VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

GENERAL NOTIFICATION SYSTEM FOR FREEIPA

SEMESTRÁLNÍ PROJEKT
TERM PROJECT

AUTOR PRÁCE   Bc. PETR KUBÁT
AUTHOR

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

OBECNÝ NOTIFIKAČNÍ SYSTÉM PRO PROJEKT FREEIPA
GENERAL NOTIFICATION SYSTEM FOR FREEIPA

SEMESTRÁLNÍ PROJEKT
TERM PROJECT

AUTOR PRÁCE   Bc. PETR KUBÁT
AUTHOR

VEDOUCÍ PRÁCE   Mgr. ADAM ROGALEWICZ, Ph.D.
SUPERVISOR

BRNO 2015

Abstrakt

Tato práce se zabýva problematikou zasílání oznámení v řešeních pro správu identit. Čtenáři jsou představeny dva takové sytémy, FreeIPA a Active Directory, se zaměřením na existující mechanismy zasílání notifikací. Zároveň je také ukázána první verze návrhu systému zasílání oznámení pro použití v projektu FreeIPA

Abstract

This thesis describes the issue of notification sending in identity management solutions. The reader is introduced to the FreeIPA and Active Directory projects, with focus on existing notification mechanisms. The author also presents an early look on the design of a notification system to be used in FreeIPA.

Klíčová slova

Keywords

Citace

General Notification System for FreeIPA

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Mgr. Adama Rogalewicze, Ph.D.

. . . .
Petr Kubát
March 20, 2016

Poděkování

Rád bych poděkoval hlavně panu Petru Špačkovi za jeho trpělivost při odborném vedení práce.

Contents

Chapter 1

Introduction

Ever since the humans inabited the Earth, one of the most important things was to know who they're dealing with using the identity of their partner. Before the spoken word, their identity mainly consisted of how they looked, sounded or maybe even smelled to others. With the development of languages came names, like "John Smith" or "Maria Garcia" which, given the tendecy of early populations being small, were sufficient as a form of identity.

However, with the dawn of the industrial age people would be moving to cities, creating larger and larger populations and with the creation of factories, employing an increasing number of people, it was important to somehow store and manage their identities and duties. At first, this was done using paper documents stored somewhere in the factory, but this solution proved to be increasingly difficult as sorting and searching through these documents had to be done manually and this took a lot of time.

With the advent of computers, this problem was solved by storing the identities of employees into databases or directory servers, from which they could be retrieved easily if needed. As the factories turned into companies, spanning multiple buildings, countries, or even continents, and computers becoming smaller, more powerful and generally available, it was important to manage stored identities and control which parts of the system the company's employees could access given their role using information management software. As a single unwanted change in this identity management could even prove fatal to the whole network, one of the most important parts a identity management system are the notifications of such changes. These notifications can be then used to respond to the event by the administrator or, more importantly, another automated system.

The main topic of this thesis are these notification systems and mechanisms in identity management software, specifically their implementation in the FreeIPA project focusing largely on interopability with third party software. Chapter 2 will be introducing the reader to the FreeIPA project and its components, while chapter 3 compares FreeIPA to Microsoft's Active Directory with emphasis on notification mechanisms. Chapter 4 will be looking at the proposed use cases for the notification system and, last but not least, chapter 5 will tackle the issues of notification sending function placement and method of transfer, as well as providing an early design of the system.

Chapter 2

FreeIPA

FreeIPA (where IPA stands for Identity, Policy and Audit) is an open-source security management solution sponsored by Red Hat aimed primarily at Linux and Unix machines [12].

The project itself combines a number of various existing open-source technologies to achieve the goal of providing centralized authentication and authorization, as well as storing important account information like users or group memberships. FreeIPA also aims to provide easy management and setup of a domain controller which would otherwise be very difficult by using the same components on your own.

This chapter will briefly introduce some of the components FreeIPA uses and describe the architecture of the resulting FreeIPA server solution, including the posibilities of extending the project.

