FIREWALL POLICY MANAGEMENT USING SECURE WEB SYSTEM

Daniel GHEORGHICĂ[1], Victor CROITORU[2]

Tehnologiile firewall au cunoscut un progres impresionant în ultimii ani, prezentând o creştere semnificativă a numărului de implementări. În schimb, tehnologiile de management al securităţii şi al firewall-urilor nu au înregistrat aceeaşi dezvoltare.

În acest context, prezenta lucrare propune un sistem securizat de tip web pentru managementul politicilor firewall. Sistemul propus permite managementul centralizat al politicilor oricărui firewall accesibil de la distanţă prin intermediul SSH (Secure Shell). Prezenta lucrare abordează procesul de administrare şi configurare centralizată a firewall-urilor ceea ce conduce la organizarea şi simplificarea acestui proces, în special în cazul implementărilor cu diverse tehnologii firewall.

In recent years firewalls have seen some impressive technological advances (e.g. stateful inspection, transparency, performance, etc.) and wide-spread deployment. In contrast, firewall and security management technology is lacking. In this paper we propose a secure web system for firewall policy management.

Our secure system implementation is able to manage any firewall products that support SSH remote management. We believe that our approach is an important step towards streamlining the process of configuring and managing firewalls, especially in complex, multi-firewall installations.

Keywords: firewall, security, management interface, GUI, PHP, UML

1. Introduction

Over the past years, there has been an accelerating growth in the use of network based services. Because of the ubiquity of computer systems and the Internet, this growth will continue. Corresponding to this continued growth in network usage, there will be increases in malicious user activity. The threat of malicious users is becoming more serious every day. Terms such as phishing, pharming, SPAM and identity theft make their way into news headlines all the time. Firewall is a widely deployed mechanism for improving the security of enterprise networks.

[1] Ph.D. Student, Faculty of Electronics, Telecommunications and Information Technology, University POLITEHNICA of Bucharest, Romania, e-mail: gheodan@gmail.com
[2] Professor, Ph.D., Faculty of Electronics, Telecommunications and Information Technology, University POLITEHNICA of Bucharest, Romania, e-mail: croitoru@adcomm.pub.ro

This paper proposes a secure system for firewalls management. The system is designed to allow management of all firewalls from a security zone, and is presented in detail in section four off the current paper.

The level of protection that any firewall is able to provide in securing a private network when connected to the public Internet is directly related to the architecture(s) chosen for the firewall by the respective vendor [1]. Generally, the most commercially available firewalls utilize one or more of the following firewall architectures:

- ***Static packet filter*** - the decision to accept or deny a packet is based upon an examination of specific fields within the IP packet and protocol headers. Before forwarding a packet, the firewall compares the IP header and Transmission Control Protocol (TCP) header against a table defined by user which contains the rules that dictate whether the firewall should deny or permit packets to pass. The rules are scanned in sequential order until the packet filter finds a specific rule that matches the criteria specified in the packet-filtering rule. If the packet filter does not find a rule that matches the packet, then it imposes a default rule.
- ***Dynamic (stateful) packet filter*** - like the static packet filter, operates at the network and transport layers (OSI layer 3 and 4). Advanced dynamic packet filter use additional state information collected from transport layer.
- ***Circuit-level gateway*** - operates at the session layer off the OSI model or as a "shim-layer" between the application layer and the transport layer of the TCP/IP stack. In many respects, a circuit-level gateway is simply an extension of a packet filter in that it typically performs basic packet filter operations and then adds verification of proper handshaking and the legitimacy of the sequence numbers used in establishing the connection. The circuit-level gateway examines and validates TCP and user datagram protocol (UDP) sessions before opening a connection, or circuit, through the firewall. Hence the circuit-level gateway has more data to act upon than a standard static or dynamic packet filter.
- ***Application-level gateway (proxy)*** - like a circuit-level gateway, an application-level gateway intercepts incoming and outgoing packets, runs proxies that copy and forward information across the gateway, and functions as a proxy server, preventing any direct connection between a trusted server or client and an untrusted host. Unlike the circuit gateway, the application-level gateway accepts only packets generated by services they are designed to copy, forward, and filter.
- ***Stateful inspection*** - combines the many aspects of dynamic packet filtering, circuit-level and application-level gateways, although stateful

inspection has the inherent ability to examine all seven layers of the OSI model.

