

Faculdade de Engenharia da Universidade do Porto



FEUP

Integrated Security Sub-System for IPBrick

Sub-Sistema de Segurança Integrado para IPBrick

Ricardo J. Moreira Teixeira

Project Report submitted to Faculdade de Engenharia da Universidade do Porto in partial fulfillment of the requirements for the degree of Integrated Master in Electrical and Computers Engineering Major Telecommunications

Supervisor: Jorge Barbosa, PhD
Co-supervisor: Eng. Helder Rocha

June 2008

Integrated Security Sub-System for IPBrick

Sub-Sistema de Segurança Integrado para IPBrick

Ricardo J. Moreira Teixeira

Graduated in Electrical and Computers Engineering at Faculdade de Engenharia da Universidade do Porto

Project Report submitted to Faculdade de Engenharia da Universidade do Porto in partial fulfillment of the requirements for the degree of Integrated Master in Electrical and Computers Engineering Major Telecommunications

Supervisor: Jorge Barbosa, PhD
Co-supervisor: Eng. Hélder Rocha

Faculdade de Engenharia da Universidade do Porto Departamento de Engenharia Electrotécnica e de Computadores Rua Roberto Frias, s/n, 4200-465 Porto, Portugal

June 2008

I dedicate this work to my family
who put up with me for (too) many years
și pentru Iulia Măierean
care mă suportă în fiecare zi.

Abstract

The present report intends to show the development of security sub-system, as well its integration with the software package of network services developed by iPortalMais, the IPBrick.

This project is part of a client demanded development to IPBrick and its outcome is intended to be commercialized as an integrant part of the latter.

With this improvement, it is intended to lay a very useful set of tools to the hands of System Administrators in a world where information security is of the utmost importance.

After a preliminary study of the solution to be applied it was developed an “on-the-fly” connection blocker and a monitoring service for accesses occurred through SSH, FTP, VPN PPTP, VPN SSL. The interface for the firewall rules ordering was also improved for easiness of use and access policies that can be associated with VPN SSL client certificates were implemented.

Thus, using exclusively open source software it was increased the security of IPBrick, continuing its amazing growth throughout the recent years.

Resumo

O presente relatório tem a intenção de mostrar o desenvolvimento de um sub-sistema de segurança, bem como a sua integração, com o pacote de software desenvolvido pela iPortalMais, o IPBrick.

Este projecto é parte de o pedido de um cliente acerca do desenvolvimento da IPBrick e o seu resultado tem como intenção ser comercializado como parte integrante da mesma.

Com este melhoramento, é intenção depositar nas mãos de Administradores de Sistemas um poderoso conjunto de ferramentas num mundo onde a segurança da informação é da maior importância.

Após um estudo preliminar das soluções a ser aplicadas foi desenvolvido um bloqueador de conexões “on the fly” e um serviço de monitorização para acesso ocorridos por meio de SSH, FTP, VPN PPTP e VPN SSL. Foi também melhorada a interface de ordenação de regras da firewall e implementado políticas de acesso que podem ser associadas a certificados de clientes VPN SSL.

Assim, usando exclusivamente software em código aberto a segurança da IPBrick foi aumentada, continuando o seu extraordinário crescimento ao longo dos últimos anos.

Index

1. INTRODUCTION.....	1
1.1. MOTIVATION.....	1
1.2. THE GOALS.....	2
1.3. REPORT STRUCTURE.....	3
2. SECURITY CONCEPTS AND TOOLS.....	5
2.1. IPBRICK.....	5
2.1.1. <i>IPBrick.I</i>	6
2.1.2. <i>IPBrick.C</i>	7
2.1.3. <i>IPBrick.GT</i>	9
2.1.4. <i>IPBrick.KAV</i>	10
2.2. LINUX SECURITY.....	12
2.3. SECURITY THREAT BY PROTOCOL.....	13
2.4. RINGS OF SECURITY.....	14
2.4.1. <i>VPN ring of security</i>	14
2.5. SECURITY SOLUTIONS.....	15
2.5.1. <i>Contrack-tools</i>	15
2.5.2. <i>Iptables</i>	16
2.5.3. <i>OpenVPN</i>	18
3. SOLUTIONS FOR THE SECURITY SUB-SYSTEM.....	21
3.1. ACTIVE CONNECTIONS.....	21
3.1.1. <i>System Breakdown Structure</i>	21
3.1.2. <i>State machine</i>	22
3.1.3. <i>Cutter</i>	23
3.1.4. <i>contrack-tools</i>	23
3.1.5. <i>iptables</i>	24
3.2. ACCESS MONITORING.....	25
3.2.1. <i>Requirement analysis</i>	25
3.2.2. <i>Log analysis</i>	26
3.3. FIREWALL RULES ORDERING.....	28
3.4. OPENVPN ANALYSIS.....	29
3.4.1. <i>Configuration file issues</i>	29
3.4.2. <i>Services integration</i>	29
4. IMPLEMENTATION.....	31
4.1. ACTIVE CONNECTIONS.....	31
4.1.1. <i>System implementation</i>	31
4.1.2. <i>PHP script</i>	32
4.1.3. <i>Web interface</i>	37
FIGURE 4.16: ACTIVE CONNECTIONS INTERFACE.....	38

4.2.	ACCESS MONITORING	38
4.2.1.	<i>Database</i>	38
4.2.2.	<i>Import script</i>	41
4.2.3.	<i>System implementation</i>	45
4.2.4.	<i>Web interface</i>	46
4.2.5.	<i>PDF report</i>	48
4.3.	FIREWALL RULES ORDERING	49
4.4.	VPN SSL ACCESS POLICIES.....	50
4.4.1.	<i>System implementation</i>	50
4.4.2.	<i>Database</i>	50
4.4.3.	<i>PHP script</i>	53
4.4.4.	<i>Web interface</i>	56
5.	TESTS AND RESULTS.....	59
5.1.	TESTBED	59
5.2.	ACTIVE CONNECTIONS TESTS	60
5.3.	ACCESS MONITORING TESTS.....	61
5.4.	FIREWALL RULES ORDERING TESTS	62
5.5.	VPN SSL ACCESS POLICIES TESTS	62
6.	CONCLUSION AND FUTURE WORK	65
	REFERENCES	67
	APPENDIX A	69

List of Figures

FIGURE 1.1: TOTAL VULNERABILITIES DISCLOSURES FROM 2000 - 2007 (IBM ISS 2008).....	1
FIGURE 2.1: IPBRICK.I SERVICES RESUME (IportalMais - Soluções de Engenharia para Internet e Redes, LDA. 2008)	6
FIGURE 2.2: IPBRICK.C SERVICES RESUME (IportalMais - Soluções de Engenharia para Internet e Redes, LDA. 2008)	7
FIGURE 2.3: IPBRICK.GT APPLIANCE (IportalMais - Soluções de Engenharia para Internet e Redes, LDA. 2008).....	9
FIGURE 2.4: IPBRICK.KAV APPLIANCE (IportalMais - Soluções de Engenharia para Internet e Redes, LDA. 2008).....	10
FIGURE 2.5: LINUX SYSTEM - A CRACKER'S MAZE (TOXEN 2003)	12
FIGURE 2.6: VIRUS PATH THROUGH A VPN (TOXEN 2003)	15
FIGURE 2.7: OPENVPN USES UDP PROTOCOL FOR EXCHANGING PACKETS	18
FIGURE 3.1: SYSTEM BREAKDOWN STRUCTURE FOR THE ACTIVE CONNECTIONS DEVELOPMENT	21
FIGURE 3.2: EXAMPLE OF THE OUTPUT OF A <code>CAT</code> TO <code>IP_CONNTRACK</code>	22
FIGURE 3.3: DIAGRAM OF THE TABLES AND BUILT-IN CHAINS OF IPTABLES (ANDREASSON 2006)	25
FIGURE 4.1: EXAMPLE TCP CONNECTION ENTRY IN THE CONNTRACK LIST	32
FIGURE 4.2: REGULAR EXPRESSION TO PARSE THE PROTOCOL OF A CONNTRACK ENTRY	32
FIGURE 4.3: REGULAR EXPRESSION TO PARSE THE TCP CONNECTION ENTRIES IN THE CONNTRACK LIST	33
FIGURE 4.4: EXAMPLE UDP CONNECTION ENTRY IN THE CONNTRACK LIST	33
FIGURE 4.5: REGULAR EXPRESSION TO PARSE THE UDP CONNECTION ENTRIES IN THE CONNTRACK LIST	33
FIGURE 4.6: EXAMPLE ICMP CONNECTION ENTRY IN THE CONNTRACK LIST.....	33
FIGURE 4.7: REGULAR EXPRESSION TO PARSE THE ICMP CONNECTION ENTRIES IN THE CONNTRACK LIST.....	33
FIGURE 4.8: EXAMPLE GRE CONNECTION ENTRY IN THE CONNTRACK LIST	33
FIGURE 4.9: GENERIC REGULAR EXPRESSION TO PARSE THE CONNECTION ENTRIES IN THE CONNTRACK LIST	34
FIGURE 4.10: COPYING OF THE DATA FROM THE MATCHING ARRAYS TO THE OBJECT-ARRAYS THAT WILL CONTAIN ALL THE CONNECTIONS DATA	34
FIGURE 4.11: GENERATION OF THE LINK FOR THE ACTION PHP SCRIPT	36
FIGURE 4.12: PHP CODE TO INVERT THE MARK WHICH IS TO UPDATE THE CONNECTION	36
FIGURE 4.13: STRING TO ISSUE THE COMMAND TO UPDATE TCP AND UDP CONNTRACK ENTRIES	36
FIGURE 4.14: STRING TO ISSUE THE COMMAND TO UPDATE ICMP CONNTRACK ENTRIES.....	37
FIGURE 4.15: STRING TO ISSUE THE COMMANDS TO BLOCK GENERIC CONNECTIONS	37
FIGURE 4.16: ACTIVE CONNECTIONS INTERFACE	38
FIGURE 4.17: DATABASE SCHEME FOR THE ACCESS MONITORING.....	38
FIGURE 4.18: FUNCTION TO SET THE FILE POINTER TO THE END OF THE LOG FILE.....	41
FIGURE 4.19: DIAGRAM OF THE ALGORITHM IMPLEMENTED IN THE PHP SCRIPT	42
FIGURE 4.20: CONDITION TO FILTER THE SSH LOG LINES	43
FIGURE 4.21: IDENTIFYING STRING FOR THE LOG LINE OF A <code>SSH ROOT</code> LOGIN ATTEMPT	43
FIGURE 4.22: EXAMPLE OF THE PHP CODE TO PARSE THE LOG LINES.....	44
FIGURE 4.23: EXAMPLE OF AN ARRAY FOR THE VPN PPTP AFTER THE PARSING.....	45

FIGURE 4.24: MANAGEMENT OF THE ACCESS MONITORING PAGE	46
FIGURE 4.25: PAGE FOR VIEWING THE RECORDS OF THE ACCESS MONITORING.....	47
FIGURE 4.26: PDF REPORT FOR THE ACCESS RECORDS SHOWN IN FIGURE 4.25.....	48
FIGURE 4.27: FIREWALL RULES ORDERING INTERFACE.....	49
FIGURE 4.28: DATABASE SCHEME FOR THE VPN SSL ACCESS POLICIES.....	51
FIGURE 4.29: DIAGRAM OF THE ALGORITHM USED TO IMPLEMENT THE POLICIES TO ONE CERTIFICATE	54
FIGURE 4.30: DIAGRAM OF THE ALGORITHM USED TO CYCLE THROUGH THE POLICIES AFFECTED TO THE CERTIFICATE AND TO ADD THE NEEDED ENTRIES TO THE DATABASE	55
FIGURE 4.31: INTERFACE FOR ADDING THE POLICY MEMBERS.....	57
FIGURE 4.32: INTERFACE FOR CREATING A VPN SSL CERTIFICATE	57
FIGURE 5.1: TESTBED SCHEMA	60

List of Tables

TABLE 2.1: SECURITY THREADS BY PROTOCOL	13
TABLE 3.1: KEY LOG MESSAGES OF THE VPN PPTP SERVER FOR ACCESS MONITORING	26
TABLE 3.2: KEY LOG MESSAGES OF THE VPN SSL SERVER FOR ACCESS MONITORING	27
TABLE 3.3: KEY LOG MESSAGES OF THE SSH SERVER FOR ACCESS MONITORING.....	27
TABLE 3.4: KEY LOG MESSAGES OF THE FTP SERVER FOR ACCESS MONITORING.....	27
TABLE 3.5: INTEGRATION OF THE VPN SSL ACCESS POLICIES WITH OTHER SERVICES OF IPBRICK.....	29
TABLE 4.1: ACCESS MONITORING EVENT CODE DESCRIPTION.....	40

Nomenclature List

AJAX	Asynchronous JavaScript And XML
CSV	Comma-Separated Values
DFA	Deterministic Finite Automaton
DHCP	Dynamic Host Configuration Protocol
DHTML	Dynamic HTML
DMZ	Demilitarized zone
DNS	Domain Name Server
FPDF	Free PDF (Library)
FTP	File Transfer Protocol
GID	Group Identification
GNU	Gnu's Not Unix
GRE	Generic Routing Encapsulation
HTML	Hypertext Markup Language
ICMP	Internet Control Message Protocol
IP	Internet Protocol
IPsec	IP Security
ISDN	Integrated Services Digital Network
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol
NFA	Nondeterministic Finite Automaton
PBX	Private Branch Exchange
PCRE	Perl Compatible Regular Expressions
PDF	Portable Document Format
PHP	PHP: Hypertext Preprocessor
PID	Process Identifier
PPP	Point-to-Point Protocol
PPTP	Point-to-Point Tunneling Protocol
PSTN	Public Switched Telephone Network
QoS	Quality of Service
RTP	Real-time Transport Protocol
SIP	Session Initiation Protocol
SOHO	Small Office or Home Office
SSH	Secure Shell
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
UID	Unique Identification
UNIX	Uniplexed Information and Computing System
VPN PPTP	Virtual Private Network based on PPTP

VPN SSL | Virtual Private Network based on SSL/TLS tunneling
XML | Extensible Markup Language

Chapter 1

1. Introduction

1.1. Motivation

The authors of *Linux Security Cookbook* tell in their book the following story, about Scott, a System Administrator of their acquaintance:

“In early 2001, I was asked to build two Linux servers for a client. They just wanted the machines installed and put online. I asked my boss if I should secure them, and he said no, the client would take care of all that. So I did a base install, no updates. The next morning, we found our network switch completely saturated by a denial of service attack. We powered off the two servers, and everything returned to normal. Later I had the fun of figuring out what had happened. Both machines had been rooted, via `ftpd` holes, within six hours of going online. One had been scanning lots of other machines for `ftp` and `portmap` exploits. The other was blasting SYN packets at some poor cable-modem in Canada, saturating our 100Mb network segment. And you know, they had been rooted independently, and the exploits had required no skill whatsoever. Just typical script kiddies.” (Barrett, Silverman and Byrnes)

This story happened 7 years ago already! Last year’s *ISS security trend report* (IBM ISS 2008) showed the amazing difference in the software vulnerabilities disclosure, from 1.533 to 6.437, from the time when the story above took place (Figure 1.1). Furthermore, in the *CSI Survey 2007* (Richardson, Roberts 2007) it is shown that almost half of the surveyed companies experienced a computer-related security incident during 2007.