2.1 Directory Server

FreeIPA's directory service is the foundation of the project as it stores various information on behalf of all of FreeIPA's components. It also plays a big role in authentication and authorization using Kerberos which will be presented in the next section.

The LDAP protocol [11] is used as a mean of communication with the server and the data itself is stored in a Directory Information Tree (DIT) which is a tree-like data structure. An example of a DIT structure can be seen in figure 2.1.
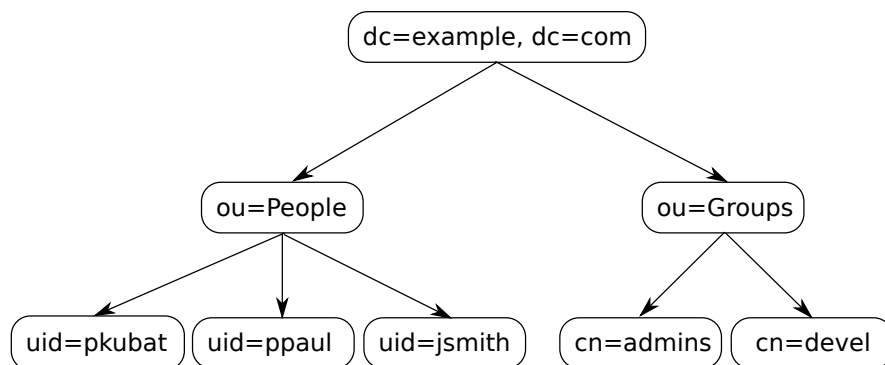

Figure 2.1: LDAP Directory Information Tree

LDAP provides several operations to use with the server [11]:

- **add, delete, modify:** These operations add, remove and modify the data contained in the DIT.

- **search, compare:** The search and compare operations are used in querying the DIT for specific information.

- **bind, unbind, abandon:** These operations can be used to authenticate to the directory, terminating the connection or abandoning a previously sent request entirely, respectively.

- **extended operations:** New operations that are not a part of the original protocol.

The actual LDAP compatible server contained in FreeIPA is implemented using the 389 Directory Server project [4].

2.2 Kerberos

Kerberos [10] is a network authentication protocol that uses symmetric encryption using a pre-shared key to authenticate a client to a network service (and vice versa) via an insecure connection using a trusted third party service called a Key Distribution Center (KDC). The resulting communication is secure because no secret keys are transported over the network in plaintext format as the KDC already contains a database of credentials for users and services in the Kerberos realm.
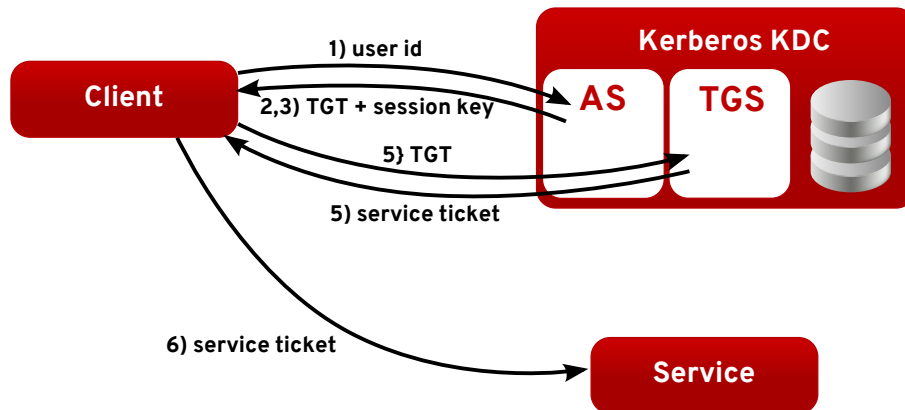
Figure 2.2: Kerberos authentication process

The process of authenticating the user to a network service is shown in figure 2.2 and can be described in these steps:

1. The user sends his principal name (an unique identifier) to the KDC's Authentication Server (AS) via a plaintext request.

