systems, applications, databases and other data storage. Because of that, they are *privileged users* which have extensive authority on each system they administrate – and often beyond that. In some cases, those privileged users do not operate with personal accounts but share dedicated super-user accounts. This raises a problem in the area of accountability, since actions cannot be attributed to a specific person once a granted permission got abused [9]. Furthermore, the user accounts with elevated access rights present a target for external as well as internal attackers and therefore provoke a high security risk.

Moreover, privileged user accounts are also included in many network devices and software products that are shipped with a default username and password combination. Sometimes, the credentials are not changed by the IT administrators, which makes it easier for attackers to break into a system and manipulate or steal sensitive data [73].

Besides the employees that use privileged identities, there are also applications and services that run under privileged user accounts, which therefore represent a security risk as well [15]. Once a central component (e.g., a directory service or database) is compromised, it is easy to sabotage other network components or steal identities and data.

The problem becomes more complicated in distributed environments, like it is the case with outsourced departments. Here, different stakeholders are connected in a heteroge-

neous environment and need to have access to the systems and data. Another example is the growing sector of cloud computing, where administrators of a service provider have access to the hardware they are managing in order to provide the infrastructure. Those could also gain access to the sensitive data or manipulate the system.

For that reason, the system administrators need to establish a solution that protects the data and restricts the access of privileged user accounts, based on the identity's authorization level. In addition, an organization might also have an interest in reducing the access of their administrators to a minimum in order to lower the possibility of getting attacked from inside the company.

In order to solve the above mentioned problems, the organization needs to introduce a dedicated security architecture that takes care of the authentication and authorization of users and applications. For that purpose, the organization has to manage all stakeholders as identities and define the access permissions and policies that guard the data. Basis for that are the business processes inside the institution, as they define the resources a stakeholder requires to accomplish a task. That is where another problem could arise since the processes have to be analyzed and sometimes changed in order to extend the existing infrastructure.

1.2 Identity and Access Management

In most organizations people work with different kinds of sensitive data while fulfilling their tasks. In order to control which person has access to which data, the IT administrators need to provide a system that enforces access control policies. In most of the cases these policies are derived from the business processes inside the organization. The essential goal is to give each person the minimum amount of access in order to secure the data and protect from faults and malicious actions. In computer science, this idea is generally referred to as the *principle of least privilege* [64]. The simplest approach of achieving that goal is to give each person an individual and thus unique identity (e.g., a personal account) that represents it on every system in the infrastructure.

In order to authenticate a user and authorize the resource access, the systems utilize the predefined information that is part of the individual identity. Controlling all that information is the purpose of the *Identity Management* (IdM). Besides creating, deleting, and modifying the identities throughout their existence as part of the identity lifecycle management, the identities need to be monitored. For that purpose, the IdM software often provides a logging component that fulfills the auditing requirements of the organization. Controlling the identities becomes even more important when external users can access internal systems or when users have access to external systems. For that purpose, the approach of a *Federated Identity Management* (FIdM) has been established, where autonomous security domains are interconnected and share identities with uniform standards [6].

On the other hand, there is the data that needs to be protected. This can be done by defining roles, access permissions, and policies that determine the necessary rights an identity

has to have in order to get access to a resource or object. This task is part of the access management and complements the identity management. The most important aspect of the access management is to apply the principle of least privilege to reduce the risk of people accessing sensitive data they should not have access to. Hence, these tasks are often combined to the term *Identity and Access Management* (IAM).

A subarea of IAM is the *Privileged Identity Management* (PIM), which mostly deals with non-personal accounts that are used by humans and software. As mentioned in chapter 1.1, users with privileged accounts are a security risk and can cause a huge damage to the organization. This is because privileged accounts have access to sensitive data and are able to change the configuration of hardware and software, most of the time without being tracked. Thus, these identities have to be managed and monitored accurately. For that purpose, the organization firstly defines who is allowed to perform specific actions. Then, the PIM has to monitor each identity and log the actions that have been performed in a session. Furthermore, a partial aspect of that is the *Privileged User Password Management* (PUPM) and the *Shared Account Password Management* (SAPM) [40].

1.3 Definitions and Notation

This section introduces basic definitions and XML notations that are used throughout the remainder of this thesis.

1.3.1 Terms and Definitions

The following terms are defined for the context of this thesis and are based on the suggestions of RFC 4949 [28].

Attributes provide information that identify a person or system entity. Before the system can utilize them, for example, for the authorization process, an authority needs to predefine and assign the attributes to the identities.

An *identity* represents and describes a person or other entity (e.g., a service or application) inside a system with various attributes. These attributes mainly focus on the personal data that make an identity unique.

A (*system*) *user* is defined as a person that uses and accesses resources that are provided by the system. In addition, also services and applications are referred to as “user”, since application-to-application scenarios are almost equal to the human-computer interaction.

Policies are the basis of access control since they render the decision which user-assigned attributes grant access.

A *role* represents a job function inside the organization to which users get assigned. Policies may define that a role is allowed to perform a specific action in the system.

An *object* or *resource* is a system component that contains or receives information. Because of that, objects are sensitive resources which need to be protected by an access control. Subjects can perform actions on them and modify their state if the permissions are valid.

Subjects are system entities that perform actions on objects, for example, change system states or the content of objects. For such actions, they require access rights which need to be assigned to each subject. Subjects might be human beings as well as applications (e.g., services or scripts). A subject itself can also be an object, depending on the situation and interaction. Users are a concrete example of a subject.

Assets are system resources that need to be protected since they contain critical or sensitive information of an organization. They are used and managed by the subjects of a system.

Accounts are system entities that are mostly assigned uniquely to an identity and used to authenticate and authorize a user. There are different types of accounts that need to be differentiated:

- Accounts of applications or services
- Shared, non-personal accounts (e.g., *admin* and *root*)
- Personal accounts with permanently enhanced permissions
- Break-glass accounts (users can elevate their permissions (e.g., in emergency cases) although they do not have these rights assigned permanently for their day-to-day tasks)

In the process of *authentication*, the identity of a user is confirmed, based on the given information (e.g., credentials such as a username/password combination or certificates).

Authorization presupposes authentication and ensures that a user is granted or denied access to a protected resource. Moreover, it is the process of specifying access rights and the assignment to the resources that should be protected. The permissions or rights of a user depend on the assigned attributes, which are evaluated as part of the policy enforcement process. The enforcement is fulfilled by an *access control*.

Accounting is the process of logging and tracking the activity of a user. The collected information (e.g., a session with every command that has been executed) can be used for auditing purpose.

Privileged User and Shared Account

First and foremost the role of a privileged user is associated with employees that have enhanced rights allocated to their identity, including system administrators, operators, and users with high-risk access to resources. Besides the personal accounts, a system also consists of pre-existing non-personal accounts that are shared by different people, such as the superuser on Windows computers (“administrator”), UNIX systems (“root”), routers, and switches. Moreover, there are privileged accounts in applications, such as a database

administration account [18]. Additionally, external partners can also have accounts in the system that are privileged and thus are a potential security risk. Furthermore, applications and services might use or run as privileged processes that can perform security-relevant functions in contrast to other processes that are not authorized to execute such actions [28].

On the other hand, there are persons that might have access to privileged accounts even though they are not authorized to use the elevated access rights that are connected with these accounts. One example are employees that still have privileges assigned to their identity, even though these should have been revoked, for example, because they have finished a project for which they required enhanced rights. In addition, a user might still have valid credentials of shared accounts which cannot be easily revoked [14]. Another example are alumni that have left the company but can still use their account because it has not been deleted yet.

The most dangerous owners of a privileged account are malicious users. They can be separated into two groups: On the one hand, the external attackers that do not know anything or very little about the system architecture and try to get access to some sensitive data and privileged accounts. On the other hand, there are the insiders, that were former employees or still work for a company and thus know things about the systems they can use to exploit loopholes in order to get access to the desired information or sabotage the infrastructure [69].

1.3.2 Notation for XML Namespaces

In order to improve readability, the XML examples in this thesis assume use of the following XML Internal Entity Declarations:

```
<!ENTITY xacml "urn:oasis:names:tc:xacml:1.0:" >
<!ENTITY xml "http://www.w3.org/2001/XMLSchema#">
<!ENTITY rule-combine
    "urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:">
<!ENTITY policy-combine
    "urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:">
<!ENTITY attribute-category
    "urn:oasis:names:tc:xacml:3.0:attribute-category:">
<!ENTITY subject-category
    "urn:oasis:names:tc:xacml:3.0:subject-category:">
<!ENTITY function "urn:oasis:names:tc:xacml:1.0:function:">
<!ENTITY subject "urn:oasis:names:tc:xacml:1.0:subject:">
<!ENTITY resource "urn:oasis:names:tc:xacml:1.0:resource:">
<!ENTITY action "urn:oasis:names:tc:xacml:1.0:action:">
<!ENTITY status "urn:oasis:names:tc:xacml:1.0:status:">
```

For example, `&xml;#string` is the same as `http://www.w3.org/2001/XMLSchema#string`.

1.4 Approach and Structure of This Thesis

Managing (privileged) user passwords with software is a common approach. Besides maintaining password lists with Excel spreadsheets that are shared via email or saved on a network storage, there are password manager tools which can provide several advantages over the lists approach.

However, these software solutions have some limitations when it comes to security goals like authorization and accounting. Once a user has access to credentials in the password manager, he can log in to the system and perform all actions that are possible with the assigned permissions. This critical problem is especially related to shared accounts (e.g., “root” on UNIX-like operating systems) since usually there is no information about the person that used the account. The password manager might provide some kind of log with information about who accessed a password at a given time but that is not enough to ensure accountability. Moreover, the management of privileged user passwords only controls who has access to the credentials but it does not offer any control about what a malicious user can do once he has gained access to a privileged account [49].

That problem could be solved by reducing the need for shared accounts and only keeping them for emergency cases. Instead, the permissions a user requires to fulfill the day-to-day tasks could be assigned to the personal account. This provides accountability and traceability because every user performs the actions with his unique identity [21].

This thesis develops a centralized authorization architecture that provides a fine-grained access control to operating-system components like the file system and to applications. However, there are situations where devices (e.g., routers and switches) cannot be integrated into the architecture because they do not provide the necessary interfaces or cannot be adjusted. For these, another solution is outlined, as well as for situations where only a password manager is capable.

The structure is as follows: Chapter 2 outlines established solutions that (partly) solve a similar problem in other environments or serve as a basis for the architecture that is developed in this thesis.

Following up on this, Chapter 3 explains which threats and challenges need to be taken into account in order to design a secure architecture for the management of privileged user passwords. Moreover, the chapter outlines typical scenarios that occur inside organizations.

After that, Chapter 4 describes the generic requirements for a privileged user password management as well as specific requirements that exist at the Leibniz Supercomputing Centre and iC Consult GmbH, which is a system integrator specialized in identity and access management.

Then, Chapter 5 analyzes established products of companies that aim to provide a solution for the management of privileged user passwords. The products are then compared to the challenges and requirements that have been outlined in the preceding chapters.

Following, Chapter 6 illustrates the components and security aspects of an architecture that provides authorization decisions, auditing, and the management of passwords. For that purpose, open standards are used, so that the generic architecture can be adapted and extended.

After that, Chapter 7 specifies a demonstrator that implements the essential parts of the architecture to show some of the scenarios of a privileged user password management.

In Chapter 8, the requirements of the Leibniz Supercomputing Centre and iC Consult are compared and the evaluated software products are analyzed regarding the suitability for these organizations.

At the end, Chapter 9 summarizes the findings and proposes topics and approaches for future work.

2 Related Work

In order to implement the idea of reducing the need for shared privileged accounts, it requires a way to model access rights and enforce them when a subject accesses an object. This chapter outlines established standards and solutions that serve as an example for or will be the basis of the architecture that is developed in Chapter 6.

There are different solutions available for web environments that solve the problem of authorization in distributed environments. The established standards are OpenID, OAuth, the Security Assertion Markup Language (SAML), and the eXtensible Access Control Markup Language (XACML). Besides that, Shibboleth is an open-source architecture that provides authentication and authorization to web applications and web services.

2.1 eXtensible Access Control Markup Language

The eXtensible Access Control Markup Language (XACML) is an XML-based open standard that specifies a format for access control policies as well as a processing model which describes the evaluation process of access requests. The response to these requests then describes who has access to what resources on which conditions. That decision is then enforced by the system that protects the access to the resource.

XACML is an *Attribute Based Access Control* (ABAC) and will be used in Chapter 6 to realize the authorization process that determines which subject is allowed to perform what action on which resource. In contrast to a *Role Based Access Control* (RBAC), XACML provides a more fine-grained way to model access rights since the flexible attributes are attached to the identity and evaluated by the rules of a policy. Instead, the RBAC approach requires to combine permissions in a role and assign it to the users. This implies that unique roles are engineered for each scenario that a user might perform and that requires different permissions. This results in more roles and a complex management of access rights [81].

The main components of the authorization decision process are

- the *subject* (user) that requests access to a protected resource,
- the *resource*, which can be data in a file, a command, an application or service as well as a system component,
- the *Policy Enforcement Point* (PEP), which performs the access control by enforcing an authorization decision on the protected resource,

- the *Policy Decision Point* (PDP), which evaluates a request and then renders an authorization decision after determining the appropriate authorization to grant,
- the *Policy Information Point* (PIP), which is a source of attribute values that retrieves attributes for the PDP if they are missing in the request message of the PEP,
- and the *Policy Administration Point* (PAP), where policies are created, managed, and deployed so that the PDP can use them [54].

The communication between the PEP and the PDP is based on a standardized request-response protocol. When the PEP recognizes the access attempt of a subject to a resource, it sends a description of the attempted resource access to a PDP in the form of an authorization decision request [56]. The request message contains attributes from a set of four different attribute types, which are listed in Table 2.1.

Table 2.1: Attributes in the XACML Request Message

Attribute	Description
Subject	Represents the entity that initiated the request and requires access to a resource.
Action	Gives information about the action that the subject wants to perform on the object/resource (e.g., read or write).
Resource	Specifies the object that a subject wants to access (e.g., the data or system component).
Environment	Contains additional information that is not related to the subject, action, or resource but is required by the PDP to evaluate the decision (e.g., the time of day or the IP address of the client from which the access request came from).

On the other hand, the response message contains information about the decision and status which represent the result of the evaluation process of the PDP. The attributes of the response message are listed in Table 2.2.

The PDP uses the request message as input in order to evaluate the appropriate *Policy*. Such policies are created by the PAP and are made up of information about the target to which the policy applies and a *Rule-Combining Algorithm* which specifies how the rules should be combined when more than one rule applies (e.g., “deny-overrides” or “allow-overrides”).

A policy contains at least one *Rule* that specifies a target to which it applies, an optional *Condition* that satisfies the rule, and the *Effect* of the rule (e.g., *Permit* or *Deny*). In the evaluation process the PDP matches the target of the policy and rule against the target that was submitted with the request message, in order to check if they are applicable.

With conditions it is possible to formulate rules that can check environment and system variables, such as the current time or the location of the device that sent the request [29].

Table 2.2: Attributes in the XACML Response Message

Attribute	Description
Decision	Contains <i>Permit</i> , <i>Deny</i> , <i>Indeterminate</i> or <i>NotApplicable</i> .
Status	Returns a status code to the PEP to inform it whether the request was correct, if attributes were missing or if the request contained a syntax error. Besides the status code, it can also contain a status message that gives more information.
Obligations or Advice	Informs the PEP that it must (obligation) or should (advice) perform an operation before or after the authorization decision is enforced.

This makes it possible to bind authorization decisions to additional requirements that have to be satisfied for a valid access but are non-static (like a permission that is assigned to the account).

Another optional part of a policy are *Obligations*, which can be used to communicate to a PEP which actions it must perform before and after the authorization decision is made. Thereby, it is possible to attach additional constraints to a decision, like triggering an event that sends a notification or writes a log entry which includes all necessary information about the request (e.g., which user wanted to access what resource and if the access was granted or denied) [29]. This feature allows the organization to ensure non-repudiation on the requests a user generates while accessing a resource.

Advanced Use Cases

With XACML it is possible to model business processes on a fine-grained level by combining attributes in an appropriate way. The following paragraphs will outline some example processes that might be required in an architecture that aims to reduce the power of privileged users.

Separation of Duties (SoD). This is an important security control feature when it comes to critical actions that should not be performed by one person only. The *four-eye-principle* or *two-man rule* is a common form of such control which aims to prevent fraud and errors. It requires that tasks and privileges are split into activities which are performed by individuals if the fulfillment by one person rises potential conflicts of interest. This ensures that privileged users cannot perform actions without being detected because another person is able to monitor all actions. Usually, these regulations are also part of the business process and formulated as business rules [5].

With XACML, such SoD rules can be formulated as policies in order to ensure data confidentiality and integrity. The basic idea is to build policies for the same target and define

critical actions that need to be performed by different subjects. An example scenario is depicted in Listing 2.1, which shows a policy that denies the access to the database audit log to any user that holds both the administrator and auditor roles [53].

Listing 2.1: Example for a Separation of Duties in XACML

```
<Policy PolicyId=" administrator :AND: auditor : disallowed "  
  RuleCombiningAlgId="&rule-combine ; deny-overrides ">  
  <Target>  
    <Subjects>  
      <Subject>  
        <SubjectMatch MatchId="&function ; string-equal ">  
          <AttributeValue  
            DataType="&xml ; string ">administrator</ AttributeValue>  
          <SubjectAttributeDesignator  
            AttributeId="urn : mynamespace : attributes : role "  
            DataType="&xml ; string "/>  
        </SubjectMatch>  
        <SubjectMatch MatchId="&function ; string-equal ">  
          <AttributeValue  
            DataType="&xml ; string ">auditor</ AttributeValue>  
          <SubjectAttributeDesignator  
            AttributeId="urn : mynamespace : attributes : role "  
            DataType="&xml ; string "/>  
        </SubjectMatch>  
      </Subject>  
    </Subjects>  
    <Resources>  
      <Resource>  
        <ResourceMatch MatchId="&function ; string-equal ">  
          <AttributeValue>database-audit-log</ AttributeValue>  
          <ResourceAttributeDesignator  
            AttributeId="&resource ; resource-id "/>  
        </ResourceMatch>  
      </Resource>  
    </Resources>  
    <Actions><AnyAction /></ Actions>  
  </Target>  
  <Rule RuleId="Deny : target : role : combination " Effect="Deny" />  
</Policy>
```

The listing starts with a `Policy` element that consists of two attributes: a unique policy identifier and the declaration of the rule-combining algorithm, which ensures that in a case where more than one policy matches, the deny decision has priority. This is important because the separation of duties must ensure that even though the user has the permissions to perform the actions independently, the policy that denies the subsequent completion takes precedence. The resulting effect ("Deny") can be seen in the `Rule` element at the

bottom of the policy. The `Target` element comprises two subjects, which need to be given for a match. If the subject has the `administrator` and `auditor` attributes assigned and if the subject performs an arbitrary action on the `database-audit-log` resource, the policy matches and the access will be denied.

Break-Glass Situations. In a break-glass scenario a user who usually does not have access to a resource but has the right to “break the glass” in an emergency situation, will be granted access to the requested resource immediately but temporarily.

In [11] Chadwick and Lievens suggest a break-glass approach for XACML which works as follows:

1. Based on a subject’s access to a resource, the PEP sends a request to the PDP which evaluates the request and might return the decision that the access is denied. However, if the policy contains the information that the subject is allowed to break the glass, the response additionally contains an obligation that informs the PEP about that.
2. The PEP can then either prompt the user for the decision or decide on its own if the break-glass action should be performed. If yes, the PEP sends another request message to the PDP which contains the action that the subject wants to perform and that it is a break-glass request.
3. The PDP then searches for a policy that contains the original action as well as the break-glass action and returns the authorization decision to the PEP. The relevant excerpt of the policy that is evaluated in this step is shown in Listing 2.2.
4. If the second response message included obligations, the PEP must take care that all are fulfilled, otherwise the break-glass has to be denied. At this point, an audit log might be created or a supervisor might be informed about the success or failure of the break-glass request.

Listing 2.2: Excerpt of a Break-Glass Policy in XACML

```
<Action>
  <Attribute AttributeId="&action ; action-id"
             DataType="&xml ; # string">
    <AttributeValue>BreakTheGlass</AttributeValue>
  </Attribute>
  <Attribute AttributeId="&action ; originalUserAction-id"
             DataType="&xml ; # string">
    <AttributeValue>originalAction</AttributeValue>
  </Attribute>
</Action>
```

Delegation of Permissions. Since XACML v3.0 the delegation of permissions is supported, which allows an authorized person (the delegator) to delegate the whole policy or parts of it to another user (the delegate). Thereby, it is possible to create temporary rules that allow a user to do something on behalf of another user [57]. The example in Listing

2.3 allows Alice to delegate the permission of performing the backup action on a database resource to any user of the network-admin group. This might be necessary because Alice is on vacation and Bob, who is a member of the network administrators group, should perform the backup tasks until Alice is back.

Listing 2.3: Example for a Policy that Allows the Delegation of Rights

```
<Policy PolicyId=" Alice : Delegation : allowed" Version=" 1.0"
  RuleCombiningAlgId=" &rule-combining ; permit-overrides ">
<Target>
  <AnyOf>
    <AllOf>
      <Match MatchId=" &function ; string-equal ">
        <AttributeValue DataType=" &xml ; # string ">
          network-admin
        </AttributeValue>
        <AttributeDesignator
          Category=" &attribute-category ; delegated :
            &subject-category ; access-subject" AttributeId=" group"
          MustBePresent=" false" DataType=" &xml ; # string "/>
        </Match>
      </AllOf>
    </AnyOf>
    <AnyOf>
      <AllOf>
        <Match MatchId=" &function ; string-equal ">
          <AttributeValue
            DataType=" &xml ; # string ">database</AttributeValue>
          <AttributeDesignator
            Category=" &attribute-category ; delegated :
              &attribute-category ; resource"
            AttributeId=" &resource ; resource-id"
            MustBePresent=" false" DataType=" &xml ; # string "/>
          </Match>
        </AllOf>
      </AnyOf>
      <AnyOf>
        <AllOf>
          <Match MatchId=" &function ; string-equal ">
            <AttributeValue
              DataType=" &xml ; # string ">backup</AttributeValue>
            <AttributeDesignator
              Category=" &attribute-category ; delegated :
                &attribute-category ; action"
              AttributeId=" &action ; action-id" MustBePresent=" false"
              DataType=" &xml ; # string "/>
            </Match>
```

```
</AllOf>
</AnyOf>
<AnyOf>
  <AllOf>
    <Match MatchId="&function;string-equal">
      <AttributeValue
        DataType="&xml;#string">Alice</AttributeValue>
      <AttributeDesignator
        Category="&attribute-category;delegate"
        AttributeId="&subject;subject-id"
        MustBePresent="false" DataType="&xml;#string"/>
    </Match>
  </AllOf>
</AnyOf>
</Target>

<Rule RuleId="Permit:Delegation" Effect="Permit"/>
</Policy>
```

The first `AllOf` defines the attribute a subject (the delegate) must have (in that case *network-admin*) so that Alice can delegate the permission. The second `AllOf` defines the resource and the third `AllOf` defines the action that Alice is allowed to delegate to a user of the *network-admin* group. The last `AllOf` specifies the subject that is allowed to delegate the permission, which is Alice.

In order to allow users to delegate permissions, the policy administration point or a similar component needs to provide some kind of self-service interface for administrators so that they can establish such temporary policies without being able to manipulate other rules that have been created by an authority.

2.2 Distributed Enforcement of XACML Policies With pam_xacml

The benefits of using a policy language like XACML can only be used if the applications support the standard. Since most of the applications do not have an integration for such policy languages, there is no way to provide centralized security policies for them directly.

However, in UNIX-like operating systems there is a level of abstraction which can be used by applications to integrate an independent authentication mechanism. This can be achieved with the *Pluggable Authentication Module* (PAM), which provides support for *authentication, account management, session management and password management* [22].

Research has shown that PAM can also be used for authorization schemes. One paper introduced *pam_xacml*, a PAM module that enables every application that implements PAM to support XACML authorization. It was developed as a prototype in the paper *Pluggable Authorization and Distributed Enforcement with pam_xacml* by A. Klenk et al. [43]. The ap-

proach allows developers to connect every PAM-enabled application to a PDP and thereby unify the authorization policies.

The PAM module can operate in a legacy mode that does not require changes to the applications and another mode that provides a more powerful authorization but therefore requires changes to the application. The module is activated like any other PAM module and can be configured with settings regarding the connection to the PDP and the path to a template that is used for authorization requests [32].

2.3 Security Assertion Markup Language

The Security Assertion Markup Language (SAML) is an XML-based open standard that specifies a format for exchanging authentication and authorization messages between an *Identity Provider* (IP) and a *Service Provider* (SP). The messages that are exchanged between the providers are called *Assertions*, which are separated into authentication, authorization, and attribute assertions. The exchange of an assertion is realized by a request-response protocol and triggered by the service provider [45].

