Usability of the Access Control System for OpenLDAP

by

Yi Fei Chen

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2019

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

This thesis addresses the usability of the Access Control System of OpenLDAP. OpenLDAP is a open source implementation of the Lightweight Directory Access Protocol (LDAP), which is a protocol that communicates with a directory service. A directory service is a database that stores information about network resources, such as files, printers and users. An access control system is the mechanism that mediates access, for example, read or write, to a resource by a user. The access control system makes these decisions based on an access control policy which states who should have access to what. We hypothesize that the access control system of OpenLDAP has poor usability. By usability, in this context, we mean how easy it is for a systems administrator to encode a high-level, informally expressed, enterprise security policy as an access control policy in syntax that OpenLDAP expects. We discuss the design and carrying out of a human-subject study to validate this hypothesis. The study consist of presenting a high-level policy to the participants and asking them to translate it into an OpenLDAP policy. The study has been approved by the University of Waterloo's office of research ethics. We have carried out the study with a total of 54 users. We present the results from analyzing the data we collected from the study. We observe that our hypothesis is validated in that only few (20%) people were able to express a high-level policy as a correct OpenLDAP policy. There is a low correlation between self reported correctness and actual correctness which suggest that people are not aware if they made any mistake in their submission. The main source of error comes from confusion about the OpenLDAP syntax and how precedence rule works.

Acknowledgements

I would like to thank my supervisor Mahesh Tripunitara for his help on my thesis

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Access control is an important part of a security system. It constrains what a user can do directly, as well as what programs executing on behalf of a user are allowed to do. In this way access control seeks to prevent activity that could lead to breach of security [19]. We refer to an user or a program that carries out an action as a *principal*. We refer to the action an user or a program performs as an *access*. There can be different levels of access, for example, read or write, we call these different levels *permissions*. The item that is read or written to is called a *resource*. The figure 1.1 below show the role of access control in the overall system system. Access control allows us to set different access permissions for different principals so that confidential resources could only be accessed by principal with higher privilege. This configuration is specified by a security policy that is maintained by a special user called the administrator.

A well-established syntax to express an access control policy is the access control list[19]. An access control list is an ordered tuple of access control rules. Each rule is composed of resource, principal and permission. These rules specifies which principal can access this resource with what kind of permission. For example in the following policy Alice can read and write the docs and Bob could only read docs.

```
docs: Alice: read,write
docs: Bob: read
```

In this thesis, we address access control in the context of OpenLDAP, which is an open source implementation of the Lightweight Directory Access Protocol (LDAP). LDAP is a protocol that communicates with a directory service. A directory service is a database that

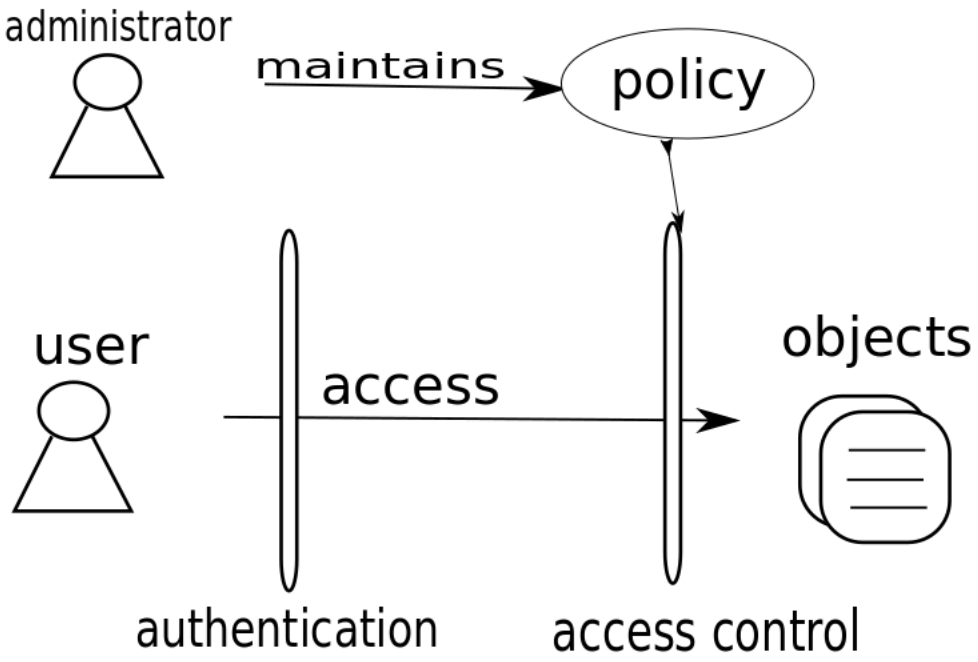

Figure 1.1: The diagram shows the function of access control in the security system. There is a usually an authentication mechanism to validate the user then the access control mechanism determines if that user has the right to access some object based on a policy maintained by the administrator.

contains information about network resources such as files, computers, printers and users in a hierarchical structure. An example of a directory service is Microsoft's Active Directory [13] . Microsoft's Active Directory is used, for example, to store login information. The login program then interacts with the directory service at the time when an user attempts to login.

OpenLDAP, as we mention above, is an open source implementation of LDAP which is compatible with GNU/Linux server. In the protocol standard there is no specification for the implementation of access control in LDAP; thus, each variant of LDAP, and in particular OpenLDAP, has its own implementation [10].

OpenLDAP uses the syntax of an access control list. OpenLDAP's syntax for access control differs from more traditional syntaxes for access control lists such as the one for the POSIX file system in that it has features to make it more flexible and expressive. These features also render the syntax more complex.

To illustrate these points, we now present an example of a OpenLDAP policy policy.

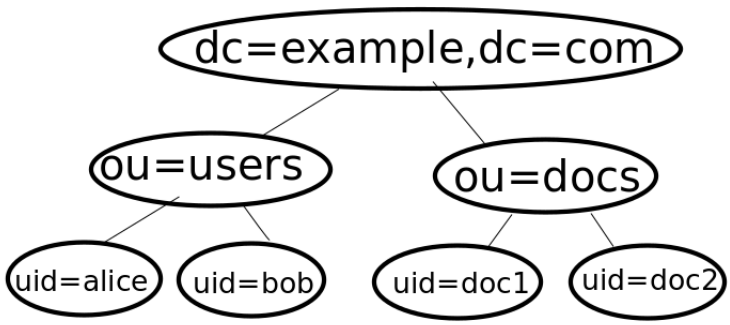

Figure 1.2: In this tree, there is a group of two users Alice and Bob and a group of documents called docs. The high-level policy we want is for Alice to able to read and write every docs and for Bob to only be able to read doc1.

```
1: to dn.subtree="ou=docs,dc=example,dc=com"
2:   by dn.subtree="uid=alice,ou=users,dc=example,dc=com" write
3:   by * break
4: to dn.subtree="uid=doc1,ou=docs,dc=example,dc=com"
5:   by dn.subtree="uid=bob,ou=users,dc=example,dc=com" read
```

The OpenLDAP police above encodes the high-level policy mentioned in Figure 1.2. Line (1) specifies all "`docs`" as the resource and then Alice is specified as the principal `write` permission is given which contain the `read` permission in OpenLDAP. At line (3) there is a control statement "`break`" which is needed for the current setup. Line (4) is where the second rule starts. Only doc1 is specified as the resource and Bob is specified as the principal. The permission for this rule is `read`.

1.1 Our work

In this study, we are investigating the usability of OpenLDAP's access control system. Usability in this context is about how easy it is for administrators to use the access control as a tool to configure the system in order to archive certain security property. In this thesis, we are focused on how easy it is to translate a high-level policy into a OpenLDAP policy. This is often a necessary step because "policies are made/implemented by multiple people" [2]. There is usually a business manager who is the policy maker. He wants certain security property based on use cases that the system would need to satisfy. The policy implementer is a system administrator who is familiar with the technology used in the system. The high-level policy, usually in English, is what the business manager gives to the system administrator who then transforms it into a policy that OpenLDAP understand. This is not the only possible task involving access control system. Sometimes the opposite happen like in the paper by Bera et al [5] where they are assembling rules in a distributed system to generate a central policy. There are other papers discussing what kind of high-level policy is the best like the paper by Maritza et al [11]. These aspects are not in the scope for this study. In this study, we focused on the task of translating the high-level policy into an OpenLDAP policy in a setup that simulates a small company.

Multiple online blog that teacher how to use OpenLDAP access control system stated that it is very hard to use:

One Stanford tutorial states "Writing Access Control Lists (ACLs) in OpenLDAP can be one of the most difficult tasks to undertake. One needs to really consider what goals they are trying to accomplish with their ACLs"[23]. This tutorial teaches all the different ways we can specify the principal and resource in OpenLDAP. These ranges from the most basic such as a subtree group to more complex like regular expression matching. We can also use attribute list or connection property to filter these expression. Therefore, the tutorial suggest that the complexity from all these different way of specifying principal and resource make OpenLDAP difficult to use.

Ingo Bente wrote up a tutorial about OpenLDAP from his own experience working with it. He said "there are multiple ways to really mess things up when changing your LDAP ACLs"[4]. He identified sequencing as a big problem. Other difficulties related to how other programs use OpenLDAP to retrieve the information and it is not obvious what access right the other programs need.

Zytrax's explained "The access directive (ACL) is brutally complex. It allows very fine control over who can access what objects and attributes and under what conditions. The side-effect of this complexity and power is that it is very easy to get olcAccess (access to) attributes/directives horribly wrong." [29] This suggest that the difficulty comes from all the different additional features in the languages that modifies the meaning of a basic rule such as the attribute list filtering of principal and the resource. Another feature that can be difficult to use is how the control statements influencing the precedence rule.