- ***Cutoff proxy*** - is a hybrid combination of a dynamic (stateful) packet filter and a circuit-level proxy. In simplest terms, the cutoff proxy first acts as a circuit-level proxy in verifying the RFC (Request for comment) - recommended three-way handshake and any required authenticating actions, then switches over to a dynamic packet filtering mode of operation. Hence, it initially works at the session layer (OSI layer 5) then switches to a dynamic packet filter working at the network layer (OSI Layer 3) after the connection-authentication process is completed.
- ***Intrusion prevention*** - provide application level analysis and verification, but current Intrusion Prevention Systems (IPS) product inspection methodologies are based primarily on signature, and try to block everything that can possibly be wrong in a given packet. As the list of known signatures grows, IPS performance slows. The rate of newly discovered known bad things on the Internet is ever accelerating and, over time, could render the use of signature-based IPS unusable. An approach that involves inline IPS products at the perimeter and passive mitigation at the core allows for "deep" network protection that spans the entire diameter of the network, not just the outer crust.
- ***Deep packet inspection*** - can be seen as the integration of Intrusion Detection Systems (IDS) and IPS capabilities with traditional stateful firewall technology. Deep packet inspection firewalls, like most stateful inspection firewalls and many IDS and intrusion detection and prevention (IDP) products, take the opposite approach. Rather than focusing on recognizing and accepting only good packets, they try to find - and then deny - only the "bad" packets. Such devices are vulnerable because they require updates whenever a new and more creative form of "bad" is unleashed on the Internet.

2. Firewall platforms

Firewall solutions are available on a number of platforms and can generally be segregated into three groups:

- ***Hardware-based products*** - are single-purpose systems. Required hardware and software are bundled in one easily installed package. Usually, hardware solutions run on a stripped down version of Unix or Linux, where all of the unnecessary Operating System (OS) components are removed. The benefits of this model are that they are fast, relatively inexpensive, and don't require the loading of the software. These solutions require another machine or directory to store and analyze the logs.

- ***Software-based products on Linux/Unix*** - are software products running on one of the many distributions of Linux or Unix. In some cases, the OS is included in the firewall product, requiring a single install on one low-cost computer.
- ***Software-based products on Windows*** - are software products running on a Windows-based OS. The primary benefit of a Windows-based firewall is its low cost, it running on an inexpensive PC. Windows is the most widely known family of OS. So it is likely that a firewall administrator will know how to use it.

For some environments, a hardware-based firewall is a great choice. For others, nothing beats a Linux or Unix-based firewall and for a great many, a Windows-based firewall is the best choice.

One of the most misunderstood terms in network security with respect to firewalls today is “OS hardening” or “hardened OS.” Many vendors claim that their network security products are provided with a “hardened OS.” What you will find in virtually all cases is that the vendor simply turned off or removed unnecessary services and patched the OS for known vulnerabilities. Clearly, this is not a “hardened OS” but really a “patched OS.”

A hardened OS is one in which the vendor has modified the kernel source code to provide for a mechanism that clearly provides a security perimeter between the non secure application software, the secure application software, and the network stack.

3. Firewall policy configuration management

A firewall, like any other network device, has to be managed by someone. Security policy should state who is responsible for managing the firewall, how will be manage and the access method.

Management of firewall policy configurations can be complex, error-prone, costly and inefficient for many large networked organizations. Implementing a firewall configuration policy either involves writing low-level command syntax via a Command Line Interface (CLI) or the use of a graphical management console. Typical errors in a firewall configuration policy range from invalid syntax to errors in properly comprehending the configuration, given its scale and complexity. The Graphical User Interface (GUI) is the most commonly used method to configure a firewall in a timely manner, especially amongst inexperienced administrators.

Actual firewall configuration tools allow individual management of the security policy. When the management of different firewall policies is required it must established one administrative session with each firewall. Each firewall will have its own security policy without any correlation with the policies of firewall from the security area.

Design of the sequence of rules in a firewall must assure that these are consistent, complete, and compact. Consistency means that the rules are ordered correctly, completeness means that every packet satisfies at least one rule in the firewall, and compactness means that the firewall has no redundant rules [2].

Firewalls are the first line of defense visible to an attacker and, by design, are generally difficult to attack directly, causing attackers to often target the administrative accounts on a firewall. The username/password of administrative accounts must be strongly protected, and the administrative channel must also high protected using Secure Socket Layer (SSL) or Virtual Private Network (VPN) technologies.