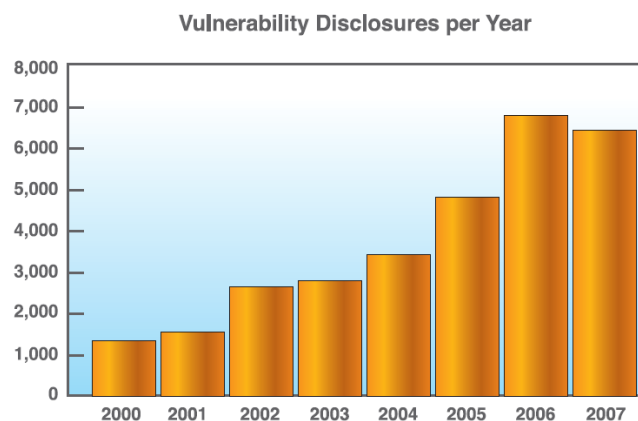


Figure 1.1: Total vulnerabilities disclosures from 2000 - 2007(IBM ISS 2008)

Security should be a major concern for every System Administrator that has machines operating in any kind of network. Attacks are not only result of someone trying to get in a private network or in its servers. In the *CSI Survey 2007* (Richardson, Roberts 2007) it may also be seen that 37 percent of the inquired attributed more than 21 percent of their cyber losses to be the cause of an inside attack.

1.2. The goals

It's the intention of this project to reinforce the security system of IPBrick by adding some extra features to it. These extra features are some specific demands of a client to iPortalMais, the company that develops the software IPBrick. Namely, it is going to be developed and integrated a connection blocker, an improved firewall rule ordering interface, an access monitoring tool and access policies for VPN SSL clients.

The connection blocker must be able to block or terminate, on the fly, any TCP/IP connection going to, through or out the server machine. This way if the System Administrator suspects that a certain system is using a specific connection to, for example, perpetrating some sort of computer crime, he can easily cut the connection first and ask questions later.

The access monitoring has to be able to monitor the accesses made to the machine by the means of SSH, FTP, VPN SSL or VPN PPTP. It has to be able to show relevant information about the access performed, as are the source IP of the access, username with which the access was made, date and time of the access, etc. It also has to create PDF reports from the filtered monitored data.

IPBrick has an interface that permits the ordering of the firewall rules. Although it's a working interface it has the inconvenient that one can only move the rule up or down one position. Imagining there is a rule in the 53th position that is needed to be moved in 3rd, the System Administrator would have to click on the rule's up button 50 times! The goal here is to create a different interface much more user friendly.

The client also wanted to restrict the access of some of his VPN clients to particular machines of his network. So for this requirement we need to create access policies applicable to VPN SSL clients, in a way that different clients may have different policies.

1.3. Report Structure

This report is divided into six chapters, structuring the work in the following manner:

The first chapter is an introduction to the project in its whole.

In the second chapter it's presented the description of the technologies and tools used.

The third chapter is the approach to the problems of the diverse aspects of the project, as the analyses of the possible solutions and the justification of the choices taken.

The fourth and fifth chapters describe the implementation of the choices taken and the tests and results.

Finally, the last chapter is a sum up analyses of the work performed and it approaches the meaning of this project in the future development of IPBrick.

Chapter 2

2. Security concepts and tools

2.1. IPBrick

This work was done taking as base the most recent version of the IPBrick software. As so, it is important to understand what this software is about and what features it presents.

In an ever growing need for information share as well as communication solution, companies can have in IPBrick a solution that adds speed, ease of use and security to these requirements. Network administration, resources management, printers, files, e-mail accounts, Internet access, security against intrusions, and all other threats that lurk every system that is not isolated from the outside world like viruses and worms. IPBrick is a complete integrated server, based on a Debian Linux distribution, having a function-directed management interface with Web access, which enables a System Administrator to perform its configuration even if she doesn't possess background knowledge of the Linux system or even networking and services knowledge. Besides the functional interface, IPBrick offers a more advanced interface where a Network and Systems Administrator has direct access to all of IPBrick's services, as being DNS, DHCP, LDAP, mail server, file server, project manager, VPN server, firewall, among others.

IPBrick is an operative system for network management with an easy and functional GUI. This way companies don't depend on Network/Systems experts to maintain its business backbone functioning and with high availability.

IPBrick establishes a standard for stability and security in data transfer. As a center of all the cooperative work, it has several integrations with this kind of software tools, as MS Outlook and SugarCRM are an example. The Linux based technology ensures confidence, speed and efficiency in data transfers.

2.1.1. IPBrick.I

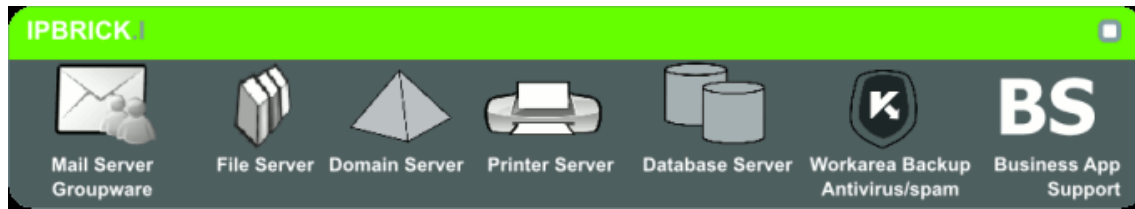


Figure 2.1: IPBrick.I services resume (iPortalMais - Soluções de Engenharia para Internet e Redes, Lda. 2008)

IPBrick.I offers all the necessary services of an Intranet server. It can function as an email, file, domain, fax, printing or backup server. Apart from the shared calendar and the contacts organizer (which can be synchronized via MS Outlook or web browser), it's possible to manage all the projects from a central point using the integrated dotProject software.

This Intranet server can function in three different modes: master, slave and as a MS Active Directory. Hence, it is possible to use IPBrick or a Windows AD to set the IPBrick users, groups, SIP phones and terminals. The IPBrick services are then able to work in a Windows or IPBrick Intranet environment.

IPBrick.I provides the following services:

- Collaborative Tools
 - E-mail - SMTP, IMAP and POP protocols
 - Address Book
 - Diary/Calendar
- File server
 - Individual and group work areas
- Domain Server
 - LDAP Protocol
- Printer Server
- Database Server
- Data Security
 - Work Area Backup Service
 - Pre-installed Kaspersky Anti-Virus for Email and Work Areas
 - Pre-installed Kaspersky Anti-Spam

- Business Applications Support
 - Document Management - iPortalDoc
 - Business Management - Gestix
 - Authentication Antispam - Spam-No
 - Enterprise Management - Primavera ERP
 - CRM - Sugar CRM

2.1.2. IPBrick.C



Figure 2.2: IPBrick.C services resume (iPortalMais - Soluções de Engenharia para Internet e Redes, Lda. 2008)

IPBrick.C is IPBrick's communication server. It provides and guarantees the security for data transfer between the Intranet and the Internet. Downloaded and uploaded data are analyzed, filtered and managed by several components of IPBrick, namely, the mail-relay, the proxy server, the Intrusion Detection System, the Anti-Virus, Anti-Spam and firewall.

As is IPBrick.C's objective to manage the connections from a company to the Internet, it controls all the available interfaces of the company to the Internet. Among others, services offered to do so are the web server, FTP server, VPN SSL, PPTP and IPsec gateways, as well as VoIP telephony and an Instant Messaging server.

VPN technology allows not only the secure flux of data, as well the choice of physical workspace in function of the company's necessities. IPBrick.C manages permanent connections between several points using IPsec and/or SSL/TLS tunnels for connection reliability.

Electronic mail and web traffic is continuously filtered through configurable rules. Statistical reports can be generated in order to present essential information in graphics of easy interpretation.

Through webmail, accessible using a SSL secured HTTP session, e-mails can be consulted and managed in a secure way, by a web browser any moment and at any time.

IPBrick.C can be set up to be placed in a firewall protected DMZ, as a communication server, or even as a complete communication server with a firewall integrated system.

IPBrick.C can import users, groups and machines (workstations and SIP phones) from an IPBrick.I or a Windows AD system. This way, installing IPBrick.C as communications server it isn't needed to redefine system information already configured in the company's Intranet server.

IPBrick.C offers the following communication, security and network management services:

- Email
 - Mail Relay - SMTP, IMAP and POP protocols
 - WebMail
- Fax 2 Mail - Mail 2 Fax
- Telephony Services
 - VoIP Gateway
 - PBX IP
 - SIP Proxy
- Web Services
 - Web Server - HTTP and HTTPS protocols
 - WebPhone
 - HTTP/FTP Proxy and Cache
- Instant Messaging
 - WebChat
 - Instant Messaging Server
- Communications Security
 - Firewall
 - VPN Server - SSL, IPSec and PPTP protocols
 - IDS (Intrusion Detection System)

- Pre-installed Kaspersky Anti-Virus for Email and Proxy
- Pre-installed Kaspersky Anti-Spam

2.1.3. IPBrick.GT



Figure 2.3: IPBrick.GT appliance (iPortalMais - Soluções de Engenharia para Internet e Redes, Lda. 2008)

IPBrick.GT completes the software solution IPBrick.I and IPBrick.C with hardware designed for enabling optimized Voice/Data-Integration. IPBrick.GT meets the demands of needed integration between traditional telephony and new VoIP SIP phones and SIP providers.

IPBrick.GT implements a full PSTN/VoIP gateway allowing the direct connection to PBX (ISDN E1/Bri and Analog Trunks), PSTN operator, LAN and Internet. The intelligent routing enables voice to flow either via VoIP or traditional ISDN and Analog technologies. IPBrick integrates traffic shaping functionality that will prioritize all telephony connections. The number of active and parallel phone calls is only limited by the given capacity of your local Internet connection bandwidth. The appliance is preinstalled with IPBrick.I and IPBrick.C and usable right away.

Characteristics:

- VoIP Data communication based on standard Internet protocols (SIP, RTP)
- Secure delivery of VoIP data through your defined VPN-tunnels
- Intelligent Voice Call negotiation to foreign VoIP-Servers, SIP Provider and to the traditional telephony network
- Isolation of the ISDN/Analog linkage to your PBX
- A migration path, incorporating your legacy ISDN/Analog phones and PBX towards VoIP technology
- Highly available VoIP company site linking
- Extensive QoS features with integrated bandwidth-management

2.1.4. IPBrick.KAV



Figure 2.4: IPBrick.KAV appliance (iPortalMais - Soluções de Engenharia para Internet e Redes, Lda. 2008)

The IPBrick.KAV is a security appliance based on the dedicated software IPBrick.IC and built to protect the enterprise in the four most important security areas on present days:

- E-mail Security
- Web Access Security
- Network Security
- Intranet Security

The IPBrick.KAV is a solution that puts together the better of two worlds in security terms: communication server and security software.

The IPBrick.KAV communication server prevents workstations from connecting to the Internet directly, not allowing trojan programs to establish tunnels with the exterior or open backdoors that allow the access to the company network from the exterior, eliminating all the problems caused by Trojans. In order to achieve this, the IPBrick makes use of a set of Proxy services.

The Kaspersky Security software eliminates on the perimeter protection assumed by the IPBrick.KAV, all the malware (trojans, worms, spyware, phishing, virus, etc) which intend to infect the computers in the internal network.

Besides, IPBrick.KAV adds a new security function, an Intranet auditor, which is able to verify and identify which workstations are infected and therefore capable of compromising the security of the Intranet.

Security characteristics:

- Web Security
 - Kaspersky Anti-virus, to remove dangerous contents existing in Web sites.

- "Black and White lists" for access control to the Internet hindering or limiting the access to certain sites.
- Mail Security
 - Kaspersky Anti-virus and Anti-Spam, clean the dangerous contents sent by e-mail.
 - Denial of service reduces the attacks that intend to block the e-mail service of the company (mail bombing).
- Network Security
 - Firewall, to block undesired access to the internal network coming from the exterior
 - VPN, to increase security level access of the users to the internal resources
 - Intrusion Detection, to identify or to alert the administrators of suspicious accesses
- Intranet Security
 - detection of infected machines in the Intranet

Additional services offered:

- WebMail: to allow you to consult your e-mail safely from any place.
- VPN SSL: to be able to access with total comfort and security the data and applications of your company from the exterior.
- VoIP: to make calls via Internet or, being outside the company, to stay in touch with all the people inside also via Internet.
- Web Server: to make contents available from your local network to the Internet

The information contained in the present subsection was collected from IPBrick's reference manual (iPortalMais - Soluções de Engenharia para Internet e Redes, Lda. 2006) and from it's web page (iPortalMais - Soluções de Engenharia para Internet e Redes, Lda. 2008).

2.2. Linux security

To expose a Linux machine to the Internet safely is not an obvious task because there are many ways that an intruder can take to broke a system, and not necessarily due to any inherently insecure issue (Toxen 2003).

As shown in Figure 2.5 can be a maze labyrinth of unsecure programs, allowing the cracker to just have to find one of the many entry points to the system, so she can soon be granted root privileges or have access to important information, like access to user accounts, access to databases, etc.

Crackers have also to concern about the cost of walking a certain part of the maze. This cost may be traduced as the time actually spent to trying to walk that path; it might also be the financial cost of purchasing new equipment to crack a certain password. While crackers have the choice to take one or other path (or simply give up), System Administrators don't enjoy of this free will.

It is of importance, and may be considered as a security augmentation itself, the improvement of the user interface with a security tool. This enhancement can prevent situations were a System Administrator may overlook a security procedure or may even be bored by the complexity of applying it.

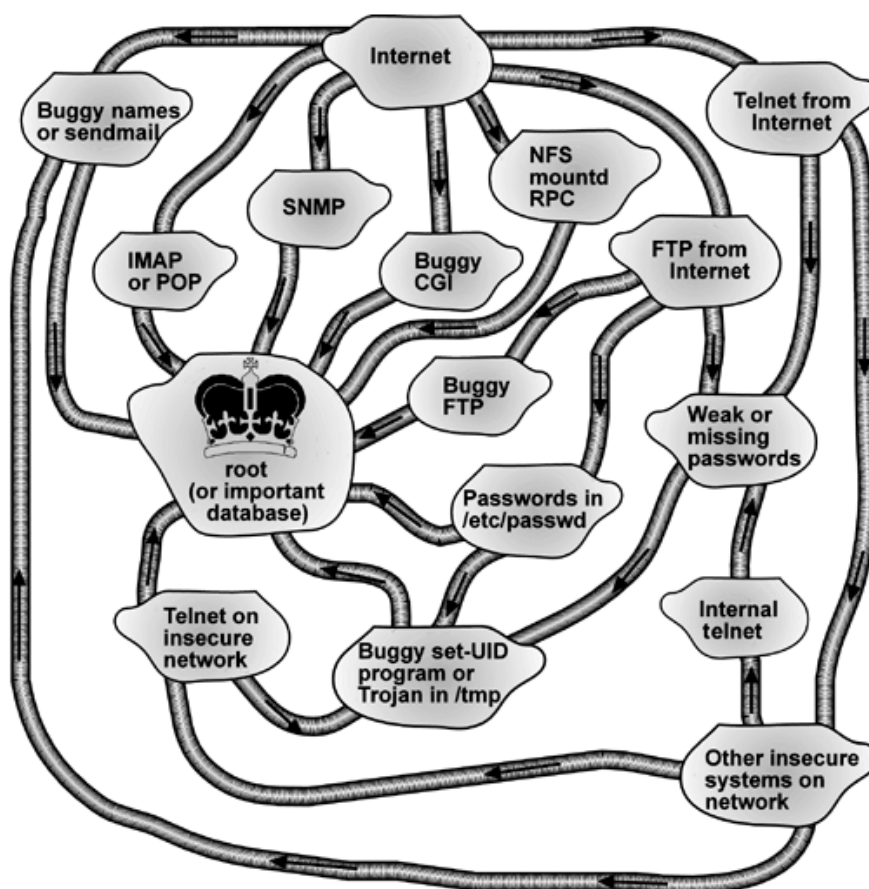


Figure 2.5: Linux system - a cracker's maze (Toxen 2003)

2.3. Security threat by protocol

Many crackers use the technique of installing, by some mean, a little server (a type of Trojan horse) to take hold of the system. This allows the cracker to remotely perform actions on the compromised system. Thus, these are programs installed on the machine, listening for packets on a predefined port; this port is known by the cracker. Through it the cracker sends the requests for the server program to execute on the system. Some popular Rootkits contain simple ones. Some more sophisticated ones require log password and perform communication encryption.