Based on my own cognitive walkthrough, in the task of translating a high-level policy into a OpenLDAP policy, there are two potential source of error. One is the ability to identify the principal and resource correctly and the other is ability to work out the precedence rule with all the different control statements that modify it.

Our hypothesis is:

> *The access control system of OpenLDAP suffers from poor usability.*

We designed an usability study where participants are first taught the basics of LDAP and OpenLDAP 's access control system. We showed them examples of different OpenLDAP policies and help them through a quick exercise. Finally they are given a high-level policy and ask to do the task of transforming it into an OpenLDAP policy. This high-level policy approximates a basic setup for a small company.

The following chapters explain the basics of OpenLDAP, the design of the study and how the data was analyzed.

Chapter 2

Overview of OpenLDAP access control system

This overview starts with the explanation of Directory Information Tree which is the back end system that LDAP connects to. Then we show the structure of OpenLDAP policy and the semantic meaning of different parts of the syntax. Lastly, the precedence rule that resolves conflicts is explained.

2.1 Directory Information Tree

As stated before in the introduction, LDAP is a protocol that communicates with a directory service and a directory service is a database that stores information about network resources in a hierarchical structure. That structure is a Directory Information Tree. There is a single root node at the top of the directory tree. Each node in the tree beside the root node have one and only one parent node. Each node can have any number of child nodes. In general, each leaf node in the tree represents one network resource such as a single user, a file or a computer. The intermediate nodes are there to give the tree its structure and organize the nodes into logical groups. For example, each department in a company can be a different subtree where the leaf nodes are the employees. Each node in the Directory Information Tree is associated with a list of attributes and each attribute has a value. The format of the value depends on the attribute. Each node in the tree belong to one objectclass. The objectclass specifies what are the mandatory and optional attributes of the node. There is a schema file that defines these objectclasses and the format of the

value for each attribute. Each node in the tree is uniquely identified by a distinguished name(DN). The distinguished name is a list of comma separated terms, where each term has the form "objectclass=name". The distinguish name of any child nodes contains the distinguish name of its parent as a suffix. This naming convention makes distinguish name form a path from the root node to the current node.

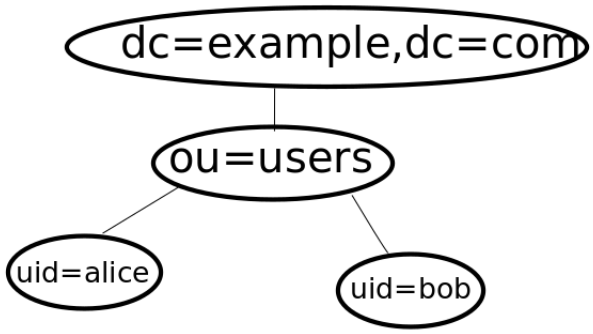

Figure 2.1: This is a simple example of a directory information tree. The two leaf node represent uses and has DN. "uid=alice, ou=user, dc=example, dc=com " and "uid=bob, ou=user, dc=example, dc=com" . Their parent is "ou=user, dc=example, dc=com" like the name suggest this node functions as a parent node to a group of users.

2.2 Extended Backus-Naur Form

In this thesis, we use the Extended Backus-Naur Form (EBNF) notation to explain the syntax of OpenLDAP policy.

The EBNF notation is composed of a list of production rules. Each production rule has ":=" to separate the right hand side from the left hand side. The left hand side is always a non-terminal and the right hand side is a sequence of terminals and non-terminals. These production rules tells us we can replace the left hand side non-terminal with the sequence on the right hand side. There is a top non-terminal. Using the production rule, we can replace that non-terminal with sequence of terminals and non-terminals. Repeat this step for every non-terminal in the sequence, until the final expression consist of only terminals, which would be a syntactically valid expression.

In EBNF notation angle bracket is used to indicate a non-terminal. The vertical bar is used

for options as you can choose one of the expressions before or following the bar. Square brackets means the terms inside are optional and square brackets with a plus sign means that the terms inside can appear one or multiple times. Braces means that the terms can appear zero or more times.

2.3 Syntax of the OpenLDAP access control policy

Below is the OpenLDAP syntax in EBNF notation taken from the OpenLDAP documentation page [**slapd**]. However this is not a complete syntax since not every non-terminal has a production rule to explain how to decompose it. Even though it is not a complete syntax, this should give a basic idea of the structure of the syntax of OpenLDAP policy.

⟨*access directive*⟩ ::= to ⟨*what*⟩ [by ⟨*who*⟩ [⟨*access*⟩] [⟨*control*⟩]]+

⟨*what*⟩ ::= *
| [dn[.⟨*basic-style*⟩]=⟨*regex*⟩ | dn.⟨*scope-style*⟩=⟨*DN*⟩]
[filter=⟨*ldapfilter*⟩] [attrs=⟨*attrlist*⟩]

⟨*basic-style*⟩ ::= regex | exact

⟨*scope-style*⟩ ::= base | one | subtree | children

⟨*attrlist*⟩ ::= ⟨*attr*⟩ [val[.⟨*basic-style*⟩]=⟨*regex*⟩]
| ⟨*attr*⟩ , ⟨*attrlist*⟩

⟨*attr*⟩ ::= ⟨*attrname*⟩ | entry | children

⟨*who*⟩ ::= *
| [anonymous | users | self | dn[.⟨*basic-style*⟩]=⟨*regex*⟩ | dn.⟨*scope-style*⟩=⟨*DN*⟩]
[dnattr=⟨*attrname*⟩]
[group[/⟨*objectclass*⟩[/⟨*attrname*⟩][.⟨*basic-style*⟩]]=⟨*regex*⟩]
[peername[.⟨*basic-style*⟩]=⟨*regex*⟩]
[sockname[.⟨*basic-style*⟩]=⟨*regex*⟩]
[domain[.⟨*basic-style*⟩]=⟨*regex*⟩]
[sockurl[.⟨*basic-style*⟩]=⟨*regex*⟩]
[set=⟨*setspec*⟩]
[aci=⟨*attrname*⟩]

⟨*access*⟩ ::= [self]{⟨*level*⟩|⟨*priv*⟩}

⟨*level*⟩ ::= none
| disclose
| auth
| compare
| search
| read
| write
| manage

⟨*priv*⟩ ::= {=|+|-} {m|w|r|s|c|x|d|0}+

⟨*control*⟩ ::= [stop | continue | break]

An OpenLDAP policy is a list of rules. A rule in the OpenLDAP nomenclature is called a directive. Let's focus on the production rule for directive It has four non-terminals : ⟨*what*⟩, ⟨*who*⟩, ⟨*access*⟩ and ⟨*control*⟩. The semantic meaning of these components are explained in the following sections. The first component is ⟨*what*⟩. This is a resource selector. It is used to specify a group of items that is being accessed. ⟨*who*⟩ is the principal selector, It specifies a group of user that's doing the access. ⟨*access*⟩ specifies what kind of permission this rule allows such as read or write. Lastly, we have ⟨*control*⟩ which influence the precedence rule. The precedence rule is explain in its own section. The ⟨*who*⟩,⟨*access*⟩ and ⟨*control*⟩ are surrounded by a square bracket with a plus sign. This means that this tuple could appear one or multiple times in a single directive. We call this tuple a *subdirective*. lastly [⟨*control*⟩] is surrounded by just a square bracket, which means that it is optional. Putting all this together, The OpenLDAP access control list is composed of a list of directives. Each directive have a resource selector at the beginning and then one or multiple subdirectives, each composed of a principal selector, an access statement and optionally a control statement.

2.4 Resource selector

Resources in OpenLDAP that we can add or modify are the nodes of the tree and the attribute values associate with each node. First, let's look at the production rule for ⟨*what*⟩ ,the resource selector. There are two options, the first one is $*$, this is a special keyword to signify everything in the tree. Next we have ⟨*scope-style*⟩ or ⟨*basic-style*⟩.

Both these styles are used to select a group of nodes in the Directory Information Tree. ⟨*basic-style*⟩ uses regular expression matching to define a group of nodes. The nodes with their DN matching the regular expression form the group of nodes selected by basic style. While ⟨*scope-style*⟩ rely on the tree structure to create a group. In ⟨*scope-style*⟩ we can define a group of nodes using subtree by just specify the root node of this subtree.

We have two optional terms for ⟨*what*⟩. The first one is ⟨*ldapfilter*⟩, this is used to filter the group of nodes specified by the previous term. The filtering is based on attribute value. The ⟨*ldapfilter*⟩ follows the the syntax of search filer specified in LDAP protocol [22]. The last optional term is a ⟨*attrlist*⟩. This can take two forms one is the form of : attributename value=expression. This has the same function as the ldap filter which is to filter the group of nodes base on attribute value. The other form ⟨*attrlist*⟩ can take is a list of attributes. As stated before the resources can be specific attributes of the nodes. If this optional term is not present then the resource selector is selecting every attribute in the node of the group, but if this attribute list is present then only attributes in the attribute list are selected. This gives finer control to how the resource are selected.

2.5 Principal selector

In OpenLDAP, the principals can be any node in the directory information tree. However most of the time the principals are the leaf node that represent a network user. Like the resource selector ⟨*what*⟩, the principal selector, ⟨*who*⟩, can use ⟨*basic-stlye*⟩ or ⟨*scope-style*⟩ to specify a group of nodes as the principal. It also has a ⟨*attrlist*⟩ to filter base on attribute value. In addition to that, you could also filter base on network connection properties like what IP address you are connection from and the security strength of that connection. The principal selector also have more keywords like anonymous, user and self. Anonymous would match anyone who is not authenticated. User are anyone who is authenticated. Self is for when the resource is the same node as the principal.