4. Firewall policy management system

As was mentioned this paper proposes a secure system for firewalls management. The system is designed modular in order to allow management of all firewalls from a security zone. Functional and architectural requirements used in the design phase of this system are:

- must be accepted only secure management operations;
- administration of more firewalls from the same interface;
- firewall security policy definition must be rule based;
- firewall rules must be analyzed for security policy compliance;
- full audit on administrative user creation and modification;
- full audit of management operations.

The system has a modular design and is implemented using Php Hypertext Preprocessor (PHP) based on Model-View-Controller (MVC) model. The MVC model has been developed, from the beginning, in order to model natural “Input - Processing - Output” flow.

The MVC model architecture (fig. 1) has its roots in Smalltalk [3], where it was originally applied to map the traditional input, processing, and output tasks to the graphical user interaction model. However, it is straightforward to map these concepts into the domain of multi-tier enterprise applications.
Components of the MVC architecture are:

- ***Model*** - represents data and the activities that govern access to and updates of this data. Often the model serves as a software approximation to a real-world process, so simple real-world modeling techniques apply when the model is defined.
- ***View*** - renders the contents of a model. It accesses enterprise data through the model and specifies how that data should be presented. It is the view's responsibility to maintain consistency in its presentation when the model

changes. This can be achieved by using a push model, where the view registers itself with the model for change notifications, or a pull model, where the view is responsible for calling the model when it needs to retrieve the most current data.

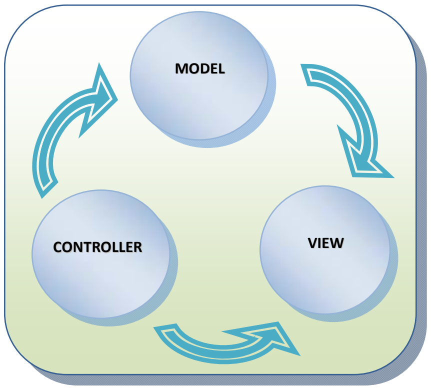

Fig. 1. MVC architecture

- ***Controller*** - The controller translates interactions with the view into actions to be performed by the model. In a Web application, they appear as GET and POST HTTP requests. The actions performed by the controller include changing the state of the model. Based on the user interactions and the outcome of the model actions, the controller responds by selecting an appropriate view.

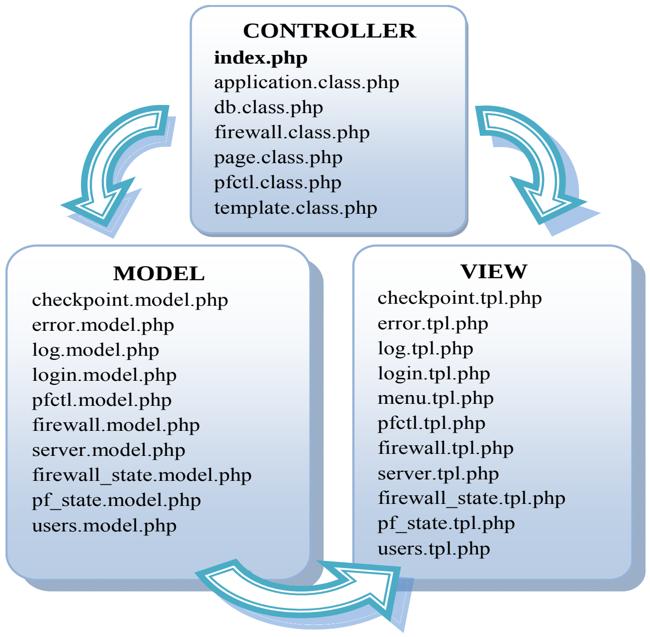

Fig. 2. MVC model files

Firewall policy management system provides a base platform and a framework for developing secure management modules. In order to demonstrate the features and the advantages of this management system two modules for Iptables and PF firewalls [4] have been developed.

The correspondence between files of the management system and MVC model is presented in fig. 2.

Further are described the most important files from the Controller layer, in order to provide a better understanding of the software program.

Because each request to the server requires application initialization, *index.php* will contains the definition of all the functions and instances and object initialization. One of the functions defined here is *handle_app_exception*, which will redirect all exceptions generated within application to a standard page for error handle. At the end *index.php* will pass the control to the application controller (*applications.class.php*). Here are defined all the pages accessible from controller, the pages with view only permission, login and start page.