The identity provider acts as an *Asserting Party* (AP) and is the central component that hosts, authenticates, and authorizes an identity. The service provider instead is the *Relying Party* (RP) that requests all required information from the identity provider [55].

In the area of federated identity management in web environments, SAML is often used to implement single sign-on between different organizations and websites. It enables administrators to centrally manage the identities and privileges [8]. Since the authentication is performed on a dedicated system (the identity provider), the threats, auditing, and control are located in one place. Hence, a service provider is never passed sensitive data like user credentials.

With SAML, one entity (e.g., a website) can pass information about a subject in the form of attributes to another party that utilizes them for an attribute-based access control [77]. However, one downside in contrast to XACML is that SAML does not define a processing model that describes how to express and interpret policies.

Certainly, SAML supports extensions which provides a mechanism to extend it with a fine-grained access control by combining it with XACML [45]. On the other hand, by adding SAML to XACML, it is possible to protect integrity and authenticity by signing access request and response messages [56]. If this is applied to the communication between the PDP and PEP, which were introduced in Section 2.1, the correctness of the messages can be verified and manipulations can be detected, which is a gain in security because attackers cannot tamper with the authorization messages and thereby exploit the access control.

2.4 OpenID

OpenID is an open standard that implements a federated authentication mechanism in order to build a decentralized security architecture for web-based services.

The architecture of OpenID consists of three components:

- the *end user* that wants to use a specific service and for this purpose has a unique OpenID identity,
- the *OpenID Provider* (OP) that hosts the OpenID identities and authenticates users,
- the *Relying Party* (RP), for example, an application or website that wants to use the OpenID information and thereto delegates the authentication to an OpenID provider.

The authentication process works as follows:

1. The end user authenticates on a website (the relying party) that provides a login with an OpenID identity by entering the previously registered identity, for example, *alice.example.com*.
2. The RP normalizes the supplied identifier and requests a document from the discovered OpenID provider.
3. The RP interprets the document and extracts necessary information to construct a new URL to which the user agent is redirected. The URL includes a URL to which the user agent should return to in case the authentication was successful and another URL to which the user is redirected in the case of a failure.
4. After the end user has been redirected to the OpenID provider that hosts the identity, the user is presented a login form. There, he enters the login identity and a password, which are evaluated by the OP after the user submitted the form.
5. The OP performs an authentication and redirects the user agent to the appropriate URL (see step 3).
6. The RP might now verify the information that has been passed by the OP and the user can use the services provided by OP since he is now authenticated [41].

Like SAML, the OpenID framework provides a single sign-on mechanism but it does not have a single sign-out mechanism. In order to authenticate a user, the OP can consult a database or directory (e.g., MySQL or LDAP) [39].

In contrast to SAML, which usually enforces that the identity provider and service provider trust each other, OpenID is used between websites that are unknown to each other. This allows a developer or administrator to establish an authorization for external parties that are not known in advance. With SAML, it is necessary to establish a trusted link between the parties first [50].

Certainly, because of the untrustworthy relationship in OpenID, the RP can only trust

the OpenID identity, which is represented by a unique Uniform Resource Locator (URL). All the other information, for example, an email address or the person's name cannot be trusted because there is no guarantee that the OP validated the information.

2.5 OAuth

OAuth is an open standard for authorization that enables third-party clients to access protected resources on behalf of a user. In contrast to SAML and OpenID it does not provide an authentication mechanism.

The advantage of OAuth is that the access to resources can be protected by the end user in a fine-grained way. It allows the user to define which application is authorized to access which data. Moreover, the user can revoke granted permissions so that an application cannot access the data beyond that revocation. In other solutions, a user would have to change the password to lock out the application.

With OAuth, the application does not require the credentials of the user to access the protected resource, which makes the credentials more secure [41]. In order to access the data on behalf of the user, the application needs to request a token, which is only issued if the user granted the necessary access rights to the application.

The three main components of OAuth are:

- the *Resource Owner* (mainly represented by a user agent (e.g., a web browser)) that wants to use a specific service,
- the *Resource Server* that hosts the user's resources which can be accessed by a Client,
- the *Authorization Server* that issues access tokens after authenticating the end user,
- the *Client* that requires access to protected resources (which are stored on the service provider) in order to fulfill the Resource Owner's request. This could be, for example, a website, desktop application, or mobile application.

The authorization process works as follows:

1. The Resource Owner wants the Consumer to access a resource that is hosted on the Service Provider.
2. The Client requests authorization either directly from the Resource Owner or transparently via the Authorization Server.
3. The Client receives a credential that represents the authorization grant from the Resource Owner.
4. The Client requests an access token from the Authorization Server, which requires an authentication first.
5. After the authentication was successful, the Client presents the authorization grant

which is validated by the Authorization Server.

6. In the case the authorization grant was valid, the Client receives the access token and connects to the Resource Server.
7. The Client authenticates at the Resource Server and presents the access token, which is validated before the access to the requested resource is granted [30].

With OAuth, it is not possible to implement complex policies based on subjects, resources, and actions. For web-based scenarios the approach of OAuth usually suffices but in network infrastructures with privileged user accounts the XACML standard offers more flexibility and functionality.

2.6 Shibboleth

Shibboleth is an implementation for a federated identity-based authentication and authorization infrastructure. It uses the Security Assertion Markup Language (SAML) standard for the exchange of messages between the Service Provider (SP) and the Identity Provider (IP). In contrast to OpenID, in a Shibboleth architecture the service provider can trust the information that it receives from the identity provider because it asserts that the user is who he pretends to be.

The main components of the Shibboleth architecture are:

- the *user* that wants to access a resource,
- the *resource* that is protected,
- the *Identity Provider* (IP) which authenticates the user,
- the *Service Provider* (SP) which performs the single sign-on process [13].

Although Shibboleth supports *Attribute Release Policies* (ARPs), the format has some downsides. For example, it does not allow to group attributes and it only supports simple conditions. Moreover, there are no obligations which is an important functionality in the area of access rights of privileged user accounts. Thus, replacing the ARPs of Shibboleth with a more flexible access control policy language like XACML is suitable for some organizations [33].

3 Threat Model

In the last couple of years the infrastructures of many organizations changed significantly. Not only the trend towards heterogeneous networks with different systems and the increasing amount of users with different access rights create a challenge. Also the distributed and dynamic environments that are comprised of systems that come and go need to be taken into account. Furthermore, the accompanying connection of systems over the Internet increases the security risks and attack vectors. Subsequently, the growing market of cloud computing and the outsourcing of departments as well as the remote administration brings up more challenging aspects.

For that reason, it is important to isolate the most critical data of the system and protected it from insiders and external attackers with contemporary security mechanisms [51]. Most notably, this applies to systems that are managed by a lot of people which have privileged access rights and use shared accounts to maintain the infrastructure. This is because the abuse of privileged user accounts is a growing threat.

In a recent study, Ponemon Institute and IBM surveyed 265 C-level executives. One result of the survey was that “ninety percent of senior executives surveyed say their company has had a data breach and almost half (forty-eight percent) expect more data breaches to occur” [36]. The chart in Figure 3.1 shows that they identify negligent insiders as the greatest threat to sensitive data, followed by lost or stolen devices. Twelve percent identified malicious insider attacks as a risk to sensitive data.

3.1 Basic Terms

In information security, there are three core goals: the *confidentiality*, *integrity* and *availability* of information and information systems. These terms are also referred to as the *CIA triad* or *information security triad* [12].

Confidentiality “[...] means preserving authorized restrictions on access and disclosure, including means for protecting personal privacy and proprietary information” [44 U.S.C., Sec. 3542].

In the area of privileged user password management a breach of confidentiality is, for example, that an unauthorized person gains access to a password (e.g., by stealing a device that contains a list of passwords or by handing a password to someone that is not authorized to know it). The consideration of confidentiality of privileged user passwords can be extended to the data that are only accessible with elevated access rights. If the confidentiality of privileged user credentials is not given, the confidentiality of the data cannot be

The source of greatest risk to sensitive data

(Two choices permitted)

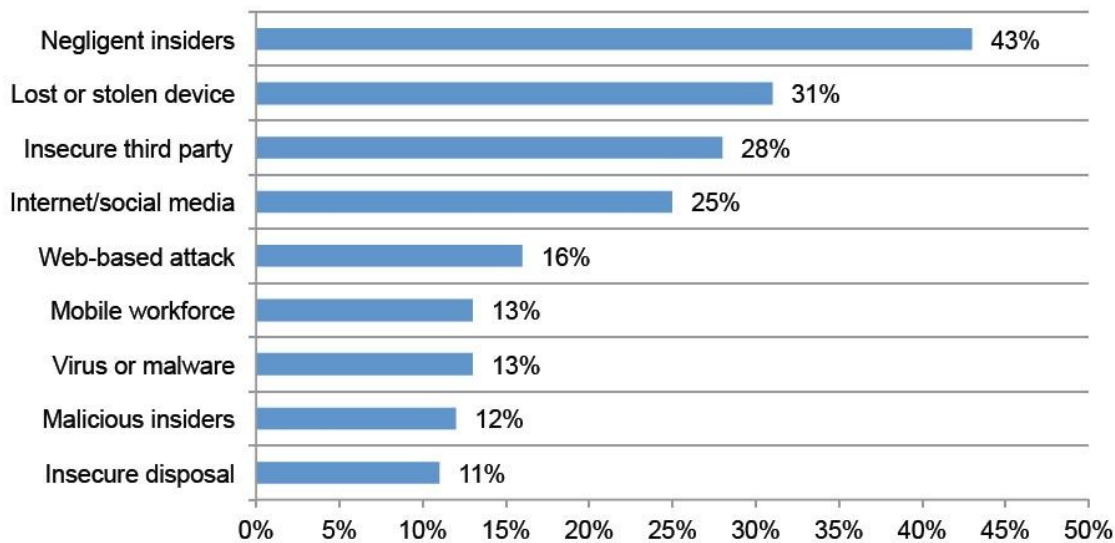


Figure 3.1: The Source of Greatest Risk to Sensitive Data

Source: Ponemon Institute and IBM survey of 265 c-level executives, February 2012

guaranteed as well.

Integrity “[...] means guarding against improper information modification or destruction, and includes ensuring information nonrepudiation and authenticity” [44 U.S.C., Sec. 3542].

If the integrity is not guaranteed, unauthorized modifications and destruction can happen without noticing, which leads to untrustworthy information and systems. This especially applies to privileged accounts since administrators can perform malicious or accidental actions that cause a breach of integrity. Moreover, if the access control cannot guarantee a separation of duties, the violation of integrity cannot be detected if the privileged user covers the tracks by manipulating audit logs.

Availability “[...] means ensuring timely and reliable access to and use of information” [44 U.S.C., Sec. 3542].

If the availability goal is not met, it will lead to a disruption of service or information will be inaccessible. Thus, the security mechanisms that protect and monitor the availability of systems need to be made accessible only to authorized administrators. If a privileged user can lock down important network components, it can cause a huge damage to the business of an organization.

Beyond that, there are other principles of information security that are used in this thesis:

Threat. “A potential for violation of security, which exists when there is an entity, circumstance, capability, action, or event that could cause harm” [28].

Attack. “An intentional act by which an entity attempts to evade security services and violate the security policy of a system. That is, an actual assault on system security that derives from an intelligent threat” [28].

Non-repudiation. “Assurance the sender of data is provided with proof of delivery and the recipient is provided with proof of the sender’s identity, so neither can later deny having processed the data” [2].

In the area of shared privileged user passwords it is important that a person cannot disclaim actions that have been performed with a shared account.

3.2 Scenarios Concerning Privileged User Password Management

Example Corporation is a fictitious organization that is representative in having an heterogeneous IT infrastructure like many other companies in the world.

The network consists of different types of servers which host or serve different types of services. There are database servers that run the Solaris operating system and serve Oracle databases that are used by various applications. Furthermore, there is a Linux web server that hosts the company’s website besides several web-based applications that allow employees, partners, and customers access to Internet and Intranet services. In order to manage the authentication and authorization of users and computers they are using a directory service that runs on a Microsoft Windows server. Last but not least there are file servers and backup servers that also host sensitive data.

The employees use desktop computers as well as mobile devices like notebooks and smartphones, which run different operating systems.

Alice, Bob, and Charlie are employees at Example Corporation but they are working in different areas. Alice is a database administrator while Bob is responsible for managing all the UNIX servers in the network. Charlie is also a server administrator but he mainly takes care of the Windows machines. Both, Bob and Charlie also administrate the routers and switches in the infrastructure. Peter is the boss of the server department while Paul is head of database department.

Scenario 1: Shared Accounts

Bob and Charlie both have access to the Windows as well as the Linux servers. For that purpose they share the privileged accounts (“root” on Linux and “Administrator” on Windows). However, they also have personal accounts which they mainly use to perform their day-to-day tasks and only in situations where a privilege elevation (e.g., with “sudo” on UNIX-like systems or the User Account Control (UAC) on Windows) cannot be applied, they use the privileged account.

Scenario 2: Password Changes

Because of the password policies, the administrators have to change the passwords on a regular basis. This requires a lot of time and effort, because of the high amount of systems they have. Since they cannot remember all the passwords, they are writing them down every time they have changed them on the target system.

Scenario 3: Simple Access Control

Moreover, Bob and Charlie are authorized to access the routers and switches and to perform the actions that are required to keep the infrastructure running. However, because of the simple access control the network components only support one privileged account that needs to be shared by Bob and Charlie. Here, they cannot use their personal account and require access to the privileged user password.

Scenario 4: Emergency Access

Bob is on vacation and one system that usually is taken care of him has an urgent problem. Since it is a critical component the problem needs to get fixed promptly. Charlie is skilled enough to find and solve the problem, but he has no credentials nor is authorized to log in on the system.

Scenario 5: Hard-Coded Credentials

Bob needs to set up a backup task on the database server that exports and transfers all the databases to the backup server. For that, he needs to insert the credentials (username and password) of a privileged database user into the backup script.

Scenario 6: Separation of Duties

Because of her day-to-day tasks, Alice is only interested in the configuration on the servers that is related to the database infrastructure. Everything else is not her business and thus she does not know how to manage the other services (e.g., the directory service) in the network. Since Bob is the server administrator, he knows about every configuration on the server-side but he is not informed about the things that are going on inside the services. So his job is to provide a working infrastructure for the services (e.g., manage the firewall and update the systems) but it is up to Alice to manage the databases. Once Alice needs some server-related changes, she has to ask Bob to take care of them.

In order to fulfill all the database-related tasks, she uses the privileged superuser account (e.g., "root" or "sysdba").

Scenario 7: External Accounts

Since Bob and Charlie also administrate systems from customer projects, they need to manage the credentials of those systems as well. For that purpose, they use a password list that is saved on a network storage.

Scenario 8: Auditing and Reporting

Peter and Paul have an interest that each of their employees have access to all resources they require to fulfill their tasks. On the other hand, they want to make sure that the system is as secure as possible and that no one interferes with someone else's work. To prevent from attacks, no administrator should have the privileges to access all systems and data in the infrastructure. Furthermore, they want to be notified at every suspicious action and to be able to see which changes have been made by which person. Additionally, Peter and Paul want to assure that unauthorized persons do not have access to systems, including former employees and partners.

Scenario 9: Fine-Grained Access Assignment

As part of a new project Alice joins the webmaster group to help them with the database layout. During that time she needs the same permissions as the webmaster group.

Scenario 10: Platform-Independent Access

In emergency cases, the administrators sometimes use their smartphone or a private computer to access the list of privileged user passwords.

Scenario 11: Access of External Administrators

The financial application is hosted on the Solaris system. Because of the application's complexity, the software is administrated by the software producer and not by the organization's administrators. For that purpose, external persons require access to the server and sufficient access rights.

3.3 Threat Targets, Threats and Attacks

An attacker might try to find vulnerabilities at the network, host, or application level [47]. Many networks use the distributed client-server architecture with separate machines for clients and servers. Here, a client communicates with services that run on one or many

servers (e.g., web or database servers). In order to represent the different components of the network in a logical way, the threat targets are grouped into three areas: *hardware*, *network*, and *account*.

3.3.1 Hardware

An attacker could try to get access to the hardware by breaking into the computing center, but it is more likely that he steals a client (e.g., a laptop or smartphone) that belongs to the organization and is authorized to connect to the infrastructure [37].

Thus, the consideration of threats to the hardware is divided into the *server* and *client* as threat targets.

Threats to the Server

An insider might be authorized to enter the server room and could steal disks with sensitive data directly without using an end device or the credentials of privileged users to gain access.

The same applies to backup devices (e.g., tapes, CDs, and hard disks) which contain sensitive data that usually is only accessible by authorized users but the backups often are less secure. In such cases, for an attacker (usually an insider) it is not necessary to possess a privileged user password since the data is not protected by an authentication and authorization mechanism. A countermeasure is to encrypt the backups and keep them in a secure place (e.g., a safe to which only managers have access to).

After stealing the hardware or copying sensitive data, an attacker could crack passwords or search for configuration files and scripts that contain hard-coded passwords in cleartext.

Threats to the Client

A common attack to spy on passwords is to compromise an end device with viruses and keyloggers, to get access to credentials.

As already mentioned, an attacker could steal an end device (e.g., a laptop or smartphone) that contains credentials. This could be cleartext password lists or application-specific password vaults that save the credentials for a user-friendly authentication.

Once the attacker has access to the privileged user passwords, he can break into the systems to which the credentials belong to. A countermeasure against that attack is to encrypt the password lists or avoid saving them on end devices. Instead, the privileged user passwords should be kept on a dedicated system so that they can be secured and centrally managed.

3.3.2 Network

Besides attacking the clients and servers, an attacker can cause security breaches to network components like routers, switches, bridges, gateways, and firewalls.

Sniffing

Instead of stealing the data from the server or capturing the traffic that is sent and retrieved directly on the host, an attacker might aim to get access to network components or directly capture the network traffic [37]. This allows the attacker to read and manipulate the data that is transmitted between the components.

Hereby, the attacker could collect more credentials of privileged accounts that allow him to access other systems. Moreover, the attacker could tamper with data, for example, manipulate an authorization decision that would deny the access but because it was tampered now grants it to the attacker.

Thus, a countermeasure is to encrypt and sign sensitive messages before they are transmitted, so that the confidentiality and integrity of the data can be guaranteed [37].

Denial of Service

Since routers and switches are important for the functionality of a network, the availability goal needs to be considered. In order to manage the network devices, administrators mostly use privileged accounts because these devices often have no or only a simple access control. This implies that they have to share the password for the privileged account and it is often not possible to log the actions an administrator performed on the device.

The knowledge of privileged user passwords allows a malicious insider to sabotage the network by misconfiguring the core network components, which results in a denial of service.

A countermeasure is to block the direct access to the target system and use a gateway that authenticates and authorizes an administrator. If the access is allowed, the gateway will establish a connection to the target system without requiring that the administrator knows the password. In addition, the gateway can log the actions that have been performed and account them to a specific person. An example for such gateway is the *Shell Control Box* sold by the company BalaBit¹.

In order to diminish the vulnerability of external attacks, the administrators could establish a firewall that blocks the direct access to servers from the Internet and only allows connections from trustworthy clients [59]. Moreover, external attacks can be reduced by limiting the authorized connections (e.g., by only allowing the access from trusted clients or over a Virtual Private Network (VPN)) [37].

¹<http://www.balabit.com>

The VPN approach comes in handy for infrastructures with systems that are administrated by outsourced departments, too. Referring to Scenario 11 (Access of External Administrators) in Section 3.2, the remote access of the external administrators could be restricted to the Solaris machine.

3.3.3 Account

The most threats come from privileged accounts, once they are abused by an attacker or insider.

Credential Theft

Besides the above mentioned theft of hardware that contains sensitive data, including the credentials of privileged accounts, an attacker might have physical access to password lists that are written down on paper and can be read by any person that has access to the office. Another critical threat is that an attacker gains access to an unencrypted password file that resides on an unprotected storage [68]. Additionally, if a privileged user forgets to log off from a shared end device, other users can exploit sensitive data, too.

A countermeasure to this threat is to establish a multi-factor authentication. Often, it depends on the system if an advanced authentication is feasible and necessary. For critical components it is a noteworthy countermeasure to protect from stolen privileged user credentials and from unauthorized insiders. Moreover, the access time could be limited so that a user is logged off automatically after a specific period of time.

Social Engineering

The technique of social engineering is an attack that tries to divulge confidential information of a person. In the case of privileged user passwords this could be the sharing with a colleague who might not be authorized to know the password. This attack is likely to happen inside the organization but also outsiders might use the attack to find out credentials [46].

A countermeasure to this threat are one-time passwords that are generated for each access to a target system.

Lack of Individual Accountability

Often, it is not possible to account the actions that have been performed by an administrator to a specific person. This is because the privileged accounts are shared, either for the ease of use or because the target system does not support an access control and personal accounts.

This causes repudiation threats which are associated with users that performed actions with a shared account but later on deny having performed these actions [48]. Thus, if possible, each administrator should be able to login to the network device with his personal account so that repudiation attacks can be avoided. For all other situations, a gateway like the Shell Control Box can help to reduce the threat.

Password Cracking

Certainly, weak passwords are easy to crack and thus are a vulnerability that needs to be taken into account, too. In order to increase the password security, the organization could define password policies that ensure strong passwords and force a user to change the password on a regular basis (e.g., every six months) [31]. However, if the infrastructure contains a lot of systems and privileged accounts, the regular password change requires quite some time and human resources. This process could be automated, for example, with a password manager but it increases the risk that the process fails and systems become inaccessible.

As mentioned before, a multi-factor authentication could be established to increase the safety. This avoids that attackers can log in to systems with stolen credentials because they additionally need to provide another factor for authentication. This could be a biometric information (like a fingerprint) or something else only the user has (e.g., a token that generates one-time passwords) [15].

Moreover, the passwords should be kept secure on a centralized system and only a few authorized administrators should have access to the passwords they need to fulfill their ever day tasks.

Brute-Force Attacks

Besides stealing the credentials with keylogging or by scanning the disk of a client or server, a very common attack against the authentication mechanism of an application is the brute-force approach that is used to find valid credentials. Here, pre-existing superuser accounts (like "administrator" or "sysdba") have a high security risk since they are always named equally and publicly known. Because of that, an attacker is able to use a brute-force approach to find the password. This attack is more complicated when it comes to personal accounts, because the attacker also has to guess the username. There are different countermeasures against such attacks, including strong password policies and lockout strategies that block a user after a few unsuccessful trials [63].

In order to mitigate successful attacks, the server administrators should also identify unused accounts, services, and open ports that can be shut down. Moreover, it is important that administrators change default passwords or keys and monitor all log files as well as services in order to recognize abnormal activities on the servers. Often, unneeded services (e.g., Telnet or DNS) are a potential pathway for attackers into the system since they are not monitored or still configured with default settings, which are not as secure as they should

be. Furthermore, unpatched services are a high danger and thus the systems should be kept up-to-date in order to reduce vulnerabilities [31]. This also applies to the underlying operating system because exploits can be used by an attacker to get unauthorized access to the system or lead to data corruption [68].

Retrieval of Plaintext Configuration Secrets

Moreover, the file systems and the data they contain (e.g., configuration files that include credentials) are threat targets as well as applications that host data (e.g., databases) or provide authentication and authorization information (e.g., directory services).

Often, reoccurring tasks that are executed automatically (e.g., cron jobs on UNIX systems that run shell scripts) communicate with services (e.g., a database) which only grant access after the client has been authorized. For that, the client has to read the credentials from a predefined place in order to transmit them to the service or application. This place is often nearby the client, for example, a shell script that includes the username and password. Instead of a username/password combination, the application-to-application authorization could be handled with cryptographic keys or other authentication factors. However, if the credentials or keys are not protected in a secure way, attackers could steal them and masquerade as the application. The same applies to configuration files that include passwords, which should only be readable by those accounts that really require access.

A good mitigation against that threat is to remove hard-coded credentials with a password management solution that reveals privileged passwords on demand [74]. Moreover, the access rights of privileged users could be restricted so that an administrator cannot read and modify files that he is not authorized to access.

Abuse of Privileged Accounts

The above mentioned negligent users cause a security threat and thus the privileged user password management solution also needs to address human error. The risk can be reduced by following the principle of least privilege and separation of duties. Besides business processes and workflows which can enforce these regulations, a technical solution is a fine-grained access control that even restricts the privileges of superusers, depending on the identity that uses the account. Moreover, logon hours can ensure that the access is only granted during predefined time intervals.

Besides administrators that act negligent, the malicious insiders which have the knowledge about the infrastructure and passwords to privileged accounts cause a security breach, too [67]. These privileges can be abused to get access to systems and data they are not authorized to or sabotage core components of the infrastructure which causes an unavailability of important systems.