2. The AS then checks the database to make sure the user exists and sends back a randomly generated session key to be used to encrypt communication with another service called a Ticket-Granting Service (TGS) encrypted with the user's secret key.

3. The AS also generates a set of credentials called a Ticket-Granting Ticket (TGT) which includes the previously generated session key and is encrypted by the secret key of the TGS.

4. After recieving the first message the client decrypts it using his secret key. This is the only time the user's key is actually used. The TGT which the client can't decrypt himself is saved in a cache on the client's side to be used later to setup a session with the TGS. At this point the user is authenticated to the Kerberos realm and doesn't have to input his secret key again for a set amount of time (commonly 10-24 hours).

5. When the user wants to authenticate agains a service in the Kerberos realm he just has to ask the TGS to send him a ticket. This request is encrypted using the session key created in previous two steps.

6. The user then authenticates to the chosen service using this ticket without the need for his secret key.

As the security of the Kerberos protocol is partly based on the time stamps of tickets, all of the clients and services in the realm have to be properly synchronized time-wise. To achieve this goal the Network Time Protocol can be used in the FreeIPA project.
FreeIPA's KDC is implemented using the MIT Kerberos [8] open source software and FreeIPA also provides its own KDC data backend called ipa-kdb which is used to both read and write user information to FreeIPA's LDAP directory service [13].

2.3   DNS

Even though it would be possible to access network services located in a FreeIPA domain directly using their IP addresses, it is much more easier to do so using domain names.
The Domain Name System (DNS) [9] is distributed naming system, that translates domain names, which can be easily memorized by humans, into IP addresses using special name servers. As such if one wants to access a network service or a webpage he doesn't have to remember its IP address, only the IP address of the name server (which is stored localy on the client machine) and the domain name of the service/webpage.
The domain name space resembles a tree structure, each node having a label that designates a part of its domain name, while the full domain name of the node can be built by concatenating this label with the domain name of its parent node.
The name space is divided into zones starting at the root of the tree structure with child nodes of the root node called top-level domains (TLD).
These zones can contain one or several domains, each domain served by one or several name servers, and can be divided into additional zones if deemed necessary.
The DNS server in FreeIPA uses an enhanced BIND name server which allows FreeIPA to store data into an LDAP directory [14]. However using FreeIPA's integrated DNS server is optional and as such the project can be used with a different third party DNS server if so desired.
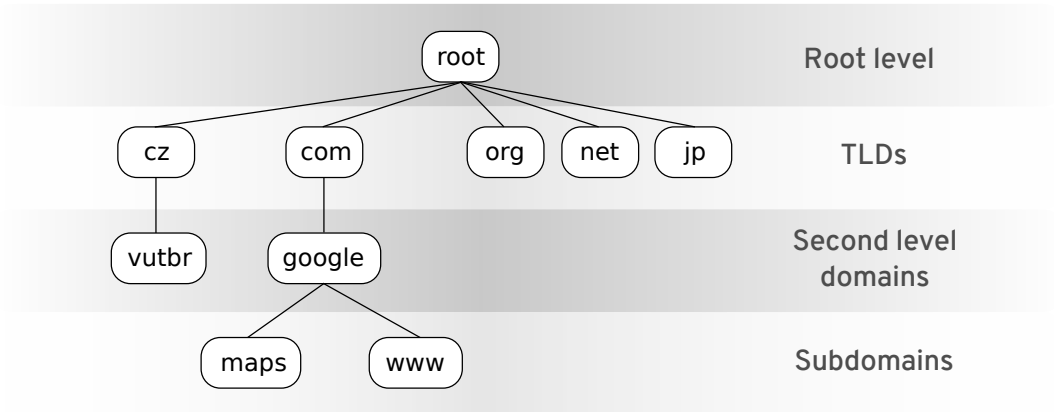
Figure 2.3: Domain name space