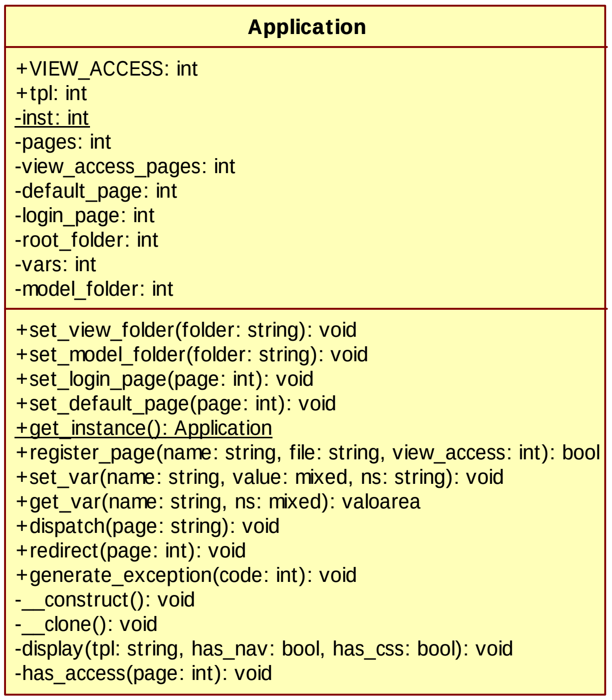
Application
+VIEW_ACCESS: int
+tpl: int
-inst: int
-pages: int
-view_access_pages: int
-default_page: int
-login_page: int
-root_folder: int
-vars: int
-model_folder: int
+set_view_folder(folder: string): void
+set_model_folder(folder: string): void
+set_login_page(page: int): void
+set_default_page(page: int): void
+get_instance(): Application
+register_page(name: string, file: string, view_access: int): bool
+set_var(name: string, value: mixed, ns: string): void
+get_var(name: string, ns: mixed): valoarea
+dispatch(page: string): void
+redirect(page: int): void
+generate_exception(code: int): void
-__construct(): void
-__clone(): void
-display(tpl: string, has_nav: bool, has_css: bool): void
-has_access(page: int): void

Fig. 3. UML component view diagram for Application class

The file *applications.class.php* contains the code for the *Application* class acting as a dispatcher for received requests and redirecting them to the corresponding page. Primary method of the application class is *dispatch*. This method load a page based on the existing GET parameter named *page* or on the argument received. If the user is not signed in it will be automated redirected to the default login page. Once the page is discovered *page.model.php* will be included and his *run* method will be called, passing the control to the model. At the end, *dispatch* will call *display* for displaying the data. Unified Modeling Language (UML) component view of the application class is shown in fig. 3.