An attacker that has access to privileged accounts can also compromise the system (e.g., by installing malicious software) and manipulate the data that is sent or retrieved. Once

a legitimate user or an attacker is authenticated, it might happen that he tries to elevate the privileges to get more access rights and take control over the system or singular applications. In order to guard against such privilege escalation attacks, processes should be configured to run under least privileged accounts [48].

In order to prevent a privilege abuse, the administrators should apply a fine-grained access control for each user. In a database scenario this can be realized with a query-level access control. Moreover, a software vulnerability could be used to escalate the access rights of an account to those of a highly privileged account. Such a vulnerability can be prevented by using the above mentioned fine-grained access control and intrusion prevention systems [68].

Another approach is to deny the login for privileged accounts (e.g., "Administrator" on Windows or "root" on UNIX-like operating systems) and let administrators authenticate with their unique identity. In order to perform their tasks, they need to be assigned the necessary permissions or have the right for privilege elevation (e.g., by using "sudo" on UNIX-like systems). However, there are situations where devices (e.g., routers) only support one privileged user or where the assignment of access rights to the identities is not possible.

Unauthorized Access to the File System

A user that has all privileges in the file system can view, edit, and delete files that contain sensitive information. This not only applies to files that belong to the operating system (e.g., configuration files and logs) but also to user-generated and application-generated data.

A countermeasure is to encrypt the data if possible or restrict the access for each individual administrator with access control lists. However, the approach of a fine-grained access control requires that permissions can be assigned to a personal account that is used by a unique identity. This often is not possible with a shared superuser account because such privileged accounts cannot and should not have access limitations. In such cases, the privileged superuser accounts could be reserved for emergency situations only, and separate accounts with the least required privileges could be created instead. One example is the creation of an account for database administrators, that only has assigned the privileges to perform actions related to the database.

Moreover, administrators often have access to directory services, databases, and audit services. This allows an attacker to spoof an identity or assign more rights to another identity without having the authorization to do so. Besides that, an attacker can steal, manipulate, or corrupt the information inside databases, which can cause unforeseen problems to the organization. Furthermore, if an attacker gains access to the audit logs, he can cover his tracks by deleting all actions that he performed. Thus, the principle of least privilege needs to be applied to the privileged accounts and only authorized administrators should have access to these accounts. The separation-of-duties approach should be used to separate privileged accounts and assign only those rights that are necessary to perform the

tasks (e.g., a router administrator should not have the privileged user password for the database that hosts the audit logs).

The principle of least privilege and separation of duties also helps to address the threats that arise from the access of external administrators, like depicted in Scenario 11 (see Section 3.2). If the privileged user account that the outsourced administration uses only has the rights to access files and databases as well as to execute specific commands they require for the support of the financial application, the other systems in the network as well as other sensitive data that is hosted on that system are more secure.

4 Requirements

The major goal of a privileged user password management solution is to increase the security by reducing the risk that comes with privileged accounts. In order to reach the goal, it might be of interest to formulate different restrictions. One example is a rule that enforces logon hours and only allows access during a given time of day, to restrict the access during non-working hours. Another example for environmental restrictions is the physical location of the client, that issued a password request. Furthermore, the organization might want to set up rules that deny the access to specific services or resources based on the IP address, once the user is logged in on the system. Other restrictions could be a rule that only allows access for one user at a time or a policy that denies multiple accesses of the same identity.

The following sections will categorize the different requirements which resulted from the scenarios in Section 3.2 and the threats in Section 3.3.

4.1 Operational and Business Requirements

Some of the business requirements for the privileged user password management solution can be derived from regulations that the organization has to address.

For example, if an organization in the United States deals with health data, it has to ensure the security and privacy of that data. The security standard is regulated in the “Health Insurance Portability and Accountability Act” (HIPAA), which prescribes that employees that have access to protected data need to be identified and that the privileges must be restricted to those that are required to fulfill the tasks [58].

In order to obey the regulations, many organizations have prepared business processes that define activities which are performed by the employees to achieve a certain goal and that regulate the responsibilities and privileges of each person. To realize the security regulations, business leaders require that the IT infrastructure provides the necessary support and that business processes are implemented there as well.

Thus, administrators need to be able to have access to the systems they are in charge of to fulfill their day-to-day tasks and keep the infrastructure running. This not only applies to the organization’s infrastructure but even more to infrastructures that are supported or operated by the organization (e.g., customer projects of a consulting firm).

The operational and business requirements are related to Scenario 8 in Section 3.2. Here, the heads of department have an interest that the employees conduct to processes and

regulations to secure the sensitive data they operate with.

4.1.1 Security Regulations

In these situations it is important that administrators have the privileges they require but it is also important that security constraints are met so that attackers or insiders cannot sabotage the infrastructure or steal sensitive data. This would endanger business associate contracts and the reputation of the organization. Thus, companies must formulate security processes and mechanisms as well as a security awareness of the employees [79].

Regarding the passwords, the organization might want to formulate special password policies for privileged identities, to make them more secure. Including the change frequency and a stronger password as for identities without elevated permissions. This could comprise the requirement for automatic password changes, too.

With a centralized password management tool, passwords could directly be changed on the target system periodically as well as after the usage of a particular account. The result is that administrators need to check out the account they want to use and that the system provides them a one-time password. Additionally, the system might block other users from accessing the account as long as it is in use (checked out) by an administrator. After the task has been finished, the user checks in the account so that others can use it. At this point, the system should change the password so that the administrator cannot re-login with the previously revealed password. An advantage of the lock-out feature is that the “check out” and “check in” actions provide information about who had access to the account at a particular time, which ensures accountability and thereby addresses the “lack of individual accountability” threat in Section 3.3. Moreover, the centralized management of passwords addresses the threat that administrators manage the passwords on their devices or in password lists.

For situations where an administrator needs to perform critical actions or requires access to sensitive data, the organization should have control mechanisms. For that purpose, the organization might prescribe a four-eye-principle which ensures that an employee cannot perform a special task without the agreement of a supervisor. An example for that is the temporary elevation of permissions which needs to be requested and approved by the supervisor or a delegate. However, there are situations which require an urgent and unplannable access. For these cases, there might be exceptional regulations that grant special privileges without having an explicit approval from a superior in advance. In access control models, the approach of flexible policies is often referred to as a *break-glass strategy* [7]. Indeed, the requirement arises that a supervisor is informed automatically so that the action can be legitimated afterwards and that the action is logged.

Subsequently, another requirement is that the solution should provide some kind of access control that allows to represent the duties and responsibilities of every stakeholder that is involved in the process of managing the privileged accounts. On the one hand there are the stakeholders that can use the passwords and on the other hand there might be a group of managers who are the only stakeholders that are authorized to view the audit logs and

reports.

4.1.2 Information Distribution

In Section 3.2, Scenario 4 (Emergency Access) showed a problem of shared accounts that is not only security-related but also has an operational aspect. For an efficient IT department it is important that the access to a system and thus the knowledge of credentials to privileged accounts are not reserved for only one person, because that would cause problems when the employee leaves the company or is on vacation.

Moreover, in emergency situations an administrator should be able to solve a problem on behalf of a colleague, although the person usually is not authorized to work on such tasks. The organization might therefore prescribe rules which regulate who is authorized in what situation to perform which actions.

One solution is that the credentials of privileged accounts are put into a safe and only authorized persons (e.g., managers) can open the safe. However, the amount of credentials in huge infrastructures is hard to handle, especially when password policies define periodic changes. Moreover, the physical approach has the downside that administrators cannot fulfill an urgent task if no authorized person is available to open the safe. In a software approach, such a process can be modeled with an emergency access request that is submitted by the administrator and then approved by an authorized person.

However, such regulations usually cause a delay and prevent the administrator from solving the problem promptly. For such cases, the business processes need to be laid out and exceptions need to be defined. This requires a fine-grained description of what administrators are allowed to do and which privileged accounts they are authorized to use. The software solution should then implement an urgent access feature that allows access without having an approval by an authorized person.

Usually, this is part of the identity and access management lifecycle and is an ongoing process. Because of the time-consuming administration costs the fine-grained rules are often not applied to each identity but rules prescribe the exposure to shared privileged accounts.

Scenario 9 (Fine-Grained Access Assignment) showed another situation where fine-grained policies can help. If Alice requires additional rights that are equal to the ones of the webmaster group, the rights to access systems, services, and data could be granted to her identity. The access to privileged user passwords should be regulated by the policies, too. After she leaves the project, the policies and access rights need to be revoked and passwords should be changed.

Thus, the privileged user password management solution needs to provide mechanisms to make the information available to more than one identity and to allow access for other users that normally do not have access. However, it also must ensure that the principle of least privilege is applied when a user gets access to privileged accounts.

4.1.3 Separation of Duties

Scenario 8 (Auditing and Reporting) showed up that one requirement should be that administrators cannot perform actions that rise conflicts of interest. The separation of duties ensures that security-relevant actions are performed by at least two persons. This ensures security because administrators cannot perform actions and cover the tracks in the audit log if the separation ensures that no administrator has the manager role that is required to access the logs.

Thereby, the requirement addresses the threat “abuse of privileged accounts” and the unauthorized access to sensitive data because the four-eye-principle as well as the separation of duties provide a control mechanism.

4.1.4 Auditing and Reporting

Scenario 8 outlined that the managers want to be able to see what administrator has access to what system and what person is accountable for performing an action.

Thus, accounting and auditing are crucial requirements when it comes to the management of privileged accounts and passwords. It is important that every action is logged and accounted to a unique identity, so that non-repudiation is guaranteed. An authorized member of the organization should have the ability to review the reports or recorded sessions, in order to find abnormal activities or to understand an attack that has happened. The required security goals for the audit log are confidentiality and integrity. It is important to ensure that only authorized persons have access to the log and that an attacker must not be able to modify or forge the log entries after compromising a system.

From a business perspective, the product should provide secure audit trails and the possibility to reconstruct the actions that preceded an attack or malicious access. Hence, the principle of least privilege should be applied to the managed accounts as well as to the identities that have access to these privileged accounts, which requires a fine-grained access control. Moreover, the solution should allow to map operational workflows to the software so that requirements like a four-eye principle or urgent access needs can be realized.

4.2 Practicability Requirements

From a user perspective, the solution should be a time saver that supports the daily work and does not add another component that complicates each task. Therefore, if the product requires an authentication, it should be usable with the personal credentials. This could be achieved by adding single sign-on which ensures that the user does not have to re-authenticate for the management tool that gives him access to the privileged accounts.

Furthermore, the password management tool should allow to copy and paste passwords

to reduce the time that is required to log in to a system and it will also avoid typing errors.

Another requirement is that the system should be easy to use and – like stated in Scenario 10 – it should be platform-independent, so that an administrator can access the tool from every client. This could be achieved by a web-based user interface that can be accessed with a web browser from every device that is authorized.

Additionally, in the password manager the credentials should be organized and grouped in a way, that allows users to manage a lot of entries in a well-arranged way. Moreover, the grouping could be used to assign access rights not only on singular entries but also on groups of credentials (e.g., a group “routers” or “Linux servers”).

Another requirement that can reduce the work for an administrator is the login capability with the personal account which is then used to perform the actions that require elevated rights. On UNIX-like systems, this can be realized with *sudo* and on machines with Windows as operating system the *User Account Control* (UAC) can be used. This can reduce the need for shared privileged accounts but it requires a fine-grained access control that allows to assign rights to each identity. However, since this is not possible on each target system (e.g., routers and switches that do not offer an access control) and it is a time-consuming task to set up the policies, the infrastructure might provide a transparent approach for such logins. The administrator would then log in to a middlebox with his personal account and the middlebox might then perform the mapping to the privileged account on the target system.

4.3 Non-Functional Requirements

Besides the business requirements and the ones concerning practicability, there are common non-functional requirements that need to be considered in the analysis for a privileged user password management solution.

4.3.1 Availability

The availability of core components in the network is important for almost all systems. Thus, a system like the Policy Decision Point needs to be fail-safe because authorization requests depend on that component. If the PDP is unavailable, the access management of all connected systems stops working, too. Hence, for all critical systems there need to be backup systems that take over when the primary system fails or is not reachable due to the failure of another network component.

The above mentioned approach of using a middlebox requires a high availability of the component because once the middlebox is not functioning, an administrator could not log in on the target system if the middlebox is the only authority that has access to the privileged user password.

4.3.2 Systems Integration

In order to reduce the complexity of having to maintain separate identities and access rights in different places (e.g., if each product requires its own user and access management), the integration into the existing infrastructure is desirable. This could be the usage of a directory or database which already provides standardized access to identities and access rights.

Moreover, the management tool should provide some kind of import feature that allows to import privileged accounts and passwords. This could be achieved manually by importing text files or by scanning systems for relevant accounts and importing them. That requirement is a time saver in the introduction phase but also when new systems are added or removed. In dynamic environments where, for example, many virtual machines are managed, this reduces the time of keeping credentials up to date.

Another point is the interaction of components in a heterogeneous network for which a solution to privileged user password management should have some kind of support. This is a requirement in many networks which have Microsoft Windows and UNIX-like servers, routers, and bridges that run different operating systems.

4.3.3 Security

An important aspect is the security and confidentiality of the components and the data they handle. The solution should ensure that only authenticated and authorized persons have access to the data and that manipulation is impossible or at least detectable. This includes that audit trails are only accessible by authorized persons and that unauthorized administrators cannot manipulate the logs directly in the file system or database.

Besides the above mentioned approach of setting one-time passwords by the password management system, another requirement might be to hide a password. This is similar to the transparent approach that a user can log in with the personal account and the system performs the mapping to the privileged account on the target system. Here, passwords would not be visible to the user and the password management system might take care of opening a connection and handling the authentication in the background. However, this approach requires that the client application can be used non-interactively.

The database which hosts the credentials and the machine in which the password management tool is running on need to be made secure, so that an attacker cannot gain access to the database and retrieve all passwords. Additionally, the administrators that manage the system need to be trusted.

Referring to the “check-out” and “check-in” feature which was already mentioned as a business requirement, the access time might be limited by the management system, so that an administrator cannot block the account for weeks but only for a few hours. This limits the time range in which the abuse of a privileged account can cause a threat like the one mentioned above in Chapter 3 where a user forgets to log off from a system.

4.4 Use Cases and Requirements at the Leibniz Supercomputing Centre

The Leibniz Supercomputing Centre (LRZ) provides supercomputer resources for research to the main universities in Munich as well as access to the Munich Scientific Network (MWN). Moreover, they operate systems where decentralized operations are inappropriate (e.g., data servers, high performance systems, and archive systems).

Besides the supercomputing resources, they also provide

- central services like email, web, name, and directory servers,
- central archive systems, including a Storage Area Network (SAN), distributed file systems, and robot-supported storage systems,
- workplace systems for various kinds of requirements (e.g., multimedia and CAD), and
- the MWN which connects the central infrastructure with the decentralized systems of over 60 locations [10].

4.4.1 Infrastructure of the LRZ

Most of the systems that are administrated by the employees of the LRZ are part of the MWN, which consists of over 103,000 decentralized systems, including hundreds of servers (with Linux and Windows as operating system), about 50 routers, over 1,300 switches, and over 2,000 access points [17]. Additionally, the administrators have access to external hardware that sometimes requires support but does not belong to the daily business.

In order to manage all the systems, the organization of the LRZ is divided into different departments. The organization structure is depicted in Figure 4.1 and shows the responsibilities of the technical departments. Although the responsibilities are theoretically separated, many projects require that different departments work together and share privileged accounts.

4.4.2 Status Quo and Requirements

The identity and access management at the LRZ is implemented with directory services like LDAP and Active Directory. The infrastructure of LRZ-SIM (Secure Identity Management) provides authentication and authorization for internal accounts as well as for projects and facilities that are supported by the LRZ.

Besides the internal accounts of LRZ employees, there are also external stakeholders that have access to some of the services, including accounts from IBM for managing the supercomputer and temporary accounts that are not personal ones.

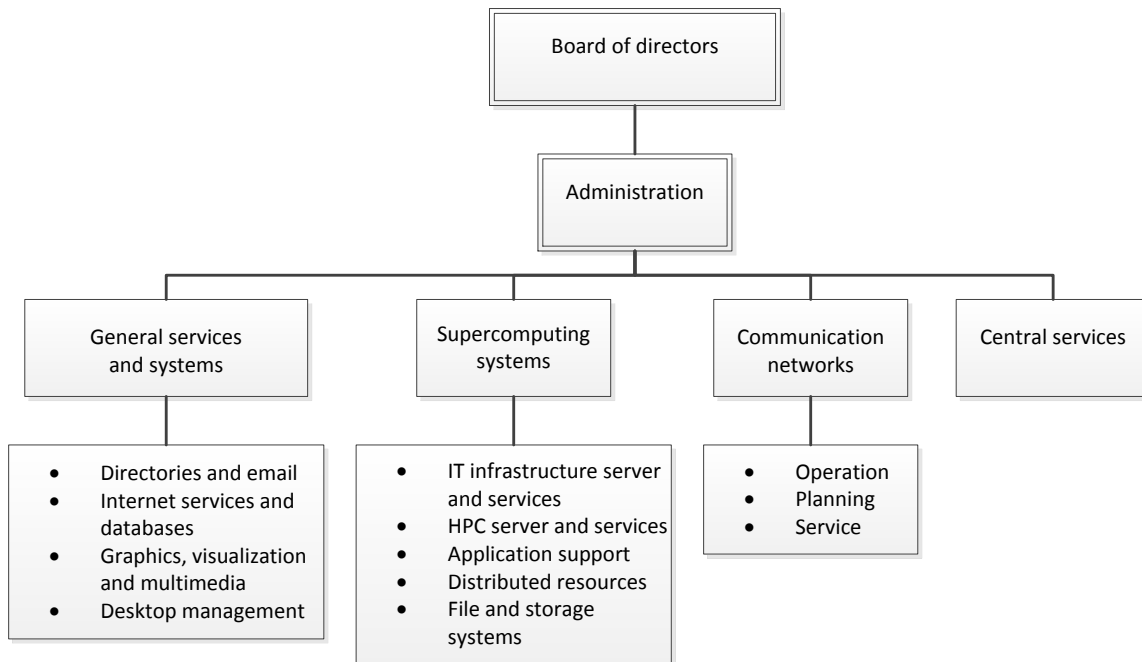


Figure 4.1: Organization Structure of the LRZ

At the moment, the management and sharing of privileged user accounts is realized by

- personal password lists which are managed by each employee individually
- and in a commercial password management system called *Password Safe Enterprise Edition* (PSE) which is sold by MATESO GmbH ¹.

Emergency Access

The LRZ has the requirement that the access to systems is allowed to every authorized person so that in emergency situations the access to privileged accounts is always guaranteed.

The requirement of a break-glass feature is already realized with the password management system. It provides a seal feature that allows authorized users to break the seal and gain access to the password if necessary. An auditor is then notified about the situation.

However, even for less urgent situations where an administrator can request an access to a privileged user password he currently does not have access to, the LRZ does not require an approval system. This is because the waiting time would be too long and thus the employees would not be able to continue their work.

¹<https://www.passwordsafe.de>

Auditing and Reporting

The data protection regulations of the LRZ require that no audit trails and no session recording is implemented. Thus, the requirement of having audit logs is not given.

However, the LRZ has the requirement for a report that shows which user has potentially access to which (privileged) accounts.

Automatic Password Change

The password policy at the LRZ requires that passwords are changed on a regular basis (at most every six months). This requires that the passwords are changed manually on the target systems as well as in the password management system and the personal password lists. The requirement for setting and changing passwords automatically is not given because the loss of control and the amount of work in the case of a failure is too high.

The requirement for keeping a password secure from colleagues is not feasible at the LRZ because responsibilities and projects overlap which requires that staff of different departments have access to the same privileged user password. This is due to the fact that the LRZ is part of the public service and thus different departments are working together which requires a password sharing.

Although the employees are encouraged to act responsibly, an insider attack would not cause the same consequences as for companies that are profit-oriented. Certainly, the LRZ needs to protect against attacks from within the MWN and keep their systems secure.

Fine-Grained Access Control

At the moment, the LRZ uses a role-based access control which does not provide any constraints to privileged accounts. Thus, a fine-grained access control in the operating system of the servers is a requirement that would be nice to have, but the implementation is contradictory to keeping the complexity of and the time for managing the identities and access privileges low. Another reason against restricting the access of privileged users is that no employee should be prevented from working freely and doing research in the field. Additionally, it is important that access is guaranteed at any time which requires that the centralized authentication and authorization infrastructure is failsafe.

Information Distribution

The solution needs to provide access to privileged user passwords to more than one person so that others can perform actions in the case the person who is mainly responsible is absent.

For the administrators the solution should be easy to use and not require much effort to

look up the credentials before they can connect to the target system. Thus, the LRZ prefers an integration into the existing infrastructure so that administrators can use single sign-on to get access to the password manager with their own account.

4.5 Use Cases and Requirements of iC Consult

iC Consult GmbH is a vendor-independent system integrator specialized in identity and access management solutions. Their portfolio includes solutions for social media and mobile login, the Internet of Things, and managed services for IAM [34].

4.5.1 Infrastructure of iC Consult

iC Consult is involved in various customer projects where they use and administrate external systems that are accessed with credentials that have to be known to everyone who is involved. Additionally, they have credentials which are required to access other external infrastructures (e.g., partner or supplier websites). Besides the shared privileged accounts in the external infrastructures, there are internal accounts that belong to the company's network.

For this reason, iC Consult requires a privileged user password management solution that allows an employee to distribute the passwords between authorized users by setting fine-grained access rights on the credentials.

4.5.2 Status Quo and Requirements

At the moment, the credentials are either stored in an encrypted file on a network storage or in a web-based wiki system. The downside of this approach is the lack of a fine-grained access control, because once a person can decrypt the file which contains all credentials or has access to the wiki, all passwords are revealed. Moreover, in order to lock someone out, all passwords need to be changed and the access to the location where the credentials are saved needs to be revoked from the identity.

Auditing and Reporting

Additionally, the requirement for accountability has been raised, which means that a privileged user password management solution should offer a reporting functionality that lists the credentials to which a specific person has access to. This allows to only change those passwords that a particular user had access to, when the person left the company. Moreover, the other users can be informed about the password change.

Another requirement is the *elevated privilege request*, which provides a mechanism for users

that are not authorized to access a specific credential to request the authorization. A supervisor can then prove the request and grant the access or deny it.

The privileged user password management solution should be a *web-based user interface*, so that credentials can be managed and obtained platform-independently. The authorization if a user is allowed to access a particular password should be integrated with the existing Active Directory. Moreover, it should be possible to control which user or group has access to a credential.

As permissions model two levels have been brought up:

- the right to *view* and use credentials, which includes the right to share, too
- the right to *edit* entries, which also allows to control which user has access.

The typical use cases are listed in Table 4.1.

Table 4.1: Use Cases at iC Consult

Use Case	Description
List entries	A user requests a list of all entries that a particular user has access to.
List permissions	A user requests a list of all users that have a <i>read</i> and/or <i>edit</i> permission for an entry.
Add entry	A user creates a new entry for a password, PIN, license key or other privilege-related data.
Edit entry	An authorized user modifies an existing entry and all users that also have access to the entry get notified about the change.
Share information	An authorized user can access an entry and share it with others by copy and paste.
Revoke access	An authorized user can revoke the permission to read and/or edit an entry.
Embed entry	A user can embed an entry into another website on the Intranet. Before the entry is shown, an authorization check is performed and when it permits the client side include of the entry.

4.6 Requirements Catalog

The above mentioned scenarios and requirements can be summarized in a list of essential features that a privileged user password management solution must have as well as desirable but not essential ones that would be nice to have. Based on the MoSCoW technique by Dai Clegg, requirements labeled as *MUST* are critical for the functionality of a privileged user password management solution, while requirements labeled as *COULD* are nice to

have. Requirements labeled as *SHOULD* provide some benefits for the solution but are not critical for the functionality of the software [44].

Systems Integration

- *Support for heterogeneous networks* [COULD]
The software supports different operating systems (especially Windows and UNIX-like systems).
- *Centralized authentication* [COULD]
The system can be integrated into the single sign-on infrastructure so that the authentication is based on the unique identity that already exists in a directory (e.g., LDAP or Active Directory).
- *Centralized authorization* [COULD]
The access rights can be retrieved from the directory and are not stored separately in the database of the management tool.
- *Account discovery/synchronization* [COULD]
The software is able to scan a specific system for relevant accounts and import them. This applies to privileged accounts like *root* and *Administrator* but also to service accounts that are executed with elevated rights. Since they are only locally available and not managed in a central directory, they need to be detected on each system individually.
Alternatively, the software could provide a bulk import feature (e.g., with CSV files) that allows administrators to manually synchronize the data set.