2.6 Access

There are two ways to specify access permission: ⟨*level*⟩ and ⟨*priv*⟩ . In the study ⟨*level*⟩ are used, since it is easier to understand. ⟨*level*⟩ has 9 possible keywords that specify what kind of access they permit. Most of them are self-evident. The table below, taken from the openldap documentaion [16] shows all the permission keywords with a brief explanation of the kind of permission they allow. The ordering that these keywords appear in the

notation is actually important. With ⟨*level*⟩ any permission that appear later include the permissions that appear earlier. For example, `write` permission contains the `read` permission. Therefore using only the ⟨*level*⟩ syntax, it is not possible for an user to write something that he cannot read. If this behaviour is desired than we need to switch to the ⟨*priv*⟩ syntax. The ⟨*priv*⟩ syntax is more powerful as it can expresses more permission setup than the ⟨*level*⟩ notation. ⟨*priv*⟩ has eight possible letters and one number each signify some kind of access. They are "m,w,r,s,c,x,d,0". Each of these signify a kind of permission. The ⟨*priv*⟩ syntax is composed of two parts an operation sign and a sequence of letters. The operation sign states what to do with the permissions represented by the sequence of letters. Plus sign means adding those permissions and minus sign means remove those permissions. Equals sign means set those permissions, which means remove all previous permissions and then add those permissions specified by the current directive. The table below is taken from the OpenLDAP documentation[16]. It explains what each kind of permission means and also matches each ⟨*level*⟩ with their equivalence in ⟨*priv*⟩.

Level	Privileges	Description
none	=0	no access
disclose	=d	needed for information disclosure on error
auth	=dx	needed to authenticate (bind)
compare	=cdx	needed to compare
search	=scdx	needed to apply search filters
read	=rscdx	needed to read search results
write	=wrscdx	needed to modify/rename
manage	=mwrscdx	needed to manage

Table 2.1: OpenLDAP permissions

2.7 Control statment

The ⟨*control*⟩ is the control statement that influences how the precedence rule determines which directive takes effect when a conflict arises. There are three possible control statements: "`stop`", "`continue`" and "`break`". However the control statement is optional, when it is not present then it works as though the "`stop`" statement is there.

2.8 Precedence rule

The semantics explained in the previous sections shows how a single directive grants a permission to a group of resources for a groups of principal. However, a single resource can potentially appear in multiple different directives and a single principal can potentially appear in multiple different directives or just different principal groups in the same directive but different subdirectives. When that happens, there is a conflict and the precedence rule is used to determine which directive takes effect. The precedence rule is quite complex and works on a case by case basis where each pair of a single resource and a single principal is considered a case. The behaviour of how OpenLDAP determines the permission for any pair of principal and resource is explained using the pseudo code below. The pseudo code is written by us based on the explanation of the control statements in the OpenLDAP documentation. The part of the system that executes this algorithm is referred to as the *reference monitor*. There are two inputs to this algorithm. One of them is a tuple T(P,R,X) that contains the information of the access control check and the other is the OpenLDAP policy.

In the tuple T, T.P is the principal that is requesting the access. T.R is the resource that is being accessed and T.X is what kind of permission is required for this access. Both T.P and T.R are a single principal and a single resource respectively.

When an access control check is initiated, the reference monitor processes the directives in the order they appear in the configuration file. However there are some modifications made to the OpenLDAP policy. These modifications are an added default end directive and an added default end subdirective. There is a directive "to * by * none stop" added at the end of the policy. It is there to catch all the cases where no other directive matches the resource and the principal in T. The subdirective: " by * none stop" is added at the end of every directive. Please note that these modifications do not actually appear on the OpenLDAP policy they are there to make the precedence rule easier to understand conceptually.

The algorithm assumes that the permission is specified using the $\langle priv \rangle$ syntax if the $\langle level \rangle$ syntax is used we can covert them to their equivalence in $\langle priv \rangle$ according to the permission table in the previous section. In the pseudo code we use the word match to means when the resource selector contains T.P or when the principal selector contains T.R. The algorithm stores a set of permissions in a variable called accumulator which is the return value of the algorithm. If T.X is one of the permissions contained in the accumulator then the access control check succeeds and the access is allowed else the access is denied.

```
1  initialize accumulator to empty set
2  lastcontrol get none
3  // iterate through each directive
4  for  each directive , d, in acl do
5  |  if d.what matches T.b then
6  |  |  // iterate through each subdirective
7  |  |  for each subdirective, s, in d do
8  |  |  |  if s.who matches T.a then
9  |  |  |  |  // modify accomulator appropriately
10 |  |  |  |  if subdirective.accessop = "+" then
11 |  |  |  |  |  accumulator ← accumulator ∪ s.accesslevels
12 |  |  |  |  end
13 |  |  |  |  if subdirective.accessop = "-" then
14 |  |  |  |  |  accumulator ← accumulator \ s.accesslevels
15 |  |  |  |  end
16 |  |  |  |  if subdirective.accessop = "=" then
17 |  |  |  |  |  accumulator ← s.accesslevels
18 |  |  |  |  end
19 |  |  |  |  // store the control statement
20 |  |  |  |  lastcontrol ← s.control
21 |  |  |  |  if s.control ≠ continue then
22 |  |  |  |  |  break
23 |  |  |  |  end
24 |  |  |  end
25 |  |  end
26 |  |  // check if the last control is break
27 |  |  if lastcontrol ≠ break then
28 |  |  |  return accumulator
29 |  |  end
30 |  end
31 end
```

The reference monitor will execute this algorithm when an access control check is initiated. There are two loops in the code. The first one at line 4 and the second one at line 7. They are not infinite loops since there are a finite number of directives and subdirectives to loop through. However, because of the default directive the algorithm is guaranteed to return at line 28. The check for control statement at line 21 is reached when the principal selector

and resource selector matches the principal and resource in T. If the control statement is "`continue`" then the algorithm continues looping through the subdirectives. If the control statement is "`break`" then the check at line 27 fails and the algorithm continues looping through the directives. If the control statement is "`stop`" then the reference monitor returns with the value in the accumulator. This means that reference monitor always returns at line 28 after processing a matching subdirective that has "`stop`" control statement. This is the case for the default end directive since it has "`stop`" as control statement and its principal and resource matches everything. The default directive returns with `none` permission. Therefore, the question is if there is another directive and subdirective that matches the principal and resource in T and have a "`stop`" so that the reference monitor return before reaching the default end directive.

2.8.1 Example of effect of precedence rule

We now discuss some examples of the manner in which a control statement impacts an OpenLDAP policy. We use a line number followed by a colon, for example, "`3:`" so we can refer to each line with clarity. The following OpenLDAP policy comprises two lines, Line (1) and (2). Line (1) gives Alice `read` permission to herself. Line (2) has no effect because Line (1) matched.

```
1: to dn.subtree="uid=alice" by dn.subtree="uid=alice" read
2:                           by dn.subtree="uid=alice" write
```

Following is the same policy as Lines (1) and (2) with the "`continue`" control statement appended to Line (1). We refer to this new policy as Lines (3) and (4) below. In the OpenLDAP policy that comprises Lines (3) and (4) below, Line (3) grants Alice `read` to herself, but is however overridden by Line (4), which grants Alice `write` to herself. Suppose Alice attempts to exercise read access. Then even though this attempt matches the principal and resource in Line (3), the "`continue`" at the end of Line (3) causes the reference monitor to continue on to line (4) which also match and gives Alice `write` permission.

```
3: to dn.subtree="uid=alice" by dn.subtree="uid=alice" read continue
4:                           by dn.subtree="uid=alice" write
```

Another example, in this following OpenLDAP policy Bob does not have `read` permission to Alice because the first directive at line (5) have the resource selector that contains Alice.

When the reference monitor process the line (5) the default subdirective will match the case when bob tries to read Alice, thus it will return with `none` permission.

```
5: to dn.subtree="uid=alice" by dn.subtree="uid=alice" read
6: to dn.subtree="uid=alice" by dn.subtree="uid=bob" read
```

However, if we use the "`break`" control statement in the first directive. The reference monitor would not return at line (7) and continues processing to line (8). Then Bob would have `read` permission to Alice.

```
7: to dn.subtree="uid=alice" by dn.subtree="uid=alice" read by * break
8: to dn.subtree="uid=alice" by dn.subtree="uid=bob" read
```

These example shows that in order to check all the potential conflict we would need to keep track of all the other directives and figure out exactly who the principal and resource are and then try to identify if these groups of principal and resource intersect with any other groups of resource and principal in another directive. This can be difficult as we need compare every directive. Changes in a single directive could have unintended consequence for other directive in the list, thus we potentially need to go through the whole access control list again to verify the correctness after each change. This process is also what makes the access control very expressive because of the many different way of grouping the principal and resource and using different control statement, we can produce very different looking access control list which grants the same permissions. So there is not a single correct implementation when we want to write an access control list for a specific high-level policy.

Chapter 3

Design of the study

In this chapter, we explain how we conceived the task for the study and how we teach the participants to use OpenLDAP access control system.