2.4 FreeIPA Architecture

While the most important components of the FreeIPA project have been described in previous sections, a number of components are still missing for the infrastructure to be complete.
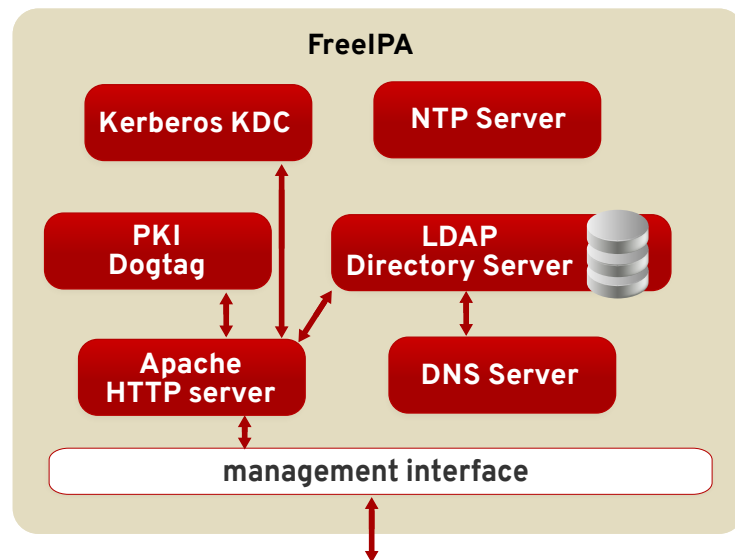

Figure 2.4: High level FreeIPA server architecture

To start with, FreeIPA can optionally use the **Dogtag project** [3] as an integrated Public Key Infrastructure that signs and publishes certificates for hosts and services inside the domain.

Secondly, the **Apache Web Server** is used to provide web based access to the management APIs using the XML-RPC and JSON-RPC APIs as well as to serve the web interface to the clients.

And last but not least, as one of the most important additions to the project, FreeIPA's **ipalib** framework is used together with the Command Line and Web based interfaces to administer the FreeIPA domain.

A high level and a detailed representation of the resulting infrastructure can be found in figures 2.4 and 2.5, respectively.
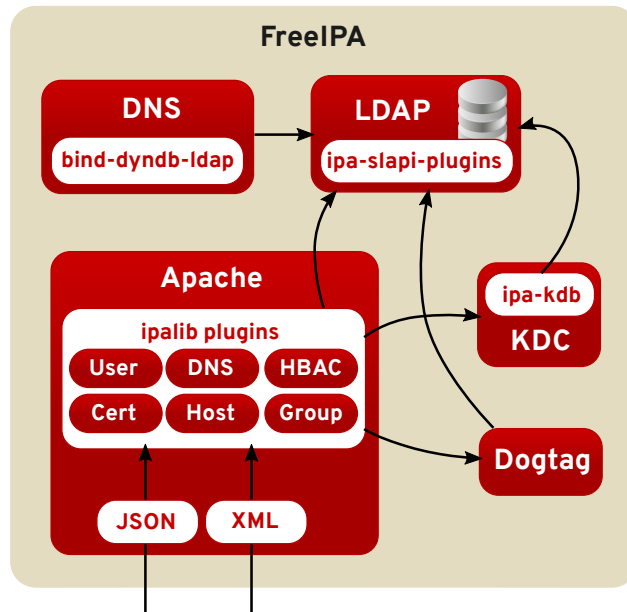
Figure 2.5: Detailed look at FreeIPA architecture

The ipalib framework is written using the Python programming language and is highly modular – most if not all of its functionality is implemented using plug-ins, which are executed on demand using the Apache Server. This framework is generally used to connect the isolated components into the FreeIPA solution, providing the ability to enroll users or services into the domain, managing the certificates used within the domain or adding Host Based Access Control rules for even beter access management.