Access Control

- *Fine-grained access control on passwords* [MUST]
The software provides different access rights that allow different levels of authorization (e.g., “read”, “write”, and “is allowed to break the glass”).
- *Emergency access* [MUST]
The system provides some kind of break-glass or seal feature that allows administrators to access privileged user passwords in emergency situations.
- *Separation of duties* [MUST]
The system offers different roles that, for example, separate a user from an auditor, so that an administrator cannot view or manipulate the audit logs or session recordings.
- *Support for standardized policy language* [SHOULD]
The software supports an open standard for defining access control policies (e.g., XACML).
- *Policy transformation mechanism from high-level policy language to* [SHOULD]

low-level MAC policies

The software transforms policies from a standardized high-level policy language (e.g., XACML) to concrete low-level mandatory access control policies (e.g., SELinux or firewall rules). This makes it possible to have a standardized process of how policies are defined and applications without support for a standardized policy language can be integrated into the concept, too.

- *Fine-grained access control on operating-system level* [COULD]
The software is able to enforce access rights in the operating system, so that even users that are logged in as privileged user can only perform actions that are assigned to their identity.

Password Management

- *Password security* [MUST]
The system stores the passwords securely so that administrators cannot bypass the management system and reveal them in the database.
- *Account check-out* [SHOULD]
The system can check-out and check-in accounts which ensures that only one user at a time can use the account and each of the two actions is logged.
- *Account check-in and one-time password* [SHOULD]
The system changes the password to a random one-time password after the account is checked in, so that a re-login with the previously known password is not possible.
- *Copy and paste a password* [SHOULD]
The system provides a user-friendly approach to copy and paste the password from the management tool.
- *Platform-independent access* [SHOULD]
The software can be accessed easily from a device and does not require a special client.
- *Limited access time* [SHOULD]
The system limits the time a user can check-out a privileged account so that it is not blocked for a very long time.
- *Hide password from the user* [COULD]
The software provides a way to open a connection to the target system without revealing the password to the user.
- *Support for applications* [COULD]
The system provides a feature that allows applications and scripts to retrieve a password. This renders hard-coded passwords unnecessary.
- *Automatic password change* [COULD]
The system is able to set new passwords and change them on the target system.

- *Password policies* [COULD]
The system can be configured to follow the organization's password policy and create secure passwords or periodically change the password.
- *Request/confirm access to privileged account* [COULD]
The system provides a way for administrators to request an access to a privileged account that is confirmed by an authorized person. This feature is different from the break-glass feature since it is used in non-emergency situations.
- *Password organization* [COULD]
Users can organize the passwords in groups or folders to have them well-arranged.

Auditing and Reporting

- *Audit logs* [MUST]
The system provides information about who accessed a privileged account or performed an action on it (e.g., changed password, broke the glass, etc.).
- *Security of audit trails* [MUST]
The system stores the audit logs and recorded sessions in a way that privileged users (e.g., database administrators) cannot manipulate them.
- *Notifications* [SHOULD]
The system informs authorized persons about noteworthy situations (e.g., a break-glass access or the request for privilege expansion).
- *Account matrix* [COULD]
The systems shows a report that lists the accounts to which a specific user has access to.
- *Session recording* [COULD]
The system can record the entered commands or a Remote Desktop Protocol (RDP) session on the target system.

5 Analysis of Established Solutions

Several companies have discovered that shared and privileged accounts are a security risk and that it requires a sophisticated password management solution to securely manage and share the passwords between administrators.

This chapter analyzes three software products that differentiate in their product range and software licenses. Hence, one commercial product of a leading software corporation, one closed-source freeware solution, and an open-source software is evaluated.

Choosing the *CA ControlMinder* as the commercial software was based on the certainty that a demo environment was available. However, there are several other companies that offer commercial PUPM software, for example,

- “Privileged Password Manager” by Dell (former Quest),
- “Security Privileged Identity Manager” by IBM,
- “Enterprise Random Password Manager” by Lieberman Software, or
- “Privileged User Manager” by Novell.

The selection of *Netwrix Privileged Account Manager* was based on the feature list as well as the certainty that it is offered as a commercial as well as free version.

The product choice of the open-source product *Soffid Identity and Access Management* was determined by the reason that there is no other open-source identity and access management suite that additionally features a privileged user password management.

In the following sections, the important product features are outlined and compared to the requirements catalog from Section 4.6.

5.1 CA ControlMinder

CA Technologies, Inc. is an American company that offers a commercial solution for privileged identity management in physical and virtual environments. The product family is called CA ControlMinder and includes features like privileged user password management, shared account management, fine-grained access control, and user activity reporting. The following information is based on the CA ControlMinder solution brief [75] and the analysis of the demo of the formerly called CA Access Control.

5.1.1 System Architecture

This subsection will outline the important aspects of the system architecture.

Systems Integration

The CA ControlMinder product is comprised of several agents that need to be installed on the servers and integrate natively with the operating system. This integration is necessary to enforce and audit the fine-grained policies. The software provides agents for all major operating systems, including Windows, Linux, and other UNIX-like systems. However, the CA ControlMinder Management Server, which is the main component, requires a Microsoft Windows Server operating system. From there, all the agents on the different end-points are controlled.

Furthermore, the software can utilize an existing directory service with users and groups. It primarily supports Active Directory and the LDAP server of Sun ONE, but it is also possible to use the UNIX Authentication Bridge (UNAB), which maps UNIX attributes to non-standard active directory attributes. The downside of this approach is that it is less integrated and may lack some features. Hence, the best results can be expected in a heterogeneous network that consists of Windows and UNIX servers.

Shared Account Management

CA ControlMinder stores critical application and system passwords in a protected data store. Once a user requires access to an account that is part of the shared account management (SAM), he needs to check-out the particular password first. This can be achieved by using a web-based user interface that enforces policies to make sure only authorized users can use a shared account.

Another SAM feature is that CA ControlMinder generates temporary one-time passwords that are changed on the end-point system (e.g., a server or database). For that, the administrators can define password policies that need to be taken into account for automatically generated passwords. It is also possible to define time intervals in which new passwords should be created.

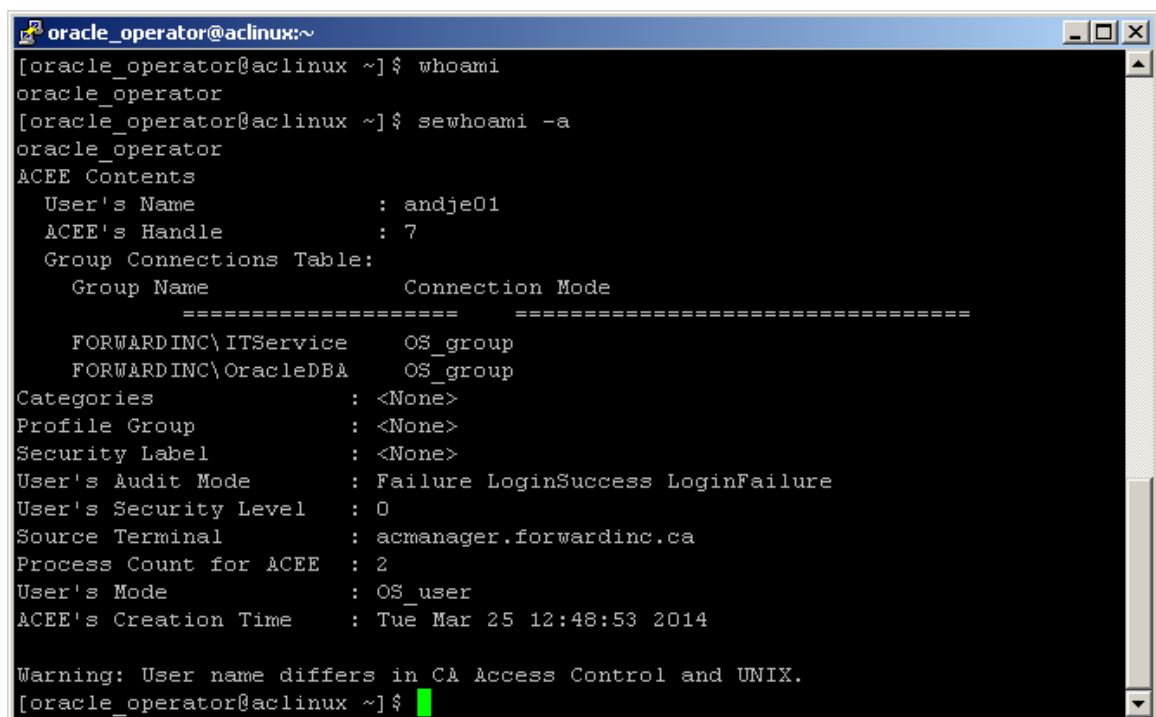
The communication between CA ControlMinder and the end-point system is based on established protocols, like JDBC for databases, SSH for UNIX-like systems, and Windows Management Instrumentation (WMI) for the communication between Windows systems.

Furthermore, the product supports the auditing and reporting of privileged accesses. For that purpose, the product can track every activity on an end-point system and correlate the various native logs that are created by the operating system and applications. The collected data is centrally saved and securely managed, so that only authorized users are allowed to access or modify the data. Moreover, the auditing daemons and logs are protected from attacks, shutdowns and tampering, which ensures integrity and availability. Additionally,

sessions can be recorded and viewed in a playback.

In order to ensure accountability, the SAM component provides an “exclusive check-out” that ensures that only one user has access to a shared account at a given time. Because of the check-out feature, the tracked actions can be assigned to the original user, which initiated the check-out. It also ensures that only authorized users can enhance their privileges (e.g., by limiting the use of the *su* command) and that the audit logs include the original account of the user that used the surrogate account.

The situation where an administrator is logged in with a shared account can be seen in Figure 5.1. The screenshot shows that user *andje01* is logged in as *oracle_operator* but the *sewhoami* command reveals that the actions are performed by *andje01*.



```

oracle_operator@aclinux:~
[oracle_operator@aclinux ~]$ whoami
oracle_operator
[oracle_operator@aclinux ~]$ sewhoami -a
oracle_operator
ACEE Contents
  User's Name           : andje01
  ACEE's Handle        : 7
  Group Connections Table:
    Group Name          Connection Mode
    =====
    FORWARDINC\ITService OS_group
    FORWARDINC\OracleDBA OS_group
Categories             : <None>
Profile Group          : <None>
Security Label         : <None>
User's Audit Mode     : Failure LoginSuccess LoginFailure
User's Security Level  : 0
Source Terminal       : acmanager.forwardinc.ca
Process Count for ACEE : 2
User's Mode           : OS_user
ACEE's Creation Time  : Tue Mar 25 12:48:53 2014

Warning: User name differs in CA Access Control and UNIX.
[oracle_operator@aclinux ~]$

```

Figure 5.1: Screenshot of PuTTY Demonstrating Non-Repudiation in CA ControlMinder

To achieve that, the CA ControlMinder inspects all relevant system calls and enforces the appropriate authorization policies. This level of control cannot be bypassed by anyone, not even by the superuser (e.g., *root* on UNIX-like systems or *Administrator* on Windows).

The additional layer that CA ControlMinder adds to the operating system provides an enhanced and fine-grained access control on Windows and UNIX-like systems. This can be seen in Figure 5.2 which shows that even though user *andje01* has elevated the privileges to those of *root*, he cannot perform all actions. Thereby, it is also possible to regulate the access based on environment attributes, like the time of day or network attributes. Furthermore, it offers the ability to assign specific administration rights to personal accounts, which usually are reserved for superusers only. This policy-based approach can be compared to

the one in Chapter 6 which is based on XACML.

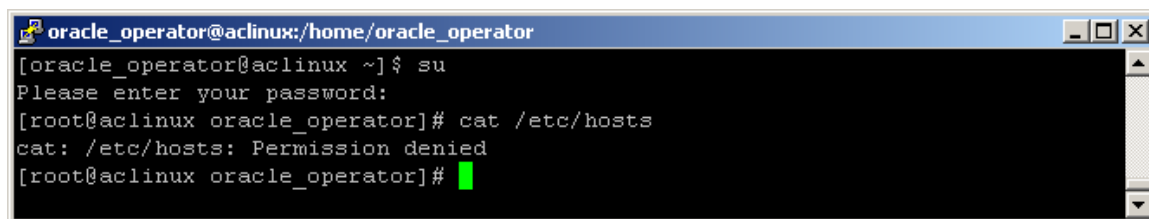


Figure 5.2: Screenshot of PuTTY Demonstrating Fine-Grained Access Control in CA ControlMinder

To protected from shoulder-surfing attacks, CA ControlMinder offers an automatic login feature that requests a password and utilizes it to log in the user to the target system as the privileged user.

For accesses to privileged accounts that are not part of the user’s role, the SAM component offers a four-eye-principle workflow that ensures that a person can use a specific privileged user but only for a short period of time. In order to use the account, the user has to submit a request via the web-baser user interface. A superior then decides whether he wants to grant or deny the request. Since this workflow requires the intervention of a second person and adds a delay, a problem arises in an emergency case. Hence, there is a feature called “break glass check-out” which allows the immediate access to a privileged account. In this case, the superior receives a notification message that informs about the emergency access.

Another aspect of shared account management is the communication between applications that use passwords to perform actions. CA ControlMinder manages the passwords of accounts like the Windows services and the Windows scheduled tasks. The shared account management can also be integrated into the run-as mechanism of Windows, which enables an application to retrieve the password from the secured data store.

A specific case of application-to-application communication is the interaction between an application (e.g., a web application) and a database. CA ControlMinder can intercept ODBC and JDBC connections and replace them with ones that include the current credentials of a privileged account. For that purpose, the administrators need to install an agent on the end-point system.

A further case are scripts and batch files that include hard-coded passwords. These can be replaced with calls to the Shared Account Management agent that checks out the password on behalf of the script.

5.1.2 Test Environment

The demo environment was provided as two pre-configured virtual machines that were executed with a VMware Player on a Linux host. The specification of the Windows server is given in Table 5.1 and the ones for the Linux server in Table 5.2.

Table 5.1: Environment Specification for CA ControlMinder (Windows Server)

Operating system	Windows Server 2003 R2 SP2
Memory	1,5 GB
Disk capacity	10 GB (C:) + 20 GB (D:)
Installed CA components	<ul style="list-style-type: none"> • CA Access Control Premium Edition R12.5 SP3 • CA Access Control R12.5 SP3 Report Portal • Enterprise Management Server • ObserveIT 5.1.0
Other software	<ul style="list-style-type: none"> • Active Directory 2013 • Active Directory Identity Management for Unix • IIS 6.0 • JBoss 4.2.3 • JDK 1.5.0 update 18 • Microsoft SQL Server 2005 • Oracle 10g Express

Table 5.2: Environment Specification for CA ControlMinder (Linux Server)

Operating system	Red Hat Enterprise Linux AS release 4
Memory	512 MB
Disk capacity	15 GB
Installed CA components	<ul style="list-style-type: none"> • CA Access Control Premium Edition for Unix R12.5 SP3 • CA Access Control Unix Authentication Broker R12.5 SP3

5.1.3 Analysis Based on the Requirements Catalog

This section evaluates the requirements based on the catalog from Section 4.6. A ✓ specifies that the requirement is fulfilled while an X signalizes that the requirement is not fulfilled. A ◇ declares that the requirement is partly fulfilled.

Systems Integration

Support for heterogeneous networks ✓

CA ControlMinder supports Windows, UNIX, Linux, and virtualized environments.

Centralized authentication ✓

The solution supports UNIX to Active Directory authentication bridging which provides a centralized authentication.

Centralized authorization ◇

The basic assignment of groups is saved in the Active Directory but all the specific roles and access rights which are configured in the web interface are stored in the database and deployed to the target system if necessary.

Account discovery/synchronization ✓

The software provides a feature called “Account Discovery” which searches for accounts on a given end-point (e.g., a MS SQL server or a server that hosts an Oracle database). The detected accounts can then be imported and a password policy can be configured. Moreover, it can be configured if the password should be changed on “check out” and “check in” actions.

Access Control

Fine-grained access control on passwords ✓

Privileged accounts can be assigned to a person individually.

Emergency access ✓

CA ControlMinder allows to specify accounts as break-glass accounts that a user can optionally display in the overview page of available privileged accounts. The user then can perform the break-glass action and has to provide a description for the reason of why the access is required. After the action has been submitted, the account is checked out for the user and the account can be used like any other account that the user is authorized to use.

Separation of duties ✓

The software separates the audit capability so that unauthorized users cannot view the

audit logs. Furthermore, the request for an access to a new account needs to be approved by another person and cannot be performed by the same administrator.

Support for standardized policy language X

The fine-grained access control of CA ControlMinder does not use an open standard for defining the policies.

Policy transformation mechanism from high-level policy language to low-level MAC policies X

CA ControlMinder does not transform policies to application-specific MAC policies.

Fine-grained access control on operating-system level ✓

With CA ControlMinder one can restrict privileged accounts based on the administrator that uses it. If, for example, Alice becomes root on a system, she might have other access rights than Bob when he is logged in as root. For that purpose, the product leaves the UNIX system of permissions intact but adds a layer of enhanced access control to it.

A special kernel module intercepts the file access operations and verifies that the user has authorization for the specific operation before returning control to UNIX. This allows an authorized person to give one person the right to view and modify a file like */etc/hosts* and deny it for another user, even though both persons use the same shared account.

Password Management

Password security ✓

CA ControlMinder encrypts all passwords with the symmetric-key block cipher RC2.

Account check-out ✓

Accounts need to be checked out before they can be used. A password policy can define if the account is then exclusively locked for the administrator or if others can login as well.

Account check-in and one-time password ✓

For each account it can be configured if the password should be changed after checking the account in. The password is then generated by on the configured password policy and changed on the target system.

Copy and paste a password ✓

The web interface allows a user to view a password as well as copy it to the clipboard.

Platform-independent access ✓

The management is realized with a web-based user interface that can be accessed from every device that runs a web browser.

Limited access time ✓

For each account a check out expiration time can be configured. This ensures that an administrator can use an account only for a given time.

Hide password from the user X

The web interface provides a way to launch login applications like PuTTY and RDP, which are started without the requirement for a password, but it is not possible to hide the options for showing the password and copying it to the clipboard. However, the approach of launching the application prevents from over-the-shoulder password theft and speeds up the login process.

Support for applications ✓

CA ControlMinder supports login applications like PuTTY and RDP that are passed the password as a parameter. Moreover, applications and scripts (e.g., batch files or shell scripts) can run on behalf of another account if executed by an authorized user. For example, Alice executes a database backup script that uses a client to contact the CA ControlMinder. The request contains the account name that should be used (e.g., "sql_backup") and the client will authenticate the currently logged in user. After that, CA ControlMinder returns the password for the requested account if the user is authorized to access the account.

Automatic password change ✓

The automatic password change is part of the password policy. The system allows administrators to configure a password expiration interval which then forces CA ControlMinder to automatically change the password.

Password policies ✓

The product supports the management of different password policies that can be applied individually to the privileged accounts. A policy can prescribe the minimum and maximum password length, the maximum repeating characters as well as the amount of upper and lower case letters, digits, and punctuations. It also supports a custom pattern and prohibited characters.

Request/confirm access to privileged account ✓

An administrator can view a list of available systems and accounts and send a request with a justification and the date until the access to the account should be granted. An authorized user can review the task and assign the requested account to the user or reject the request.

Password organization ✓

The web interface displays all accounts that a user is allowed to use in a list. This list can be filtered by an end-point (e.g., a host) or account name. Additionally, accounts can be grouped into "containers" (e.g., one container for SSH accounts and another one for Windows accounts). However, this grouping is defined by the person who manages the accounts, a normal user cannot re-arrange the grouping in custom containers.

Auditing and Reporting

<i>Audit logs</i>	✓
The logs contain detailed information about which user checked out/in an account, who broke the glass, and who approved an account request.	
<i>Security of audit trails</i>	X
The logs are saved as cleartext in the database and can be viewed and modified by the database administrators.	
<i>Notifications</i>	✓
The software allows to configure an email notification for specific events (e.g., when a break-glass action is performed or when a new request is available). Moreover, it displays notifications in the web interface (e.g., directly after the user logged in) to inform the user about new tasks.	
<i>Account matrix</i>	✓
The web interface can list all accounts that are assigned to a specific user.	
<i>Session recording</i>	✓
CA ControlMinder supports the recording of session which is realized by the integrated software from the company <i>ObserveIT</i> ¹ . It allows to record SSH, Telnet and Console sessions on UNIX-based systems as well as Windows sessions, which meet the compliance requirements of PCI, HIPAA, ISO 27001 and SOX.	

5.2 Netwrix Privileged Account Manager

Founded in 2006, Netwrix offers solutions for change auditing in Microsoft Windows environments. One of their products is the Netwrix Privileged Account Manager, which is available as a closed-source freeware. It features the centralized management of generic identities (e.g., “enable” on Cisco or “root” on UNIX) as well as local and domain accounts. Additionally, it provides different roles and the auditing of all managed accounts. Moreover, it can be integrated with an Active Directory and automatically update the passwords of local Windows and domain accounts on a regular basis or after each usage.

5.2.1 System Architecture

The Management Server component of the product can be installed on any Microsoft operating system since Windows XP. It requires Microsoft SQL Server as a back-end and the

¹<http://www.observeit.com>

installation of Microsoft Internet Information Services (IIS), because the web-based user-interface of the Privileged Account Manager is based on ASP.NET [16].

On the client-side it requires a web browser and the Microsoft Silverlight plugin, from where the web-based management interface is used. The authentication of the web interface supports the login of domain accounts which are centrally managed in the Active Directory. However, the role assignment needs to be managed in the product which means that they are stored in the Microsoft SQL database and cannot be retrieved from the directory.

The web interface is depicted in Figure 5.3 and shows the account management page of a user that has full permissions. A user that has been assigned one of the roles *System Administrator*, *Account Manager* or *Account Operator* can access a privileged account and check it out. This action locks the account for other users and creates an entry in the audit log, which can only be viewed by a user that has the role *System Administrator* or *Report Viewer*. Once a privileged account is checked out, the software enables the button to reveal the password.

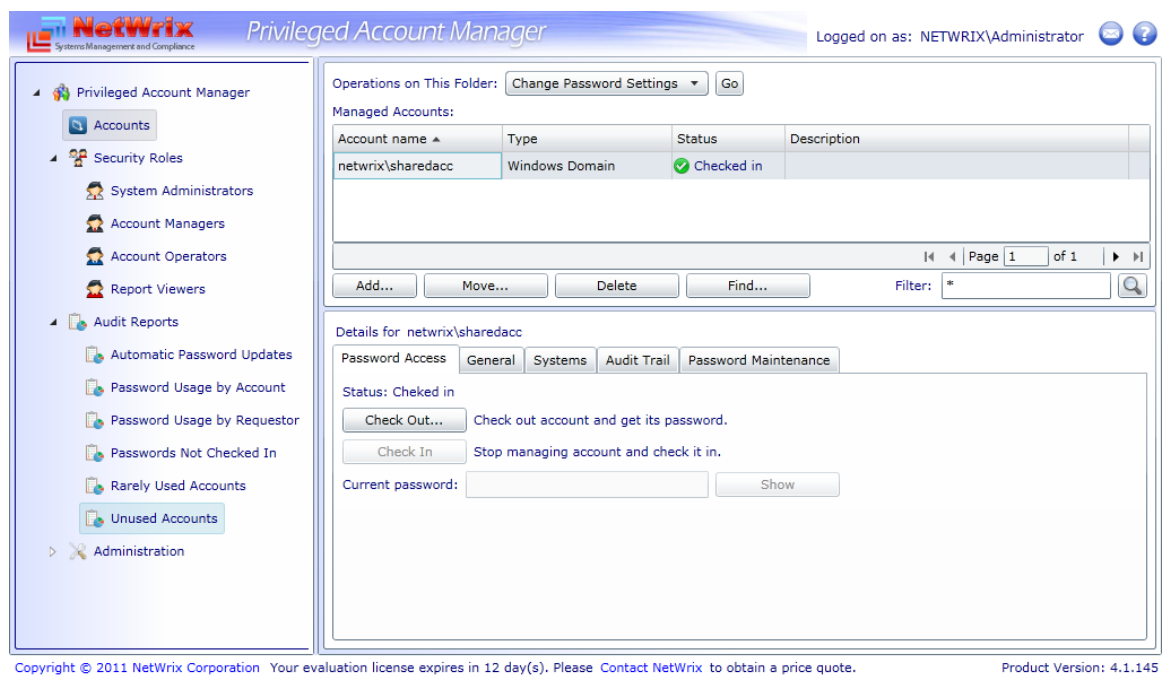


Figure 5.3: Netwrix Privileged Account Manager

The product supports three types of managed accounts:

- Generic accounts that are added by entering a username and password,
- Windows local accounts that are added by entering a username of an account that exists on the same system where the Privileged Account Manager is installed on, and
- Windows domain accounts that are added by providing a username of a domain

account which is centrally managed in an Active Directory.