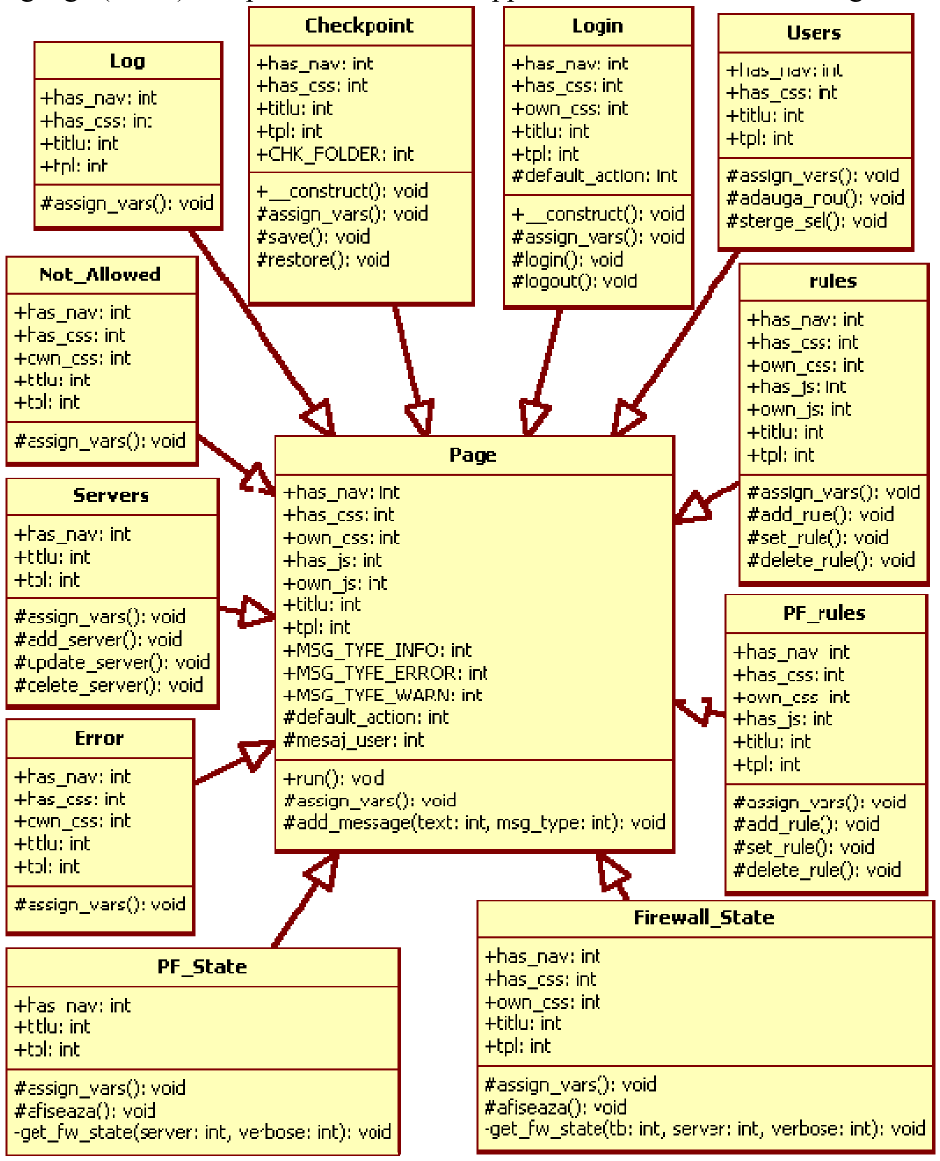

Fig. 4. UML component view for *Page* class

The *Page* class defines a template for model pages and functions for calls routing to other methods. The *run* method is the most imported method defined in this class being inherited by all the model pages, which extends the class *Page.* UML component view of the Page class and model pages that extend this class are shown in fig. 4. In order to assign default values for template pages *assign_var* method will be used. Method *add_message* is used for displaying information or noncritical messages to the user.

The primary role for *Template* class is to display the HTML templates, being the interface to the View layer from MVC. The *assign_var* method assigns values to a template variable, using *$name* variable as parameter. The *display* method is used to display the template specified in argument, using as variable *$tpl_file*, which represents the name of the template file (without extension).

Application data is stored and maintain in a Structured Query Language (SQL) database. The interface between the model pages and data is realized with a DB abstract layer. By parsing all commands related to the database through the general interface, it becomes possible for an application to use one of the many existing databases. DB abstract level gives a driver compatible with the database. A graphical diagram of this process is represented in fig. 5.

DB abstract layer is modeled in a class named Db in Db.class.php. The UML component view diagram is shown in fig. 6.

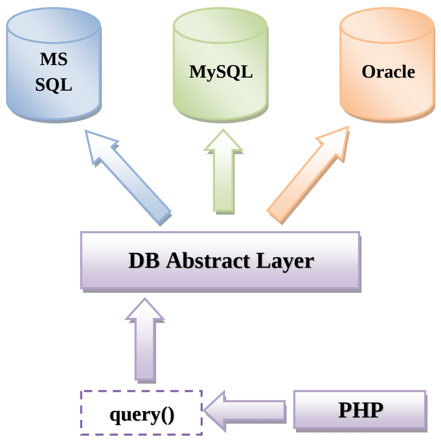

Fig. 5. DB Abstract layer representation

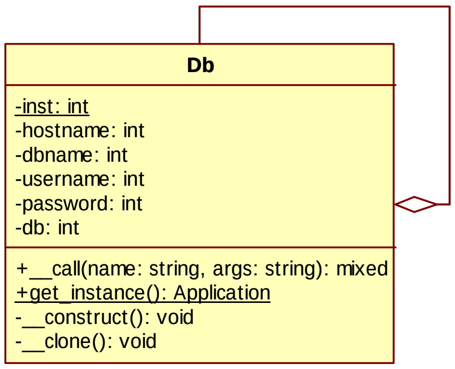

Fig. 6. UML Component View for Db class

The workflow of the application is shown in fig. 7, describing step by step the classes and methods called.