2.5 Extending FreeIPA

2.5.1 Extending the Framework

As previously stated, the ipalib framework is highly modular and extensible with a couple of different ways to add new (or modify current) functionality [2]:
First off one can directly extend existing objects of the ipalib framework. These objects are mainly responsible for executing various commands sent using the XML and JSON APIs and as such extending or modifying the objects, eg. adding a parameter or rewriting an object's label, can alter the default behaviour of those commands.
The second way is to extend methods of an object stored in the LDAP database by adding callbacks to these methods. Callbacks are user-defined functions that are called at various stages of exectution of the method:

Pre callback

    This callback is called before executing the method's code. Used for modifying arguments and data validation.

Post callback

    This callback is called after executing the method's code and allows for analyzing the result of the command.

Exc callback

This callback is called in case there is an error during the execution of the method's code. Can be used to recover from the error.

Interactive callback

Allows a command to decide if additional parameters should be requested from the user.

2.5.2 Extending the Directory Server

The 389 DS used in the FreeIPA project has two ways of creating extensions for its operations, first one achieved by writing a server plug-in, while the second one uses LDAP's Content Synchronization Operation protocol.

Server Plug-ins

There are several types of plug-ins that can be used when trying to extend the directory server [15]:

Pre operation

A pre operation plug-in is executed before starting the LDAP operation. Mostly used for data validation.

Post operation

A post operation plug-in is executed after performing the LDAP operation. Can be used for notifications.

Entry storage and fetch

Executed before writing and reading data from the database, respectively. An example of this type of plug-in would be encryption/decryption of data saved in the database.

Extended operation

Executed when the client calls an extended operation.

Syntax

Run when getting a list of candidates for a search. Can be used to modify comparison operations.

Matching rule

Run when the client sends a request with an extensible matching search filter.

One can see that there are some similarities to callbacks of ipalib's extensions, especially the way of calling pre and post operations plug-ins is basically the same as pre and post callbacks differing only in when they are called. While pre and post callbacks are bound to a specific method of a specific command object (eg. before or after executing a user-add command), the pre and post operations plug-ins are bound to an LDAP operation and as such are run much more frequently.

Content Synchronization Operation

The Content Synchronization Operation (syncrepl) is an extension of the existing search operation provided by LDAP and is mostly used, as its name implies, as a mean of directory content synchronization across different applications. The operation can be configured to provide a client with changes made to a subtree of a DIT and can be used similarly to 389DS's pre/post operations without the need to extend the whole add/modify/delete operation. As syncrepl extends the search operation the client can use its content control attributes to specify the content changes it wants to recieve, namely base DN and search scope to control the scope of the operation and filter to further specify which entries' changes the client is interested in.

After using the syncrepl operation (with refreshAndPersist mode) the client is sent initial content located in the DIT after which the server will only send notifications about changes within the tracked content. Updates in the DIT can generate add, modify and delete result messages according to the changes made and upon recieving these changes the client has to determine the differences by himself as the server will send all of the entry's attributes on its modification (in case of add or modify state change) or none at all (in case of delete state change).

Chapter 3

Active Directory

3.1 Structure and Components

This section is based on the official documentation for Active Directory (AD) [7].

AD is a distributed directory service used by the Microsoft Windows Server operating systems and, while similar in the components used, is quite different to the FreeIPA project. Objects located in a AD organized network are structured into a logical structure, the basis of which form forests and domains.

Forests are the security boundaries of the logical structure and can be structured to provide data and service autonomy and isolation, removing the dependency on physical topology.

Each forest is able to hold several **domains**. These domains can be structured to provide data and service autonomy but not isolation as to allow for replication optimalization. This logical structure can also be used to control access to data for each domain seperately.
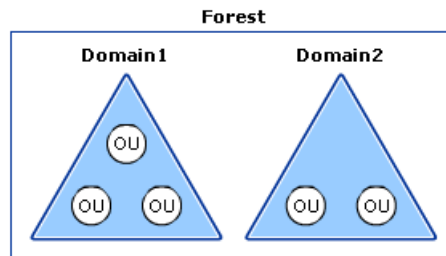

Figure 3.1: An Active Directory forest and its domains [7]

Each domain is connected to one or more **domain controllers**. A domain controller is a service that stores directory data and manages user and domain interactions. Each domain controller consists of a **schema**, which defines objects and attributes that can be store in the directory, a **data store**, which manages the storing and retrieving of data, and the actual **database**. The data store uses a number of interfaces to access stored data as can be seen in figure 3.2, one of which is the same protocol as the 389DS uses in FreeIPA – LDAP.