The generic accounts are static and only reveal the password that has been entered. The other two account types can be configured with advanced settings. These enable that the software changes the password on the target system after the used account has been checked in. Furthermore, a task can be configured that changes the password after a given time periodically (e.g., every day at 4 a.m.). This increases the security because administrators cannot log in to a system without using the Privileged Account Manager which gives them a temporary password.

The audit reports are based on the SQL Server Reporting Services which are part of the Microsoft SQL Server and allow users to analyze special databases and generate reports. However, the audit logs and thus the information of who used what account at which time are stored in plaintext in the database. This allows every database administrator to view and manipulate the audit data.

Certainly, there are some other drawbacks:

- Although it is possible to create folders and organize passwords in groups (e.g., group them by projects or system types), it is not possible to assign folders to users or only grant access to singular entries.
- The feature that changes the password on the target system only works on Microsoft Windows operating systems. Despite the software is targeted for homogeneous networks, the credentials of other operating systems can be managed with the Generic Account type.
- There is no session audit or recording for target systems, which means that in case of an attack, only the information is given which user had checkout the account in a specific time range. So it is not possible to see which actions the user performed on the system. This requires separate products or system features in order to establish an audit trail on each target system which allows traceability.
- In contrast to CA ControlMinder, the Netwrix Privileged Account Manager cannot bypass the password and open a program like PuTTY.

5.2.2 Test Environment

The environment was realized with two virtual machines in Oracle VirtualBox on a Linux host. The first virtual machine is a Windows domain controller that provides an Active Directory for the Netwrix Privileged Account Manager which is installed in the second virtual machine. The domain controller provides the domain “netwrix.lo” which is joined by the host on which the Netwrix Manager runs on. Thus, users that are managed in the Active Directory can log on to the Windows system in the second virtual machine as well as log in to the web-based interface that Netwrix provides.

The specification of the Windows domain controller is given in Table 5.3 and the one for

the Netwrix Manager in Table 5.4.

Table 5.3: Environment Specification for Windows Domain Controller

Operating system	Windows Server 2008 R2 SP1
Memory	1 GB
Disk capacity	20 GB
Installed software	<ul style="list-style-type: none"> • .NET Framework 3.5.1 • Active Directory • DNS server

Table 5.4: Environment Specification for Netwrix Privileged Account Manager

Operating system	Windows Server 2008 R2 SP1
Memory	1 GB
Disk capacity	25 GB
Installed software	<ul style="list-style-type: none"> • .NET Framework 3.5.1 • APS.NET • IIS 6.0 • Microsoft Silverlight • Microsoft SQL Server 2008 • Netwrix Privileged Account Manager 4.1.145

5.2.3 Analysis Based on the Requirements Catalog

This section evaluates the requirements based on the catalog from Section 4.6. A ✓ specifies that the requirement is fulfilled while an X signalizes that the requirement is not fulfilled. A ◇ declares that the requirement is partly fulfilled.

Systems Integration

Support for heterogeneous networks X

The software only supports Windows operating systems and cannot manage accounts on other target systems that run other operating systems.

Centralized authentication ✓

The solution supports Active Directory authentication for domain logins to the web-based interface.

Centralized authorization ◇

The assignment of roles to identities is saved in a SQL database and needs to be performed via the web interface.

Account discovery/synchronization ◇

The software can discover local Windows and domain accounts on request or with scheduled tasks.

Access Control

Fine-grained access control on passwords X

There is no possibility to assign passwords or folders to specific identities.

Emergency access X

The software does not offer a break-glass nor a seal feature.

Separation of duties ✓

The software separates the audit capability so that unauthorized users cannot view the audit logs. This is realized through roles that need to be assigned to the users.

Support for standardized policy language X

Netwrix does not provide a fine-grained access control nor does it use an open standard for defining policies.

Policy transformation mechanism from high-level policy language to low-level MAC policies X

Netwrix does not provide a support for a high-level policy language.

Fine-grained access control on operating-system level X

The NetWrix Privileged Account Manager does not offer an integration into the operating system, it only manages accounts and the passwords.

Password Management

<i>Password security</i>	✓
Passwords are saved as ciphertext in the database.	
<i>Account check-out</i>	◇
Accounts need to be checked out before the password can be revealed. The configured password setting defines if the password is automatically changed after checkout. It does not provide a feature to restrict the usage to one user at a time.	
<i>Account check-in and one-time password</i>	X
The product does not set one-time passwords when the account is released.	
<i>Copy and paste a password</i>	✓
The web interface allows to view a password as well as copy it to the clipboard.	
<i>Platform-independent access</i>	✓
The management is realized with a web-based user interface that can be accessed from every device that runs a web browser.	
<i>Limited access time</i>	✓
The password settings can define a maximum checkout time, which ensures that an administrator cannot use an account as long as he wants. This only works for managed local and domain accounts where the password can be changed by the tool.	
<i>Hide password from the user</i>	X
There is no way to hide the password from the user.	
<i>Support for applications</i>	X
There is no support for applications or scripts.	
<i>Automatic password change</i>	✓
The automatic password change is part of the password policy. The system allows to configure an interval in which the password has to be changed by the system. However, this only works for managed local and domain accounts. All other accounts have to be managed manually.	
<i>Password policies</i>	◇
The product provides one password policy that applies to all passwords. It is not possible to define more than one policy or apply a policy to specific accounts. One can set the password length, the minimum number of upper and lower case characters as well as the number of digits and non-alphanumeric characters.	

Request/confirm access to privileged account X

It is not possible to request access to accounts that have not been assigned to a user in advance.

Password organization ✓

The web interface allows to organize the passwords in virtual folders, which can have different password settings. However, it is not possible to assign different users to folders or passwords.

Auditing and Reporting

Audit logs ✓

The log contains information about which user checked out a privileged account and who revealed or changed the password.

Security of audit trails X

The logs are saved as plaintext in the database, so that every database administrator could manipulate them.

Notifications X

The software does not generate notifications about events.

Account matrix X

Since all administrators that are authorized to access the account manager can view all entries, there is only the list of available managed accounts and role assignments.

Session recording X

There is no session recording feature available.

5.3 Soffid Identity and Access Management

Soffid is a Spanish company that was founded in 2013. Their product portfolio includes an identity and access management solution as well as enterprise single sign-on and identity federation. To date, their solutions are the only open-source products on the market. Although the source code of the products is published as open source under the GNU General Public License (GPL), they offer commercial versions with support for enterprises, too.

5.3.1 System Architecture

Like any other IAM software, the product manages users, hosts, applications, and the like. Additionally, it features the control of privileged user accounts as well as an auditing feature that logs, which user requested access to the account's password [71].

The server components of the product support the Microsoft Windows Server operating system as well as Linux, whereas the enterprise single sign-on component can be installed on all recent Windows and Linux systems. As a back-end it supports the major databases (Oracle, MySQL, and SqlServer).

In addition it is possible to extend the product with different kinds of addons. Thereby, it is possible to add a connector for Active Directory or LDAP servers, which allows to manage the users, groups, and roles in a directory instead of a relational database.

The architecture with the core components is depicted in Figure 5.4.

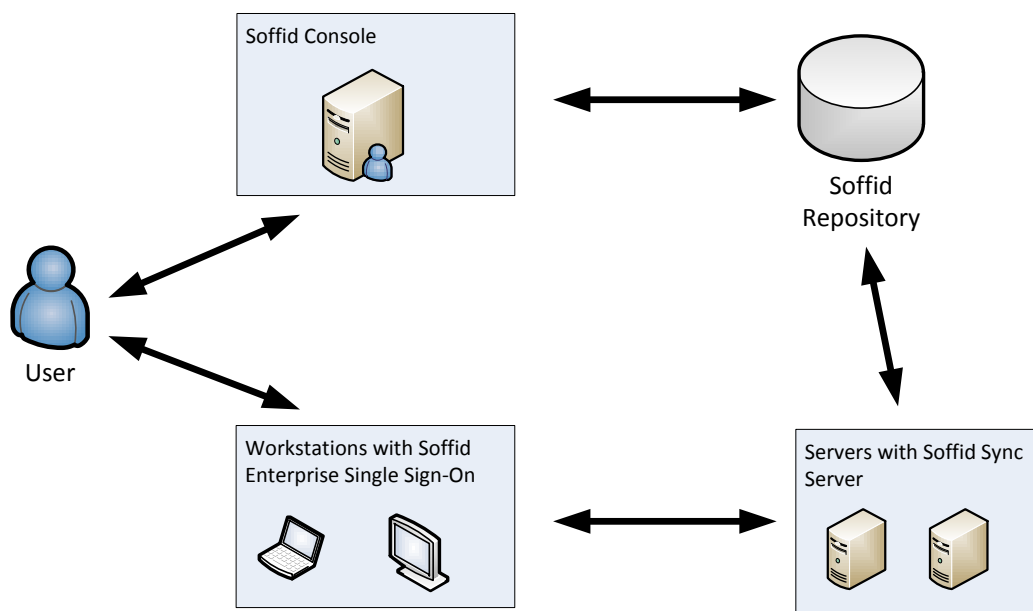


Figure 5.4: Architecture of Soffid Identity and Access Management

Besides the back-end, which Soffid calls *Repository*, there are two more components which belong to the core of the architecture: the *Soffid Console* and the *Soffid Sync Server*. Both require a recent Java Runtime Environment (JRE) since the web services depend on the JBoss Application Server. The console component represents the front-end by which the system is managed. It provides a web-based user interface for administrators and a self-service portal for all users. The self-service portal allows authorized users to list all privileged accounts they have access to. Moreover, it is possible to view the password of an account if the password policy permits that action. An example of how the self-service portal looks like when a user clicked on “query password” to display the password of a privileged

account is shown in Figure 5.5.

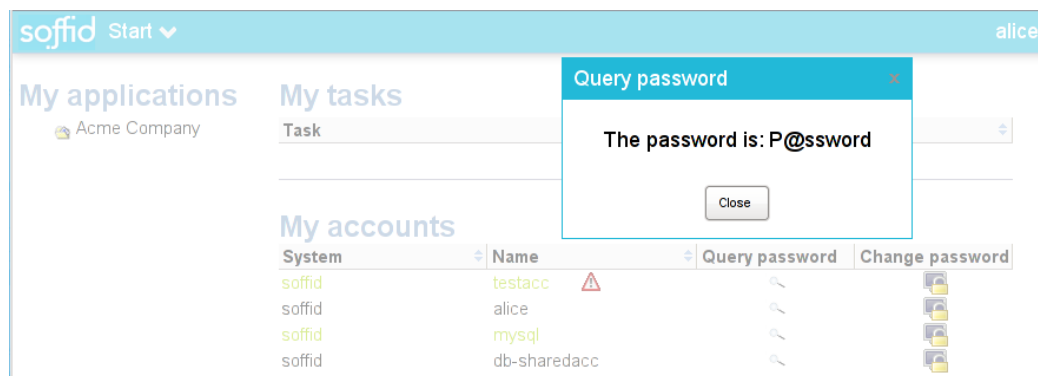


Figure 5.5: Revealing a Password in the Soffid Self-Service Portal

The Sync Server is installed on every system that needs to be integrated into the IAM infrastructure, as it connects to the repository and acts as a (web) interface for other services. It polls the repository for changes and enforces them on the system, including the management of users, groups, and passwords. The Soffid IAM solution supports two different password policy types: user-provided passwords and automatically generated passwords. In the latter case, the Sync server changes passwords of privileged accounts in the configured interval.

Each user login to the web interface is logged and the change action of a privileged password in the self-service portal is also visible in the audit log. Although the audit log can only be accessed through the web interface by authorized people, the data is saved unprotected in the repository. Thereby, every database administrator can view, delete, and manipulate the audit trails.

The default access control can be extended with an XACML addon, which allows administrators to formulate more complex authorization rules [70]. Thereby it is possible to restrict the access to privileged accounts of a certain user in a fine-grained way and bind it to environment conditions (e.g., a time interval in which the user has access).

5.3.2 Test Environment

The environment was realized with one virtual machine in Oracle VirtualBox on a Linux host. The specification is given in Table 5.5.

5.3.3 Analysis Based on the Requirements Catalog

This section evaluates the requirements based on the catalog from Section 4.6. A ✓ specifies that the requirement is fulfilled while an X signalizes that the requirement is not fulfilled. A ◇ declares that the requirement is partly fulfilled.

Table 5.5: Environment Specification for Soffid Identity and Access Management

Operating system	Ubuntu 12.04.4 LTS
Memory	1 GB
Disk capacity	8 GB
Installed software	<ul style="list-style-type: none">• MySQL Server 5.5.35• OpenJDK 7• Soffid IAM console 1.3• Soffid Sync server 1.3

Systems Integration

Support for heterogeneous networks

✓

The software supports Windows as well as UNIX-like systems and can manage accounts on those systems.

Centralized authentication

✓

The software can be integrated with Active Directory and LDAP. A single sign-on feature can be additionally installed but is only supported on Windows. Moreover, Soffid can be extended with addons and provides an integration with additional identity providers (e.g., Shibboleth).

Centralized authorization

◇

The group assignments and authorization are saved in the database. However, with the XACML addon it is possible to have a centralized generic authorization mechanism that can be used in the web interface.

Account discovery/synchronization

X

The software does not provide an account discovery feature.

Access Control

Fine-grained access control on passwords

✓

Privileged accounts can be assigned to a person individually.

Emergency access

X

The software does not provide a break-glass nor a seal feature.

Separation of duties ✓

The software separates the audit capability so that unauthorized users cannot view the audit logs.

Support for standardized policy language ✓

Soffid IAM can be extended with an XACML addon that implements the capability of defining fine-grained access control policies based on an open standard.

Policy transformation mechanism from high-level policy language to low-level MAC policies X

The solution does not offer a transformation mechanism for the XACML policies.

Fine-grained access control on operating-system level X

Soffid does not provide an operating-system layer that enforces access rights.

Password Management

Password security ✓

The passwords are saved as ciphertext in the database.

Account check-out X

An administrator can view all passwords of the assigned accounts without having to check them out.

Account check-in and one-time password X

There is no check-out and check-in feature and also no one-time password mechanism. If the user wants to change the password of a privileged account, he can do so if the password policy allows it.

Copy and paste a password ✓

The web interface allows an authorized user to view a password as well as copy it to the clipboard.

Platform-independent access ✓

The management is realized with a web-based user interface that can be accessed from every device that runs a web browser. With an addon it is also possible to implement a web-based single sign-on mechanism.

Limited access time X

Soffid cannot limit the access time of an account, which is due to the fact that there is no check-out and check-in feature.

5 Analysis of Established Solutions

Hide password from the user X

There is no way to hide the password from the user.

Support for applications X

Soffid does not offer a support for applications.

Automatic password change ✓

The automatic password renewal is part of the password policy. The system allows administrators to configure a password expiration interval which then forces Soffid to automatically change the password.

Password policies ✓

The product supports the management of different password policies that can be applied individually to the privileged accounts. A policy can prescribe the minimum and maximum password length, the amount of upper and lower case letters, numbers, and symbols. It also supports a custom regular expression and prohibited characters.

Request/confirm access to privileged account X

There is no feature that allows users to request an access to accounts that have not been assigned to the administrator.

Password organization X

The web interface displays all accounts that a user is allowed to use in a list. The list cannot be filtered or grouped.

Auditing and Reporting

Audit logs X

Soffid provides audit logs for actions that are performed in the identity and access portal but it does not log when an administrator reveals the password of a privileged account or changes it.

Security of audit trails ◇

The logs are saved in cleartext and could be modified by any database administrator. Since Soffid does not create audit logs that are related to the privileged account usage, it is not a security issue.

Notifications X

The software does not send any notifications.

Account matrix ◇

The web interface can list all groups that are assigned to a specific privileged account but it cannot show the users. So one needs to list all users that belong to the group in order to find the users that have access to the account.

Session recording

X

Soffid does not provide a session recording feature.

5.4 Summary of Software Analysis

The summary in Table 5.6 shows that CA ControlMinder fully supports 22 of the 27 requirements that have been developed in Section 4.6, while Netwrix Privileged Account Manager fulfills nine and Soffid Identity and Access Management ten of the requirements.

CA ControlMinder is a complex software solution that is suited for organizations that require a password management solution that is integrated into the infrastructure and thereby provide a fine-grained access control on the operating-system level of the servers. From all requirements that are labeled as *MUST*, the product does not satisfy the “security of audit trails”. This disadvantage can be compensated by ensuring that only authorized administrators have access to the database that hosts the logs.

The Netwrix Privileged Account Manager fulfills three out of six “must have” requirements. This applies to the “fine-grained access control on passwords”, the “emergency access”, and “security of audit trails”. The absence of an emergency access is compensated with the missing control of who is authorized to access which password. Since all administrators have access to all passwords in the manager, it is always guaranteed that all passwords are accessible. However, the missing access control is a downside that makes the software only suitable for organizations that want to manage the privileged user passwords in a central store where all administrators have access to.

Although Soffid Identity and Access Management first and foremost is an open source IAM product, it provides a privileged user password management component that fulfills three out of six “must have” requirements. The missing emergency access and audit logs for revealed passwords are two downsides that could be removed by an organization by implementing it in the open-source code.

Both of the last mentioned products do not provide a fine-grained access control in the file system or operating system. However, the expandability of the open-source software with the standardized policy language XACML provides a basic approach for a transformation of high-level access policies to low-level policies. This, for example, can be utilized to implement operating system specific agents that convert the policies to access control lists of the file system or to policies of SELinux so that privileged user accounts have restricted access rights.

Table 5.6: Summary of the Software Analysis Based on the Requirements Catalog

Requirement	CA	Netwrix	Soffid
Systems Integration			
Support for heterogeneous networks	✓	X	✓
Centralized authentication	✓	✓	✓
Centralized authorization	◇	◇	◇
Account discovery/synchronization	✓	◇	X
Access Control			
Fine-grained access control on passwords	✓	X	✓
Emergency access	✓	X	X
Separation of duties	✓	✓	✓
Support for standardized policy language	X	X	✓
Policy transformation mechanism from high-level policy language to low-level MAC policies	X	X	X
Fine-grained access control on operating-system level	✓	X	X
Password Management			
Password security	✓	✓	✓
Account check-out	✓	◇	X
Account check-in and one-time password	✓	X	X
Copy and paste a password	✓	✓	✓
Platform-independent access	✓	✓	✓
Limited access time	✓	✓	X
Hide password from the user	X	X	X
Support for applications	✓	X	X
Automatic password change	✓	✓	✓
Password policies	✓	◇	✓
Request/confirm access to privileged account	✓	X	X
Password organization	✓	✓	X
Auditing and Reporting			
Audit logs	✓	✓	X
Security of audit trails	X	X	◇
Notifications	✓	X	X
Account matrix	✓	X	◇
Session recording	✓	X	X

6 Architecture

The fundamental components for a secure architecture are authentication, authorization, accounting and auditing. The first three are commonly referred to as AAA and can be seen as a framework for providing the necessary information in order to control and audit the access to resources as well as enforcing policies [60]. The generic architecture has been proposed as “AAA Authorization Framework” by the Network Working Group [27].

An approach to solve the shared account problem is to deny the (direct) use of such identities. Once every user has its own account on a system that he is allowed to work on, the accounting is assured, since every action is assigned to a unique identity that is not shared between several administrators. To implement this, there has to be an architecture that allows a fine-grained access control which is centralized and easy to handle. Thereby it is possible to assign each identity the appropriate permissions that are necessary to fulfill the work assignments.

However, this approach only works for systems that can be integrated into the existing identity and access management infrastructure because the users have to be maintained on the systems and the system itself has to be adjusted to make use of the security infrastructure. Certainly, this is not the case for some network devices, like routers or switches. These devices often only support one privileged account which has to be shared. Moreover, these devices cannot be integrated into the infrastructure if they do not support the required protocols out of the box. For them, the idea of having a unique identity per user will not work.

Moreover, in many infrastructures that use virtualization and environments in different locations (e.g., customer projects), the number of individual accounts that would need to be maintained usually is not manageable. However, with an attribute-based access control, the complexity can be reduced and can serve as an approach for individual accounts that are assigned the necessary privileges. In situations where this is not possible, privileged shared accounts can only be reduced and need to be monitored in order to detect anomalies.

This chapter will propose a way to establish a centralized access control by using a standardized policy language like XACML and an approach to solve the problem of shared accounts on devices that cannot be integrated into the access control infrastructure directly.

6.1 Components

In order to build an architecture that solves the problem, there need to be several components that are responsible for different tasks in the authorization process. The overall architecture, which will be discussed in more detail in this section, is depicted in Figure 6.1. The idea for this architecture is based on the policy framework architecture of the Internet Engineering Task Force (IETF) [24]. The upper half of the figure shows the process of defining policies (at the *Policy Administration Point (PAP)*) and deploying them to the components that host them. The lower half shows the enforcement process that is triggered by a subject which requests an access to a protected resource. Thus, the component which enforces the access rights (the *Policy Enforcement Point (PEP)*) needs to retrieve the policy decision from a dedicated server (the *Policy Decision Point (PDP)*).

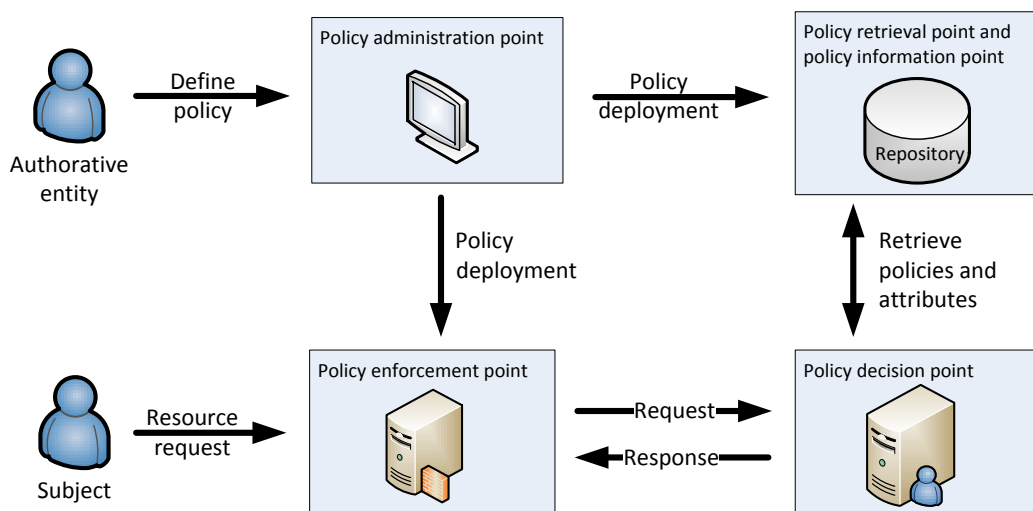


Figure 6.1: IETF Policy Framework Architecture

The reason for choosing such architecture is based on the following key requirements, which have been outlined in Section 4.6:

- Centralized authorization,
- Fine-grained access control on operating-system level,
- Fine-grained access control on passwords, and
- Support for standardized policy language.

The next subsections will describe the necessary components in more detail.

6.1.1 Policy Administration Point

The policy administration point (PAP) is an abstract component inside the network. It represents the initial place where authorization policies are created and maintained by an authoritative entity (e.g., a manager or system administrator). Without that starting point, the other components, including the PDP and PEP, could not protect the resources. Since policies are defined in XML, the PAP can be any piece of software that provides an interface to build the policies [52].

These policies are based on business-level requirements and prescribe the actions an identity is allowed to perform as part of their daily work. Before the policies can be created, the business process needs to be analyzed and modeled in a way, that allows to transform the business needs into technology-level counterparts. One way to achieve that transformation has been described in [80] by Wolter et al..

The management software should store the policies in a repository, like in a database or the file system. In the XACML terminology, the policy repository is called policy retrieval point (PRP) [26]. Furthermore, the software should be able to distribute the policies to one or more known policy decision point(s). The process depends on the capability of the target device and it might be necessary to transform the policies into a specific format [78].

A clever policy management tool might support the discovery of resources by analyzing the network topology and collecting the devices and actors inside the network (e.g., users and applications) [78].

In its most simplistic form, the authoritative entity could use a text editor to maintain the policies in an XML file. The result would then be transferred to the repository and the target devices that enforce the policies.

6.1.2 Policy Decision Point

The policy decision point (PDP) has already been introduced in Section 2.1 as part of the eXtensible Access Control Markup Language (XACML). It uses the policies which have been created at the policy administration point to answer the incoming authorization requests.

The PDP might be implemented at a network node alongside the policy enforcement point or located on a central remote server. While evaluating the appropriate policy decision, the PDP may use additional protocols (e.g., LDAP or SNMP) to retrieve additional information from different back-ends or achieve functionality such as user authentication or accounting [25]. Moreover, it may contact the policy retrieval point (PRP) in order to retrieve the required policies as well as the policy information point (PIP) to obtain additional attributes.

