Network Switch a Centralized Access Approach

Abhay Kumar Singh

Abstract— This work gives a mechanism for doing authentication and authorization between managed element and server from a single database using a Centralized controller which can control a multiple switches. This work allows having one or more authentication servers for the switches to authenticate against which centralizes the authentication databases, making it easier to manage switch. Moreover, switch continues to support the pre-existing local authentication which works as a fallback in case of loss of connectivity to authentication server. Command authorization on per user basis is added which makes possible to have authorization of user to execute specific commands. Old access level authorization is continued to support as well. Protocol client is added and integrated into the existing system. As a part of this this work Remote authentication is supported meaning that authentication has not to be done by each switch by its own. Authentication database is shared with each other by switches now. Therefore each switch need not to be configured individually for a specific user and password in the network which will make the process of adding/modifying users very fast as opposed to time consuming in a large switch network and it is no more a security concern also. Chances of misconfiguration and mismatch are minimized.

Key words: AAA, API, Authentication, Authorization, C, Database, Ftp, NAS, Session, Switch, Telnet, SSH.

I. INTRODUCTION

The Telecommunication switch which is working or deployed in the field is based on the control plane and data plane architecture and it is a vendor specific. Each vendor has its proprietary control plane and data plane architecture which cannot be changed by the end user. If any modification we need inside the switch control plane or data plane we need to report to the vendor so the vendor modify the things specific to the request and provide a firmware/software image to the end user and then end user install this on the router/switch to make the service availability. We have seen in the field that many requests from the customer or service provider is mainly with the data control side. Now in this architecture the problem is how to manage all the switches with a single controller because the telecom Switch can be accessed using telnet, ssh, ftp and sftp methods with appropriate username and password. Various kinds of configurations and modifications can be done to existing configuration of the switch. Configurations can be changed or added by commands on command line interface.

Many users can be added giving them different access rights and grouping them into different groups permitting or limiting their rights to access the switch and to execute the commands. User can be deleted or added and passwords can be changed.

Problem is that if one user is deleted or added or password is changed then this process has to be repeated on every switch in the network. And if network is large with more than 500 switch with 100 users then this is process is lengthy and time consuming. There can be chances of miss-configuration like password mismatch and it can be security concern also. Numbers of switches and users might grow depending upon the network.

II. PROBLEM STATEMENT

Currently most of the switches, supports authentication which is purely local, meaning that authentication is done by each switch by its own. Authentication database is not shared by switch with each other; therefore each switch has to be individually configured for a specific user and password in the network.

Moreover, currently few command privilege levels can be assigned to a user to authorization. These levels are mapped to few command groups. It is not possible to have authorization of a user to execute specific commands. So switch supports only access level authorization. It is possible to do remote authentication and authorization so that this process can be centralized. More than one authentication sever can be configured to provide redundancy also. An option can be to have a controller switch either this controller can be a switch or a window machine may be our personal computer which can control a group of switches. By using this mechanism for authentication and authorization we can have client server architecture.

III. CURRENT IMPLEMENTATION

User interface commands lets add a user account to the local database. The account includes a user name, access level, and password. The hierarchy of access levels in order of greatest privilege to least privilege is:

- ADMIN
- ACCESS_LEVEL1 for potentially dangerous configuration or complex troubleshooting commands
- ACCESS_LEVEL2 for node-level, service-affecting commands
- ACCESS_LEVEL3 for basic service configuration
- ANYUSER for display commands and basic user commands.

The privilege of the user that is being added must be less than the user level at which command for adding user is being run. For example, if you are logged in at ACCESS_LEVEL2 level, you can create user accounts with ACCESS_LEVEL1 or ANYUSER privilege. The commands available to a particular user have either the same or a lesser privilege level than that of the user-account.

With SERVICE access, for example, commands can be used that require ACCESS_LEVEL2, ACCESS_LEVEL3, or ANYUSER privilege.

Manuscript received on April, 2013.

Abhay Kumar Singh, M.Tech (Pursuing) , Department of Computer Science, JAMIA HAMDARD, New Delhi

Supervised by: Suraiya Praveen, Assistant Professor, Department of Computer Science, JAMIA HAMDARD, New Delhi

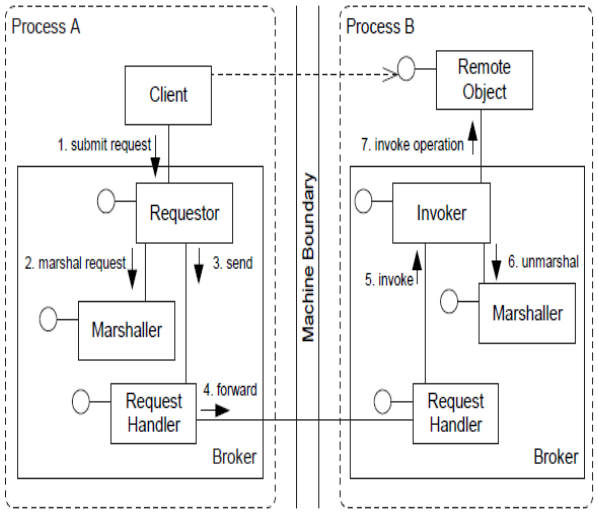
Currently command adds a user and its access level to the local database only which is local authentication and authorization. There is a limit on maximum number of user accounts on a switch.

IV. ARCHITECTURE OVERVIEW

A broker component is responsible for coordinating communication, such as forwarding requests as well as for transmitting results and exceptions. The Broker architectural pattern can be used to structure distributed software systems with decoupled components that interact by remote service invocations.