3.1 Policy

As stated before the focus of the study is for the participants to translate a high-level policy into a corresponding OpenLDAP policy that accomplish the goals of the high-level policy. In the study, the high-level policy is one that imitates the setup of a small company. we decide not to design the study base on multiple simple task that accomplish a single goal at a time because a major step that can cause difficulties is how we organize the directives and if we need to separate them or merge together. If the study is a series of simple tasks of one goal at a time then there is less opportunity for the participants to reorganize the directives. Therefore, we design the the task to be composed of list of multiple goals that can potentially interfere with each other. Given the time limitation, we decided that a single task that is composed of multiple goals would be enough. For this study, the participants are given a high-level policy composed of a list of goals that we want to accomplish and they are specifically told that there are no ordering for these goals and there is no one to one mapping between each policy goal and a access control directive. This is to remind them that there are always multiple possible OpenLDAP policy that can satisfy the same high-level policy. The first list below is the high-level policy that is given to the participant as the goal of the task of the study.

The high level policy of the study

1. *admin has write permission to everything in the tree.*
2. *every user has write permission to her/his own attributes (displayName, mobile, telephoneNumber, and userPassword).*
3. *humanresources has write permission to all employee entries.*
4. *sambaservice has read permission to all userPasswords.*
5. *managers of engineering and accounting have write permission to all their users.*
6. *anonymous has authentication permission to all userPasswords.*
7. *all employees can read the displayName, mobile and telephoneNumber attributes of all other employees .*

Goals taken directly from Bente's blog [4]

A There is a technical admin user that has write access to everything. LDAP admins have write access to everything by using their personal user accounts.

B A user has write access to a certain set of his attributes, including the password. This is mandatory to support any reasonable kind of self service. However, a user should not be able to change all of his attributes.

C For example, a user should not be able to change hist email address (otherwise, your IT department might have a really bad day). Obviously, a user should not be able to read or change the password of other users.

D Members of User Help Desk and HR have access to manage user accounts by using their own, personal user accounts. This involves task like resetting passwords, deactivating existing accounts and provisioning of new accounts.

E There will be technical service accounts. Think of a Samba server that requires read permissions on some user attributes (like sambaNTPassword which sadly is still a thing). Such accounts should have read access to basically the entire LDAP tree, but should avoid to be given write access. You don't want to misuse your technical admin account as a service account either.

F This one is a bit tricky. Depending on your business, you will need some sort of group membership management. In order to keep your LDAP admins from freaking out, you need to come up with a distributed approach. We did the following: For each group, there is a specific set of managers. Each manager has write access to his group, so that he can add new members and remove existing members, but not to other groups.

G Anonymous cannot read or write anything.

This policy of the study is inspired by Bente [4] who sought to migrate an existing system to the OpenLDAP directory service. This policy has seven goals, Goals (1)–(5) are directly from Bente [4], except that my goals are simplified and my goals all have the same simple sentence structure : principal has permission to resource. This makes it easier for participants to figure out what the principal resource and permission are for each goal. The last two of my goals was not explicitly stated in Bente's policy but they are very common uses

cases.

Bente's Goals (A) and (B) merged into one goal since those two goals were very similar. They were both about some kind of administrator having write access to everything. This translate into my goal (1) : the system administrator have write to everything. My goal (2) is for self service where each user can change their own attributes. We did not include the constraint where some attribute are not writable by oneself. My goal (3) is give HR the ability to manage all employees. This is a translation of goal (D) from Bente. My goal (4) is to allow certain programs to work with the information stored in the directory tree. Specifically sambaservice is the SMB protocol of the windows system. This goal corresponds to Bente's goal (E). My goal (5) is for each department to be managed by their own managers which corresponds to Bente's goal (F). My goal (6) is an one rule that needs to be included in every working OpenLDAP policy since it is needed for user to login to the LDAP system. My goal (7) is for employee to know some basic contact information of other employees which is a very logical requirement.

This policy is applied to the LDAP tree below. Ingo Bente did not show what kind of tree he was working with. The tree below is a simple one that represents a small company with three departments that has managers for engineering and accounting departments. The participants are given the tree diagram below. They also have access to a program called ApacheDirectoryStudio which is used to examine Directory Information Trees. This program shows each node with its attribute and its place in the directory tree structure.

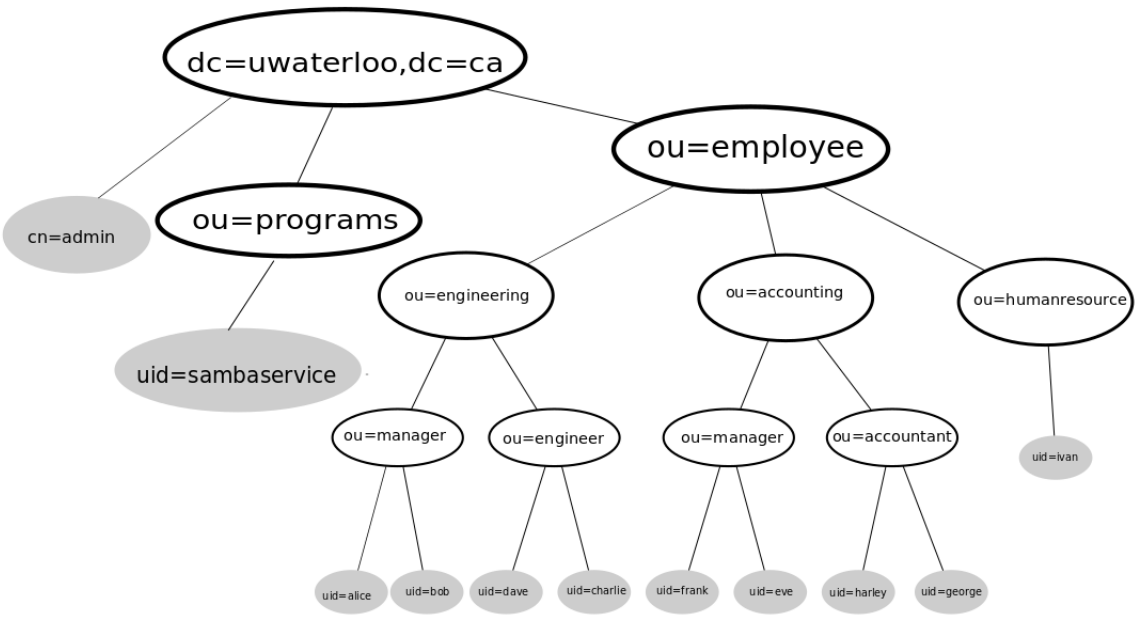

Figure 3.1: LDAP tree for the study

3.2 Syntax

The syntax used in this study is a scaled down version of the full syntax for OpenLDAP. The following is the reference sheet giving to the participants to show the restricted syntax.

```
syntax of each acl rule

 to RESOURCE by PRINCIPAL PERMISSION
     [by PRINCIPAL PERMISSION]
     ...
```

```
     [by PRINCIPAL PERMISSION]
     [by * break]
 to RESOURCE by PRINCIPAL PERMISSION
     ...

RESOURCE :
    dn.subtree="domain name" [attrs="attribute name","attribute name"]
    * [attrs="attribute name","attribute name"]


PRINCIPAL :
    dn.subtree="domain name"
    *
    users
    anonymous
    self


PERMISSION :
    none
    auth
    read
    write


The order of the rules in an access control list matters.
```

In this restricted syntax, only "dn.subtree" and the special character $\star$ are used as a way to specify the resource. This syntax also include the attribute list use to specify the set of attributes for the resource group. The principal selector also uses the "dn.subtree" syntax and four other possible keywords : $\star$, user, anonymous and self. The restricted syntax uses only the default "`stop`" control statement except we give the option of adding "by * break'' at the end of the directive to change the precedence rule. The simpler $\langle level \rangle$ syntax for permissions are used. This simplified syntax contains most source of difficulties in OpenLDAP access control system, while not overloading the participants with complexity like more detailed ways to specify the resource and principal and additional ways we can modify how the precedence rule works with more control statements.

3.3 Potential difficulties in the task

To write the OpenLDAP policy that encodes the policy from the first section, a participant would first need to identify the principal, resource and permission that would satisfy each case mentioned in the policy goal, then they would try to write down a set of directives that accomplish that policy goal, preferable in as few directives as possible. This step is repeated for all 7 policy goals. After finishing the previous step for all the policy goals, the participant puts all the directives together into a single list and then check the ordering of the directives which means going through each policy goal and trying to figure out if one of the directive conflict with another. This involves checking if the resource in one directive intersect another and then if it the principal intersect. If such intersection exist the participant needs to try to figure out if with the current order of the directives in the access control list is correct or some kind of fix is necessary, such as changing the ordering of the directives or decompose one directive into multiple different directives that cover different part of the resource and principal. Another task during this process is to try to merge multiple directives into one directive in order to simplify the access control list. The merge is possible when the two directive have the same resource.

3.4 Training a participant

The participants were first taught the scaled down version of the syntax by going through the syntax reference sheet which explains what each expression means. After, they were given the following examples.