The PDP fulfills the requirement for a centralized authorization because it represents a dedicated and central component that provides authorization decisions to other decentralized systems.

6.1.3 Policy Enforcement Point

Like the PDP, the policy enforcement point (PEP) was already mentioned in Section 2.1. This component handles the user's resource request and generates an authorization request based on the user's action. The authorization request is then sent to the PDP which either returns a permit or deny response. Based on that, the PEP will enforce the authorization decision by granting or denying the user's resource request.

In order to provide the adequate information to the PDP, so that it can evaluate the applicable policy, the PEP often has to be specific to the application. That is why the PEP is usually located at a network node [25]. One example are accesses to the resources of a database, that consists of tables, records, and attributes. In order to model a fine-grained access control, the PEP needs to transmit all the necessary information to the PDP, for example, which user gains access to which column of a record. These policy attributes are very specific to that kind of service and they cannot be reused for another type of service that needs to transmit different attributes as part of the authorization request.

The PEP is the entity that protects different kinds of resources, which are summarized in the following subsections.

File System Protection

The requirement for a fine-grained access control in the operating system includes the access to the data that resides in the file system. There, not only sensitive business or customer-related data is saved but also configuration files that control the behavior of the operating system, services, and applications. Thus, the access rights of privileged users need to be controlled so that negligent administrators and insider attacks can be contained.

Almost every file system and operating system supports its own access control to provide a security mechanism for the data. The traditional file permissions in UNIX-like operating systems only allow a simplistic security concept, based on the three classes *user*, *group* and *others*. These classes might have attached none or a combination of these permissions:

- *read*: grants the permission to read a file or – if it is a directory – read the names of the files inside the directory
- *write*: grants the permission to modify a file or – if it is a directory – modify the entries inside the directory (create/edit/rename a file)
- *execute*: grants the permission to execute a file or – if it is a directory – access a file's content and meta information

Since the three classes are very limiting when it comes to a fine-grained access control that needs to support many users and groups that require different permissions on an object, the UNIX-like operating systems support access control lists (ACLs). These ACLs provide a more flexible way to define which objects a subject is allowed to access. Since the traditional permissions and the ACLs are discretionary access control mechanisms, the

task of setting up the access rights for a lot of users is a very time-consuming task.

On the other hand, a mandatory access control mechanism is more flexible and secure as it not only uses the subject and object information, but also takes additionally rules into account.

Two possible approaches for solving the access control problem on a Linux operating system are *Security-Enhanced Linux* (SELinux) and *AppArmor*. Both are Linux kernel security modules that allow administrators to set up security policies for users and programs. It is also possible to construct a mapping between files and security contexts to establish a protection for the file system.

Like stated in Section 6.1.1, the policy manager software that represents the policy administration point might transform the high-level policies to low-level SELinux or AppArmor policies, if the security module itself does not have the ability to contact a policy server. Thus, the PAP needs to transform and deploy the policy rules to the PEP.

The concrete implementation of a transformation mechanism is beyond this thesis and thus mentioned as a topic for a future work in Section 9.2.2.

Protecting Applications

In order to use authorization policies inside an application, the software either has to implement an interface that utilizes the policies directly from the policy decision point, or the software needs to support a high-level application programming interface that acts as a broker between the policy server and the application.

One example is the *pluggable authentication module* (PAM) mechanism in UNIX-like operating systems. This can be used by applications to implement an independent and high-level authentication scheme.

Like stated in Section 2.2, the `pam_xacml` module adds support for XACML authorization to every application that is PAM-enabled. If the application does not utilize PAM nor does it offer an interface for a direct use of the policy server, the approach will not work. Alternatively, if the application provides an access control that can be managed programmatically, the policy administration point might transform the high-level policies into the application's low-level format. The approach is equal to the one mentioned in the previous subsection.

Protecting Data Stores

A special kind of applications are databases, which provide a collection of data that is organized in some kind of schema. In relational database management systems like MySQL, there are tables which consist of columns and rows which can be managed by using SQL.

The access to tables, rows and columns can be protected by using the built-in access control

lists. However, the high-level approach of modeling attribute-based access control policies with XACML provides a more expressive and manageable way of controlling the access rights. In [38] Jahid et al. showed an approach how high-level XACML policies can be compiled into low-level access control lists which are supported by the database system. Especially on UNIX-like operating systems, the control of who is authorized to log in to the database can be integrated with XACML by using the `pam_xacml` module, since major database systems provide PAM authentication modules.

Besides managing the access rights, the sensitive data might additionally be encrypted before it is stored in the database. This not only protects the data from unauthorized access by the database administrator which has elevated privileges to access all the data. It also increases the security in the network if only the encrypted data is transferred.

Sensitive data like passwords, PINs, product or license keys can then only be decrypted by designated applications that reside on trusted systems. These applications are in charge of revealing the decrypted information only to authorized users. This can be achieved by using asymmetric cryptography, like outlined in more detail in Section 6.2.1.

Protecting a database like that comes in handy when privileged user passwords are saved, so that password manager tools can access them. The database management system could then enforce the policy decision which it received via the PAM module from the policy decision point. The policies contain which administrator is authorized to access which rows and columns.

Protecting Network Devices

The usage of `pam_xacml` is easy for applications that support PAM, but in almost every network there are also devices that only support one privileged account. This might be a root account on a router or switch, which can only be accessed over a few protocols (e.g., SSH or Telnet).

Since those devices normally do not support the use of a PDP and cannot be extended to use a centralized security infrastructure, one has to find a way to solve the problem of users that share the privileged account. One approach is to deny the direct access to the device and force all users to connect through a *jump server*. This jump server can be compared to a *bastion host*, which is located on the public side of the demilitarized zone (DMZ) and thus is unprotected by a firewall. In order to connect to the local area network from the Internet, one has to connect through the bastion host [19].

The architecture is depicted in Figure 6.2 and shows the interaction with the PDP. The administrator who wants to connect from the client device to the target system (e.g., a router) cannot establish a direct connection, because the authentication mechanism might deny it or the user is not allowed to know the privileged user password that is required to log in to the device. Instead, he has to connect to the jump server and authenticate with the personal account and request a forwarding connection that is established by the middle box on behalf of the user. By this, the jumper server can check if the subject is authorized

to connect to the target system.

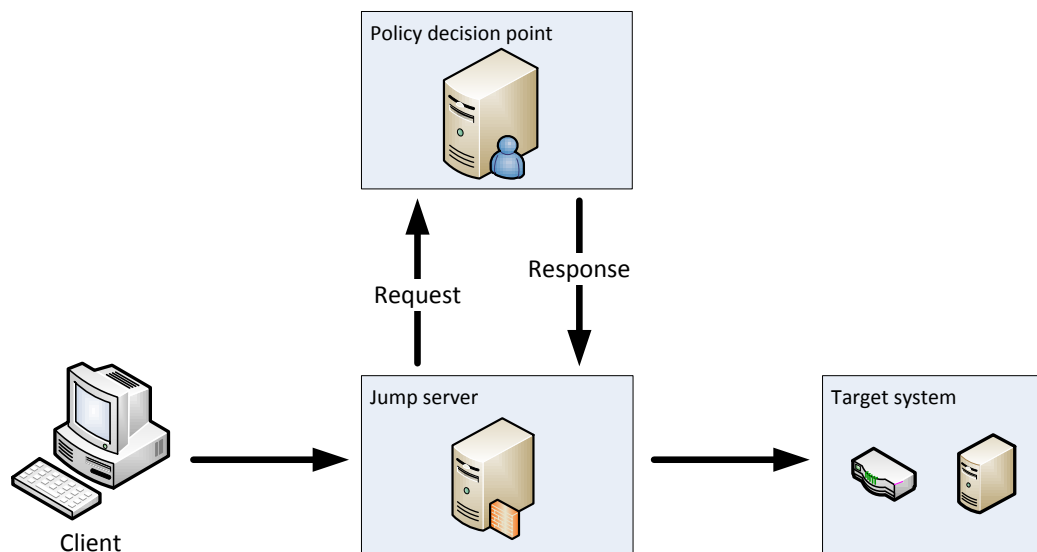


Figure 6.2: Jump Server

This authorization is realized by contacting the PDP and enforcing the authorization decision that it returns. If the PDP returns a permit, the jump server can open a connection on behalf of the subject. Since the jump server establishes the connection, the user will not see any credentials (e.g., the username/password combination of the target system's privileged account). Making the shared account transparent and hiding the sensitive credentials is one important security aspect and might be a requirement of the organization, too.

For auditing purpose, the jump server could monitor the commands that have been entered or record the whole session. It is also possible to only allow one user at a time to be connected to the target system, so that it is always clear which actions have been performed by which user. This ensures accountability since no one uses the privileged account directly but has to authenticate at the jump server with the personal account.

Certainly, this approach has some drawbacks. One is the security problem that a user might be authorized to connect to the target system and then establish a reverse tunnel from the target system back to the client. This would produce a second connection that does not use the jump server and the user would be able to circumvent the auditing mechanism and other security features that the jump server would have provided.

Another aspect is that the jump server should support different protocols and many target systems. For smaller networks, an approach like port address translation (PAT) might work. Here, one would assign separate ports to each supported protocol on a server and other ports for the same protocols on another server. However, the number of available ports is limited and thus larger networks would require several jump servers, which not only forces a user to remember the port but also the responsible jump server.

As an alternative to PAT, one could think of a transparent approach that utilizes the firewall, which filters the packages and changes the destination IP address if necessary, so that the package is sent to the jump server. Additionally, the original destination IP address needs to be placed somewhere, so that the jump server can extract it and use it to establish the connection to the target system.

As mentioned before, the jump server needs to support several protocols, for example, SSH, Telnet, SNMP, and so on. This can be achieved on a per-service basis or a software that provides an interface for each supported protocol and acts as an intermediary. In the first case each service needs to support mechanisms for authentication and authorization that can be integrated with an PDP or can utilize established APIs like PAM, so that a fine-grained access control can be applied for each service individually. On the other hand, the services need to support a forwarding mechanism to the target system and security features like an audit trail and session recording to guarantee accountability. These requirements could be implemented directly into the service, for example, into the Telnet server. Whereas in the case of an intermediary software it is necessary that it behaves like a man-in-the-middle and arbitrates between the client and the target system. For that purpose, the software needs to understand each protocol, in order to handle the data stream, provide authentication, and audit trails.

However, the man-in-the-middle approach of a jump server might be against the business law and regulations of the organization. Certainly, the aspect of hiding the passwords from the user cannot be realized without a man-in-the-middle box that takes care of the authentication on the target system. Consequently, the persons in charge need to decide whether they can trust their administrators and give them access to the passwords so they can log in to the target system directly. On the other hand they might decide that they cannot trust their employees and go with the man-in-the-middle system.

6.1.4 Password Manager

Sometimes, the approach of having a jump server in between the client and a target system is not feasible. This could be the case when the target system only supports an interactive authentication, for example, partner websites that require a login in order to access a protected area. Another reason are credentials that are associated with a project which need to be made known to different team members or personal identification numbers (PINs) that are shared between several administrators. In this case, there are shared accounts to which authorized users need full access to. This problem can be solved with password managers which not only store credentials but also product or license keys and other access-related sensitive data that needs to be protected. Comprehensive software permits the integration into the organization's infrastructure in order to use the existing identities and access policies. This allows users to authenticate against a directory (e.g., LDAP or Active Directory) with their centrally managed identity.

However, to provide a fine-grained access control, the application needs to implement an appropriate interface. This could be the support of XACML, which then fits into the architecture mentioned above. Certainly, the password manager can also be web-based

and integrated into an existing web access management solution, which would handle the authentication (e.g., with a single sign-on) and authorization. This could also be realized with XACML by implementing the request-response protocol or by using one of the libraries which exist for various programming languages (e.g., Java, PHP, Ruby, and Python).

In order to fulfill the requirement for an emergency access and the demand for a separation of duties, the password manager can utilize the XACML-integrated mechanisms. This also applies to the delegation of privileges which makes it possible to give someone the right to access a privileged user password for a given time period (e.g., if Alice is on vacation she can assign all or some of her privileges to Bob if the policies allow it). This guarantees that in emergency situations other administrators have access to the passwords.

The requirement for accountability can be realized by logging who accessed and revealed the password of a privileged account. If everyone authenticates at the password manager with the personal account, the necessary information is available. Moreover, by implementing a “check-out” and “check-in” mechanism it is possible to ensure that only one administrator has access to a privileged user password. Thereby, all actions between a check-out and check-in can be accounted to a specific identity.

The requirement for automatically changing a password on the target system requires an interface so that the password can be changed programmatically. On UNIX-like operating systems this could be realized by using an SSH connection that authenticates with the privileged user’s credentials (e.g., “root” and the current password) and executing the `passwd` command. This could be implemented by a task that resides on the same machine as the password manager and is authorized to query the password database for accounts including the corresponding passwords that need to be changed. After that, the automated task would generate a new password based on some predefined password policy and change it on the target system as well as in the password database.

6.1.5 Audit Logging System

To ensure non-repudiation, each action should be logged with the information of who performed the action on what resource at which point in time. For ease of maintainability and security reasons, there should be a centralized component in the network that is dedicated for logging.

There, the log entries of all relevant network components, including servers, applications, firewalls, routers and switches, and databases should be accumulated. Moreover, the authorization decisions should be logged, too. For later forensics, not only the response of the PDP is important but also the request and obligation handling needs to be protocolled, so that a full audit trail is given and a security issue can be detected.

Therefore, the PDP and PEP need to implement a logging facility. In all major operating systems, there is a logging mechanism to which the messages can be sent. This allows to configure the built-in logging mechanism to forward all messages to a central server,

without having to configure each application individually.

On UNIX-like operating systems, the de-facto standard for logging system events is the *Syslog Protocol* [35]. With the PAM module *pam_xacml*, which was introduced in Section 2.2, it is possible to log the request as well as the authorization decision to the system log of the PEP. If the logging daemon is configured to use a centralized logging server, the data will be stored on the dedicated system.

The logging requirement also applies to the jump server, which not only should log the authentication and authorization process but also the information about when it retrieved a particular entry from the password database.

Besides logging all read accesses to the privileged user passwords in the database, all modifications and deletions should be audited as well, to ensure a consistent audit trail throughout the lifetime of a resource. Thus, it should be possible for an auditor to check who had access to a password or who initiated a password change.

Furthermore, all performed actions (e.g., privilege elevation, typed-in commands, etc.) should be recorded to the logging system, too.

6.2 Security Considerations

The security of privileged user passwords is an important aspect but also the security of the systems on which administrators perform actions with privileged accounts needs to be considered. This can be achieved by integrating an *intrusion detection system* (IDS) that monitors the network and systems for malicious activities and reporting the incidents to dedicated persons [62]. One example for such an IDS is the open-source software *Samhain*, which can detect the tampering of databases and configuration files as well as the manipulation of log files.

6.2.1 Encryption of Credentials

One approach for protecting sensitive information is to use cryptography. An established variant is the public-key cryptography (PKC), which uses two separate keys, one of which is placed in a public file and another one that has to be kept secret (the private key).

In order to save sensitive data like shared passwords of privileged accounts, license keys and the like securely, one might want to use cryptography, too. This requires the establishment of a public-key infrastructure (PKI) in the network.

In the case of the aforementioned password manager, the application would use the public key to encrypt the password and transfer the ciphertext to the database, where it is saved securely. At the time when the jump server requires a particular password to establish the connection to the target system on behalf of the user, it would query the database and retrieve the credentials including the encrypted password. The ciphertext then is decrypted

on the jump server by using the private key and the plaintext password can then be used for the authentication process.

Disadvantages of the traditional public-key cryptography are that it involves a complex infrastructure and that – in the case of a password database – all assets are encrypted with the same public key. This means that an attacker gains access to all passwords if he can steal the private key. In order to prevent the theft and reduce the security issue, the servers that require the private key to decrypt the ciphertext need to be trusted systems.

Another solution to the problem is to not encrypt all passwords with the same key. Instead, only those passwords are encrypted with the same key which can be access by the same identity. All the other passwords that the identity must not have access to are encrypted with another key. However, this approach requires to encrypt the same password with different keys if different users are authorized to know the password. So if Alice and Bob are allowed to use the root password and both have different key pairs, the database needs to contain the ciphertext two times: once it was encrypted with the public key of Alice and the second time it was encrypted with Bob's key.

Since this approach is hard to maintain, the demonstrator in Chapter 7 will use the first idea which encrypts all data with the same key. However, there are more complex solutions to this problem, like the attribute-based encryption which is discussed as a future work in Section 9.2.1.

6.2.2 Secure Audit Logs

All major operating systems provide some sort of system logs which are important for the security of a system. The logged data should provide helpful information for the analysis after a break-in and give hints about failed attack attempts. Usually, the general activity of users and programs is logged, as well as the system resource usage and program crashes or errors.

Audit trails are used for *accountability, intrusion detection, problem identification, and reconstruction of events* [72]. Because of that, an attacker will try to cover the tracks and erase all actions that have been recorded in logs and might potentially whistle-blow him. Hence, the audit logs need to be made secure in order to prevent an attacker from accessing and manipulating them.

This not only applies to external attackers but also to insiders. Thus, the organization should have rules for the separation of duties, which regulate that employees who administrate the access control mechanisms are not the same as those who administrate and monitor the audit trail [72].

For investigations of security violations, not only the actions of an attacker are relevant. Additionally, the circumstances that allowed the attack are of importance, too. This includes the authorization decisions that were made by other systems (e.g., the PDP), which also have to be saved in a secure audit log.

The log management infrastructure should be implemented by using centralized log servers and log data storage [42]. This reduces the administration time and increases the security as audit logs are not saved decentralized on each device which might get compromised by an attacker.

The first step to secure logging is using a separate logging server instead of having the logs on the same device on which the user operates on. In the case of an attack, the logs on the compromised system would not be secure anymore. However, if the logs reside on a special system that does not offer much possibilities for a break-in (e.g., services without security holes), the logs are more secure from tampering.

Although the central logging server is a security improvement, it is still necessary to establish more properties which meet the important security goals, including the *integrity*, *confidentiality*, and *availability* of log files [42].

The *integrity* requirement of log files can be fulfilled with the forward-secure stream integrity approach. Bellare and Yee were the first who combined message authentication codes (MACs) with log entries. Their approach generates a MAC for every single log entry by using a different MAC key for each entry. The key K_i for log entry i is obtained from key K_{i-1} of the previous epoch. After the MAC for the new log entry has been generated, the key K_{i-1} is deleted, so that an attacker can get the key K_i but does not get any knowledge about K_j with $j < i$. However, the initial key K_0 needs to be kept secure and should only be known to auditors since this key can verify all log entries [3].

This approach allows auditors to detect the tampering (e.g., the modification or deletion of log entries) but does not securely store the entries, so that only authorized people have access to the log entries. The *confidentiality* of log files can be guaranteed by restricting the access to a small group of authorized auditors. This could be realized by formulating fine-grained XACML policies that control the access on a per-file basis or, for example, if a database is used, on a per-entry level. Moreover, the confidentiality and security can be increased by encrypting the log entries and only allowing auditors to decrypt the ciphertext.

In [65] Schneier and Kelsey presented an approach that uses per-record encryption keys and permission mask that allow a selective disclosure of log entries. Furthermore, their approach ensures that users that possess a decryption key cannot make undetectable changes.

The *availability* of log files can be ensured by keeping multiple copies and backups of the data in different locations so that a damage or deletion of one copy does not cause a loss of data [42]. Moreover, the log infrastructure itself, especially the server which hosts the log service, should be fail-safe, for example, by having standby facilities that take over in the case of a system or network failure.

6.3 Analysis Based on the Requirements Catalog

This section evaluates the requirements based on the catalog from Section 4.6. A ✓ specifies that the requirement is fulfilled while an X signals that the requirement is not fulfilled. A ◇ declares that the requirement is partly fulfilled.

Systems Integration

Support for heterogeneous networks ◇

The architecture and components from Section 6.1 are suitable for heterogeneous networks. Certainly, the operating system specific additions which have been exemplified for UNIX-like systems need to be ported to other operating systems, for example, Windows. This especially applies to the enforcement of policy decisions and the PAM mechanism that is not available on Windows. An proposal on how this can be implemented in a future work is outlined in Section 9.2.3.

Centralized authentication ✓

The above mentioned architecture does not require a specific authentication infrastructure. The fine-grained access control and password manager can be based on a directory (e.g., Active Directory or LDAP) as well as the integrated user management of the operating system (e.g., */etc/passwd* on UNIX-like systems).

Centralized authorization ✓

The authorization is realized with XACML policies and a centralized policy decision point that evaluates the policy decision requests from different policy enforcement points, including servers, routers, and applications.

Account discovery/synchronization ◇

The goal of the proposed architecture in this thesis was to reduce the need for shared privileged accounts that are not maintained in a central user repository (e.g., with an identity and access management solution). Because of that, the account synchronization might only be required for the password manager that has been addressed in Section 6.1.4. However, the password manager was intended to be used for accounts that reside on systems or in networks which are not part of the organization's network. In such cases, an account discovery feature usually cannot scan the systems and extract the privileged accounts.

Access Control

Fine-grained access control on passwords ✓

The access to passwords can be controlled by defining fine-grained policies that are enforced by a password manager which then only displays accounts and reveals passwords

to authorized users.

Emergency access ✓

The architecture ensures that every administrator that is authorized to access a system can do so with the personal account. Moreover, XACML policies can define break-glass rules that allow administrators to perform actions they are usually not authorized to (see Section 2.1).

Separation of duties ✓

The fine-grained access control of XACML allows to define policies that separate actions from each other that rise potential conflicts of interest (see Section 2.1).

Support for standardized policy language ✓

XACML, the policy language which has been used for the architecture, is an open standard.

Policy transformation mechanism from high-level policy language to low-level MAC policies ◇

The transformation from XACML policies to, for example, SELinux policies is possible and can be analyzed in more depth as a future work, like addressed in Section 9.2.2.

Fine-grained access control on operating-system level ✓

The proposed approaches in Section 6.1 showed that a fine-grained access control on an operating-system level is possible. There are established solutions to protect the file system as well as PAM modules to connect services and applications. This ensures that access rights can be set on a fine-grained basis and unauthorized users (e.g., external administrators) do not have unrestricted access rights.

Password Management

Password security ✓

The passwords are saved as ciphertext in the database and only trusted systems have the private key to decrypt the passwords.

Account check-out ✓

The password manager can implement an account check-out feature which logs that an administrator started to use the privileged account. Moreover, it can ensure that only one user at a time can reveal the password.

Account check-in and one-time password ✓

The feature of changing a password automatically can be combined with an action that deallocates a privileged user password in the password manager. The tool would then perform a password change on the target system and save it in the password database.

Copy and paste a password ✓

The password manager can reveal a password that has been retrieved from the database so that a user can use it as well as copy it to the clipboard.

Platform-independent access ✓

If the password manager is implemented as a web-based application it can be accessed by every device that provides a web browser.

Limited access time ✓

In XACML a policy might have a limited access time specified as obligation. The policy enforcement point then has to ensure that once the time is exceeded, the user is disconnected. In that case it might inform a service which is part of the password manager that it must perform a check-in action on that account so that other administrators can use it.

Hide password from the user ✓

The jump server provides a way to hide the password from the user and establish a connection after authenticating and authorizing a user.

Support for applications ✓

Applications can utilize the fine-grained access control by implementing PAM. Moreover, trusted applications can query the password database and decrypt the ciphertext in order to use the credentials to perform an action. The limitation of having one public/private key pair for all passwords can be overcome with an attribute-based encryption that allows to encrypt singular passwords individually. This approach is discussed as part of the future work in Section 9.2.1.

Automatic password change ✓

The automatic password renewal functionality can be part of the password manager but requires operating-system specific interfaces that can be used to change the password (e.g., SSH on UNIX-like systems).

Password policies ✓

The password policies can be part of the password manager and ensure that user-defined passwords as well as automatically changed passwords follow the policies.

Request/confirm access to privileged account ◇

This feature can be implemented as part of the password manager or it can be realized with an external tool, for example, in an issue tracking system that allows any administrator to request additional privileges. An authorized user might then decide if he wants to grant the access rights by creating a new XACML policy for that specific user.

Password organization ✓

The password manager might offer some kind of organization in groups and folders.

Auditing and Reporting

Audit logs ✓

The architecture can use the integrated logging daemon that might be configured to protocol the action as well as the policy decision on a central server.

Security of audit trails ✓

The logs might be saved on a central server as well as encrypted and protected from tampering. On UNIX-like systems, there are logging daemons like *syslog-ng*¹ which offer a cryptographic signature feature.

Notifications ✓

Notifications can be sent as part of an obligation or at the time when an action is performed in the password manager (e.g., when a request to a privileged user password is generated).