Broker component achieves better decoupling of clients and servers. Servers register themselves with the broker, and make their services available to clients through method interfaces. Clients access the functionality of servers by sending requests via the broker. A broker's tasks include locating the appropriate server, forwarding the request to the server and transmitting results and exceptions back to the client. By using the Broker pattern, an application can access distributed services simply by sending message calls to the appropriate object instead of focusing on low-level inter-process communication. In addition, the Broker architecture is flexible, in that it allows dynamic change, addition, deletion, and relocation of objects. The Broker pattern reduces the complexity involved in developing distributed applications, because it makes distribution transparent to the developer. It achieves this goal by introducing an object model in which distributed services are encapsulated within objects. Broker systems therefore offer a path to the integration of two core technologies: distribution and object technology.They also extend object models from single applications to distributed applications consisting of decoupled components that can run on heterogeneous machines and that can be written in different programming languages

A. Broker Architecture Pattern



The Broker architectural pattern has some important benefits:

Location Transparency: As the broker is responsible for locating a server by using a unique identifier, clients do not need to know where servers are located. Similarly, servers do not care about the location of calling clients, as they receive all requests from the local broker component.

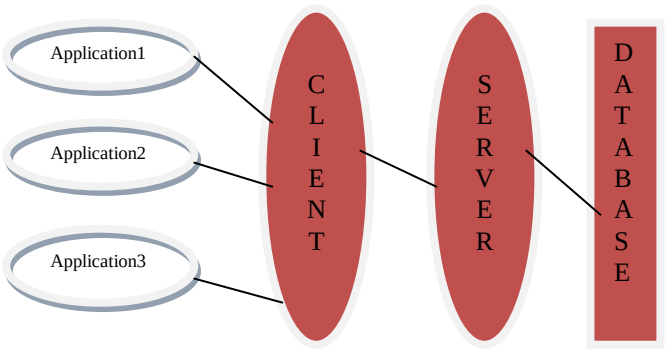
Changeability and extensibility of components: If servers change but their interfaces remain the same, it has no functional impact on clients. Modifying the internal implementation of the broker, but not the APIs it provides, has no effect on clients and servers other than performance changes. Changes in the communication mechanisms used for the interaction between servers and the broker, between clients and the broker, and between brokers may require you to recompile clients, servers or brokers. However, you will not need to change their source code. Using proxies and bridges is an important reason for the ease with which changes can be implemented.

Portability of a Broker system: The Broker system hides operating system and network system details from clients and servers by using indirection layers such as APIs, proxies and bridges. When porting is required, it is therefore sufficient in most cases to port the broker component and its APIs to a new platform and to recompile clients and servers. Structuring the broker component into layers is recommended, for example according to the Layers architectural pattern. If the lower-most layers hide system-specific details from the rest of the broker, you only need to port these lower-most layers, instead of completely porting the broker component.

Interoperability between different Broker systems: Different Broker systems may interoperate if they understand a common protocol for the exchange of messages. This protocol is implemented and handled by bridges, which are responsible for translating the broker-specific protocol into the common protocol, and vice versa.

Reusability: When building new client applications, you can often base the functionality of your application on existing services. Suppose you are going to develop a new business application. If components that offer services such as text editing, visualization, printing, database access or spreadsheets are already available, you do not need to implement these services yourself. It may instead be sufficient to integrate these services into your applications

V. SWITCH CLIENT AS BROKER



VI. RESULT AND ANALYSIS

The result of this work is to provide functionality to:

- Have a single user authentication database via the client Server Architecture.
- Allow customizable per user command authorization.

In the current software, the Switch's authentication is purely local, meaning that each switch does its own authentication. However,

the Switches do not share their authentication databases with each other; therefore each switch in the network has to be individually configured for a specific user and password. The implementation of the client server architecture will allow the users to configure one or more authentication servers for the switches to authenticate against. This then centralizes the authentication databases, making it easier for the end users to manage their switches. Moreover, while the current software has five (5) command privilege levels: admin, ACCESS_LEVEL3, ACCESS_LEVEL2, ACCESS_LEVEL1 and ANYUSER, that can be assigned to a user to authorize command groups.

The software does not allow the users to configure authorization of a user to execute specific commands. The implementation of the client server architecture protocol will allow the users to configure command authorization on a per user basis, as well as offer the historical privilege level authorization

REFERENCES

1. Avgeriou Paris, Uwe Zdun. Architectural patterns revisited: a pattern language. 10th European Conference on Pattern Languages of Programs (EuroPlop 2005); July 2005; Irsael, Germany.
2. Buschmann F, Meunier R, Rohnert H, Sommerlad P and Stal M. Pattern-Oriented Software Architecture: A System of Patterns. Chichester: John Wiley & Sons, 1996
3. Kernighan, Dennis M. Ritchie. The C Programming Language. Englewood Cliffs, NJ:Prentice Hall,1988
4. Manual. VxWorks Programmer's Guide 5.3.1. Alameda, CA: Wind River Systems Inc, 4 may 1998
5. Alex Berson. Client Server Architecture McGraw-Hill, 1996
6. Douglas Comer. Interworking with TCP/IP: Prentice Hall 2006
7. Forouzan : TCP/IP protocol suite: 2nd Edition : Tata Mcgraw Hills
8. James Rumbaugh,Ivar Jacobson,Grady Booch :The Unified Modeling LanguageReference Manual : second Edition : Pearson Education
9. Erich Gamma, Richard Helm, Ralph Johnson :Design Patterns: Elements of Reusable Object-Oriented Software : Pearson Education, 1-Oct-1994
10. Joshua Kerievsky: Refactoring to Patterns:Pearson Education, 05-Aug-2004