These servers use the TCP or UDP protocol to communicate, usually listening to high port numbers, above 1023. It is of extreme importance to use a tool, like `netstat`, to be able to monitor those high numbered ports (Toxen 2003).

Having a tool that presents active connections in these high numbered ports is valuable, even more valuable if it permits in the next instant to block the suspect connection.

Examples of threats by protocol are presented in Table 2.1 (some of them taken from (Black Hat 2001)).

Table 2.1: Security threads by protocol

Protocol	Security threats
TCP	Trojans, Rootkits, Brute-force authentication attacks, etc.
UDP	Trojans, Rootkits communication, UDP flood attack, RIP attacks, etc.
ICMP	ICMP packet magnification (ICMP Smurf), Ping of death, ICMP flood attack, ICMP nuke attack, ICMP Tunneling, IRDP attacks etc.
GRE	GRE tunnel intrusion

2.4. Rings of security

Rings of security is a term used in Bob Toxen's book (Toxen 2003) to describe the security levels or barriers that a cracker has to break until it gets control of the system or has access to some type of sensible information.

Nowadays the original UNIX architecture of a single ring of security is not enough. A single firewall is not the ideal protection when internal attacks are a big percentage of the threats that a company is subjected to, as described in annual CSI survey (Richardson, Roberts 2007). Many companies just rely on a well crafted firewall to protect them from a possible attack, having behind it totally unsecured other systems. Of course that it is easy to see that a network designed this way has to possess, not only a perfectly implemented firewall, as well a System Administrator that is able to respond very rapidly and efficiently to changes in the network and in used programs by updating the firewall rules accordingly to these changes. Common sense tells us that nothing in life is perfect!

Instead of a single ring of protection, in a system with multiple rings of security is much harder to be compromised. If this is done correctly, even if a cracker gets past one ring there will be another ring to stop her and possibly even a third one. In a system implementation of this kind a cracker will have to follow a sequence of at least two "hard-to-follow" path segments to get to any goal that will cause substantial harm.

"Rings of Security" have been added to Linux in several places. It is the reason root should not be allowed to use telnet. This restriction requires a cracker to break two passwords (root's and some ordinary user's) before he can log in as root remotely via telnet. This creates two rings of Security.

This is why IPBrick, besides having a firewall, has several others rings of security, like the proxy, the Intrusion Detection System, Kaspersky Anti-virus, etc. This way it is, for example, less probable that a network user can go to a phished website, download malicious software and install it in the machine, having it afterwards infecting the private network.

2.4.1. VPN ring of security

People have the tendency to believe that successfully implementing a VPN (and only to permit access to their private network through the latter) is enough to prevent hazardous events to happen in their unsecured LAN. Although it represents an additional ring of security in their network, it is not enough to prevent most of the common threats. As an example, a common Windows virus, with which the VPN client machine was infected, can spread through the VPN to the private network that the client machine is connecting to (Toxen 2003).

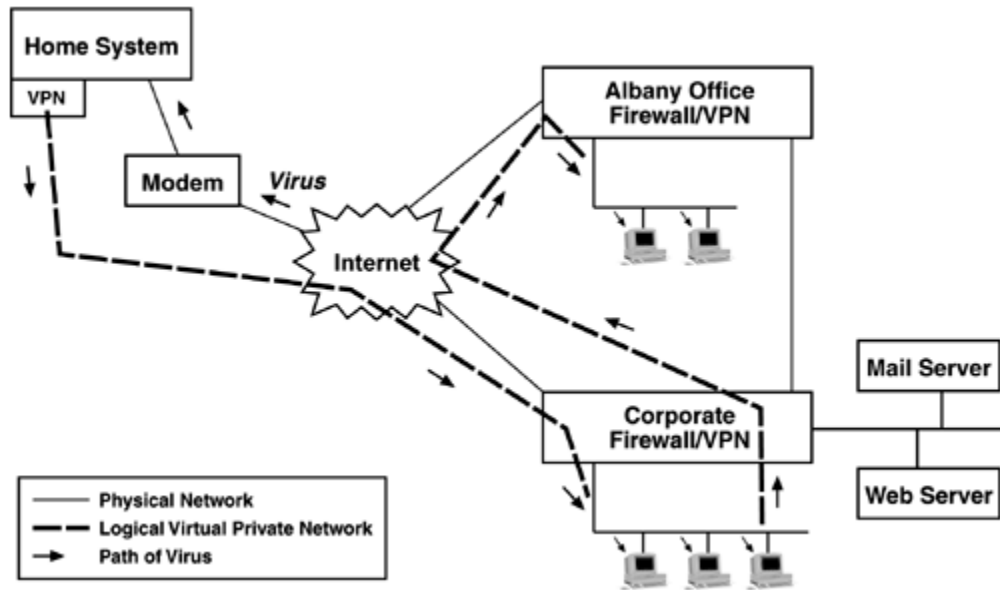


Figure 2.6: Virus path through a VPN (Toxen 2003)

Although Figure 2.6 refers to the path of a virus in a VPN, it is easy to imagine that a compromised VPN certificate from a client would gain access to the entire corporate internal network.

By this example we can see that an important additional ring of security would be to restrict, selectively, the access of VPN clients to the machines of the private network it is permitted to connect to.

2.5. Security Solutions

2.5.1. Contrack-tools

Contrack-tools presents itself as an userspace daemon, `contrackd`, and an interface for the command line, `contrack`. These tools enables System Administrators to interact with the connection tracking system (State Machine) of the popular Linux package, `iptables`, at userspace level.

This program was created by Pablo Neira Ayuso, an active contributor and member of the Netfilter Core Team, being the latter the developers of the most well known Linux command line program for the configuration of the Linux kernel packet filtering ruleset: `iptables`.

The userspace daemon `contrackd` has two main functionalities:

- to implement an high availability cluster based on iptables, as it is demonstrated on the project webpage, in the document “Simple high-availability Primary-Backup testbed” (Ayuso, Test Case 2007).
- to collect statistics from the Connection Tracking System of the firewall.

The `contrack` command line program enables to perform the following operations related to the Connection Tracking System:

- List the connection or expectation tracking list
- Search for a particular entry in the table
- Delete an entry from the table
- Create new entries in the table
- Display a real-time event log
- Flush the whole given table

The author affirms that `contrack` can be used to kill an established TCP connection without adding an iptables rule. Using for this a sane stateful ruleset which would block a packet that does not match any existing entry in the Connection Tracking Table. Basically, the idea consists of removing the entry that talks about the victim TCP connection. Thus, the client experiences a connection hang. Moreover, since `contrack` is not dependent of the layer 4 protocol, you can use to kill whatever layer 4 network flow (UDP, SCTP, ...).” (Ayuso, `contrack-tools: Connection tracking userspace tools for Linux 2008`)

2.5.2. Iptables

The Netfilter Project is the one who created iptables, the userspace tool to control Linux’s firewall. Since the release of version 2.4 of the Linux kernel, in January 2001, this userspace tool as been the standard to interact with the packet filtering framework, which provides the firewall service in Linux.

After these years, Iptables is now the most formidable of the open source firewalls, being able to compete with most commercial proprietary firewalls. Features like comprehensive protocol state tracking, packet application layer inspection, rate limiting, and a powerful mechanism to specify a filtering policy puts it as a very reliable, competent and powerful firewall, suitable for commercial use. It is the default userspace tool for packet manipulation of most of all the Linux distributions.

There is a common misunderstanding about what exactly are iptables and Netfilter. Putting it in a simply way, Netfilter is the packet filtering framework that iptables, the userspace tool, uses to manipulate the Linux kernels 2.4.x and 2.6.x packet filter ruleset. The Netfilter is also the name of the project that supplies other programs related with the Linux package filtering.

IPTables state machine

Iptables possesses a module that is called the state machine. It is used to track the connections passing through the machine. Connection tracking is done to let the Netfilter framework know the state of a specific connection. This feature makes iptables a stateful firewall, this meaning that it can be created rules based on a certain connection state. Like this, with iptables, it is possible to create tighter rulesets than with its non-stateful counterparts.

As to which iptables its concerned, there are four types of states for the tracked connections. These are NEW, ESTABLISHED, RELATED and INVALID. With the `--state` match it is possible to control who or what is allowed to initiate new sessions.

All of the connection tracking is done by a special framework within the kernel called contrack. contrack may be loaded either as a module, or as an internal part of the kernel itself. Although contrack lacks protocol specific states, it is possible to load additional modules so that the state machine of iptables may handle and classify protocol specific attributes. These extra modules are intended to handle TCP, UDP or ICMP protocols among others. This gathered information is used to allow contrack to classify the packets with its particular states. In the case of UDP streams, they are generally identified by their destination IP address, source IP address, destination port and source port.

With the contrack module, the possibility to turn on and off defragmentation is no longer present in the kernel configuration. As the connection tracking can not work properly without defragmenting packets, defragmenting has been incorporated into contrack and is carried out automatically. The only possibility to disable packet defragmentation is by turning off the connection tracking. Defragmentation is always performed if connection tracking is turned on.

The connections are tracked in the PREROUTING and OUTPUT chains of iptables. The inbound connections to the machine are tracked and its state calculated in the PREROUTING chain. On the other hand, if a new connection is originated from the machine itself, the connection's state will be calculated and tracked within the OUTPUT chain. This way it is used the first chains that handle the connections, both for inbound connections and for outbound ones.

2.5.3. OpenVPN

OpenVPN is a full-featured open source SSL VPN solution that can serve the most varied finalities. It can be used for a broad range of purposes as it is remote access, site-to-site VPNs, Wi-Fi security, and enterprise-scale remote access solutions with load balancing, failover, and fine-grained access-controls. Being a simple to implement service, OpenVPN is a cost-effective, lightweight alternative to other VPN technologies, targeted for the SME and enterprise markets.

Security overview

OpenVPN has two authentication modes:

- Static Key: Use of a pre-shared static key
- TLS: Use of SSL/TLS + certificates for authentication and key exchange

The static key mode consists of the generation of a key that is shared between the two machines before OpenVPN starts the tunnel. This static key consists of four independent keys: HMAC send, HMAC receive, encrypt, and decrypt.

In SSL/TLS mode, it is used bidirectional authentication (i.e. each side of the connection must present its own certificate). Upon success, OpenSSL's RAND_bytes function is used for encryption/decryption and HMAC key source material generation and exchanging over the SSL/TLS connection. This mode never uses any key bidirectionally. Thus, each peer has a distinct sent HMAC, received HMAC, packed encryption, and packed decryption key. If `--key-method 2` is used, the actual keys are generated from the random source material using the TLS PRF function. If `--key-method 1` is used, the keys are generated directly from the OpenSSL RAND_bytes function. `--key-method 2` was introduced with OpenVPN 1.5.0 and will be made the default in OpenVPN 2.0.

To avoid latency bottleneck the SSL/TLS rekeying is done with a set transition-window parameter, where the old and new keys are permitted to co-exist.

Because SSL/TLS is designed to operate over a reliable transport, OpenVPN provides a reliable transport layer on top of UDP (Figure 2.7).

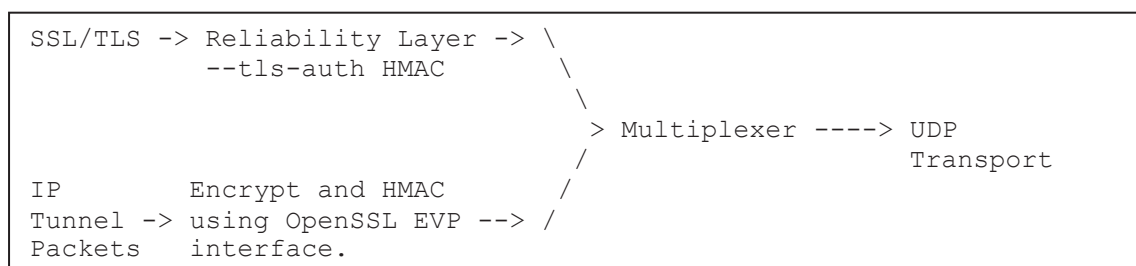


Figure 2.7: OpenVPN uses UDP protocol for exchanging packets

After each peer has its set of keys, the tunnel forwarding operation commences.

The encrypted packet is formatted as follows:

- HMAC(explicit IV, encrypted envelope)
- Explicit IV
- Encrypted Envelope

The plaintext of the encrypted envelope is formatted as follows:

- 64 bit sequence number
- payload data, i.e. IP packet or Ethernet frame

The HMAC and explicit IV are outside of the encrypted envelope.

The per-packet IV is randomized using a nonce-based PRNG that is initially seeded from the OpenSSL `RAND_bytes` function.

HMAC, encryption, and decryption functions are provided by the OpenSSL EVP library, allowing the user to select an arbitrary cipher, key size, and message digest for HMAC. This library uses BlowFish as the default cipher and SHA1 as its default message digest.

The mode where the users choose a pre-shared passphrase (or static key) in conjunction with the `--tls-auth` directive to generate an HMAC key to authenticate the packets that are themselves part of the TLS handshake sequence is one of the advantages of OpenVPN. This protects against buffer overflows in the OpenSSL TLS implementation, because an attacker cannot even initiate a TLS handshake without being able to generate packets with the correct HMAC signature.

OpenVPN multiplexes the SSL/TLS session used for authentication and key exchange with the actual encrypted tunnel data stream. OpenVPN provides the SSL/TLS connection with a reliable transport layer (as it is designed to operate over). The actual IP packets, after being encrypted and signed with an HMAC, are tunneled over UDP without any reliability layer. So if `--proto udp` is used, no IP packets are tunneled over a reliable transport, eliminating the problem of reliability-layer collisions. Of course that in the case of tunneling a TCP session over OpenVPN running in UDP mode, the TCP protocol itself will provide the reliability layer.