Active Directory also uses, similarly to FreeIPA, DNS as its domain controller location mechanism and Kerberos for user authentication.
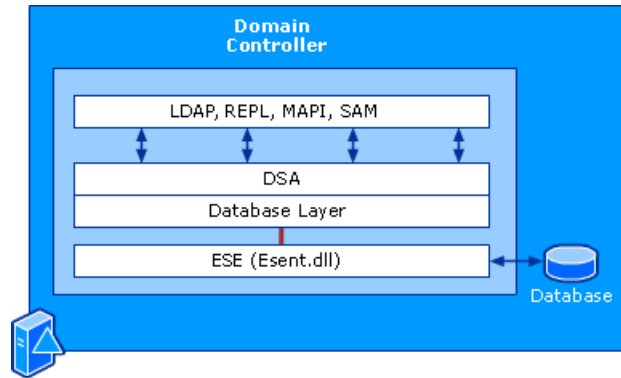
Figure 3.2: Active Directory Domain Controller [7]

3.2 Notification Mechanisms

As AD does not by itself provide a complete notification system, this section describes mechanisms contained in AD and Windows operating systems in general that can be used to send notifications between applications.
The following mechanisms are available for users of AD directory service:

Change Notifications
> This mechanism allows for a client to register with a domain controller to recieve notifications for changes in the AD domain service. The notification control is specified in a LDAP asynchronous search operation.
> The user can define a number of search parameters: scope, to monitor just the object or its immediate children, filter, which allows for object filtering and attributes, a list of the changed object's attributes that will be returned [6].

Polling
> The second notification mechanism available in AD is directly polling the AD directory service for changes. There are two types of polling available. Polling using Directory Synchronisation control, which is an LDAP server extension that allows to check for changes since a previous state, while the other type – querying using a special uSNChanged attribute – is used when Directory Synchronisation is unavailable or inefficient [6].

These mechanisms are, however, primarily used to mantain consistency of data between the AD directory service and other applications [6]. As such there is one additional mechanism that can be used not only with AD, but generally with any application running on a Windows operating system and that is **Event Tracing**(ET) [5].
ET is a mechanism that allows for application defined events to be logged into a log file and optionally recieved by other applications in real time. ET has three major components:**controllers**, that control the ET sessions, **providers** that write events into a ET session and **consumers** which consume one or more event sessions either directly or from a log file.

11

Chapter 4

Use Case Analysis

In this chapter the author analyses potential use cases of the notification system based on the research of both Active Directory and FreeIPA projects.

4.1 Active Directory Use Cases

As mentioned in section 3.2 notification systems integrated in Active Directory (change notifications and polling) have generally only one use and that is **keeping data integrity** between applications that use a copy of data stored in the directory server and the actual directory server. Even though this use case is not the primary use of FreeIPA's notification system, similar behaviour could be easily introduced by sending changed data with every generated notification.

When it comes to the general event tracing mechanism of the Microsoft Windows Server, it is mainly used for **logging purposes** by a myriad of applications running on the server. One of the most interesting of these logs is the **security event log**. This session logs various security related issues such as every successful/failed login attempts, group membership changes or failed access attempts to secured objects.

Additionaly, while researching Active Directory's notification mechanisms a specific use case for user written scripts came up repeatedly and that is **password expiry notifications**. As AD notifies the user of this fact only as he tries to log in with his expired password, said scripts monitored the password expiry time by using a polling mechanism for the expiry attribute and notified the user directly via email. As such notification on password expiration should be one of the primary use cases of FreeIPA's notification system. However, as being notified on every login/logout or group membership change would be unnecessary and maybe even bothersome, the system should instead notify on an action with possibly malicious intent, eg. **repeated login attempts** or the **addition of a user into a highly priviledged group** (admins, sales, etc.).