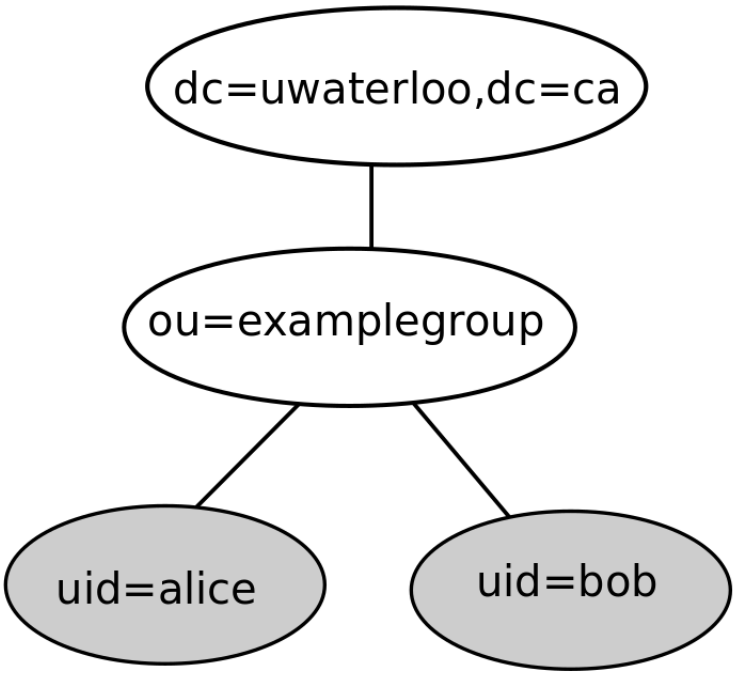

Figure 3.2: Example of the LDAP tree used for the study

The following is the policy that all the examples want to accomplish

1. Alice has write permission to herself
2. Bob has write permission to himself
3. Alice and bob has read permission to each other.

Example 1:

```
1: to dn.subtree="uid=alice,ou=examplegroup,dc=uwaterloo,dc=ca"
2:    by dn.subtree="ou=examplegroup,dc=uwaterloo,dc=ca" read
3:    by dn.subtree="uid=alice,ou=examplegroup,dc=uwaterloo,dc=ca" write
4: to dn.subtree="uid=bob,ou=examplegroup,dc=uwaterloo,dc=ca"
5:    by dn.subtree="ou=examplegroup,dc=uwaterloo,dc=ca" read
6:    by dn.subtree="uid=bob,ou=examplegroup,dc=uwaterloo,dc=ca" write
```

Example 2:

```
7:  to dn.subtree="uid=alice,ou=examplegroup,dc=uwaterloo,dc=ca"
8:     by dn.subtree="uid=alice,ou=examplegroup,dc=uwaterloo,dc=ca" write
9:     by dn.subtree="ou=examplegroup,dc=uwaterloo,dc=ca" read
10: to dn.subtree="uid=bob,ou=examplegroup,dc=uwaterloo,dc=ca"
11:    by dn.subtree="uid=bob,ou=examplegroup,dc=uwaterloo,dc=ca" write
12:    by dn.subtree="ou=examplegroup,dc=uwaterloo,dc=ca" read
```

The first two example has the same strategy of using one directive for when Alice is the resource and another directive for when Bob is the resource. For each directive there are two subdirectives one to give `read` permission to each other and the other to give `write` permission to themselves. The difference between the examples is the ordering of the subdirectives. Example 1 fails to accomplish the policy, as Alice and Bob does not have permission to write to themselves. If we look at the case where Alice tries write to herself. The resource selector at line 1 and the principal selector at line 2 would match that access control check, thus the reference monitor would return at line 2 with `read` permission which means that Alice can not write to herself. The same thing would happen for Bob with line 4 and line 5. Example 2 fixes this problem by changing the ordering of the subdirectives so that the first subdirectives are for writing to themselves and the second subdirectives for reading each other. This ordering prevent the issue encounter at examples 1. If we look at the case where Alice write to herself then line 7 and line 8 would match and give the corresponding `write` permission. The case where Bob reads Alice would match line 7 and line 9 and gives the correct permission. line 8 does not interfere with this case since the principal selector would not match.

Example 3:

```
13: to dn.subtree="ou=examplegroup,dc=uwaterloo,dc=ca"
14:    by dn.subtree="ou=examplegroup,dc=uwaterloo,dc=ca" read
15: to dn.subtree="uid=alice,ou=examplegroup,dc=uwaterloo,dc=ca"
16:    by dn.subtree="uid=alice,ou=examplegroup,dc=uwaterloo,dc=ca" write
17: to dn.subtree="uid=bob,ou=examplegroup,dc=uwaterloo,dc=ca"
18:    by dn.subtree="uid=bob,ou=examplegroup,dc=uwaterloo,dc=ca" write
```

Example 4:

```
19: to dn.subtree="uid=alice,ou=examplegroup,dc=uwaterloo,dc=ca"
20:    by dn.subtree="uid=alice,ou=examplegroup,dc=uwaterloo,dc=ca" write
21: to dn.subtree="uid=bob,ou=examplegroup,dc=uwaterloo,dc=ca"
22:    by dn.subtree="uid=bob,ou=examplegroup,dc=uwaterloo,dc=ca" write
23: to dn.subtree="ou=examplegroup,dc=uwaterloo,dc=ca"
24:    by dn.subtree="ou=examplegroup,dc=uwaterloo,dc=ca" read
```

Example 5:

```
25: to dn.subtree="uid=alice,ou=examplegroup,dc=uwaterloo,dc=ca"
26:    by dn.subtree="uid=alice,ou=examplegroup,dc=uwaterloo,dc=ca" write
27:    by * break
28: to dn.subtree="uid=bob,ou=examplegroup,dc=uwaterloo,dc=ca"
29:    by dn.subtree="uid=bob,ou=examplegroup,dc=uwaterloo,dc=ca" write
30:    by * break
31: to dn.subtree="ou=examplegroup,dc=uwaterloo,dc=ca"
32:    by dn.subtree="ou=examplegroup,dc=uwaterloo,dc=ca" read
```

The following examples use the same strategy of matching a single directive for each policy goal. Example 3 fails because of the ordering of the directives. The goal of Alice writing to herself is supposed to be fulfill by directive at line 15 and the goal of Bob writing to himself is supposed to be fulfill by directive at line 17. However, for both these cases, the reference monitor will return after line 14 because the resource selector at line 13 and the principal selector at line 14 would also match all these cases and thus the reference monitor gives `read` permission. Therefore example 3 fail to accomplish the policy.

Example 4 tries to fix the mistake in example 3 by reorder the directives. However, If we look at the case where Alice tries to read Bob the reference monitor will return after line 22 with the default subdirective. The resource selector at line 21 matches Bob which is the case we looking at, although the principal selector at line 22 does not match the principle, we need to remember the added default subdirective with a principal selector that matches everything thus the reference monitor would return after line 22 with `none` permission for Alice tries to read Bob. The same thing happens when Bob tries to read Alice the reference monitor will return after line 20, thus example 4 also fails to accomplish the policy.

Example 5 fixes the problem with example 4 by adding the "by * break" statements. For the case when Alice tries to read Bob, although line 28's resource selector would still match. Line 30's control statement will tell the reference monitor to continue looking at

the following directives thus the reference monitor will return at line 32 and give `read` permission. The same happens for Bob trying to read Alice with line 27.

Example 6:

```
33: to dn.subtree="ou=examplegroup,dc=uwaterloo,dc=ca"
34:    by self write
35:    by dn.subtree="ou=examplegroup,dc=uwaterloo,dc=ca" read
```

The last example teaches the participant how to use the keyword self and how to merge two directives into one. Line 33 and Line 34 gives Alice and Bob write permission to themselves as this leverage the power of the keyword self which is when the principal and resource are the same node, thus line 34 would only match when Alice and Bob tries to write to themselves. Line 25 would catch the other case which is When Alice and Bob would try to read each other.

After the examples we proceed into an exercise where the goal is change to

1. Alice has write permission to herself
2. Bob has write permission to himself
3. Alice and bob has read permission to each other's display name but not userpassword

This exercise requires the participants to use the attribute list in the resource selector since we need to select only the displayname attribute not the userpassword attribute.

3.5 Ethics Clearance

We applied for, and received clearance, from the Office of Research Ethics at the University of Waterloo. Our study requires such clearance because it involves human subjects. We determined there are minimal risk to the participants since the task only involve them sitting in front of the computer and typing a few commands. The sessions are record but their information are kept anonymized to preserve their privacy. This information is also kept secure in the primary investigators office.

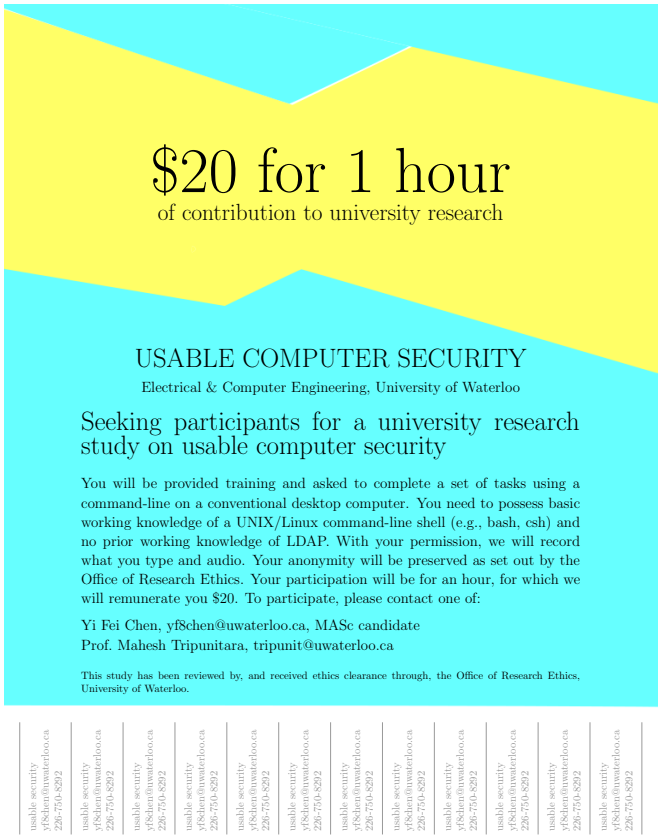

Figure 3.3: Recruitment poster used for the study

During the recruitment process we only require the participants to be familiar with UNIX command line. Since we want the participants to be familiar with tree structure and some notion of access control and it would be difficult to ask people if they have some knowledge of access control. However that is a ambiguous question, thus we ask for familiarity in with UNIX since the UNIX file system is one examples of access control system. Anyone familiar with the UNIX command should also be able to navigate through a tree structure. We assume the participants who fulfill the requirement are very similar in technical knowledge to junior system administrators.