Chapter 3

3. Solutions for the security sub-system

3.1. Active connections

3.1.1. System Breakdown Structure

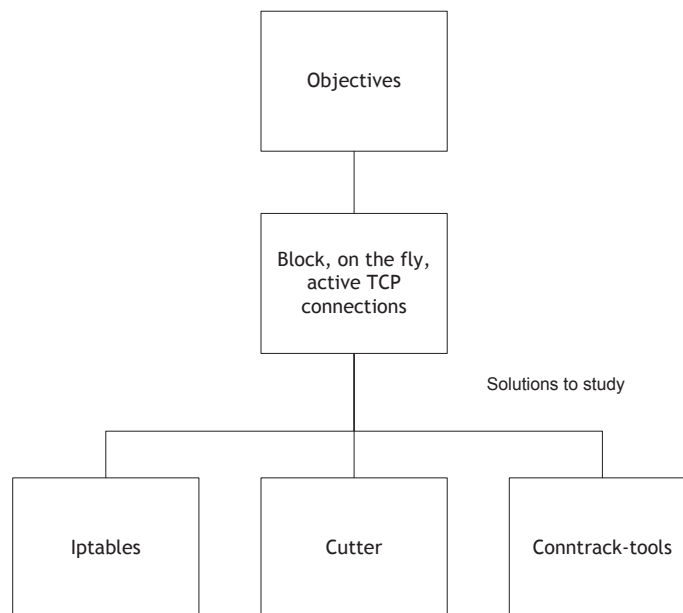


Figure 3.1: System Breakdown Structure for the active connections development

As it may be seen in Figure 3.1, for this part of the development, it was accorded that the objective to achieve was to develop a tool that would be able to immediately block, with a simple command from the System Administrator, any active TCP connection going to, through or from the machine.

Having this objective in mind, it was studied various solutions to accomplish, or even over pass, the expectations of the client.

So, to meet client's expectations, it was studied cutter, a program that sends reset messages to both connection peers; contrack-tools, to manipulate iptables state machine and as well as iptables itself, because PBrick's firewall is based in it.

3.1.2. State machine

First there was the need to understand where to find the information about what connections were actually passing in the machine. Although there are programs like `netstat` and `tcpdump` that can do the job, there is one particular file that was found to be easily parsed to extract the useful information. The state machine of `iptables` has a pseudo-file where it is kept the records of all the connections, and their state, passing in the Linux system. This pseudo-file is situated in `/proc/net/nf_contrack`. By issuing the command

```
cat /proc/net/nf_contrack |grep -v 127.0.0.1
```

it's printed a list of all the connections tracked by the `contrack` module of `iptables`, except the ones that have the string "127.0.0.1"; this ones are internal connections of the system. An example of this output can be seen in Figure 3.2.

```
udp      17 6 src=192.168.69.197 dst=192.168.69.255 sport=138 dport=138
packets=1 bytes=240 [UNREPLIED] src=192.168.69.255 dst=192.168.69.197
sport=138 dport=138 packets=0 bytes=0 mark=0 use=1
tcp      6 426417 ESTABLISHED src=192.168.69.19 dst=192.168.69.176
sport=39157 dport=22 packets=5677 bytes=335289 src=192.168.69.176
dst=192.168.69.19 sport=22 dport=39157 packets=5506 bytes=665840 [ASSURED]
mark=0 use=1
udp      17 28 src=192.168.69.230 dst=192.168.69.255 sport=137 dport=137
packets=3 bytes=234 [UNREPLIED] src=192.168.69.255 dst=192.168.69.230
sport=137 dport=137 packets=0 bytes=0 mark=0 use=1
tcp      6 431992 ESTABLISHED src=192.168.69.59 dst=192.168.69.176
sport=58912 dport=902 packets=35506 bytes=3902072 src=192.168.69.176
dst=192.168.69.59 sport=902 dport=58912 packets=52721 bytes=45717188
[ASSURED] mark=0 use=1
tcp      6 431999 ESTABLISHED src=192.168.69.59 dst=192.168.69.176
sport=37778 dport=902 packets=76839 bytes=4017994 src=192.168.69.176
dst=192.168.69.59 sport=902 dport=37778 packets=77625 bytes=27155364
[ASSURED] mark=0 use=1
udp      17 6 src=192.168.69.2 dst=192.168.69.255 sport=138 dport=138
packets=2 bytes=472 [UNREPLIED] src=192.168.69.255 dst=192.168.69.2
sport=138 dport=138 packets=0 bytes=0 mark=0 use=1
udp      17 20 src=192.168.69.176 dst=192.168.69.2 sport=32854 dport=53
packets=1 bytes=72 src=192.168.69.2 dst=192.168.69.176 sport=53 dport=32854
packets=1 bytes=142 mark=0 use=1
```

Figure 3.2: Example of the output of a `cat` to `ip_contrack`

Later on, the lines that don't correspond to actual active connections (as the connections labeled `UNREPLIED` in Figure 3.2) have to be discarded for being of no actual interest in the context of this work.

The state machine of `iptables` associates to the TCP protocol packages special states. Using these special states is possible to further filter the entries of the `contrack` list, enhancing the performance of the script that will parse this information.

There is no actual documentation enumerating and explaining the TCP packets special states, there is just the reference that these states can be found in the header file `ip_contrack_tcp.h` (can be consulted in Appendix A). Thus, analyzing this file the following conclusions were taken:

- All the packets marked with the state `CLOSE` are no longer part of active connections and so they can be eliminated from the entries to be parsed.
- If an entry contains the `WAIT` string, means that it is no longer an actual active connection, as so it can also be discarded from the parsing.
- The states with the `SYN` string and with `LAST_ACK` means that these packets correspond to the handshake stage of the TCP connection. Most probably in a few milliseconds after the `cat` command is performed this will be already an active connection, so this entry is not discarded.
- `NONE` and `ESTABLISHED` states correspond to active connections and so they will be included in the `contrack` entries to be parsed.
- Due to the lack of documentation on these states, the `LISTEN` and `MAX` states have unknown meaning; so they are also parsed.

With this information in hand it is now needed to find a solution to block the desired connection “on the fly”.

3.1.3. Cutter

Cutter is a program that, supposedly, terminates a connection on the fly by sending raw packets, both to the client as to the server of the connection.

Although the author claims that it’s a working program (Lowth 2005), it was never tested, by the latter, in a Debian Linux distribution. The program was successfully compiled and the commands to block the connections were issued without being presented any error, but any TCP, UDP, GRE or ICMP protocol connections were unsuccessfully blocked.

In the end, it was concluded that this program doesn’t work with the present IPBrick distribution of Linux.

3.1.4. contrack-tools

In its webpage (Ayuso, `contrack-tools`: Connection tracking userspace tools for Linux 2008), `contrack-tools` author affirms that by removing the entry of the connection in the connection tracking list is enough to drop the connection in question. After several exchanged e-mails with the author, that lead to a bug fix of the program, it was clear that in the test machine the program didn’t performed as expected: with any entry being deleted from the

connection tracking list, a second or two after, the connection would reappear and no connection would be terminated.

Nevertheless, using this program it was possible to successfully manipulate the mark on a specific connection. This way we established a specific mark to blocked connections (10000) and added rules to the firewall that would drop packets marked with “10000”.

3.1.5. iptables

It would be possible to use iptables as the solution to add or delete rules, on the fly, by simply issuing a shell command. Although, due to IPBrick’s firewall configuration methods, it would be necessary to create a complex tracking system for the rules added by this development so that the fully integration would not be compromised.

As so, it was asserted that the best method to fulfill the goals of this development was to use a set of iptables rules that, matching against the conntrack mark “10000”, blocks the desired connection. For setting the marks “on the fly” it will be used the program conntrack-tools.

The connections to block can have three possible origin-destination pairs, each of these being handled by different built-in chains of the iptables structure (Figure 3.3). These are:

- INPUT chain: Handles connections that have its destination the system itself.
- OUTPUT chain: Handles connections that originate from the system itself.
- FORWARD chain: Handles connections that have its destination another host on the network.

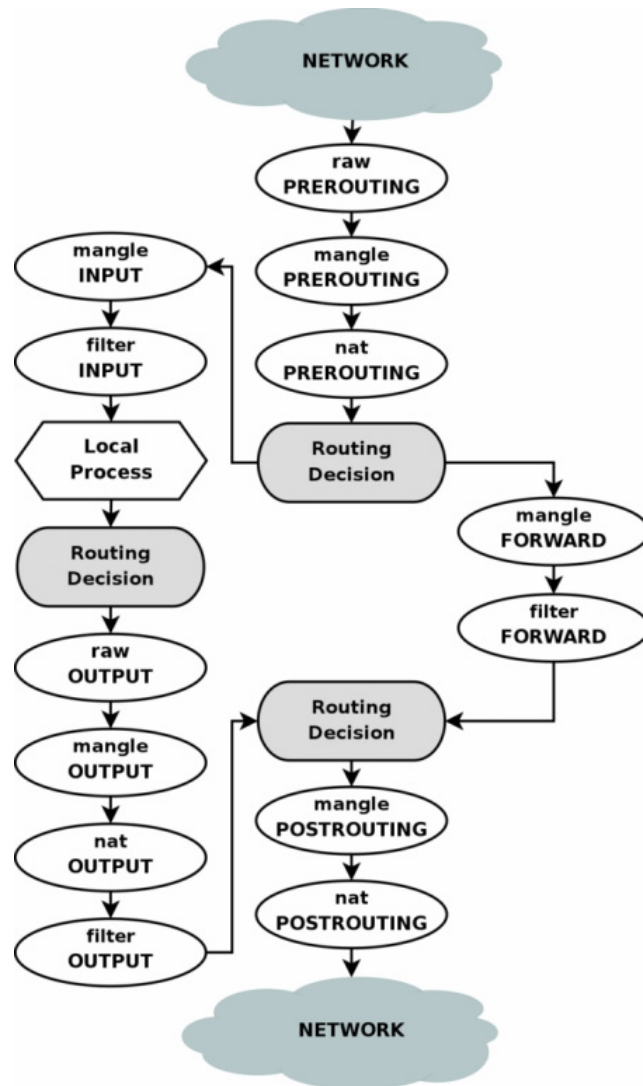


Figure 3.3: Diagram of the tables and built-in chains of IPTables (Andreasson 2006)

3.2. Access monitoring

3.2.1. Requirement analysis

The main required functionalities for this development can be categorized by the following:

- Services: being able to monitor SSH, FTP, VPN SSL and VPN PPTP accesses to the machine.
- Logged information: it should be shown relevant information about the access logged, specifically, begin and end dates and time, IP source, username used for the access.

Table 3.2: Key log messages of the VPN SSL server for access monitoring

Information retrieved	Example log message
Successful log in Date and time of event Username Source IP address	Thu Mar 6 17:52:44 2008 192.168.96.200:1103 [bobby] Peer Connection Initiated with 192.168.96.200:1103
Successful log out / Time out disconnection Date and time of event	Thu Mar 6 18:30:30 2008 bobby/192.168.96.200:1055 [bobby] Inactivity timeout (--ping-restart), restarting
Blocked access Date and time of event	Tue Mar 18 13:44:53 2008 bobby/192.168.96.90:1111 write UDPv4 []: Operation not permitted (code=1)

Table 3.3: Key log messages of the SSH server for access monitoring

Information retrieved	Example log message
Successful log in Date and time of event Username	Mar 19 16:39:54 fara sshd[13531]: (pam_unix) session opened for user operator by (uid=0)
Successful log out Date and time of event Source IP address	Mar 19 16:41:08 fara sshd[13531]: Connection closed by ::ffff:192.168.69.24
Wrong password input Date and time of event Username Source IP address	Mar 20 19:46:21 fara sshd[17969]: Failed keyboard-interactive/pam for operator from ::ffff:192.168.69.24 port 38810 ssh2
Wrong username input Date and time of event Username Source IP address	Mar 19 16:44:14 fara sshd[13584]: Illegal user oprator from ::ffff:192.168.69.24
Blocked access Date and time of event Source IP address	Mar 19 18:52:32 fara sshd[13608]: Read error from remote host ::ffff:192.168.69.24: Connection reset by peer
Root log in attempt Date and time of event Source IP address	Mar 19 16:45:57 fara sshd[13608]: Failed none for root from ::ffff:192.168.69.24 port 60225 ssh2

Table 3.4: Key log messages of the FTP server for access monitoring

Information retrieved	Example log message
Successful log in Date and time of event Username Source IP address	Mar 5 18:21:23 fara proftpd[12099]: fara.prostie.ro (192.168.69.24[192.168.69.24]) - USER administrator: Login successful.
Successful log out Date and time of event Source IP address	Mar 5 18:22:16 fara proftpd[12099]: fara.prostie.ro (192.168.69.24[192.168.69.24]) - FTP session closed.
Wrong password input Date and time of event Username Source IP address	Mar 5 18:24:20 fara proftpd[12132]: fara.prostie.ro (192.168.69.24[192.168.69.24]) - USER administrator (Login failed): Incorrect password.
Wrong username input Date and time of event Username Source IP address	Mar 5 18:34:56 fara proftpd[12225]: fara.prostie.ro (192.168.69.24[192.168.69.24]) - USER kimze: no such user found from 192.168.69.24 [192.168.69.24] to 192.168.69.202:21

3.3. Firewall rules ordering

The existing interface of the ordering of the firewall rules presents itself as a list of firewall rules. For ordering a rule, the user has to click on the link presented in the line of rule as many times as positions it has to be moved.

The best way to make the ordering of the firewall rules an easy process is to apply to the existing interface, a JavaScript library that will make this process as simple as dragging and dropping the rule.

Using the scriptaculous framework (Fuchs n.d.) and the PHP wrapper class SLLists (Neustaetter 2006), we have an easy way of implementing the functionality of having a sortable and to translate the resulting outcome into PHP. SLLists features the following functionalities:

- SLLists - constructor that basically sets the path to the JavaScript files
- addList - adds a list or other element as a new sortable entity
- printTopJS - prints the JavaScript into the head of a PHP file
- printForm - prints an HTML form that contains the hidden inputs needed. Alternatively users can create their own forms or use the printHiddenInputs functions to put these hidden inputs in existing forms
- printBottomJS - prints the JavaScript that should go right before the closing body tag
- getOrderArray - returns an array with items and their order after being passed an input with the serialized scriptaculous list

3.4. OpenVPN analysis

3.4.1. Configuration file issues

In order to block or permit access to internal machine, it will be used firewall rules that will match against the specific IP address that the certificate is attached to.

To do this it is needed to add the capability of per certificate configuration, being a certificate configuration file used to attribute a specific IP, chosen by IPBrick, to the VPN SSL client machine.

3.4.2. Services integration

As this development will be using information from the LDAP server, to be able to easily add defined machines and machine groups to the access, it will be also affected by services that interact with the machine management system of IPBrick. Therefore it was assessed which services, in fact, have influence on the defined policies that contain IPBrick defined groups or machines.

Table 3.5: Integration of the VPN SSL access policies with other services of IPBrick

Services	Actions performed
DHCP	Modify and delete
DNS	Modify and delete
Reverse DNS	Modify and delete
Network interfaces management	Modify
VoIP	Delete
LDAP	Modify and delete

In Table 3.5 we can see what type of action, and from which services, can influence the integrity of defined policies. This happens because all the services presented also use the information that is defined in IPBrick's LDAP. This way if its information is changed by any of the configuration pages of this services, this change of information will also affect the VPN SSL access policies configuration. In the case of the network interfaces management, the information changed is of the private network interface's IP. So, it will be needed to create functions that will update the database information of the VPN SSL implementation, when these actions are performed.