4.2 FreeIPA Use Cases

This section describes the use cases as suggested by FreeIPA's community.

Some of the use cases proposed by the community in the project's specification intersect with use cases resulting from the analysis of Active Directory's use cases (user password expiration, group membership changes and repeating login attempts) in the previous section. Use cases not yet mentioned are as follows:

User enrollment, deletion, edit

User enrollment (and generally user-based operations) are one of the most frequent and thus important events in a identitz management software. As such a notifications of such changes sent possibly via email could be beneficial to some groups (HR, sales). Additionaly, using this notification system, an automated system could help setup, or delete, user related environments in the system (eg. home folders) on user creation or deletion, respectively.

Certificate creation/expiry

As the Certmonger client-side daemon already takes care of monitoring certificate expiry and optionally their renewal, this use case would be mainly about sending a notice to the concerned groups about the impending situation. However, in environments not running Certmonger on the client machines, it could be used to connect to another automated renewal system.

Chapter 5

System Design

In this chapter the author tackles one of the most difficult questions to answer – from where and through what means FreeIPA's notifications should be sent, resulting in the basis of a design for the actual notifiaction system.

5.1 Sender Placement

Given the current extensibility of the project as described in section 2.5, there are three possible general placements that can be used:

1. Notifications sent directly from the 389DS would allow for the notifications to be delivered even when using an LDAP client different from the FreeIPA client which is apparently quite common. However, adding code directly to the directory server (using the plug-in system) might cause drops in performance.

2. Notification senders located in the ipalib framework would allow for more precise calls of notification sending methods and possibly better performance. This approach would also provide the user with the ability to send notifications that would not be limited to changes in the 389DS.

3. A hybrid approach that combines the previous two approaches and thus adds senders to both the 389DS and the ipalib framework for different operations depending on the severity of hits on performance.

While there are no ready-made recievers contained in Active Directory that could help with the choice of approach, it still offers valuable information on what properties the receivers (and senders) should have:

Generalization
All of AD's available notification mechanisms use a general user-defined notification reciever. This allows the user to have control over how the notifications will be processed and responded to.

Variety
AD offers the user a number of different ways of sending notifications. As such the user can decide if he wants to recieve notifications any time an attribute is changed, or if he wants to check for changes himself periodically.

Given the analysis of AD's notification mechanisms, the author currently opts for the **hybrid approach** with a receiver defined by the user. However, further testing will be needed to see if this approach is possible when it comes to hits in performance as a result of introducing the notification plug-ins.

5.2   Method of Transfer

There are many different protocols that can be used for the actual notification transfer, like the Advanced Message Queuing Protocol or Fedora's own Fedora Infrastructure Message Bus, however the **D-Bus message system** has been chosen by the author as the method of transfer.

The D-Bus message system is a communication mechanism between processes running on the same host that is present on nearly any Linux operating system, which is one of the reason thats make it so attractive to be used as the transport medium for notifications in FreeIPA. The architecture of the D-Bus system is quite similar to Microsoft's Event Tracing, with the difference of D-Bus not logging sent messages in a file. The architecture consists of **clients**, who are each sending and recieving messages and a **dbus daemon** running the actual bus. D-Bus messages are transported over the bus by the dbus daemon, to which any number of processes can be connected at any given time [1].

D-Bus also has APIs (bindings) written in a number of languages, including the **C and Python languages** used by 389DS and ipalib framework, respectively, for both the sending and recieving side. Together with the bus being able to send messages from multiple sources, to multiple recievers, this allows D-Bus to be used with any of previously mentioned sender placements, including the hybrid one.

There were some concerns that by using D-Bus, any change to the data sent to the recievers would mean the need to rewrite the reciever to accomodate to the changes made. However, using the Python bindings and its argument type discovery function, it should be possible for the reciever to work even after the changes have been introduced to the senders. In case the reciever is written in another lower-level binding, the reciever will have to be editted accordingly, but that is to be expected.