Information Letter for Usability of openldap Access Control Lists (ACLs)

You are invited to participate in a research study conducted by Yi Fei Chen, under the supervision of Prof. Mahesh Tripunitara of the University of Waterloo, Canada. The objectives of the research study are to assess the usability of openlda Access Control Lists (ACLs). The study is likely to be part of an MASc thesis.

If you decide to volunteer, you will be asked to complete a 1 hour session. 15 minutes will be spent on training, and the remainder on some tasks we ask you to perform. With your permission, we will perform an audio recording of your session for later analysis and collection of data, such as how long you took to complete a particular task, whether you appeared to have difficulty and for us to ensure that we accurately record the results.

We require participants to have some working knowledge of a Unix/Linux shell (e.g., bash, csh), and no prior knowledge of ldap. Participation in this study is voluntary. You may decline to answer any questions that you do not wish to answer and you can withdraw your participation at any time. There are no known or anticipated risk from participating in this study. Any information about you will be kept confidential. All of the data will be aggregated and no individual will be identifiable from the aggregated results. The data, with no personal identifiers, collected from this study will be maintained on a password-protected computer database in a restricted access area of the University. As well, the data will be electronically archived after completion of the study and maintained for two years and then erased. Should you have any questions about the study, please contact either Yi Fei Chen (yf8chen@uwaterloo.ca) or Mahesh Tripunitara (tripunit@uwaterloo.ca). Further, if you would like to receive a copy of the results of this study, please contact either investigator.

I would like to assure you that this study has been reviewed and received ethics clearance through the Office of Research Ethics at the University of Waterloo ORE#40009. However, the final decision about participation is yours. If you have any comments or concerns resulting from your participation in this study, please contact the Office of Research Ethics, at 1-519-888-4567 ext. 36005 or ore-ceo@uwaterloo.ca.

Figure 3.4: The consent letter informs the participants what they are doing during the study and they are recorded with a screen capture and audio recording.

CONSENT FORM

By signing this consent form, you are not waiving your legal rights or releasing the investigator(s) or involved institution(s) from their legal and professional responsibilities.
Title of the study:Usability of OpenLDAP access control
I have read the information presented in the information letter about a study being conducted by Yi Fei Chen, under the supervision of Mahesh Tripunitara of the Department of Electrical and Computer Engineering at the University of Waterloo. I have had the opportunity to ask any questions related to this study, to receive satisfactory answers to my questions, and any additional details I wanted.
I am aware than my session is going to be audio recorded to ensure an accurate recording of my responses and for later analysis and data collection.
I was informed that I may withdraw my consent at before and during the study without penalty by advising the researcher. However, I will not be able to request withdrawal of the data after study participation is completed. This project has been reviewed by, and received ethics clearance through, the Office of Research Ethics at the University of Waterloo. I was informed that if I have any comments or concerns resulting from my participation in this study, I may contact the Director, Office of Research Ethics at 1-519-888-4567 ext. 36005 or ore-ceo@uwaterloo.ca.

☐ I am aware than my session is going to be screen captured and audio recorded in order document any mistakes the participant makes for later analysis and data collection.

☐ I give permission for the use of anonymous quotations in any thesis or publication that comes from this research.

I agree of my own free will to participate in the study.

Participants name: _____

Participants signature: _____ Date _____

Researchers/Witness signature _____ Date _____

Figure 3.5: The consent form just ask them to acknowledge that they have given their permission for the recordings.

3.6 Limitations

Our study has some limitations, which we discuss in this section. One limitation for the study, it that we assume that the directory information tree is static. A dynamic tree could create more difficulties, but then we would need to constraint the study to the most likely changes such as adding user or removing a user from a group. There could also be more drastic changes like adding new attribute to every user.

Another aspect to consider is that the task of the study is focused on writing OpenLDAP policy base on a high-level policy. However the opposite case could also occur like when we need to update the system but was unable to find the original high level policy. In this case we would need to figure out the original high-level policy base on the current OpenLDAP policy that is in the system.

Chapter 4

Data analysis

There were a total of 54 people who participate in the study.

The first four participants were treated as pilot participants, in that they were used to fine-tune the manner in which we conducted the study for the remainder of the participants. Two of them accomplished the task without too much difficulty. One of them made the mistake of switching principal and resource in the syntax and another had difficulty with the self keyword. Thus the training script was changed to explicitly mention these issues as mistakes to avoid.

The actual study was conducted on 50 participants. However two of the participants had a lot of difficulty with the task. Both of them gave up halfway through the session and did not finish the task. Thus those two submissions are discarded. Out of the remaining 48 submissions only 20.8% got everything correct with +/-11.5% for the 95 percent confidence interval. Since the high-level policy is given in English there could be some interpretation difference. More specifically, for goal 5 "managers of engineering and accounting have write permission to all their users". The term "their users " could have two different interpretations. If we look back at the directory tree the engineering department has two sub groups engineers and mangers. The accounting department also have two subgroups accountants and managers. One interpretation of the high-level policy is that "their users " means everyone in the department including the managers. The other interpretation is that "their users" only include the workers of the department but not the managers. For the correctness both of these interpretation are considered valid.

During the study, we note down the time it takes for each participant to accomplish the task. The timer starts after introducing the high-level policy and showing the participants the directory tree. The timer stops after we confirmed with the participants that they feel

that they did the best they could. We always the participants the option of taking as much time as they want. After they finish the task, we also ask them how confident they are that their OpenLDAP policy accomplishes the goals listed in the high-level policy.

The average time it takes to accomplish the task is 25.73 minutes with standard deviation of 6.84 minutes. The point-biserial correlation between time taken and correctness is only 0.105 which is very low. This is not very surprising since the participants know that there are 30 minutes allocated for the task and most people use all the time provided. The participants was also ask for how well do they think they did between 0 to 100 percent. The average of the self reporting is 71.10 percent with a standard deviation of 14.17%. The point-biserial correlation between self reported correctness and actual correctness is also very low at 0.0275. The low correlation tells us the participants are not certain if they made any mistakes in their submission.

Each submission is also evaluated based on the if the submission accomplish each individual policy goal. There are 7 high-level policy goals, 1 point is giving if their OpenLDAP policy fulfills the requirement for every possible combination of principal and resource mentioned in that policy goal. There is a extra point awarded for checking that the submission does not give extra permission to principal and resource pair not mentioned in the high-level policy. Since a single syntax mistake can causes the whole submission to fail in the eyes of the system. We fix the following simple syntax mistakes: missing quotation marks, extra quotation mark/bracket and spacing issue. These issue are fixed because with a robust error detection tool, these error should be obvious to anyone writing the policy. They appear in the study since the tool used did not have a very good error message system to show where the syntax errors are. For all other syntax mistakes, we remove all directives that cause a syntax issue and evaluate the left over directives as is. Based on those criteria, the following table shows the number of participants that accomplish each goal in the high-level policy.

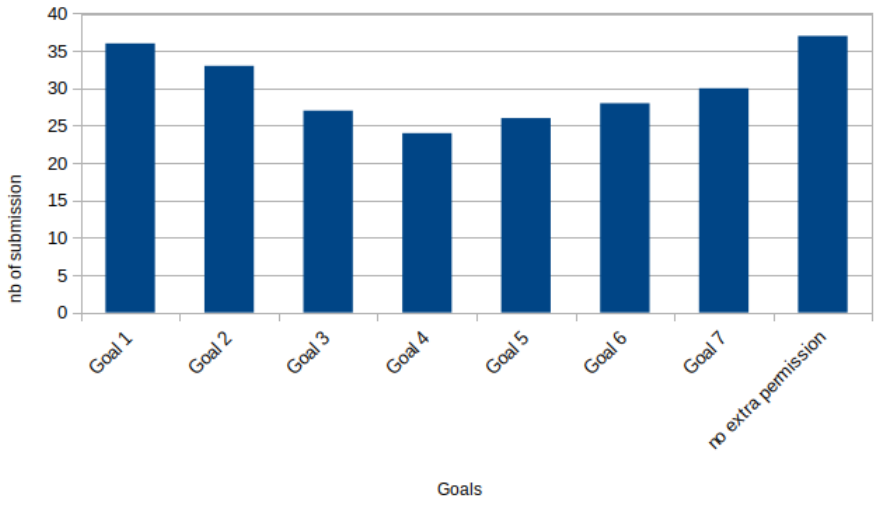

Figure 4.1: Number of submission that accomplish each goal

We also compiled to see how many goals is accomplish by each submission. The following table shows what number of submission scored how many points.