Chapter 4

4. Implementation

4.1. Active connections

4.1.1. System implementation

Having in mind what was discussed in chapter 3 about `iptables` built-in chains, three rules were added to the firewall. These rules were inserted in IPBrick's PostgreSQL database, in the existing table for the firewall. This way, it dynamically generates the file `firewall.sh`. This file is the script that contains the commands for initializing IPBrick's firewall. The resulting commands added to this script file are:

```
iptables -A INPUT -m connmark --mark 10000 -j DROP
iptables -A OUTPUT -m connmark --mark 10000 -j DROP
iptables -A FORWARD -m connmark --mark 10000 -j DROP
```

These rules will make the packets marked with “10000”, in the `conntrack` entries, to be dropped by IPBrick's firewall. It was needed to add three rules, one for each built-in chain of `iptables`. The mark “10000” was chosen for being known as a mark unused by any of the services of IPBrick®.

Next it was needed to install `conntrack-tools`, the program that allows interaction with the Connection Tracking System of Linux. As we can learn from `conntrack-tools` webpage (Ayuso, `conntrack-tools`: Connection tracking userspace tools for Linux 2008) we need two libraries that were not previously installed on IPBrick: `libnfnetlink`, a library that “provides a generic messaging infrastructure for in-kernel netfilter subsystems (such as `nfnetlink_log`, `nfnetlink_queue`, `nfnetlink_conntrack`) and their respective users and/or management tools in userspace” (Netfilter core team 2007), and the `libnetfilter_conntrack` library, a “userspace library providing a programming interface (API) to the in-kernel connection tracking state table” (Netfilter core team 2007).

Both the libraries and `conntrack-tools` were downloaded from their sites, compiled and made a Debian package out of them for further future installation in IPBrick systems. The version of `conntrack-tools` that was installed was 0.9.5, the latest stable version at the time. The library version dependencies for this version of `conntrack-tools` were 0.0.25-1 for `libnfnetlink` and 0.0.82-1 for `libnetfilter_conntrack`.

4.1.2. PHP script

Now that the system is set up, the functionalities will be implemented through a PHP script.

The file `nf_contrack` has the permission 440 (only readable by `root` and its group), so it cannot be read by the system user that is running the IPBrick PHP script (`www-data`). To do so it was used the socket system already implemented in IPBrick to issue the command

```
cat /proc/net/nf_contrack | grep -v 127.0.0.1
```

as `root`, through the PHP script. The output of this command is then passed to an array, being each line of the output an element of the array. This way the subsequent parsing of the information becomes easier.

Secondly, the array is filtered, using the `strpos()` function, to remove the entries that don't represent active connections. These connections are the ones that contain one of the following strings: `UNREPLIED`, `WAIT` or `CLOSE`. It was a rational option to do the filtering of the `contrack` entries in these two steps. This way it was taken advantage of the hybrid NFA/DFA engine of the GNU `grep` program (Friedl 1997). Thus, the benchmarking of the script showed that it was faster to do the filtering of the string `127.0.0.1` with the `grep` command instead of using the PHP function. The rest of the filtering was done by the PHP function as result of the same benchmarking showing no improvement in doing otherwise.

The filtered array is then separated into 4 other arrays, one for each type of connection protocol. This connection protocols are: `TCP`, `UDP`, `ICMP` and other connection protocols. After being separated, each one of them is parsed using the `preg_match()` function of PHP.

```
ipv4      2 tcp      6 431966 ESTABLISHED src=192.168.69.59 \
dst=192.168.69.176 sport=58940 dport=902 packets=28785 bytes=2658792 \
src=192.168.69.176 dst=192.168.69.59 sport=902 dport=58940 packets=58029 \
bytes=64790576 [ASSURED] mark=0 use=1
```

Figure 4.1: Example TCP connection entry in the `contrack` list

In Figure 4.1 we see an example TCP connection entry in the `contrack` list. The entries in the `contrack` list, as said before, are first parsed for separating each protocol. This is done by the regular expression of Figure 4.2 so that each protocol can be parsed by a specific regular expression. The specific regular expression for TCP entries is shown in Figure 4.3.

```
/^[a-zA-z0-9]+[ ]+[0-9]+[ ]+([a-z]+)/
```

Figure 4.2: Regular expression to parse the protocol of a `contrack` entry

```
/^.*src=([0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}) .*dst=([0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}) .*sport=([0-9]+) .*dport=([0-9]+) .*src=.*dst=.*sport=.*dport=.*mark=([0-9]+) /
```

Figure 4.3: Regular expression to parse the TCP connection entries in the contrack list

```
ipv4      2  udp      17 179 src=192.168.69.202 dst=192.168.69.2 sport=882 \
dport=2049 packets=15 bytes=1940 src=192.168.69.2 dst=192.168.69.202 \
sport=2049 dport=882 packets=14 bytes=2016 [ASSURED] mark=0 use=1
```

Figure 4.4: Example UDP connection entry in the contrack list

Figure 4.4 shows an example UDP connection entry in the contrack list. These protocol entries are parsed by the regular expression in Figure 4.5.

```
/^.*src=([0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}) .*dst=([0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}) .*sport=([0-9]+) .*dport=([0-9]+) .*src=.*dst=.*sport=.*dport=.*mark=([0-9]+) /
```

Figure 4.5: Regular expression to parse the UDP connection entries in the contrack list

```
ipv4      2  icmp      1 25 src=192.168.69.6 dst=192.168.69.10 type=8 code=0 \
id=33029 [ASSURED] src=192.168.69.10 dst=192.168.69.6 type=0 code=0 \
id=33029 mark=0 use=1
```

Figure 4.6: Example ICMP connection entry in the contrack list

Figure 4.6 shows an example ICMP connection entry in the contrack list. These protocol entries are parsed by the regular expression in Figure 4.7.

```
/^.*src=([0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}) .*dst=([0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}) .*sport=([0-9]+) .*dport=([0-9]+) .*src=.*dst=.*sport=.*dport=.*mark=([0-9]+) /
```

Figure 4.7: Regular expression to parse the ICMP connection entries in the contrack list

```
ipv4      2  gre      47 598 timeout=600, stream_timeout=18000 \
src=192.168.1.3 dst=84.91.32.190 srckey=0x4000 dstkey=0x280 packets=5 \
bytes=285 [UNREPLIED] src=84.91.32.190 dst=195.23.114.78 srckey=0x280 \
dstkey=0x4000 packets=0 bytes=0 mark=0 secmark=0 use=1
```

Figure 4.8: Example GRE connection entry in the contrack list

For all the connections that are not of the protocols shown above, as it is the case of the GRE protocol (Figure 4.8), it was used the more generic regular expression presented in Figure 4.9.

```
/^.*src=([0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}) .*dst=([0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}) .*src=.*dst=.*mark=([0-9]+) /
```

Figure 4.9: Generic regular expression to parse the connection entries in the contrack list

With this parsing done, the values of the matching array are copied to an object-array that will contain all the information of all the connections of a particular protocol. The properties of each element of this object array will be data of a given connection. In Figure 4.10 it can be seen an example of the copying of the values to the object-arrays that will have the information of the TCP protocol, the ICMP protocol and for the generic connections. The object-array for the UDP protocol is similar to the TCP one, just changing its name from `$conn_data_tcp` to `$conn_data_udp`.

```
<?
.
.
.
$conn_data_tcp[$a]->srcip = $match1[1];
$conn_data_tcp[$a]->destip = $match1[2];
$conn_data_tcp[$a]->srcport = $match1[3];
$conn_data_tcp[$a]->destport = $match1[4];
$conn_data_tcp[$a]->mark = $match1[5];
.
.
.
$conn_data_icmp[$c]->srcip = $match3[1];
$conn_data_icmp[$c]->destip = $match3[2];
$conn_data_icmp[$c]->type = $match3[3];
$conn_data_icmp[$c]->code = $match3[4];
$conn_data_icmp[$c]->id = $match3[5];
$conn_data_icmp[$c]->mark = $match3[6];
.
.
.
$conn_data[$d]->proto = $match0[1];
$conn_data[$d]->srcip = $match4[1];
$conn_data[$d]->destip = $match4[2];
$conn_data[$d]->srcport = "";
$conn_data[$d]->destport = "";
$conn_data[$d]->mark = $match4[3];
.
.
.
?>
```

Figure 4.10: Copying of the data from the matching arrays to the object-arrays that will contain all the connections data

The object-arrays for the TCP and UDP connections will have the following properties:

- `srcip`: The source IP of the connection
- `destip`: The destination IP of the connection
- `srcport`: The source port of the connection
- `destport`: The destination port of the connection
- `mark`: The current mark that the connection possesses

In addition to the `srcip`, `destip` and `mark` properties, the object-array for the ICMP connections will have the following properties:

- `type`: ICMP type of the connection¹
- `code`: ICMP code of the connection¹
- `id`: ICMP id of the connection

For the generic connection type, the object-array will be the same as for the TCP and UDP connections, but with the following differences:

- The `srcport` and the `destport` properties won't contain data.
- It will have one more property, the `proto` property. This property will contain to what protocol refers the `conntrack` entry.

Using then a mixture of PHP and HTML, these object-arrays are used to display and identify, in the web interface of IPBrick, the active connections presently going through the system.

In the end of the parsing, each object-array is checked if it contains any data with the function `count()`. If it returns a true value, i.e., a greater value than "0", using HTML a table is created and the connections are displayed, using a `for()` cycle for that purpose.

For each connection it is also created a link to another PHP script that blocks or unblocks the chosen connection. With the link to the PHP script (using the GET method) it's also passed the parameters needed to block/unblock the selected connection. In Figure 4.11 it's shown, as an example, the HTML/PHP code to generate that link, in the case of the ICMP connections.

¹ The meaning of these ICMP types and codes can be consulted in the online document ICMP TYPE NUMBERS (IANA, et al. 2008)

```
<a href="corpo.php?pagina=view_active_conn_acc&proto=icmp&s_ip=<?echo
$conn_data_icmp[$i]->srcip;?>&d_ip=<?echo $conn_data_icmp[$i]-
>destip;?>&icmp_type=<?echo $conn_data_icmp[$i]->type;?>&icmp_code=<?echo
$conn_data_icmp[$i]->code;?>&icmp_id=<?echo $conn_data_icmp[$i]-
>id;?>&mark=<?echo $conn_data_icmp[$i]->mark;?>"><?if ($conn_data_icmp[$i]-
>mark==10000) ?> Desbloquear ligação <? else ?> Bloquear ligação ?></a>
```

Figure 4.11: Generation of the link for the action PHP script

The second PHP script is a, so called, action script. It receives the information about the connection in question and executes the command necessary to block or unblock the connection.

Basically, what this script does is to invert the mark that the present connection possesses in the iptables state machine, from 0 to 10000 or vice-versa, using the simple equation

$$new_mark = |current_mark - 10000| \quad . \quad (4.1)$$

This formula, translated into PHP, is represented in Figure 4.12.

```
<? $mark = abs($mark-10000); ?>
```

Figure 4.12: PHP code to invert the mark which is to update the connection

Then, the PHP script was made to issue a command, as `root`, through the socket system of IPBrick. This will be the command that will actually update the state of the desired connection. If it is a TCP, UDP or ICMP connection it will update the connection entry of the `nf_conntrack` module using the `conntrack-tools` program. In the case it is a generic connection it will issue a series of commands, respectively:

1. It adds one rule to the PREROUTING chain and another to the OUTPUT chain of `iptables` that marks the connection with the desired conntrack mark.
2. Waits 10 seconds issuing the command `sleep`.
3. Deletes the firewall rules previously added to `iptables`.

These commands are passed as strings to be executed as `root` user. For the TCP and UDP connection it is used the string of Figure 4.13 to issue the update command to `conntrack-tools`.

```
<? $command = "conntrack -U conntrack -s ".$s_ip." -d ".$d_ip." -p
".$proto." --sport ".$s_port." --dport ".$d_port." -m ".$mark; ?>
```

Figure 4.13: String to issue the command to update TCP and UDP conntrack entries

The concatenated variables on this string are the values that were passed with the GET method to this action script. For the ICMP connections it was used the string of Figure 4.14 and for the generic connections the string used is presented in Figure 4.15.

```
<? $command="contrack -U contrack -s ".$s_ip." -d ".$d_ip." -p ".$proto."
--icmp-type ".$icmp_type." --icmp-code ".$icmp_code." --icmp-id ".$icmp_id."
-m ".$mark; ?>
```

Figure 4.14: String to issue the command to update ICMP contrack entries

```
<? $command="iptables -t mangle -A PREROUTING -p ".$proto." -s ".$s_ip." -d
".$d_ip." -j CONNMARK --set-mark ".$mark.";iptables -t mangle -A OUTPUT -p
".$proto." -s ".$s_ip." -d ".$d_ip." -j CONNMARK --set-mark ".$mark."; sleep
10; iptables -t mangle -D PREROUTING -p ".$proto." -s ".$s_ip." -d ".$d_ip."
-j CONNMARK --set-mark ".$mark.";iptables -t mangle -D OUTPUT -p ".$proto."
-s ".$s_ip." -d ".$d_ip." -j CONNMARK --set-mark ".$mark." &"; ?>
```

Figure 4.15: String to issue the commands to block generic connections

After executing the command, a small JavaScript script calls the original page so that the user can perform other operations.

4.1.3. Web interface

In accordance with IPBrick's ease of use, the most straightforward way for the user to interface with this tool was to create an additional web page in the web interface of IPBrick.

In this web page there are listed the various connections to the machine (Figure 4.16). These connections are organized in 4 different tables: TCP, UDP, ICMP and other active connections. Each connection is identified by its source IP, source port, destination IP and destination port. By pressing the link, in the action column, the user blocks/unblocks the desired connection, changing its state description from "Enabled" to "Blocked" or vice-versa.

Using a clear, simple, but yet powerful interface, the user has a very direct way of blocking the undesired connections, without having to deal with the complexity of `iptables`, `contrack-tools` and the Linux networking architecture.