To accomodate for some of the use cases proposed in chapter 4, a **lightweight monitoring daemon** will designed and implemented to be used to check the expiry date of users' passwords and certificates and send notifications to the bus when necessary.

Chapter 6

Conclusion

The aim of this thesis was to introduce the reader to the FreeIPA identity management solution, its components and management framework, and its need for a notification system that would allow for a better administration. The FreeIPA project was then compared to another identity management system, for which Microsoft's Active Directory was selected, with focus on the analysis of its notification mechanisms and cases in which they are used. This has however proven to be quite a difficult task as the uses of Active Directory's notifications were found to be quite limited.

Additionaly an early look on the coming design of FreeIPA's notification was presented, mainly focusing on placement of notification sending parts of the system and the way of transport of those notifications. As part of describing the design decisions behind the transport of notifications, the D-Bus message system was introduced to the reader.

With this task complete, the author will now discuss the results presented in this thesis with the FreeIPA community, choose which use cases to use in the final system based on their feedback and after that focus on completing its design.

Another very important task is the testing of effects of various sender placements on the performance of FreeIPA and its components, as described in chapter 5. This testing will be worked upon in tandem with the design of the notification sending part of the system and is expected to have a strong influence on the way the system will be designed.

Bibliography

[1] dbus. [online], [cit. 2015-12-12].
http://www.freedesktop.org/wiki/Software/dbus/.

[2] Alexander Bokovoy. Extending FreeIPA. [online], 2011 [cit. 2015-12-12].
https://abbra.fedorapeople.org/freeipa-extensibility.pdf.

[3] Dogtag. Dogtag Certificate System. [online], [cit. 2015-12-12].
http://pki.fedoraproject.org/.

[4] Red Hat. 389 Directory Server. [online], [cit. 2015-12-12].
http://directory.fedoraproject.org/.

[5] Microsoft. Event Tracing. [online], [cit. 2015-12-12].
https://msdn.microsoft.com/en-us/library/bb968803%28v=vs.85%29.aspx.

[6] Microsoft. Tracking Changes. [online], [cit. 2015-12-12].
https://msdn.microsoft.com/en-us/library/ms677974%28v=vs.85%29.aspx.

[7] Microsoft. Active Directory Collection. [online], [cit. 2016-01-06].
https://technet.microsoft.com/en-us/library/cc780036%28v=ws.10%29.aspx.

[8] MIT. Kerberos: The Network Authentication Protocol. [online], [cit. 2015-12-12].
http://web.mit.edu/kerberos/.

[9] P. Mockapetris. Domain Names - Implementation and Specification, RFC 1035.
[online], November 1987 [cit. 2015-12-27]. https://tools.ietf.org/html/rfc1035.

[10] C. Neuman, T. Yu, S. Hartman, and K. Raeburn. The Kerberos Network
Authentication Service (V5), RFC 4120. [online], July 2005 [cit. 2015-12-25].
https://tools.ietf.org/html/rfc4120.

[11] J. Sermersheim. Lightweight Directory Access Protocol (LDAP): The Protocol, RFC
4511. [online], June 2006 [cit. 2015-12-12]. https://tools.ietf.org/html/rfc4511.

[12] The FreeIPA Team. About FreeIPA. [online], [cit. 2015-12-12].
http://www.freeipa.org/page/About.

[13] The FreeIPA Team. Kerberos. [online], [cit. 2015-12-25].
https://www.freeipa.org/page/Kerberos.

[14] The FreeIPA Team. DNS. [online], [cit. 2015-12-27].
https://www.freeipa.org/page/DNS.

[15] T. Čapek and E. D. Ballard. Red Hat Directory Server 10 Plug-in Guide. [online], June 2015 [cit. 2016-01-01]. https://access.redhat.com/documentation/en-US/ Red_Hat_Directory_Server/10/html/Plug-in_Guide/.