Account matrix ✓

The authorization decision is based on the policies that define which administrator is authorized to access a specific password or perform an action as a privileged user. Thus, the policy administration point can give the information.

Session recording ◇

The above mentioned architecture does not record sessions but this can be realized by installing a software like *ObserveIT*² on the servers.

6.4 Summary

The architecture proposed in this chapter is an approach for an infrastructure that uses open standards. The policy-based authorization with the XACML standard can be utilized to establish a fine-grained access control on the operating-system level of the servers as well as in applications and web-interfaces. Thereby, privileged user passwords that are managed in a central and secure database can be accessed by administrators as well as applications. The access control on the policy enforcement points guarantees that only authorized users have access to privileged user credentials.

The approach of a jump server provides accountability and authorizes administrators that want to connect to network devices. With access control security policies like SELinux it is possible to reduce the power of privileged accounts in the file system and protect sensitive data from unauthorized access.

A centralized logging system that provides integrity and confidentiality of the audit trails

¹<http://www.balabit.com/de/network-security/syslog-ng>

²<http://www.observeit.com>

ensures that malicious actions can be detected and only authorized persons have access to the logs.

The full architecture is depicted in Figure 6.3 and shows the basic XACML infrastructure, the jump server, and password manager component. Moreover, the central audit logging system is included but for clearness the links to all the systems that send the log information have been omitted.

The summary of the architecture requirements and the comparison to the three analyzed software products of Chapter 5 is given in Table 6.1.

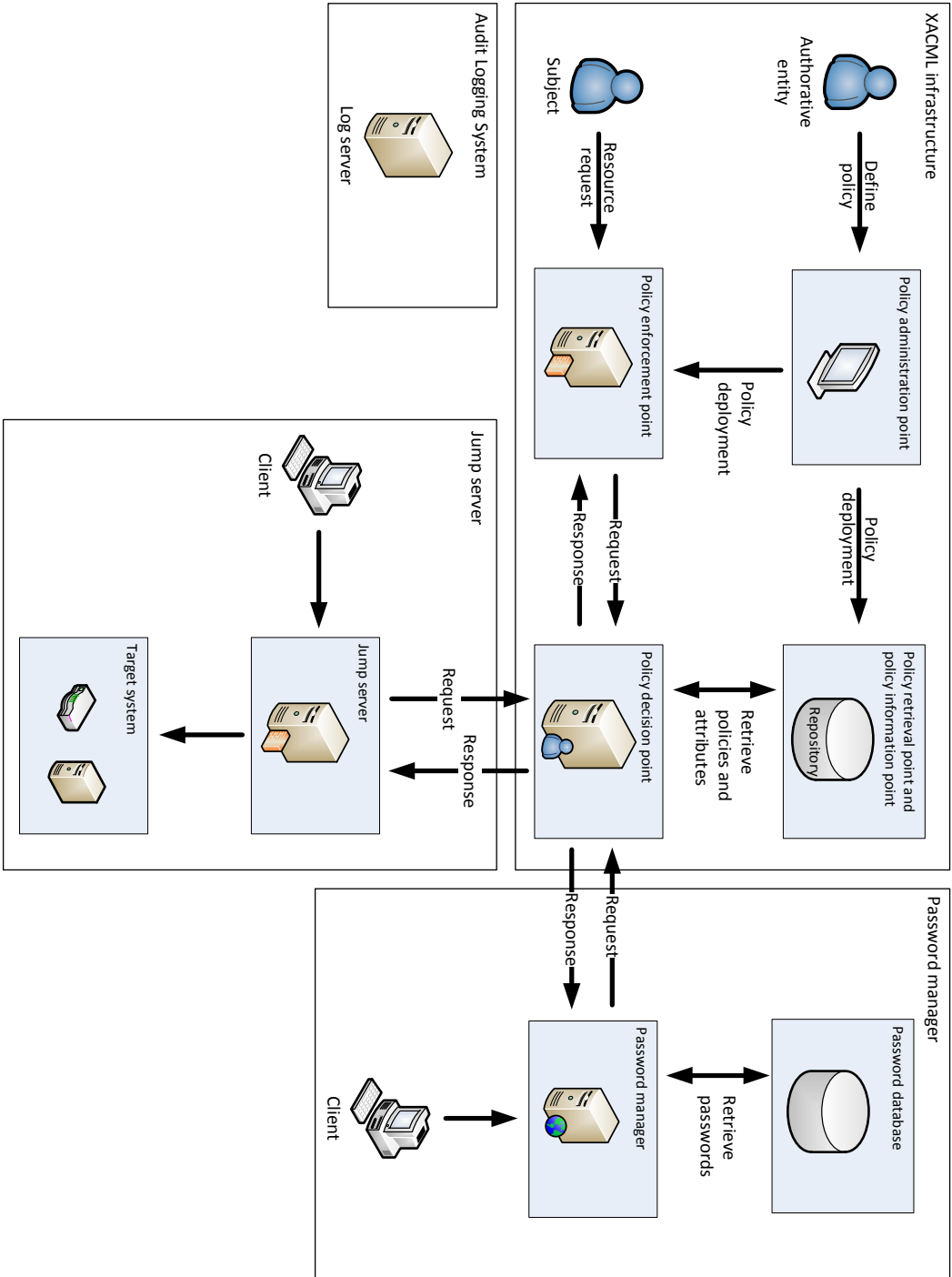


Figure 6.3: Full Architecture

Table 6.1: Summary of the Architecture Based on the Requirements Catalog

Requirement	Architecture	CA	Netwrix	Soffid
Systems Integration				
Support for heterogeneous networks	◇	✓	X	✓
Centralized authentication	✓	✓	✓	✓
Centralized authorization	✓	◇	◇	◇
Account discovery/synchronization	◇	✓	◇	X
Access Control				
Fine-grained access control on passwords	✓	✓	X	✓
Emergency access	✓	✓	X	X
Separation of duties	✓	✓	✓	✓
Support for standardized policy language	✓	X	X	✓
Policy transformation mechanism from high-level policy language to low-level MAC policies	◇	X	X	X
Fine-grained access control on operating-system level	✓	✓	X	X
Password Management				
Password security	✓	✓	✓	✓
Account check-out	✓	✓	◇	X
Account check-in and one-time password	✓	✓	X	X
Copy and paste a password	✓	✓	✓	✓
Platform-independent access	✓	✓	✓	✓
Limited access time	✓	✓	✓	X
Hide password from the user	✓	X	X	X
Support for applications	✓	✓	X	X
Automatic password change	✓	✓	✓	✓
Password policies	✓	✓	◇	✓
Request/confirm access to privileged account	◇	✓	X	X
Password organization	✓	✓	✓	X
Auditing and Reporting				
Audit logs	✓	✓	✓	X
Security of audit trails	✓	X	X	◇
Notifications	✓	✓	X	X
Account matrix	✓	✓	X	◇
Session recording	◇	✓	X	X

7 Specification and Implementation of a Demonstrator

This chapter describes a demonstrator that implements parts of the architecture that has been proposed in Chapter 6. Figure 7.1 shows the system architecture, which consists of a *client*, a *jump server* that connects to a target system, and a *policy decision point* that performs the authorization evaluation. Moreover, the architecture includes a *password database* that contains encrypted passwords which are used by the jump server to perform the authentication on the target system.

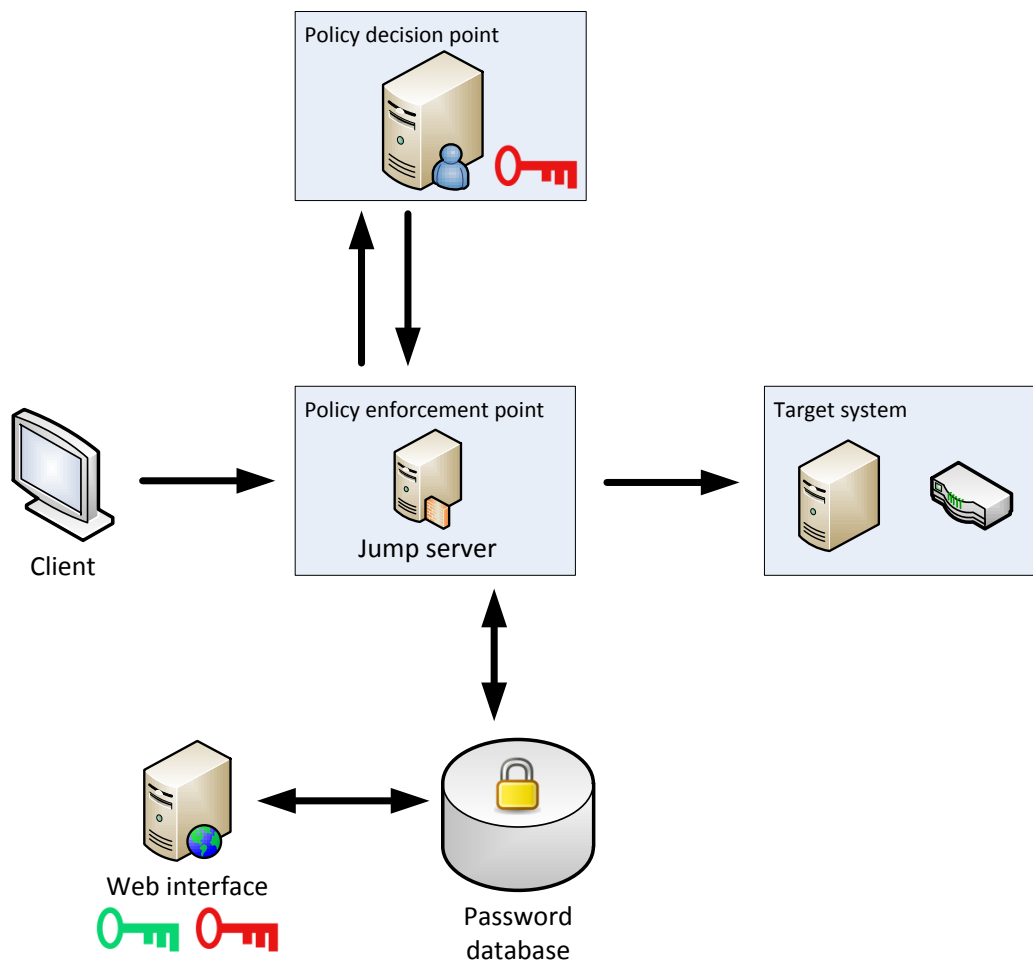


Figure 7.1: Architecture of the Demonstrator

The environment was realized with two virtual machines in Oracle VirtualBox on a Linux host. The first virtual machine represents the policy decision point, a second one provides the jump server, and a third machine contains the password database and a simplified web interface.

The specification of the first virtual machine is given in Table 7.1, the one for the jump server in Table 7.2, and the specification of the password database and web interface is summarized in Table 7.3.

Table 7.1: Environment Specification for the Policy Decision Point

Operating system	Debian 6.0.8
Memory	384 MB
Disk capacity	8 GB
Installed software	<ul style="list-style-type: none">• Apache Xerces 2.9.1• gSOAP 2.7.10• OpenJDK Runtime Environment 1.6.0 build 27• pam_xacml 0.2-alpha• SimplePDP• Sunxacml 1.2

Table 7.2: Environment Specification for the Jump Server

Operating system	Debian 6.0.8
Memory	384 MB
Disk capacity	8 GB
Installed software	<ul style="list-style-type: none">• MySQL client• sshpass 1.04

Table 7.3: Environment Specification for the Web Server

Operating system	Debian 7.4
Memory	1 GB
Disk capacity	8 GB
Installed software	<ul style="list-style-type: none">• Apache 2.2.2• MySQL Server 5.5• PHP 5.4.4

7.1 Authorization for PAM-Enabled Applications

The setup uses the *pam_xacml* module which was introduced in Section 2.2. In order to force a PAM-enabled program like *sudo* to use the PAM module, one has to add a similar line like the one in Listing 7.1 to the configuration file (e.g., */etc/pam.d/sudo*).

Listing 7.1: PAM Configuration for *sudo*

```
account required pam_xacml.so requestBuilder=INTERNAL
requestTemplate=/etc/pamxacml/sudo_template.xml
pdpRequester=SIMPLE pdpEndpoint=PDP-server:1234
```

This configuration forces PAM to call the “*pam_xacml.so*” module with some parameters. One of them is “*pdpEndpoint*”, which tells the *pam_xacml* module to which host and on which port it has to connect to reach the PDP service. The other important parameter is “*requestTemplate*”, which specifies the path to the XML template that makes up the request message.

The request message transmits the information that is necessary for the PDP to evaluate the request and find the appropriate policy. The simplest request template submits

- the *subject* that invoked the request (e.g., the username) and
- the *resource* that the subject wants to access (e.g., the name of the service or application)

The template furthermore could send information about the *environment* (e.g., the name of the host from which the request came from) or the *action* that a subject wants to perform on a resource. An example for a request message is shown in Listing 7.2. Here, the user *alice* wants to access the resource *sudo*. Thus, the *Subject* attribute contains the username and the *Resource* attribute provides the name of the command. Moreover, the *Environment* attribute contains the hostname, which is *my-hostname* in this example. The *Action* attribute is a static string that was not used but other applications could insert in-

formation like “read”, “write”, or “delete”, so that rules can be built that evaluate concrete actions.

Listing 7.2: Example Request Message for sudo

```
<Request xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="...">
  <Subject>
    <Attribute AttributeId="&subject;subject-id"
      DataType="&xml;#string">
      <AttributeValue>alice</AttributeValue>
    </Attribute>
  </Subject>
  <Resource>
    <Attribute AttributeId="&resource;resource-id"
      DataType="&xml;#string">
      <AttributeValue>sudo</AttributeValue>
    </Attribute>
  </Resource>
  <Action>
    <Attribute AttributeId="&action;action-id"
      DataType="&xml;#string">
      <AttributeValue>PamXacml Authorization</AttributeValue>
    </Attribute>
  </Action>
  <Environment>
    <Attribute AttributeId="urn:pamxacml:hostname"
      DataType="&xml;#string">
      <AttributeValue>my-hostname</AttributeValue>
    </Attribute>
  </Environment>
</Request>
```

7.2 Policy Decision Point

The authors of *pam_xacml* used the *sunxacml* implementation with a simple Java-based web service as a policy decision point. Although the *pam_xacml* module also supports other PDPs as well, it was the best to choose an equal setup, which is why the demonstrator uses the same PDP (“SimplePDP”) that was shipped with the *pam_xacml* module.

The PDP is configured to listen on a TCP port to allow other machines to connect to the centralized PDP server. Once the PDP web service receives a request, it uses the *sunxacml* library to evaluate it. Thereto, it checks the policies which have been provided as a text file that holds the XACML policies in XML syntax. Based on that, the PDP builds a response

message that includes the authorization decision. An example for a response is depicted in Listing 7.3. The `Decision` attribute contains the decision that has been evaluated, which is “Permit” in this example. Hence, the PEP is allowed to grant the access to resource “sudo” to the user. Moreover, the `Status` attribute contains a status code that informs the PEP that no errors occurred.

Listing 7.3: Example Response Message That Permits the use of sudo

```
<Response>
  <Result ResourceID="sudo">
    <Decision>Permit</Decision>
    <Status>
      <StatusCode Value="&status;ok" />
    </Status>
  </Result>
</Response>
```

7.3 Jump Server

A jump server for the SSH protocol can be realized with established software and a few configuration changes. One approach is to use the `ProxyCommand` setting of OpenSSH and only allow logins with public key authentication.

At first, the public key of each user that is allowed to connect to the jump server is added to the `authorized_keys` file of a dedicated user (e.g., “proxy”). In front of the key, the `ProxyCommand` is added, which always executes a script after the user has successfully been authenticated. An example for an `authorized_keys` file that executes the script `/usr/local/bin/authorize_user` is shown in Listing 7.4.

Listing 7.4: `ProxyCommand` in `authorized_keys` File

```
command="/usr/local/bin/authorize_user alice" ssh-rsa AAAAB3...
command="/usr/local/bin/authorize_user bob" ssh-rsa AAAAB3...
```

For each user, there is a public key and the `ProxyCommand` that executes the “`authorize_user`” script with the username as a parameter. The script is the only command a user is able to execute and it will be called automatically. This ensures that a user can only use the jump server to connect to another server but cannot run any other command on the jump server. Thereby, the server can be protected from attacks. The approach allows to check the username of the connecting user in the script and to perform an authorization check. Thereto, the script can use the central PDP and check if the user is authorized to connect to the target system, which is transmitted by the SSH client as a command and used as a variable in the script.

The user connects to the target system via the jump server by using the local SSH client and executing the command shown in Listing 7.5.

Listing 7.5: Command to Reach a Target System via the Jump Server

```
$ ssh -t proxy@jump-server target-system
```

The command will connect to the jump server as user “proxy” and transmits the hostname (e.g., “target-system”) of the machine to which the administrator wants to connect to as the command. Instead of treating the command as the real hostname, it could also be a unique identifier that is used to get the real hostname from a database.

Once the script has sent a request message to the PDP and received a positive authorization response, it can establish the connection to the target system. For that purpose, it needs to collect the username (e.g., “root”) and password for the given target system from a database.

The password is saved as ciphertext in the database and is decrypted with a private key on the jump server. With these credentials the script then connects to the target system and appends the SSH session to the client’s session. The administrator can then work on the target system as if he connected directly to the machine. As the script established the second part of the connection, the user never gets in touch with the credentials.

7.4 Web Interface and Password Database

The web-based user interface demonstrates a management platform for passwords like the password manager proposed in Chapter 6. A user can create new entries, modify existing ones or delete an entry. Each password is encrypted with a public key whereas the reveal function for a password is realized by using the private key to decrypt the ciphertext.

Besides revealing a password in the web interface, the jump server can retrieve the encrypted password and decrypt it with the private key. After that, the password can be used to establish a connection to the target system.

Concluding, the jump server and the web server are the only components that deal with cryptographic keys. The database only contains encrypted passwords and the server on which the database is hosted does not have access to the public or private key.

8 Evaluation

At first, this chapter will analyze the differences between the requirements of the Leibniz Supercomputing Centre and iC Consult. After that, the applicability of the proposed architecture from Chapter 6 and the analyzed software products from Chapter 5 will be evaluated.

8.1 iC Consult Compared to the Leibniz Supercomputing Centre

The requirements of iC Consult and the LRZ, which have been discussed in Chapter 4, are summarized in Table 8.1. A \checkmark declares that the organization has the requirement while an X signals that the requirement is unwanted or not given. A \diamond specifies that the organization classifies the requirement as “nice to have”.

The requirements analysis in Chapter 4 showed that the main difference between the LRZ and iC Consult is that most of the systems that the administrators of the LRZ operate, belong to their organization, while employees of iC Consult not only need to manage privileged user accounts of their internal infrastructure but also those of their customer’s infrastructures. This implicates that the privileged user password management at the LRZ can be more integrated into the infrastructure than at iC Consult. However, at both organizations the management solution has to be capable of being integrated into the existing authentication infrastructure and the authorization should be centralized as well.

The requirement for a support of heterogeneous networks is given at the LRZ but at iC Consult the requirement only applies to the internal infrastructure. Moreover, because of the external systems that iC Consult is in charge of, the fine-grained access control on the operating-system level can only be implemented in each infrastructure separately and if the particular customer wants it. At the LRZ, the integration can be accomplished easier because of the coherent infrastructure.

The fine-grained access control on passwords is important for both organizations, as well as the emergency access and separation of duties. Thus, the privileged user password management needs to provide an access control that at least supports a role-based authorization schema but an attribute-based access control might be more flexible.

Another difference between the organizations can be seen at the “request/confirm access to privileged account” requirement. In contrast to the LRZ, iC Consult requires a request process for new privileged accounts that are approved by an authorized person.

The auditing and reporting section shows that the LRZ has the business regulation to not

Table 8.1: Requirements of the Leibniz Supercomputing Centre Compared to iC Consult

Requirement	LRZ	iC Consult
Systems Integration		
Support for heterogeneous networks	✓	✓
Centralized authentication	✓	✓
Centralized authorization	✓	✓
Account discovery/synchronization	✓	◇
Access Control		
Fine-grained access control on passwords	✓	✓
Emergency access	✓	✓
Separation of duties	✓	✓
Support for standardized policy language	◇	◇
Policy transformation mechanism from high-level policy language to low-level MAC policies	◇	◇
Fine-grained access control on operating-system level	✓	◇
Password Management		
Password security	✓	✓
Account check-out	◇	◇
Account check-in and one-time password	X	◇
Copy and paste a password	✓	✓
Platform-independent access	✓	✓
Limited access time	✓	✓
Hide password from the user	X	◇
Support for applications	✓	✓
Automatic password change	X	◇
Password policies	✓	✓
Request/confirm access to privileged account	X	✓
Password organization	✓	✓
Auditing and Reporting		
Audit logs	X	✓
Security of audit trails	-	✓
Notifications	✓	✓
Account matrix	✓	✓
Session recording	X	✓

track the actions that are performed by administrators. This includes audit logs and session recording. On the other hand, iC Consult operates various external infrastructures with several stakeholders that have access to the systems. Thus, audit trails are necessary to ensure accountability.

8.2 Applicability Analysis for the Leibniz Supercomputing Centre

This section analyzes the applicability of the architecture that was proposed in Chapter 6 and the software that was analyzed in Chapter 5, based on the requirements from Section 4.4.

8.2.1 Applicability of the Architecture

The centralized approach of the architecture as well as the policy-based authorization language XACML can provide the basis for a privileged user password management at the LRZ. XACML fulfills the requirements for a separation of duties and the definition of rules that allow an emergency access. With the transformation of high-level policies to low-level policies a fine-grained access control can be realized so that privileged users do not have unrestricted access on all systems.

Certainly, the LRZ noted that managing the access rights for every single administrator is too time consuming and complex in their infrastructure. An alternative is to combine the administrative tasks into generic groups that have the attributes assigned, so that database administrators have other access permissions than operating system administrator, even if they share the same privileged account. This reduces the complexity of the access management but still gives a security improvement.

Besides that, the architecture adds another authorization concept to the infrastructure. Thus, the organization might need to manage a PDP besides the existing Active Directory and LDAP servers.

The approach of having a jump server to connect to target systems (e.g., routers and switches) is already fulfilled with the BalaBit Shell Control Box.

8.2.2 Applicability of the Analyzed Software

The CA ControlMinder provides a lot of features and fulfills most of the requirements that were outlined in the requirements catalog of Section 4.6. Thus, it would be suitable for the LRZ because features such as the auditing and session recording can be deactivated so that the product adheres to the business regulations.

The Netwrix Privileged Account Manager misses some of the requirements that the LRZ

has, for example, a fine-grained access control on passwords as well as notifications and a seal functionality.

The Soffid Identity and Access Management software suits a bit better than the Netwrix product but it also misses some requirements, including the seal functionality. However, because of the open-source approach and the LRZ's close integration in research, the software could be extended to the needs of the organization.

8.3 Applicability Analysis for iC Consult

This section analyzes the applicability of the architecture that was proposed in Chapter 6 and the software that was analyzed in Chapter 5, based on the requirements from Section 4.5.

8.3.1 Applicability of the Architecture

The attribute-based approach of the architecture provides a more flexible access control than a role-based access control. Once the central components like the policy decision point, policy administration point, and policy repository are established, other systems and applications can utilize the centralized authorization.

However, most of the existing business applications cannot be integrated into the proposed architecture until they provide adequate interfaces or directly implement the support for a standard like XACML. Thus, a component like the password manager can be implemented by using standardized interfaces but it would be an in-house development.

In order to leverage the proposed architecture not only for the internal infrastructure but also for external infrastructures, the architecture might be extended to a federated privileged user password management so that policies could be deployed to different systems.

8.3.2 Applicability of the Analyzed Software

For iC Consult, a product like the CA ControlMinder suite seems to be applicable as it fulfills almost all requirements that have been analyzed in this thesis. The software could be integrated into the internal infrastructure as well as into the external infrastructure of their customers. However, for the centralized privileged user password management that iC Consult requires for their customer projects, a less extensive product might be applicable as well.

In contrast, the Netwrix Privileged Account Manager misses several features and is thus not suitable for the needs of the organization. One example is the absence of an access control that provides an assignment of passwords to groups or singular administrators.

The open-source software from Soffid has an integrated identity and access management but misses some important features, like the emergency access and audit logs. Such features could be integrated but it requires time and effort and would not come close to the marketability of a commercial product.

9 Conclusion and Future Work

This chapter will summarize the thesis and outline possible topics for future work that can be based on the findings of this thesis.

9.1 Summary

This thesis showed that administrators use privileged accounts to perform day-to-day tasks and that they often have unrestricted access to business-critical systems and data. In consequence, the unlimited access rights of such privileged accounts can lead to negligent actions and insider attacks that are difficult to prevent. Moreover, the sharing of privileged accounts between several employees increases the security risk because the accountability and non-repudiation is violated.