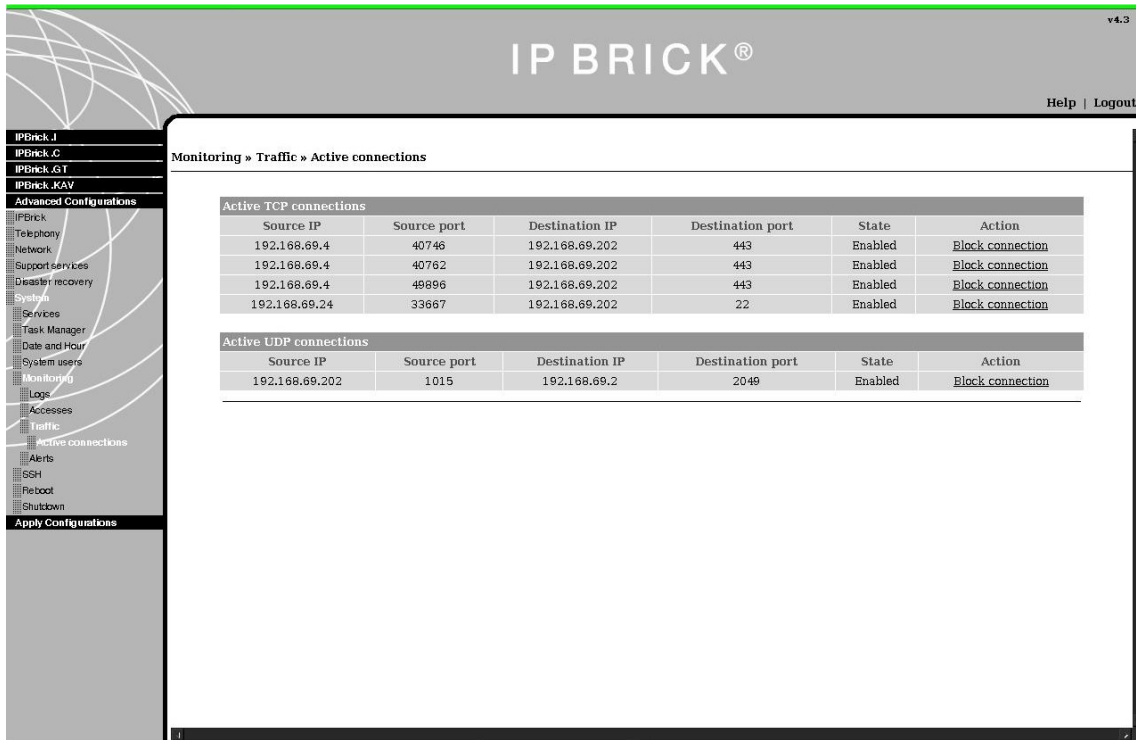


Figure 4.16: Active connections interface

4.2. Access Monitoring

4.2.1. Database

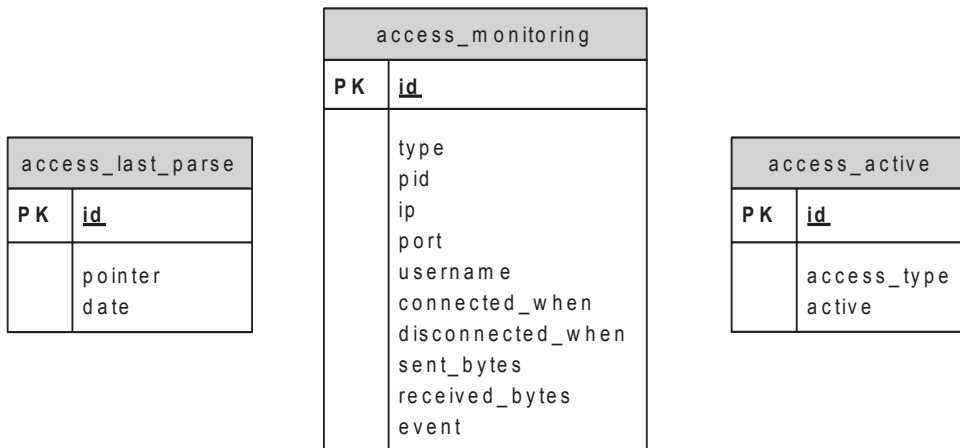


Figure 4.17: Database scheme for the access monitoring

In Figure 4.17 it's shown the diagram of the database implementation for the logging of the accesses to the machine.

The table `access_monitoring` will be the one containing all the information that will be parsed, both from the syslog file, as from the `openvpn-1.log` file. The attributes of this table are explained next:

- `id`: the numerical identification of the entries in the table; it's also its primary key.
- `type`: to which type of access the database entry refers. It can take four values:
 - `ftpd`
 - `sshd`
 - `ssl`
 - `ppp`
- `pid`: the process ID of the program that created the log
- `ip`: which IP accessed the machine
- `port`: what port used to access the machine
- `username`: which username was used to access the machine
- `connected_when`: when was the connection began
- `disconnected_when`: when the connection was terminated
- `sent_bytes`: the bytes the machine sent due to that connection; it's exclusive for the VPN PPTP connection
- `received_bytes`: the bytes the machine received due to that connection; it's exclusive for the VPN PPTP connection
- `event`: the present state of the access; it can take the nine values presented in Table 4.1

Table 4.1: Access monitoring event code description

Value	Meaning	Refers to which accesses
1	Connected	All
2	Disconnected normally (by the server or the client)	SSH, FTP, VPN PPTP
3	Wrong pass inserted	SSH, FTP, VPN PPTP
4	Wrong username inserted	SSH, FTP, VPN PPTP
5	Disconnected due to blocking	SSH, VPN PPTP, VPN SSL
6	Disconnected due to connection timeout	VPN PPTP
7	Disconnected due to timeout or blocking	FTP
8	Disconnected; connection attempt as root user	SSH
9	Disconnected normally or due to connection timeout	VPN SSL

As for the table `access_last_parse` of Figure 4.17, it will be used to keep track of both the date of the last parsing, as the last position of the file position indicator used by the file system pointers of PHP. The attributes of the table are explained next:

- `id`: the numerical identification of the entries in the table; it's also its primary key. It will contain two identifying values:
 - 0: ID of the entry for the syslog file.
 - 1: ID of the entry for the `openvpn-1.log` file.
- `pointer`: the last position of the file position indicator from the last used file system pointer; the position is measured in bytes from the beginning of the file.
- `date`: it's a UNIX timestamp that represents the moment when the last parsing was made.

The table `access_active`, represented in Figure 4.17, is used to store the information of what access types to parse. The attributes of the table are explained next:

- `id`: the numerical identification of the entries in the table; it's also its primary key.
- `access_type`: identifies the access type. It contains the following values:
 - `ftp`: identifier of the FTP access type.

- `ssh`: identifier of the SSH access type.
- `ppp`: identifier of the VPN PPTP access type.
- `ssl`: identifier of the VPN SSL access type.
- `active`: this attribute takes a form of a Boolean variable: 1 for enabled, 0 disabled.

4.2.2. Import script

As discussed before, the most efficient and direct way to address the problem was to create a script that does the parsing of the syslog and the `openvpn-1.log` files. Thus, again it was used PHP scripting to create this import script.

The diagram of the algorithm in which this script was based is presented in Figure 4.19. It was made various functions that, more or less, correspond to the blocks presented in the diagram. Thus, there is the need to further analyze how this algorithm implementation was putted forward.

Firstly the function `getLastParseInfo()` queries and parses the result of the query of the database table `access_last_parse`. The values retrieved are the last parse date and the value of the last file position indicator. After the return of the function being compared to the date of the backup log file, the right file is open. Using the PHP function `fseek()`, the file pointer is updated with the value of the attribute `pointer` of the previously queried table `access_last_parse`. This way it will only read, parse and process the new lines of the log file - a time saving solution.

Next, after the function `getActiveAccessLogs()` queries the table `access_active`, it is assessed if there is any access that has the attribute `active` with the value 1. If not, again using the `fseek()` function, the file system pointer (the `$fp` variable in Figure 4.18) is updated.

```
<? ... fseek ($fp, 0, SEEK_END); ... ?>
```

Figure 4.18: Function to set the file pointer to the end of the log file

In the case of any of the flags are active (the attribute `active` of the table `access_active`), the function `filterLog()` is called. This function filters the log lines of the accesses set to be analyzed and groups the lines by access type. This is accomplished by doing a series of `if()..else if()`. As an example of the conditions used, the one for the case of SSH log lines is shown below.

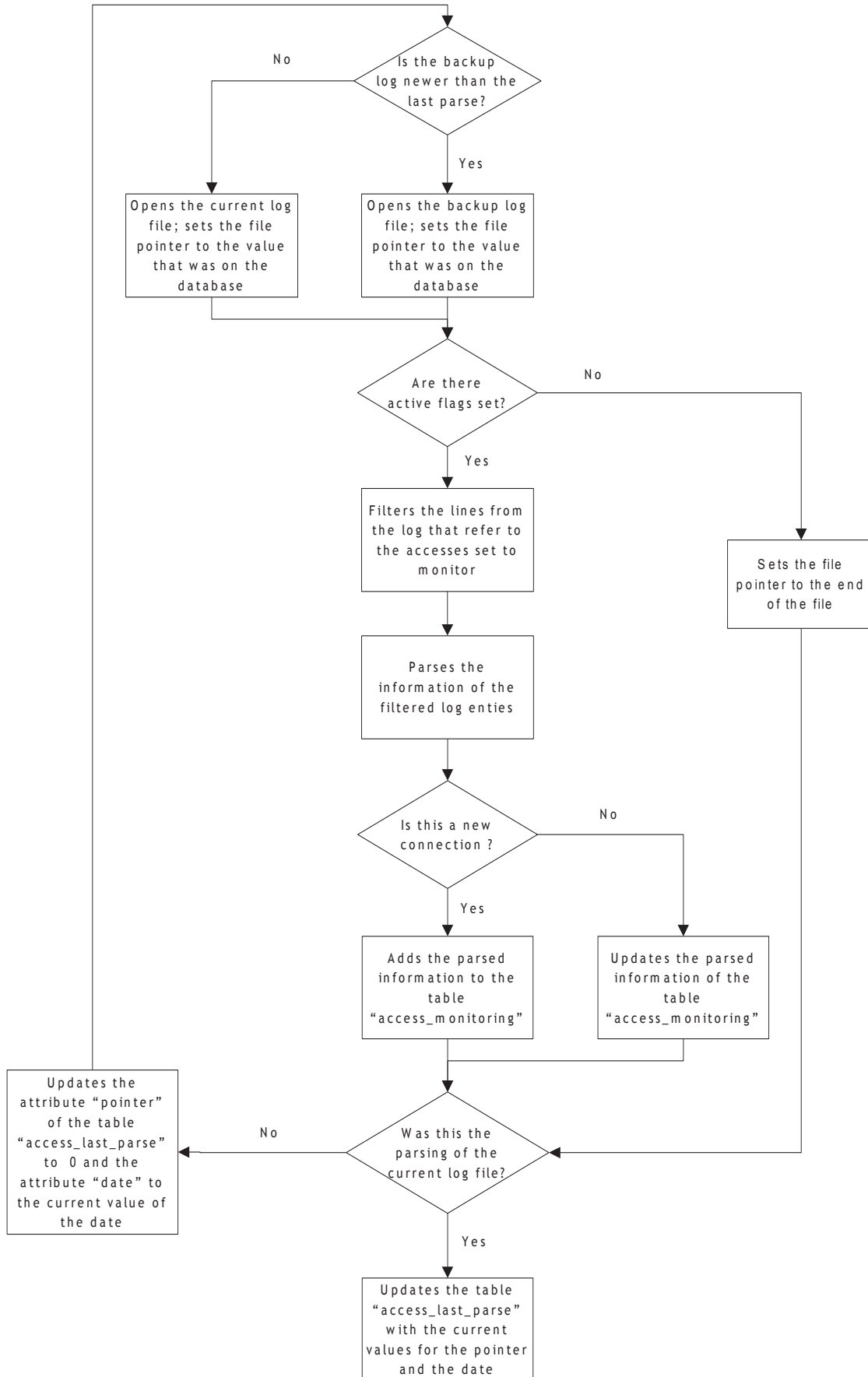


Figure 4.19: Diagram of the algorithm implemented in the PHP script

```
<? ... $active_logs[0]->active == "t" && strpos ($contents, "sshd(") ... ?>
```

Figure 4.20: Condition to filter the SSH log lines

If the monitoring of that particular access is active and if the identifier string as a match in the line being analyzed, the content of that line is copied to a multi-level array. This multi-level array has two levels of keys. The first is the PID of the program that generated the log. The second level key is the numbering of the lines that belong to the same PID. This is accomplished by parsing the PID information of the filtered line, using the `preg_match()` function, and passing its match as the first level key. In the case of the VPN SSL connections the identifier is the IP:port pair because its log doesn't show the PID value.

At this instance of the PHP script we have up to four two-level arrays. These arrays already have the log lines that belong to a certain PID grouped together.

Next, each array is tested if it exists. If it does, that means that the type of access which the array belongs to was marked to be monitored and that there was information in the log file about it. Thus the array is started to be parsed.

The parsing is done doing a two level iteration of the array. The first level cycles through the PID numbers that the array keys have. The second level iteration cycles through the log lines that were written by the process that had the PID.

In each second level iteration, the log line is tested against a specific identifying string. This string will be a common set of characters that belong to a log line that has relevant information for the monitoring of the particular access. As an example of this identifying string, Figure 4.21 shows the string used to identify the line that contains the information that a `root` login attempt, through SSH, took place.

```
<? ... $identifier="Failed none for root from ::ffff:"; ... ?>
```

Figure 4.21: Identifying string for the log line of a SSH `root` login attempt

Once again, the function `strpos()` was used to search the particular identifying string inside the log. This was done because it was assessed, by benchmarking, that it was much faster to first use the `strpos()` to identify if the log line had some useful information than to use the function `preg_match()` to identify and immediately parse the information of the line. This is because `preg_match()` uses the much more complex and system-consuming PCRE engine to do the matching.

When the log line is matched against the identifier, it is parsed to store its information in the table `access_monitoring` of the database. Having in mind that performance is an issue when running so many services at the same time, the use of the `preg_match()` function was deprecated. The benchmarks done showed an average increase of about 50% when other, less system-consuming functions were used to parse the file.

```

<? ...
else if (strpos ($data_line, "Connection closed by"))
    {
        $data_array = explode (" ", $data_line);
        $data_array = array_values(array_filter($data_array,
"validElement"));

        $ip = explode (":", $data_array[8] );
        $ip[3] = trim ($ip[3]);
        $ssh_access[$value] = array_merge($ssh_access[$value], array(
"disconnected_when" => $data_array[0]." ".$data_array[1]." ".date("Y")."
".$data_array[2], "ip" => $ip[3], "event" => 2));
    }
... ?>

```

Figure 4.22: Example of the PHP code to parse the log lines

As it can be seen in Figure 4.22, it was used “lighter” functions, like the `trim()` and `explode()` functions, to parse the information. This increased the complexity of the programming of code itself, as in cases of one having to take care of situations were the day as just one digit instead of two and therefore there’s more a space in the line, which changes the key of the array that has the information needed to put in the database. For a quick explanation of this example parsing code:

1. If the string `$data_line` contains the identifier string “Connection closed by”, this log line string is parsed for information.
2. The log line is divided by its space characters using the `explode()` function.
3. The resulting array is filtered by removing the empty values and renumbering the keys of the array. This is done by using the PHP function `array_filter()` that filters the array using a callback function. If the callback function returns `FALSE` the value of the array is discarded. The callback function `validElement()` returns true if the array value is bigger than 0.
4. Using another `explode()` the array value with the key 8 (ex.: `::ffff:192.168.69.24`) will be divided by the character “:”, resulting in the array `$ip`, that will have in it’s 4th element the IP plus the trailing empty space.
5. The trailing empty spaces are trimmed with the function `trim()`.
6. Next, the information parsed is added to a two-level array structure made to be easy to add its values to the database table. Figure 4.23 shows the first-level keys of this array, that are the PID values. The second level keys are the other attributes of the database table `access_monitoring`.

```
Array
(
    [13769] => Array
        (
            [connected_when] => Mar 13 2008 17:53:34
            [username] => administrator
            [ip] => 192.168.69.24
            [event] => 2
            [disconnected_when] => 13 2008 18:12:49
            [sent_bytes] => 1234532
            [received_bytes] => 9856
        )
    )
)
```

Figure 4.23: Example of an array for the VPN PPTP after the parsing

In the end of the parsing each four arrays (one for each access type) are cycled through a `foreach()` function. Each first level element is tested to see if it has the second level key `connected_when`. If it doesn't, the function `updateAccessByPID()` is called to update the connection that already has an entry in the database table `access_monitoring`. In the case were the array has a parsed new connection it calls the function `insertAccess()`, which inserts the connection entry in the database.