When you start the application and the page *index.php* is automatically called the following methods:

- *set_view_folder* - set the directory where the application load templates from;

- *set_model_folder* - set the directory where the application load model classes from;
- *register_page* - register pages in the controller. Each application module will have at least one model page. This method (register_page) keeps all the model pages in a vector, and search through it in order to decide the current page at an request;
- *set_login_page* - decide which page from vector is the login page;

set_default_page - set default page after user authentication, to know to what page of the vector to be redirected after the user and password were entered correctly.

Finally, is called the *dispatch* method that yields control of the *Application* class.

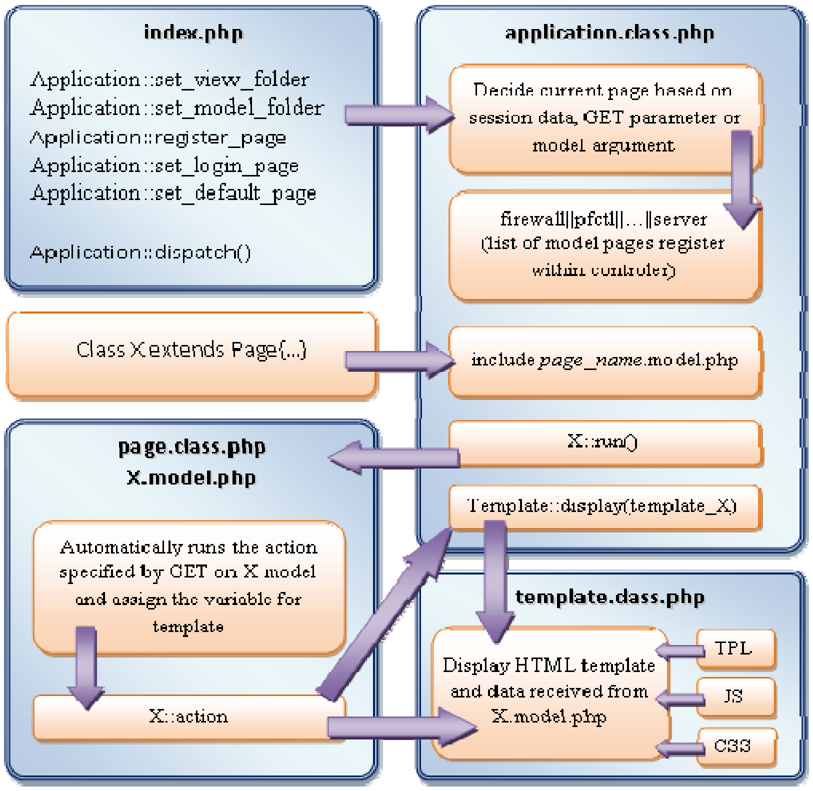
index.php
Application::set_view_folder
Application::set_model_folder
Application::register_page
Application::set_login_page
Application::set_default_page
Application::dispatch()
application.class.php
Decide current page based on session data, GET parameter or model argument
firewall||pfctl||…||server (list of model pages register within controler)
Class X extends Page{...}
include page_name.model.php
X::run()
Template::display(template_X)
page.class.php
X.model.php
Automatically runs the action specified by GET on X model and assign the variable for template
X::action
template.class.php
Display HTML template and data received from X.model.php
TPL
JS
CSS

Fig. 7. Application workflow

Application class decides the current page based on the existence of the session, GET parameter or argument received. Search in vector and redirects to the corresponding page. After calling the run method, controller assigns the control to the model class, automatically runs the action specified in the GET parameter and associate variables for the template.

Finally, after the model process data, *dispatch* will call the *display* method for displaying them to user. *Application* class will use *Template* class for displaying a template in format HTML with the data received from model.

In fig. 8 is shown a typically usage scenario for the secure management system. Management system software is installed and configured on the application server. Data transmitted between the client web browser and server are encrypted using SSL tunnel with a 1024-bit private key. After user authentication and HTTP session establishment the administrator can define the managed firewall equipments.

Once the firewalls are defined the next step is to define the firewall security policy by setting the rules. Rules are created one by one through the *rules.model.php* page for the entire security area. Modular design of the application allows integration with any formal rules verification or correlation algorithms [5].

The defined security policy is applied to firewalls using Secure Shell (SSH) encrypted channels after validation. All the actions made through system are logged for auditing.