However, the requirements analysis has shown that shared privileged accounts are required for emergency situations and that the utilization of personal accounts often is not feasible or cannot be implemented on some systems because they only provide a simple access control with one privileged account.

For that purpose, a privileged user password management solution needs to consider more special use cases than a normal password manager that manages non-privileged credentials. Certainly, usual password managers can help sharing the credentials and can provide a fine-grained access control to prevent that unauthorized users have access to passwords. Moreover, extensive software products can increase the security with frequent password changes and one-time passwords that are automatically changed on the target systems.

However, the threat model has shown that not only the privileged user passwords need to be protected from attacks but also the threats that arise from administrators using privileged accounts need to be considered. The analysis showed that in order to ensure accountability on shared accounts, the password manager needs to log every access to a privileged user password. Besides that, it might also provide a check-out and check-in feature for passwords to restrict the access to one user at a time and use the action as an indication which administrator was logged in to the system at a given time.

Although it requires a lot of effort to define the access rights of each administrator, it is an improvement to the security to reduce the privileges to a minimum with centrally managed policies that are enforced on the target systems and in applications. Thereby, administrators can use their own personal account that has all privileges attached to fulfill the

every job. For situation where administrators require urgent access to privileges they are usually not authorized to use, the architecture also needs to provide business processes that are implemented in security policies. Besides that, there need to be password vaults for exceptional cases where a shared privileged account is required, including components that cannot be integrated in a central access control infrastructure, for example, routers or systems in foreign networks.

Besides following the principle of least privilege by establishing a fine-grained access control, the architecture should provide a secure audit trail in order to ensure non-repudiation and review the actions that were performed during an attack. This requires the authenticity and integrity of logs as well as a separation of duties that ensures that only authorized persons have access to the audit logs and that an attacker cannot cover his tracks by manipulating the logs.

The proposed architecture showed that business processes can be modeled with an access control policy language like XACML and that applications can utilize a centralized component to enforce the authorization decisions. With XACML, the separation of duties, the delegation of permissions, the principle of least privilege, and emergency accesses can be realized directly in the authorization policies. A jump server can help to protect from unauthorized accesses to target systems as well as ensure accountability and non-repudiation.

The analysis of three software solutions showed that the functional range differs widely and that it depends on the organization's requirements what product suits best. Some of them only provide a password manager that allows administrators to access the credentials of privileged accounts while extensive products also considered the requirements that exist beyond the access to the privileged user password. For example the emergency access or the enforcement of the principle of least privileges in the operating system of the managed systems. As mentioned before, the management of fine-grained access rights on the operating-system level becomes more complex than the management of who is authorized to access what password.

However, for some organizations the establishment of a more complex privileged user password management infrastructure might be suitable. Especially for those that have outsourced the administration of their network and want to ensure that privileged users cannot perform actions beyond their area of responsibility. Moreover, audit trails can help to detect malicious actions and theft of business secrets.

9.2 Future Work

This thesis mainly covered basic approaches that can be improved in future work. This includes security enhancements as well as improvements to the proposed architecture. Some of them have already been addressed in the previous chapters and reference to the subsequent sections which will outline proposals for future research.

9.2.1 Attribute-Based Encryption For Sensitive Data

In the area of asymmetric cryptography, the attribute-based encryption provides a solution to the problem where different people with different key pairs want to have access to the same encrypted data. In this case, the traditional encryption scheme requires to encrypt the same data (e.g., a password) for every known public key of a subject that might demand access to it. This is because it is necessary to have the public key of the receiver that will decrypt the message before being able to encrypt the data for the receiving subject [20].

Hence, an alternative encryption scheme has been proposed by Adi Shamir in 1984 [66]. The idea behind the *Identity-Based Encryption* (IBE) was to replace the digital certificates with an identifier, like the user's email or IP address. Certainly, this approach also requires to encrypt the data with different public keys when more than one person has to be able to decrypt the data.

Based on the identity-based cryptosystem, Sahai and Waters proposed the *Fuzzy Identity-Based Encryption* [61] in combination with the *Attribute-Based Encryption* (ABE) [23], which aimed to overcome the limitations of having to encrypt the data with multiple public keys. The idea behind their approach was to define attributes that make up the identity of a subject. An encrypted message can be decrypted if and only if the receiver's attributes ω are close to the attributes $\omega' \subset \omega$ that are part of the public key that was used to encrypt the message. They invented a "set overlap" distance metric which compared the two attribute sets and determined, if the subject is allowed to encrypt the data or not.

As a follow up work of the attribute-based encryption scheme, Bethencourt et al. proposed the *Ciphertext-Policy Attribute-Based Encryption* (CP-ABE) [4], which attaches an access structure to the ciphertext and attributes to the private key. The access structure of the ciphertext is formulated as a boolean expression over the attributes, which ensures that only a user can decrypt the ciphertext, whose attributes pass through the access structure.

In contrast to an approach that uses secret sharing, the CP-ABE provides *collusion-resistance*. This is because it makes it impossible that two or more users collude their attributes and thereby can decrypt the ciphertext which they could not have accomplished individually.

This approach can be used for a database that hosts sensitive data like passwords. The data is encrypted and includes an access structure that defines which attribute combination is allowed to decrypt the ciphertext. On the other hand, each subject has attributes attached to its private key, which are used to determine if the decryption is allowed.

In the case of the jump server, it is imaginable that the PEP or the PDP collect the encrypted password for the requested target system from the database and use the user's private key to decrypt it if possible. Thereby, an attacker that obtains access to the private key on the PDP could not decrypt all privileged user passwords. Instead, the attacker needs to collect several user-specific keys and combine them to be able to access all passwords.

9.2.2 Transformation of High-Level Policies to Low-Level Policies

Since not all operating system components and applications support an interface for a centralized authentication component like the policy decision point, it is necessary to establish a mechanism that transforms high-level policies (which are defined, e.g., with XACML) to application-specific low-level policies.

One example is the system-level transformation of XACML policies to SELinux policies which allows to map a policy like “Bob is authorized to view and edit the file */etc/hosts*” to a concrete mandatory access control rule that SELinux can enforce.

In [1] Alam et al. have proposed an architecture and a transformation mechanism for XACML policies to SELinux policies. In future work, other transformation mechanisms need to be considered, for example, network-level policies that allow to transform XACML policies to firewall rules [76].

9.2.3 Expansion to Other Operating Systems

Although the software analysis in Chapter 5 included Windows as well as Linux operating systems, the architecture in Chapter 6 mainly focused on open standards and UNIX-like operating systems.

Thus, the transfer of the XACML architecture to Windows servers can be examined in a future work. For that purpose, a replacement for the PAM approach needs to be found, so that applications can utilize XACML through a standard interface. There already exist implementations of policy decision point servers that run on Windows. Moreover, there are libraries for different programming languages that can be used in applications to implement a policy-based authorization with XACML directly. One example is the open-source library XACML.NET¹ for applications that are based on the Microsoft .NET Framework.

Furthermore, the fine-grained access control on the operating-system level is an open topic that can be researched. An approach for enforcing the policy decisions in the file system of Windows are *File System Filter Drivers*, which can intercept the system calls and provide authorization checks based on the decision of a policy decision point.

Another aspect is the automatic password change that requires an appropriate mechanism for Windows systems. This could be realized with a dedicated service that is installed on the server and that performs the local password change. Instead of using a custom implementation, the architecture could utilize Microsoft’s *Cusrmgr.exe tool*² or *PsPasswd*³ by Mark Russinovich which can be used to remotely change the password of an account.

¹<http://mvpos.sourceforge.net>

²<http://support.microsoft.com/kb/272530/en-us>

³<http://technet.microsoft.com/en-us/sysinternals/bb897543.aspx>

9.2.4 Virtual Environments

The virtualization technique provides a lot of advantages, including the reduction of costs and increase of availability, but it also creates new security challenges regarding the privileged user password management. This is because administrators that have access to the host machine can often access all guest machines that are virtualized by the hypervisor.

Moreover, virtual machines can be created and moved easier than physical machines which increases the amount of privileged accounts as well as the effort to monitor the network for malicious actions. This requires that the access rights of privileged users comply with the principle of least privilege and that administrative tasks are conformable with the separation of duties.

In a future work, the proposed architecture could be enhanced with a concept that addresses the hypervisor and secures the virtualized environments.

Bibliography

- [1] Masoom Alam, Jean-Pierre Seifert, Qi Li, and Xinwen Zhang. Usage Control Platformization via Trustworthy SELinux. In *Proceedings of the 2008 ACM Symposium on Information, Computer and Communications Security, ASIACCS '08*, pages 245–248, 2008.
- [2] William C. Barker. Information Security. *Recommendations of the National Institute of Standards and Technology*, August 2003.
- [3] Mihir Bellare and Bennet Yee. Forward Integrity For Secure Audit Logs. In *Technical Report, Computer Science and Engineering Department, University of San Diego*, November 1997.
- [4] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-Policy Attribute-Based Encryption. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy, SP '07*, pages 321–334. IEEE Computer Society, 2007.
- [5] R.A. Botha and J.H.P. Eloff. Separation of duties for access control enforcement in workflow environments. *IBM Systems Journal*, 40:666–682, 2001.
- [6] Daan Broeder, Bob Jones, David Kelsey, Philip Kershaw, Stefan Lüders, Andrew Lyall, Tommi Nyrönen, Romain Wartel, and Heinz J Weyer. Federated Identity Management for Research Collaborations. Technical report, CERN, April 2012.
- [7] Achim D. Brucker and Helmut Petritsch. Extending Access Control Models with Break-glass. In *Proceedings of the 14th ACM Symposium on Access Control Models and Technologies, SACMAT '09*, pages 197–206. ACM, 2009.
- [8] Axel Buecker, Paul Ashley, and Neil Readshaw. Federated Identity and Trust Management. Technical report, Tivoli Software, May 2008. <http://www.redbooks.ibm.com/redpapers/pdfs/redp3678.pdf>.
- [9] Axel Buecker, Barry Evans, and Dirk Rahnenfuehrer. Centrally Managing and Auditing Privileged User Identities by Using the IBM Integration Services for Privileged Identity Management, May 2010. <http://www.redbooks.ibm.com/redpapers/pdfs/redp4660.pdf>.
- [10] Leibniz Supercomputing Centre. The LRZ in a nutshell. <https://www.lrz.de/wir/lrz-flyer/lrz-flyer.pdf>, July 2012. [Accessed March 20, 2014].
- [11] David Chadwick and Stijn Lievens. Break the Glass Profile For XACML v2.0 and v3.0. February 2011. <https://lists.oasis-open.org/archives/xacml/201011/>

doc00000.doc.

- [12] John Chirillo and Edgar Danielyan. *Sun Certified Security Administrator for Solaris 9 & 10 Study Guide*. McGraw-Hill Professional Publishing, June 2005.
- [13] Shibboleth Consortium. How Shibboleth Works: Basic Concepts. <http://shibboleth.net/about/basic.html>. [Accessed December 26, 2013].
- [14] IBM Corporation. Avoiding insider threats to enterprise security. <http://public.dhe.ibm.com/common/ssi/ecm/en/wgw03016usen/WGW03016USEN.PDF>, October 2012. [Accessed February 20, 2014].
- [15] Lieberman Software Corporation. Privileged Identity Management. 2010.
- [16] NetWrix Corporation. NetWrix Privileged Account Manager, Datasheet. http://www.netwrix.com/download/Datasheets/Privileged_Account_Manager_Datasheet.pdf. [Accessed February 28, 2014].
- [17] Leibniz-Rechenzentrum der Bayerischen Akademie der Wissenschaften. Jahresbericht 2012. <https://www.lrz.de/wir/berichte/JP/JPBer2012.pdf>, July 2013. [Accessed March 20, 2014].
- [18] IBM developerWorks. IBM Security Privileged Identity Manager Technical White Paper. https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/Wc42dcb83bbad_485d_a112_1d03408e24c0/page/IBM%20Security%20Privileged%20Identity%20Manager%20Technical%20White%20Paper, February 2013. [Accessed November 5, 2013].
- [19] Kurt Dillard. Intrusion Detection FAQ: What is a bastion host? <http://www.sans.org/security-resources/idfaq/bastion.php>. [Accessed February 11, 2014].
- [20] Nishant Doshi and Devesh Jinwala. Updating Attribute in CP-ABE: A New Approach. *IJCA Proceedings on International Conference in Distributed Computing and Internet Technology, ICDCIT:23–28*, January 2013.
- [21] Garo Doudian. Generic Accounts and Non-Repudiation. <http://www.giac.org/paper/gsec/3940/generic-accounts-non-repudiation/106335>, June 2004. [Accessed November 29, 2013].
- [22] Open Software Foundation. Unified login with pluggable authentication modules (PAM). <http://www.opengroup.org/rfc/rfc86.0.html>, October 1995. [Accessed February 11, 2014].
- [23] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based Encryption for Fine-grained Access Control of Encrypted Data. In *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS '06*. ACM, 2006.
- [24] Network Working Group. Policy Framework Architecture. <http://tools.ietf.org/html/draft-ietf-policy-arch-00>, August 1999. [Accessed February 11, 2014].

- [25] Network Working Group. A Framework for Policy-based Admission Control. <http://tools.ietf.org/html/rfc2753>, January 2000. [Accessed February 10, 2014].
- [26] Network Working Group. AAA Authorization Framework. <http://tools.ietf.org/html/rfc2904>, August 2000. [Accessed February 9, 2014].
- [27] Network Working Group. Generic AAA Architecture. <http://tools.ietf.org/html/rfc2903>, August 2000. [Accessed January 17, 2014].
- [28] Network Working Group. Internet Security Glossary, Version 2. <http://tools.ietf.org/html/rfc4949>, August 2007. [Accessed October 23, 2013].
- [29] Michael Hafner, Mukhtiar Memon, and Muhammad Alam. Modeling and Enforcing Advanced Access Control Policies in Healthcare Systems with SECTET. September 2007.
- [30] Dick Hardt. The OAuth 2.0 Authorization Framework. <http://tools.ietf.org/html/rfc6749>, October 2012. [Accessed April 5, 2014].
- [31] Red Hat. Red Hat Enterprise Linux 6 Security Guide. https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux/6/pdf/Security_Guide/Red_Hat_Enterprise_Linux-6-Security_Guide-en-US.pdf, 2013. [Accessed January 7, 2014].
- [32] Tobias Heide. Introduction to pam_xacml. http://pamxacml.sourceforge.net/Documents/pam_xacml.htm. [Accessed December 26, 2013].
- [33] Wolfgang Hommel. Using XACML for Privacy Control in SAML-based Identity Federations. In *In Proceedings of the 9th Conference on Communications and Multimedia Security (CMS 2005)*, September 2005.
- [34] iC Consult. Independent Consulting leads to intelligent solutions. <http://www.ic-consult.com/en-US/portfolio.html>. [Accessed January 17, 2014].
- [35] IETF. The Syslog Protocol. <http://tools.ietf.org/html/rfc5424>, 2009. [Accessed March 16, 2014].
- [36] Ponemon Institute. The Business Case for Data Protection: What Senior Executives Think about Data Protection. http://www.ponemon.org/local/upload/file/Business_Case_for_Data_Protection_WP.pdf, February 2012. [Accessed February 1, 2014].
- [37] SANS Institute. A Layered Security Model: OSI and Information Security. <http://www.giac.org/paper/gsec/3908/layered-security-model-osi-information-security/106272>, June 2004. [Accessed February 3, 2014].
- [38] Sonia Jahid, Carl A. Gunter, Imranul Hoque, and Hamed Okhravi. MyABDAC: Compiling XACML Policies for Attribute-based Database Access Control. In *Proceedings of the First ACM Conference on Data and Application Security and Privacy, CODASPY '11*, pages 97–108, 2011.

- [39] Yu-Lin Jeng. An OpenID Based Authentication Mechanism in a Distributed System Environment. *International Journal of Computer and Communication Engineering*, September 2012.
- [40] Köhler. Jochen. Privileged Identity Management. Administrative Zugriffe auch in der Wolke unter Kontrolle. http://www.securitymanager.de/magazin/privileged_identity_management.html, November 2012. [Accessed October 18, 2013].
- [41] Pauli Kaila. OAuth and OpenID 2.0. *Proceedings of the seminar on network security*, December 2008.
- [42] Karen Kent and Murugiah Souppaya. Guide to Computer Security Log Management. *Recommendations of the National Institute of Standards and Technology*, September 2006.
- [43] Andreas Klenk, Tobias Heide, Benoit Radier, Mikaël Salaün, and Georg Carle. Pluggable Authorization and Distributed Enforcement with pam_xacml. *Informatik Aktuell*, pages 253–264. Springer, 2009.
- [44] Janet Kuhn. Decrypting the MoSCoW Analysis. <http://www.itsmsolutions.com/newsletters/DITYvol5iss44.pdf>, November 2009. [Accessed March 20, 2014].
- [45] Rebekah Lepro. Cardea: Dynamic Access Control in Distributed Systems. November 2003. <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.140.8932&rep=rep1&type=pdf>.
- [46] Sophos Ltd. Security Threat Report 2013. <http://www.sophos.com/de-de/medialibrary/PDFs/other/sophossecuritythreatreport2013.pdf>, 2013. [Accessed November 2, 2014].
- [47] J.D. Meier, Alex Mackman, Michael Dunner, Srinath Vasireddy, Ray Escamilla, and Anandha Murukan. Improving Web Application Security: Threat Modeling. <http://msdn.microsoft.com/en-us/library/ff648644.aspx>, June 2003. [Accessed December 3, 2013].
- [48] J.D. Meier, Alex Mackman, Michael Dunner, Srinath Vasireddy, Ray Escamilla, and Anandha Murukan. Improving Web Application Security: Threats and Countermeasures. <http://msdn.microsoft.com/en-us/library/ff648641.aspx>, January 2006. [Accessed November 5, 2013].
- [49] Russell Miller. Beyond Passwords: a fine-grained approach to Privileged Identity Management. *Privileged Identity Management and Virtualization Security*, January 2013. <http://transform.ca.com/rs/catech/images/beyond-passwords-a-fine-grained-approach-to-privileged-identity-management-wp.pdf>.
- [50] Alan Nagelberg. Identity Federation Concepts. http://assets1.csc.com/cybersecurity/downloads/FIM_White_Paper_Identity_Federation_Concepts.pdf, July 2010. [Accessed December 26, 2013].

- [51] AEP Networks. How to Protect Your Critical Resources with Identity-based Access Control. 2008.
- [52] OASIS. Architecture of Policy Administration Point (PAP) and related components. <https://wiki.oasis-open.org/xacml/Policy%20Administration%20Point%20Architecture>. [Accessed February 9, 2014].
- [53] OASIS. XACML Profile for Role Based Access Control (RBAC). <http://docs.oasis-open.org/xacml/cd-xacml-rbac-profile-01.pdf>, February 2004. [Accessed February 25, 2014].
- [54] OASIS. OASIS eXtensible Access Control Markup Language (XACML) Version 2.0. http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf, February 2005. [Accessed February 26, 2014].
- [55] OASIS. Security Assertion Markup Language (SAML) V2.0 Technical Overview. <https://www.oasis-open.org/committees/download.php/20645/sstc-saml-tech-overview-2%200-draft-10.pdf>, March 2008. [Accessed February 25, 2014].
- [56] OASIS. SAML 2.0 Profile of XACML. <http://docs.oasis-open.org/xacml/3.0/xacml-profile-saml2.0-v2-spec-cs-01-en.pdf>, August 2010. [Accessed February 25, 2014].
- [57] OASIS. XACML v3.0 Administration and Delegation Profile Version 1.0. <http://docs.oasis-open.org/xacml/3.0/xacml-3.0-administration-v1-spec-cd-1-en.html>, August 2010. [Accessed February 9, 2014].
- [58] U.S. Department of Health & Human Services. Security Standards: Administrative Safeguards. <http://www.hhs.gov/ocr/privacy/hipaa/administrative/securityrule/adminsafeguards.pdf>, September 2007. [Accessed March 20, 2014].
- [59] California Department of Technology. Network Architecture Standard. http://www.servicecatalog.dts.ca.gov/services/professional/security/docs/3117_Network_Architecture_Standard.pdf, July 2013. [Accessed March 1, 2014].
- [60] N. Papatheodoulou and N. Sklavos. Architecture & system design of Authentication, Authorization, & Accounting services. *EUROCON 2009*, IEEE:1831–1837, May 2009.
- [61] Amit Sahai and Brent Waters. Fuzzy Identity-Based Encryption. In *Advances in Cryptology - EUROCRYPT 2005*, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, May 2005.
- [62] Karen Scarfone and Peter Mell. Guide to Intrusion Detection and Prevention Systems (IDPS). *Recommendations of the National Institute of Standards and Technology*, February 2007.
- [63] Karen Scarfone and Murugiah Souppaya. Guide to Enterprise Password Manage-

- ment. *Recommendations of the National Institute of Standards and Technology*, April 2009.
- [64] Fred B. Schneider. Least Privilege and More. *IEEE Security & Privacy*, pages 55–59, 2003.
- [65] Bruce Schneier and John Kelsey. Secure Audit Logs to Support Computer Forensics. *ACM Trans. Inf. Syst. Secur.*, pages 159–176, May 1999.
- [66] Adi Shamir. Identity-based Cryptosystems and Signature Schemes. In *Proceedings of CRYPTO 84 on Advances in Cryptology*, pages 47–53. Springer-Verlag New York, Inc., 1985.
- [67] Adam Shostack. Experiences Threat Modeling at Microsoft. http://blogs.msdn.com/cfs-file.ashx/___key/communityserver-components-postattachments/00-08-99-18-06/Shostack_2D00_ModSec08_2D00_Experiences_2D00_Threat_2D00_Modeling_2D00_At_2D00_Microsoft.pdf, 2008. [Accessed February 11, 2014].
- [68] Amichai Shulman. Top Ten Database Security Threats. http://www.schell.com/Top_Ten_Database_Threats.pdf, 2006. [Accessed November 20, 2013].
- [69] George Silowash, Dawn Cappelli, Andrew Moore, Randall Trzeciak, Timothy J. Shimeall, and Lori Flynn. Common Sense Guide to Mitigating Insider Threats, 4th Edition. Technical report, Carnegie Mellon University, December 2012. http://www.ncix.gov/issues/ithreat/docs/Common_Sense_Guide_to_Mitigating_Insider_Threats.pdf.
- [70] Soffid. XACML. <http://confluence.soffid.org/display/SOF/XACML>. [Accessed February 25, 2014].
- [71] Soffid. Identity and Access Management, Data Sheet. http://soffid.com/wp-content/uploads/2013/04/Soffid-Product-Sheet_v0.11.pdf, April 2013. [Accessed January 17, 2014].
- [72] Marianne Swanson and Barbara Guttman. Generally Accepted Principles and Practices for Securing Information Technology Systems. <http://csrc.nist.gov/publications/nistpubs/800-14/800-14.pdf>, September 1996. [Accessed February 11, 2014].
- [73] Bob Tarzey, Cleve Longbottom, and Mariateresa Faregna. Privileged user management. http://www.ca.com/Files/SupportingPieces/quocirca_priviledgd_usr_mgmt_oct_19_09_219925.pdf, 2009. [Accessed October 18, 2013].
- [74] CA Technologies. CA ControlMinder Shared Account Management. <http://www.ca.com/us/~/media/Files/DataSheets/ca-controlminder-shared-account-management.PDF>, 2013. [Accessed January 16, 2014].
- [75] CA Technologies. Privileged Identity Management with CA ControlMinder. <http://www.ca.com/au/~/media/Files/SolutionBriefs/ca->

- controlminder-solution-brief.pdf, 2013.
- [76] Tugkan Tuglular. Firewall Configuration Management Using XACML Policies. In *Telecommunications Network Strategy and Planning Symposium, 2008. Networks 2008. The 13th International*, pages 1 – 29, 2008.
- [77] Valerio Venturi, Tom Scavo, and David Chadwick. Use of SAML to retrieve Authorization Credentials. <http://www.ogf.org/documents/GFD.158.pdf>, November 2009. [Accessed February 25, 2014].
- [78] Dinesh C. Verma. Simplifying Network Administration Using Policy-based Management. *Network. Mag. of Global Internetwkg.*, 16(2):20–26, March 2002.
- [79] Harry L. Waldron. Security Is a Business Requirement. <http://technet.microsoft.com/en-us/library/cc512684.aspx>, July 2007. [Accessed February 20, 2014].
- [80] Christian Wolter, Andreas Schaad, and Christoph Meinel. Deriving XACML Policies from Business Process Models. In *Proceedings of the 2007 International Conference on Web Information Systems Engineering, WISE'07*, pages 142–153, 2007.
- [81] Eric Yuan and Jin Tong. Attributed Based Access Control (ABAC) for Web Services. In *Proceedings of the IEEE International Conference on Web Services, ICWS '05*, pages 561–569, 2005.