Subsequently, if the log file analyzed was the current one, the script calls the function `updateLastParseInfo()` that will update the database table `access_last_parse` with the present values for the file position indicator and date. By the other hand, if the opened file system pointer refers to the backup log, the table is updated with the present date and the value 0 for the `pointer` attribute. The script will start again to check if the backup file is newer than the value in the database. This time the value in the database will be newer than the backup file and the script will open the current log file and parse it from the beginning.

4.2.3. System implementation

All of the four configuration files of the services to be monitored were customized in order to output to `syslog` just the essential debug needed to track the accesses. So, it was needed to change the debug parameter in each of the services configuration file.

The script explained in the last subsection was made to be run each 10 minutes. This way the load on the machine remains low for only few lines are parsed each time the import script is run. Using the `crontab` file of IPBrick services, it was added a line to run the import script.

4.2.4. Web interface

For this functionality it was created two web pages in the configuration interface of IPBrick. It was created a webpage for easily managing which of the services have its monitoring active (Figure 4.24). In this webpage it is displayed a table that shows all of the services possible to be monitored and its present state. To change the state one has just to click the link that has the name of the service. This link leads to a second page where there is a drop-down style box, inside a HTML form, with the available states for the monitoring of the selected access type. After selecting the desired state, the user presses the submit button making the monitoring state of the selected type to change and for the main page of the access monitoring management to be displayed with the updated state.

For the display of the records of the accesses monitored it was used, as a model, a previous developed web page for displaying statistics that uses the Prototype (Prototype Core Team 2007), the scriptaculous (Fuchs n.d.) and the MochiKit (Mochi Media, Inc. 2006) frameworks, as well the SLists PHP class (Neustaetter 2006). After customization of the previously mentioned page, the access monitoring records page uses these frameworks to switch between the filtering parameters and the records (as well to change the links from “Filter” to “Records”), to sort the columns of the table of records and to do the Ajax database requests each time the user filters the displayed information.

The screenshot shows the IPBrick web interface. The top header displays the IPBrick logo and version v4.3. The sidebar on the left contains navigation options under 'Advanced Configurations' and 'Apply Configurations'. The main content area is titled 'Monitoring » Accesses » Management' and contains a table with the following data:

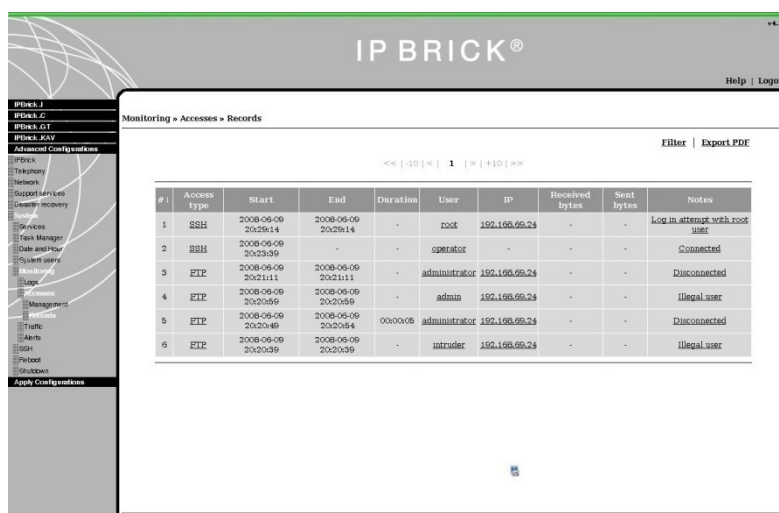
State	Services
Enabled	SSH
Enabled	FTP
Disabled	VPN PPTP
Disabled	VPN SSL

Figure 4.24: Management of the access monitoring page

The filtering parameters that can be set through the filter form are:

- The access types to display
- The IP address
- The username
- The accesses active in a date interval. For this it was used the DHTML/JS Calendar (Bozan 2008) to make it easy to set the date interval, by showing a calendar where the user can select the dates from it.
- The access note to be displayed (“Connected”, “Illegal password”, “Timeout”, etc.)

When the filtering parameters are set, the user presses the “Filter” button and the parameters of the filter are read by a JavaScript file. This file does an AJAX request that calls a PHP script that constructs the database query accordingly to the parameters of the filter, queries the database and generates an XML file with the information retrieved from the database. A function is then called that parses the XML file and updates the displayed web page, hiding the filter and showing a table with the parsed XML information. The resulting table is shown in Figure 4.25.



#	Access type	Start	End	Duration	User	IP	Received bytes	Sent bytes	Notes
1	SSH	2008-08-09 20:29:14	2008-08-09 20:29:14	-	root	192.168.08.24	-	-	Log in attempt with root user
2	SSH	2008-08-09 20:29:39	-	-	operator	-	-	-	Connected
3	FTP	2008-08-09 20:21:11	2008-08-09 20:21:11	-	administrator	192.168.08.24	-	-	Disconnected
4	FTP	2008-08-09 20:20:59	2008-08-09 20:20:59	-	admin	192.168.08.24	-	-	Illegal user
5	FTP	2008-08-09 20:20:49	2008-08-09 20:20:54	00:00:05	administrator	192.168.08.24	-	-	Disconnected
6	FTP	2008-08-09 20:20:39	2008-08-09 20:20:39	-	intruder	192.168.08.24	-	-	Illegal user

Figure 4.25: Page for viewing the records of the access monitoring

In this table, the System Administrator can further filter the displayed information by simply clicking the item he wants to filter the table by. Clicking the values presented in the columns “Access type”, “User”, “IP” or “Notes”, a JavaScript function is called that adds the value clicked to the previous filter parameters and runs the AJAX call explained earlier, with the additional parameter. This way the user can easily narrow down the initial search.

The access monitoring records page is also shown in the interface for the FTP, VPN PPTP and VPN SSL services. When accessed through the links presented in these interfaces, the records shown will be just the ones referring to that particular service.

4.2.5. PDF report

In each of the pages of the access monitoring records there is a link that the user can press in order to generate a PDF report of the records that are currently displayed. Figure 4.26 is an example of a generated PDF report.

Report created by IPBriick - iPortalMais at 2008-Jun-09 20:29

Filter parameters upon report creation									
Access(es):		SSH, FTP, VPN PPTP and VPN SSL							
User:		IP:		Notes:					
Active accesses from:				to:					

No.	Access type	Start	End	Duration	User	IP	Received bytes	Sent bytes	Notes
1	SSH	2008-06-09 20:29:14	2008-06-09 20:29:14	0h:0m:0s	root	192.168.69.24			Log in attempt with root user
2	SSH	2008-06-09 20:23:39		0	operator				Connected
3	FTP	2008-06-09 20:21:11	2008-06-09 20:21:11	0h:0m:0s	administrator	192.168.69.24			Disconnected
4	FTP	2008-06-09 20:20:59	2008-06-09 20:20:59	0h:0m:0s	admin	192.168.69.24			Illegal user
5	FTP	2008-06-09 20:20:49	2008-06-09 20:20:54	0h:0m:5s	administrator	192.168.69.24			Disconnected
6	FTP	2008-06-09 20:20:39	2008-06-09 20:20:39	0h:0m:0s	intruder	192.168.69.24			Illegal user

Figure 4.26: PDF report for the access records shown in Figure 4.25

Clicking the link, a JavaScript function is called, which builds a link to a PHP file, passing the filter parameter by the GET method. This function then opens a browser window with the built link.

The PHP file uses the filter parameters to query the database for the same records that were displayed in the records page, with which (after some information processing) it creates a CSV file, storing it in a temporary directory. This CSV file will be the actual information that the table of the PDF report will contain.

The process of generating the PDF file is done by the included FPDF library (Plathey 2004). After extending the FPDF class included in this PHP script in order to customize the two tables present in this report, the functions of this library are called. With these functions the header table, containing the filter parameters passed to this script, is created and the CSV file is read, parsed and used to fill the table of the records monitored. Before using a FPDF function to output the resulting PDF to the previously opened window, the CSV file is deleted from the temporary file.

4.3. Firewall rules ordering

The page for ordering the firewall rules was modified so that a user can easily drag and drop the firewall rules to the desired place and then, when pressing the submit button, all the changes are written to the database table `firewall`, that later will be used to write the firewall script file.

State	Rule	Interface	Protocol	Module	Source IP	Port	Destination IP	Port	Identifier	Policy
enable	INPUT	eth1	udp					5080		ACCEPT
disable	INPUT	eth0	tcp		192.168.69.0/24			194		DROP
disable	INPUT	eth0	tcp		192.168.69.0/24			5180		DROP
enable	INPUT			connmark					--mark 10000	DROP
enable	INPUT	eth0			0/0		0/0			ACCEPT
enable	INPUT	lo			127.0.0.0/8		127.0.0.0/8			ACCEPT
enable	INPUT	lo			192.168.69.178					ACCEPT
enable	INPUT	eth1	tcp		0/0		10.0.0.253	80		ACCEPT
disable	INPUT	eth1	tcp		0/0		10.0.0.253	443		ACCEPT
enable	INPUT	eth1	tcp		0/0		10.0.0.253	25		ACCEPT
enable	INPUT	eth1	tcp		0/0		10.0.0.253	110		ACCEPT
enable	INPUT	eth1	tcp		0/0		10.0.0.253	143		ACCEPT
disable	INPUT	eth1	tcp		0/0		10.0.0.253	22		ACCEPT
disable	INPUT	eth1	tcp		0/0		10.0.0.253	20		ACCEPT
disable	INPUT	eth1	tcp		0/0		10.0.0.253	21		ACCEPT
enable	INPUT	eth1	icmp						--icmp-type echo-reply	ACCEPT
enable	INPUT	eth1	icmp						--icmp-type destination-unreachable	ACCEPT
enable	INPUT	eth1	icmp						--icmp-type time-exceeded	ACCEPT
enable	INPUT	eth1	tcp					1723		ACCEPT
enable	INPUT	eth1	47							ACCEPT
enable	INPUT	ppp+			192.168.69.0/24		192.168.69.0/24			ACCEPT
enable	INPUT	eth1	tcp		0/0	80			!-syn	ACCEPT
enable	INPUT	eth1	tcp		0/0	443			!-syn	ACCEPT
enable	INPUT	eth1	tcp		0/0	25			!-syn	ACCEPT

Figure 4.27: Firewall rules ordering interface

Using the scriptaculous framework (Fuchs n.d.) and the PHP wrapper class SLLists (Neustaetter 2006), each “tr” tag of the HTML table is transformed into a “dragable” item (highlighted line in Figure 4.27). Each line of the HTML table is also given a specific identification using the “id” attribute. This is the initial position of the rule in the rules list.

For the submit of the new order of the firewall rules, it is used a customized version of the `printForm()` function of the SLLists, which uses a scriptaculous function to read the present order of the rules; then it prints an HTML form that passes this new order, by the POST method, to the action PHP script, called upon submit.

The PHP action script called previously does a database query, retrieving the values of the attribute `order` of the table `firewall`, in ascending order. Through a `for()` cycle it is checked if there is some change in the rules order, comparing the initial position of the rule in the list (through the previously mentioned “id”) with the actual position. If the initial position is different from the new, a PHP function is called that updates the rule entry in the database; changes it to the value of the `order` attribute that the previously rule in that position had.

This way, in the end of the cycle, all the `order` attributes are updated without changing IPBrick's value scheme for ordering of the firewall rules; this way it is assured the integrity of the firewall system.

4.4. VPN SSL access policies

4.4.1. System implementation

The first step for the implementation of this development was to create an additional directory for the client configuration files, which will contain the defined fixed IP for each VPN SSL certificate having access policies. This client configuration directory was created in the path `/etc/openvpn/1/ccd/`.

The configuration file of the OpenVPN server, `server.conf`, was also modified so that it includes the client configuration files. As the OpenVPN server configuration file is generated by IPBrick, this modification just implied adding a supplementary entry in the database table `vpnssl` - the one used by IPBrick to generate the previously mentioned file.

4.4.2. Database

For implementing the VPN SSL access policies it was necessary to create four new tables in the database of IPBrick, namely, `vpnssl_policies`, `vpnssl_policies_members`, `vpnsslcert_policies` and `vpnsslcert_firewall`. It was also needed to add the column `ip`, to the table `vpnsslcert`. The scheme of this database implementation is presented in Figure 4.28; the arrows indicates a relationship of N:1 between two database tables.

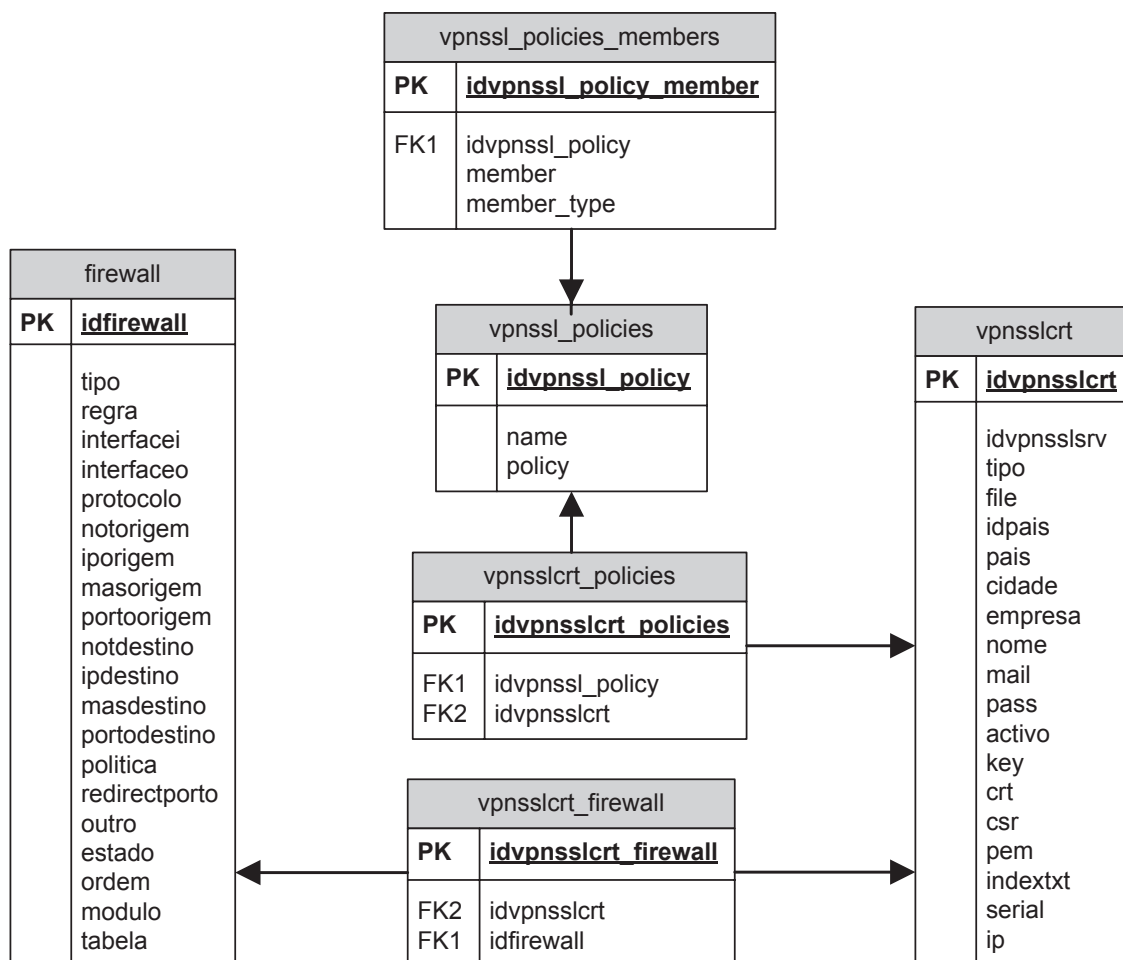


Figure 4.28: Database scheme for the VPN SSL access policies

In the above figure we can see that the main table of this implantation is the `vpnssl_policies` table, which is the one that contains the numerical identification of the policies. The `name` column will be the one containing the user-given names for the created policies, while the `policy` column will contain one of two numerical values: 1 for the permission policies or 2 for the restriction policies.