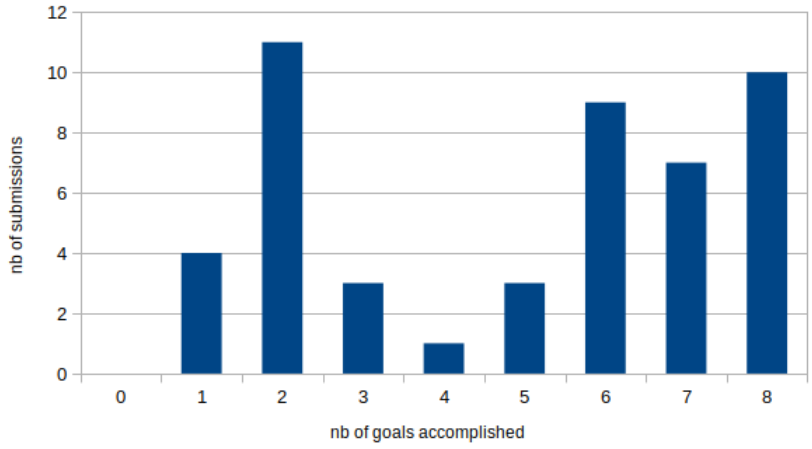

Figure 4.2: Number of submission vs number of goals accomplished

We can see that there are actually lot of people who got six and seven points and there is

	Scale Mean if Item Deleted	Scale Variance if Item Deleted	Corrected Item Total Correla-tion	Cronbach's Alpha if Item Deleted
policy1	4.19	5.47	0.38	0.81
policy2	4.25	5.55	0.30	0.83
policy3	4.40	4.50	0.79	0.75
policy4	4.46	4.72	0.67	0.77
policy5	4.42	4.72	0.67	0.77
policy6	4.37	5.05	0.51	0.80
policy7	4.31	4.86	0.62	0.78
no extra permission	4.17	5.63	0.31	0.82

Table 4.1: Reliability analysis

another group that only got a one or two right. This seems to imply that there are two groups in the participants. One group that understood most of OpenLDAP access control system and the other that has lots of difficulty with it. However this can be misleading, because of the precedence rule one directive could influence all directives after it. Thus one submission that only got one or two points could be just that they made mistake on one control statement that derail all the other directives after it. Half of the submissions that only got around 2 points is because they did not use "by * break" after the first directive which was necessary in order to not interfere with the other directives. Another reason that there are a big group of people who only got a few right is that they made the mistake of confusing where the principal and the resource should be placed in the syntax. Thus they fail lots of policy goals. Interestingly for those that got confused about the placement of principal and resource in the syntax. For directives that uses keywords such as self or anonymous, they would write the directive with the correct syntax. This is probable due to the fact that the reference sheet would remind them that these keywords are the principal and where it should be place in the syntax. While when the principal and resource both use "dn.subtree" syntax, they would get confused.

Using reliability test we can see how difficult each policy goal is and which one are more correlated with the correctness of the submission. The goal that is least correlated with correctness should be the most difficult to accomplish.

The first and second policies are the least correlated with successful, since there are more

submissions that accomplishes these goal. The majority of the submissions follow the order that the goal are list in the high-level policy so the directive that accomplish the first goals are usually the first directives in the list and since the precedence rule favours the directives that appears first. These directive are less likely to be affected by any precedence issue , thus higher rate of success. The "no extra permission point" is also not very correlated. The participants are fairly conservative when writing permission very few people actually try to write a directive that can potentially gives extra permissions and then use the precedence rule to restricted it. The submission that gives extra permission are all caused because they misunderstood how the syntax works.
The 3rd goal is most correlated with success followed by goal 4 and 5. These goals contain errors that participants are more likely to make. One source of error comes from the fact that if the submission give the employee read permission to each other appear before directives that try to accomplish goal 3 and 5, the directive for goal 3 can interfere with the other directives that accomplish goal 3 and 5. For goal 4, a common error is that when the high-level policy refer to the resource as all userpasswords the participants gets confused and only use the employees userpasswords as resource. In their mind only the employees are the users for this case. Interestingly goal 6, has the exact same resource but less participants made the same mistake on it. Some participants that got goal 4 wrong got goal 6 correct.

Chapter 5

Recommendations

There were many participants that made the mistake of switching the principal and resource, thus we should change the syntax so that the principal appear first and then the resource. This matches more closely to the English sentence structure of subject-verb-object. The principal corresponds to the subject and the resource corresponds to object. This new syntax would feel more natural to English speaker as the ordering of the principal and resource in the OpenLDAP syntax would match the order in the English language.

Another recommendation would be to change the default subdirective in order to minimize the need for control statement. Using the current OpenLDAP syntax two directives that has the same resource selector would cause conflict even if the principal selector does not intersect. If we change the default sub directive so that if the resource selector intersect but no principal selector intersect then the reference monitor would continue looking at other directives without need to add a break statement at the end. In the current syntax, if we want to minimize the use of control statements we need to merge as many directive as possible that targets the same resource. If this change is implemented then we only need to consider the precedence rule, when the resource selector and the principal selector both intersect with another directive instead of current syntax where only the resource selector needs to intersect.

5.1 Future work

The previous recommendations focused on the more obvious mistakes made by a lot of participants, however the problem of ordering the directives does appear to affect some of the participants. Solutions to fix this problem would require more work.

Like other access control system such as svnauthz for the SVN version control system, there is a tool in OpenLDAP that can verify for a specific pair of principal and resource what kind of access is allowed. The tool is called slapacl. The inputs to it are a DN that serve as the principal, another DN for the resource, an attribute name and the kind of access permission that is being checked. The tool returns "allowed" or "denied" to tell us is the current access succeed or not with the current OpenLDAP policy. This tool was not used in the study. If introduced for the participants that were confused about the syntax, the tool can clarify what the current OpenLDAP policy is doing. However, this tool can only be of limited help since it is restricted to a specific principal and resource at a time, so not a group of principal and resource. For example, if we look at the task in the study there are more than 400 possible pairs of principal and resource. This tool does not single-handedly solve the problem of ordering the directives, since looking through each pair will be a very time consuming process and potential very tedious. There are many similar pairs of principal and resource that have the same permission, thus it is still up to the participants to find the edge cases. A better auditing tool would be one that can figure out which directives are actually conflicting with each other and states all the conflicting cases in a succinct way. It should point out the directive that has precedence and propose some possible alternative if the precedence is not working as what the user has in mind.

Another possible solution to the sequencing issue is to make a completely new tool that uses a different conceptual model for the access control policy which avoids the need of a precedence rule by being internal consistent. The new model would use a new syntax based on declarative language which describe the end result with out caring about the details. This new syntax would still need to model the access relationship between resources and principals. It should also still uses the tree structure of the nodes to group them together. After devising such a model, we can make a tool that converts any policy in the new syntax into an OpenLDAP policy and then conduct a study to verify if the new model is indeed easier to user than the current OpenLDAP model.

Chapter 6

Related work

Our work relates usability with access control. While some instance it can be argued that there is a trade off in system with usability and security. Such as the papers by Raz et al [6] and Feth [9] that tries to find the balance between security and usability.

However, this view is only valid when looking at normal users for the system. If we look at the experiences of the system administrator, then usability and security goes hand in hand. As explained by Saltzer and Schroeder [18]. They explain the principal of Psychological Acceptability which is that human interface must be easy to use. The user must have a correct mental image of the goals and the mechanism of the system. In the instance they user they are talking about is the administrator.

There are multiple papers testing different systems to see if users does indeed have correct mental image. Raja et al [17] showed "Most of the participants did not have a useful mental model of of firewalls". Felt et al [8] show that Android users do not understand the permission system. This lack of understanding could lead to security venerability. Xu et al [28] show that security misconfigurations happen because user does not fully understand the system and use trial and error to configure it.Tan et al [24] showed users are more likely to accept security prompt when there is a brief explanation even if they don't really understand it.

In order to increase the usability there are paper that focus on the kind of policies that people want and implement. Smetters et al [21] examine how uses share their files and notice that they prefer sharing sharing the whole folder rather than configure each file individually. Motiee et al [14] also examine the policy of users for sharing files and see if they follow the principal of least privilege. They showed Windows users do not follow the principal of least privilege.

There are multiple aspect to configure the security policy. The following paper concentrate on the step of how security requirement becomes a high level policy and how we can improve this step. Werlinger et al [27] examine how security administrator communicates with different stake holder in the context of security. Karp et al [12] presents some guideline to make access control part of the user work flow thus easier to configure and less likely to make mistakes. Another asepct that is being investigate is the different model for security we can have. More specifically how the different syntax models the security requirements Olson et al [15] propose a different model for access control syntax that is more expressive and based on the semantics of Transaction Datalog. Beckerle et al [3] propose some metrics to evaluate the usability of access control policy.

These model of secuirty can be easily translate from one to another as the paper by Barkley [1] show an role based model can be implemented using access control list with group mechanism.

There are other research on increasing the usability of the access control of specific system with a new tool. Such as Xiang Cao et al's [7] who designed a wizard that walks though the decision process of configuring access control for WebDAV. Ueno et al [25] propose a what you see is what you get access control interface for mashups. Schreuders et al [20] propose a alternative for Linux program security mechanism that can help user with limited knowledge about security. There is also the work of Reeder et al's [26] who also investigate the conflict resolution of access control system for windows file system and arrives at the conclusion that the conflict resolution based on "the specificity-based method provides substantial usability gains for tasks that require a policy author to make changes to a default decision issued by the conflict-resolution method." Which in fact is what OpenLDAP does when the control statement is the default.

Bibliography

[1] John Barkley. "Comparing Simple Role Based Access Control Models and Access Control Lists". In: *Proceedings of the Second ACM Workshop on Role-based Access Control*. RBAC '97. Fairfax, Virginia, USA: ACM, 1997, pp. 127–132. ISBN: 0-89791-985-8. DOI: 10.1145/266741.266769. URL: http://doi.acm.org/10.1145/266741.266769.