Management system data are stored in a relational database using MySQL implementation of *Db* class.

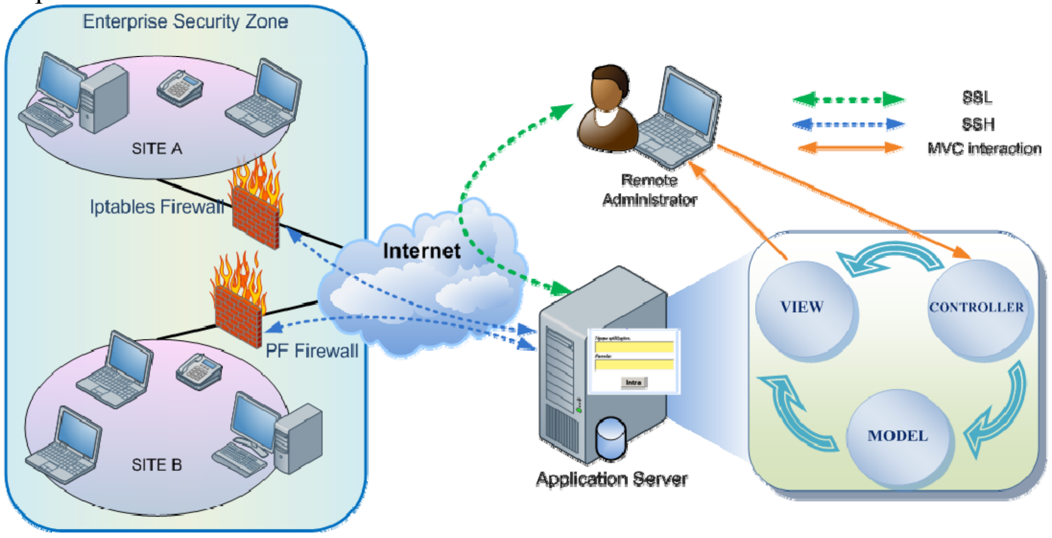

Fig. 8. A common usage for the secure management system

System high availability (HA) configuration can be obtained using clustering. Front-end component that contains web interface and scripts can be clustered using load balancing services (WEB farming) between two or more servers and backend component represented by database can be clustered by using databases specific HA technology. Special attention is needed for SSH public key distribution over the frontend web farm servers [6].

5. Conclusions

In the context of rising security requirements and the continuing need for firewall configuration, the proposed secure management system appears with his simplicity and usability. The ability to easily manage a firewall configuration is very important in any circumstances. Our proposed secure management system provides a different approach on configuring firewalls. Some of the features of the proposed system are:

- all management operations are secure;
- secure access to management system;
- high scalability and extensibility;
- online firewall configuration backup / restore;
- firewall policy creation, optimization and correlation;
- configuring multiple firewall in the security zone;
- complete operations audit;
- simple and intuitive Web based GUI;
- point in time restore facilities.

The management system can be extended with additional modules for formal rules verification and global security correlation [7]. It can be easily integrated with other Operations Support Services (OSS) or ServiceDesk applications in order to satisfy any enterprise requirements.

R E F E R E N C E S

[1]. *H.F. Tipton, M. Krause,* Information Security Management Handbook 6th ed., Auerbach Press, Florida, 2007

[2]. *M.G. Gouda, X. Liu,* Firewall design: consistency, completeness, and compactness, Distributed Computing Systems, Proceedings, 24th International Conference on, 2004. pp. 320- 327

[3]. *W.R. Lalonde, J.R. Pugh*, Inside Smalltalk (**Vol. One**), London, Prentice-Hall, 1990

[4]. *G. Daniel,* Retele IP arhitecturi si management (IP networks architecture and management), Bucharest, UPB, PhD student research report number 2 (Adviser V. Croitoru), 2007

[5]. *A.X. Liu,* Formal Verification of Firewall Policies, ICC '08. IEEE International Conference on, pp. 1494-1498., **Vol. 1**, 2008

[6]. *J. Frihat, F. Moldoveanu, A. Moldoveanu*, General Guidelines for the Security of a Large Scale Data Center Design, U. P. B. Sci. Bull., Series C, **Vol. 71**, Iss. 3, 2009

[7]. *J.L. Thames, R. Abler,* Implementing distributed internet security using a firewall collaboration framework. Richmond, SoutheastCon, Proceedings, IEEE, pp. 680-685, 2007.