[2] Lujo Bauer et al. "Real Life Challenges in Access-control Management". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '09. Boston, MA, USA: ACM, 2009, pp. 899–908. ISBN: 978-1-60558-246-7. DOI: 10.1145/1518701.1518838. URL: http://doi.acm.org/10.1145/1518701.1518838.

[3] Matthias Beckerle and Leonardo A. Martucci. "Formal Definitions for Usable Access Control Rule Sets from Goals to Metrics". In: *Proceedings of the Ninth Symposium on Usable Privacy and Security*. SOUPS '13. Newcastle, United Kingdom: ACM, 2013, 2:1–2:11. ISBN: 978-1-4503-2319-2. DOI: 10.1145/2501604.2501606. URL: http://doi.acm.org/10.1145/2501604.2501606.

[4] Ingo Bente. *keeping your sanity while designing openldap acls*. medium. 2015. URL: https://medium.com/@moep/keeping-your-sanity-while-designing-openldap-acls-9132068ed55c.

[5] P. Bera, S. K. Ghosh, and P. Dasgupta. "Policy Based Security Analysis in Enterprise Networks: A Formal Approach". In: *IEEE Transactions on Network and Service Management* 7.4 (2010), pp. 231–243. ISSN: 1932-4537. DOI: 10.1109/TNSM.2010.1012.0365.

[6] Christina Braz, Ahmed Seffah, and David M'Raihi. "Designing a Trade-off Between Usability and Security: A Metrics Based-model". In: *Proceedings of the 11th IFIP TC 13 International Conference on Human-computer Interaction - Volume Part II*. INTERACT'07. Rio de Janeiro, Brazil: Springer-Verlag, 2007, pp. 114–126. ISBN: 3-540-74799-0, 978-3-540-74799-4. URL: http://dl.acm.org/citation.cfm?id=1778331.1778344.

[7] Xiang Cao and Lee Iverson. "Intentional Access Management: Making Access Control Usable for End-users". In: *Proceedings of the Second Symposium on Usable Privacy and Security*. SOUPS '06. Pittsburgh, Pennsylvania, USA: ACM, 2006, pp. 20–31. ISBN: 1-59593-448-0. DOI: 10.1145/1143120.1143124. URL: http://doi.acm.org/10.1145/1143120.1143124.

[8] Adrienne Porter Felt et al. "Android Permissions: User Attention, Comprehension, and Behavior". In: *Proceedings of the Eighth Symposium on Usable Privacy and Security*. SOUPS '12. Washington, D.C.: ACM, 2012, 3:1–3:14. ISBN: 978-1-4503-1532-6. DOI: 10.1145/2335356.2335360. URL: http://doi.acm.org/10.1145/2335356.2335360.

[9] Denis Feth. "User-centric Security: Optimization of the Security-usability Trade-off". In: *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ESEC/FSE 2015. Bergamo, Italy: ACM, 2015, pp. 1034–1037. ISBN: 978-1-4503-3675-8. DOI: 10.1145/2786805.2803195. URL: http://doi.acm.org.proxy.lib.uwaterloo.ca/10.1145/2786805.2803195.

[10] R. Harrison. *Lightweight Directory Access Protocol (LDAP): Authentication Methods and Security Mechanisms*. RFC 4513. RFC Editor, 2006. URL: https://tools.ietf.org/html/rfc4513.

[11] Maritza L. Johnson et al. "Laissez-faire File Sharing: Access Control Designed for Individuals at the Endpoints". In: *Proceedings of the 2009 Workshop on New Security Paradigms Workshop*. NSPW '09. Oxford, United Kingdom: ACM, 2009, pp. 1–10. ISBN: 978-1-60558-845-2. DOI: 10.1145/1719030.1719032. URL: http://doi.acm.org/10.1145/1719030.1719032.

[12] Alan H. Karp and Marc Stiegler. "Making Policy Decisions Disappear into the User's Workflow". In: *CHI '10 Extended Abstracts on Human Factors in Computing Systems*. CHI EA '10. Atlanta, Georgia, USA: ACM, 2010, pp. 3247–3252. ISBN: 978-1-60558-930-5. DOI: 10.1145/1753846.1753966. URL: http://doi.acm.org/10.1145/1753846.1753966.

[13] Microsoft. *Active Directory Domain Services Overview*. Microsoft. 2019. URL: https://docs.microsoft.com/en-us/windows-server/identity/ad-ds/get-started/virtual-dc/active-directory-domain-services-overview.

[14] Sara Motiee, Kirstie Hawkey, and Konstantin Beznosov. "Do Windows Users Follow the Principle of Least Privilege?: Investigating User Account Control Practices". In: *Proceedings of the Sixth Symposium on Usable Privacy and Security*. SOUPS '10. Redmond, Washington, USA: ACM, 2010, 1:1–1:13. ISBN: 978-1-4503-0264-7. DOI:

10.1145/1837110.1837112. URL: http://doi.acm.org/10.1145/1837110.1837112.

[15] Lars E. Olson, Carl A. Gunter, and P. Madhusudan. "A Formal Framework for Reflective Database Access Control Policies". In: *Proceedings of the 15th ACM Conference on Computer and Communications Security*. CCS '08. Alexandria, Virginia, USA: ACM, 2008, pp. 289–298. ISBN: 978-1-59593-810-7. DOI: 10.1145/1455770.1455808. URL: http://doi.acm.org/10.1145/1455770.1455808.

[16] OpenLDAP. *OpenLDAP Software 2.4 Administrator's Guide*. OpenLDAP Foundation. 2011. URL: https://www.openldap.org/doc/admin24/access-control.html.

[17] Fahimeh Raja et al. "It's Too Complicated, So I Turned It off!: Expectations, Perceptions, and Misconceptions of Personal Firewalls". In: *Proceedings of the 3rd ACM Workshop on Assurable and Usable Security Configuration*. SafeConfig '10. Chicago, Illinois, USA: ACM, 2010, pp. 53–62. ISBN: 978-1-4503-0093-3. DOI: 10.1145/1866898.1866907. URL: http://doi.acm.org/10.1145/1866898.1866907.

[18] J. H. Saltzer and M. D. Schroeder. "The protection of information in computer systems". In: *Proceedings of the IEEE* 63.9 (1975), pp. 1278–1308. ISSN: 0018-9219. DOI: 10.1109/PROC.1975.9939.

[19] R. S. Sandhu and P. Samarati. "Access control: principle and practice". In: *IEEE Communications Magazine* 32.9 (1994), pp. 40–48. ISSN: 0163-6804. DOI: 10.1109/35.312842.

[20] Z. Cliffe Schreuders, Tanya McGill, and Christian Payne. "Empowering End Users to Confine Their Own Applications: The Results of a Usability Study Comparing SELinux, AppArmor, and FBAC-LSM". In: *ACM Trans. Inf. Syst. Secur.* 14.2 (Sept. 2011), 19:1–19:28. ISSN: 1094-9224. DOI: 10.1145/2019599.2019604. URL: http://doi.acm.org/10.1145/2019599.2019604.

[21] D K Smetters and Nathan Good. "How Users Use Access Control". In: *Proceedings of the 5th Symposium on Usable Privacy and Security*. SOUPS '09. Mountain View, California, USA: ACM, 2009, 15:1–15:12. ISBN: 978-1-60558-736-3. DOI: 10.1145/1572532.1572552. URL: http://doi.acm.org/10.1145/1572532.1572552.

[22] M. Smith. *Lightweight Directory Access Protocol (LDAP): String Representation of Search Filters*. RFC 4515. RFC Editor, 2006. URL: http://www.rfc-editor.org/rfc/rfc4515.txt.

[23] stanford. *OpenLDAP ACL Examples*. stanford. 2015. URL: https://uit.stanford.edu/service/directory/aclexamples.

[24] Joshua Tan et al. “The Effect of Developer-specified Explanations for Permission Requests on Smartphone User Behavior”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI ’14. Toronto, Ontario, Canada: ACM, 2014, pp. 91–100. ISBN: 978-1-4503-2473-1. DOI: 10.1145/2556288.2557400. URL: http://doi.acm.org/10.1145/2556288.2557400.

[25] Nachi Ueno et al. “Soramame: What You See is What You Control Access Control User Interface”. In: *Proceedings of the Symposium on Computer Human Interaction for the Management of Information Technology*. CHiMiT ’09. Baltimore, Maryland: ACM, 2009, 5:38–5:41. ISBN: 978-1-60558-572-7. DOI: 10.1145/1641587.1641592. URL: http://doi.acm.org/10.1145/1641587.1641592.

[26] Robert W. Reeder et al. “More than skin deep: Measuring effects of the underlying model on access-control system usability”. In: May 2011, pp. 2065–2074. DOI: 10.1145/1978942.1979243.

[27] Rodrigo Werlinger et al. “Security Practitioners in Context: Their Activities and Interactions with Other Stakeholders Within Organizations”. In: *Int. J. Hum.-Comput. Stud.* 67.7 (July 2009), pp. 584–606. ISSN: 1071-5819. DOI: 10.1016/j.ijhcs.2009.03.002. URL: http://dx.doi.org/10.1016/j.ijhcs.2009.03.002.

[28] Tianyin Xu et al. “How Do System Administrators Resolve Access-Denied Issues in the Real World?” In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. CHI ’17. Denver, Colorado, USA: ACM, 2017, pp. 348–361. ISBN: 978-1-4503-4655-9. DOI: 10.1145/3025453.3025999. URL: http://doi.acm.org/10.1145/3025453.3025999.

[29] zytrax. *OpenLDAP Samples*. 2015. URL: http://www.zytrax.com/books/ldap/ch6/index.html#ex-parts.