As each policy can have several members and even several member-types, there's a relationship of 1:N between the table `vpnssl_policies` and the table `vpnssl_policies_members`. The attributes of the `vpnssl_policies_members` table and its functional meaning are presented next:

- `idvpnssl_policy_member`: the numerical identification of the entry of each policy member.
- `idvpnssl_policy`: it's the foreign key from the table `vpnssl_policies` and it indicates to which policy the member belongs to.
- `member`: it's the actual value of the member that IPBrick will use to identify it. It can take the form of a UID number for IPBrick's registered machine, a GID for

registered machine groups, an IP subnet (ex.: 192.168.69.128/25), an IP value for a machine defined by its IP or an IP range (ex.: 192.168.69.202–192.168.69.240).

- `member_type`: like the name implies, is the type of member that the database entry refers to. It can take the following values:
 - 3: registered machine groups
 - 4: registered machines
 - 5: IP subnet
 - 6:IP defined machine
 - 7:IP range

The table `vpnsslcert` was an already existing table in IPBrick's database. This table contains the certificates for the VPN clients as well all the information related to them. Now that all the certificates that will have policies will be forced to have a particular virtual IP address, it is necessary to associate the IP with the certificate in question. To manage this it was added the column `ip` to this table. In this new attribute it is kept the reference to what subnet the particular certificate was assigned, being the real IP address calculated later on.

The table `vpnsslcert_policies` is a connection table for creating the relation of N:N between the tables `vpnssl_policies` and `vpnsslcert`. This relation exists because each certificate can have various policies associated to it; as well each policy can be associated with various certificates. Thus, this table has the foreign keys `idvpnssl_policy` and `idvpnsslcert` referring the tables named previously.

The table `firewall`, shown in the database model, already existed in IPBrick's database and was not modified. The relation between the tables `vpnsslcert` and `firewall` was implemented by adding the table `vpnsslcert_firewall`. This way IPBrick keeps the records of the firewall rules added to implement the VPN SSL access policies.

4.4.3. PHP script

For the implementation of the VPN SSL access policies it was needed to create various functions that could be called in the various events that changes the configuration of the VPN SSL access policies.

The events that change the VPN SSL are:

- Certificate creation
- Certificate revocation
- Certificate modification
- Policy member modification
- Modifying the IP of a IPBrick registered machine
- Modifying an IPBrick group of machines
- Erasing an IPBrick registered machine
- Erasing an IPBrick group of machines

Upon creating of a VPN SSL client certificate, the function `vpnssl_build_firewall()` (described in Figure 4.29) is called having as reference the new id of the created VPN SSL client certificate, retrieved from the database. This was the main function created. It searches and retrieves the necessary information from all the policies that the user set to be active in the created certificate and builds the needed firewall rules.

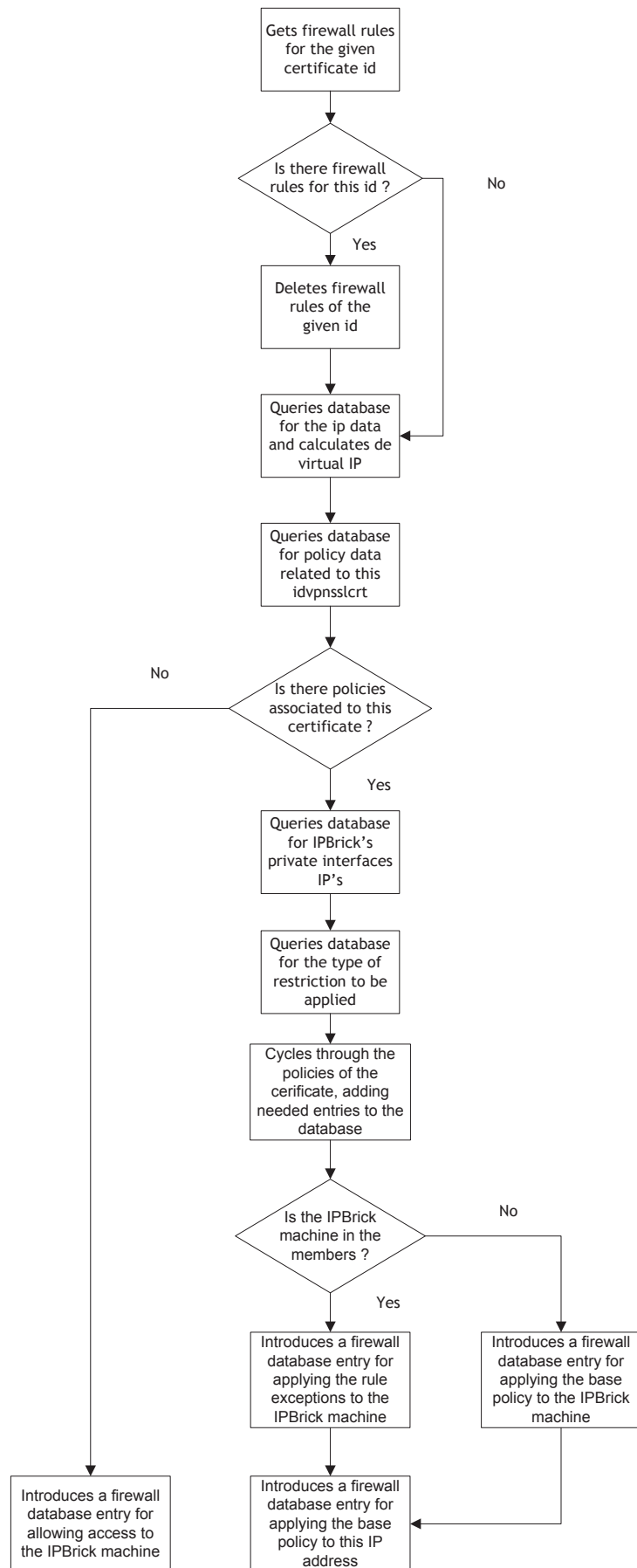


Figure 4.29: Diagram of the algorithm used to implement the policies to one certificate

In Figure 4.30 is shown in detail how is implemented the block that cycles through the policy members and creates the firewall rules.

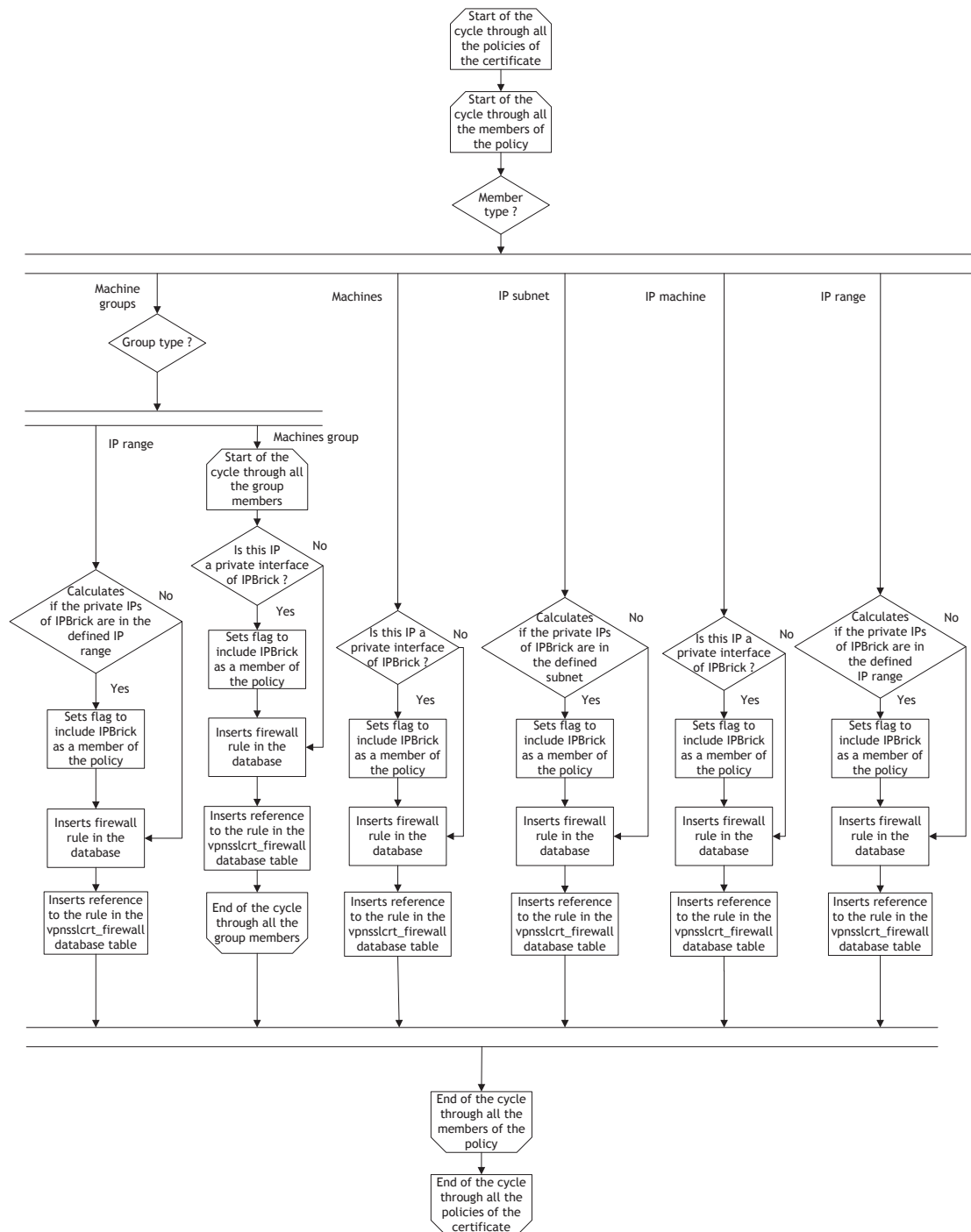


Figure 4.30: Diagram of the algorithm used to cycle through the policies affected to the certificate and to add the needed entries to the database

When a certificate is changed, it is only needed to call this function to update the firewall rules that implement the desired set of policies. In the case of the certificate being revoked, the status of the certificate in the database table `vpnsstcrt` is set to revoked and all the

entries of this certificate in the table `vpnsslcert_policies` are deleted. Thus, upon calling this function it will only erase the firewall rules from the database.

In the other cases the situation is more complex. As the actions of modifying or deleting groups and machines are spread throughout a variety of services in IPBrick, it was created a set of function to deal with these situations.

When a machine group is modified or erased, a function is called that queries the database for all the certificates that the modified machine group included. Then it checks whether if the group was erased or just modified. If erased, it erases all the references to the former machine group from the table `vpnssl_policy_members`. Then, in a `for()` cycle, it calls the function `vpnssl_build_firewall`, passing as argument the id of the certificates resultant from its first query.

In the case of a machine IP change or deletion it's even more complex. A function is called that checks if the computer belongs to any machine group. If so, it retrieves the id of the certificates that have polices containing that group(s). It then checks if there is any certificate with policies having the machine in question. If the machine was deleted it erases all the references to the former machine(s) from the table `vpnssl_policy_members`. Having at this point the arrays of ids of certificates that have the machine as individual and as in part of a machine group, it merges the two arrays and filters the duplicated values out of it. Using then a `for()` cycle, it calls the function `vpnssl_build_firewall`, passing as argument the id of the certificates present in the merged array.

4.4.4. Web interface

It was created an interface, within the page of the VPN SSL service, where the System Administrator can view the existent policy, separated by the categories "Permission Policies" and "Restriction Policies". In this interface it's possible to add, modify and delete the policies. The configuration/modification page is presented in Figure 4.31. In it can be modified all the possible members of the selected policy.

Upon creation of the certificate, the System Administrator can choose the permissions to associate to the certificate by a simple interface that lists the permissions available in a box and lets the user to selected them and associate them to the certificate, appearing in the box of the policies associated with the certificate (Figure 4.32).

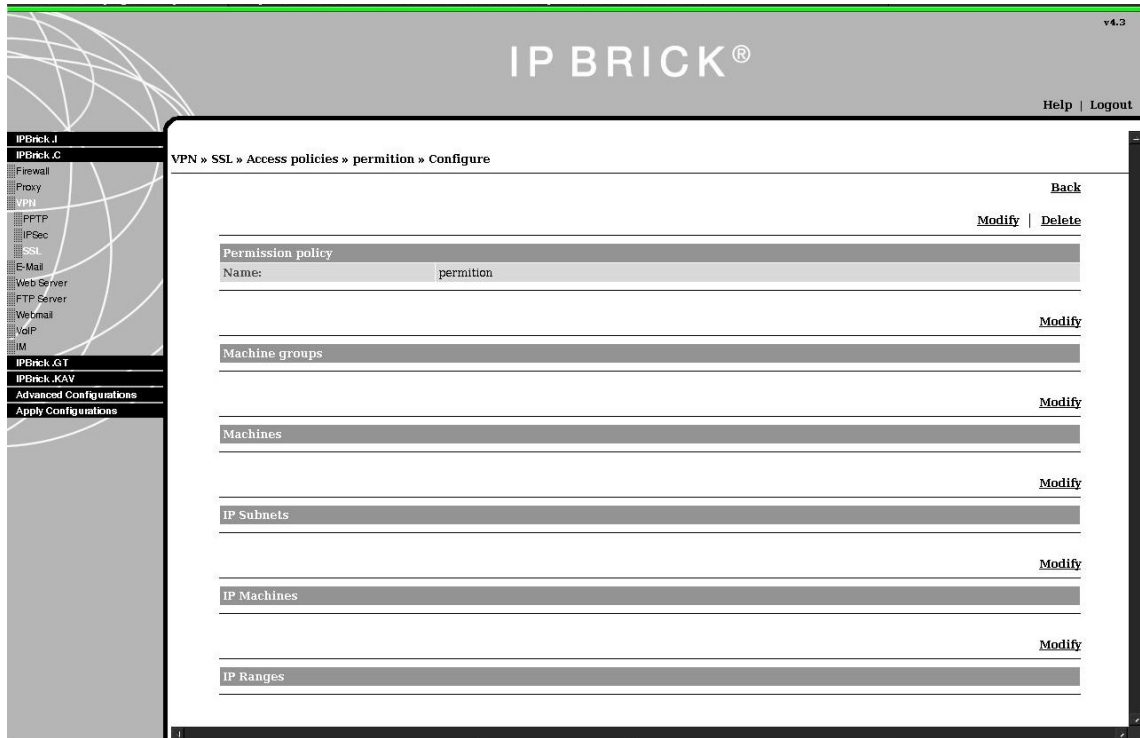


Figure 4.31: Interface for adding the policy members

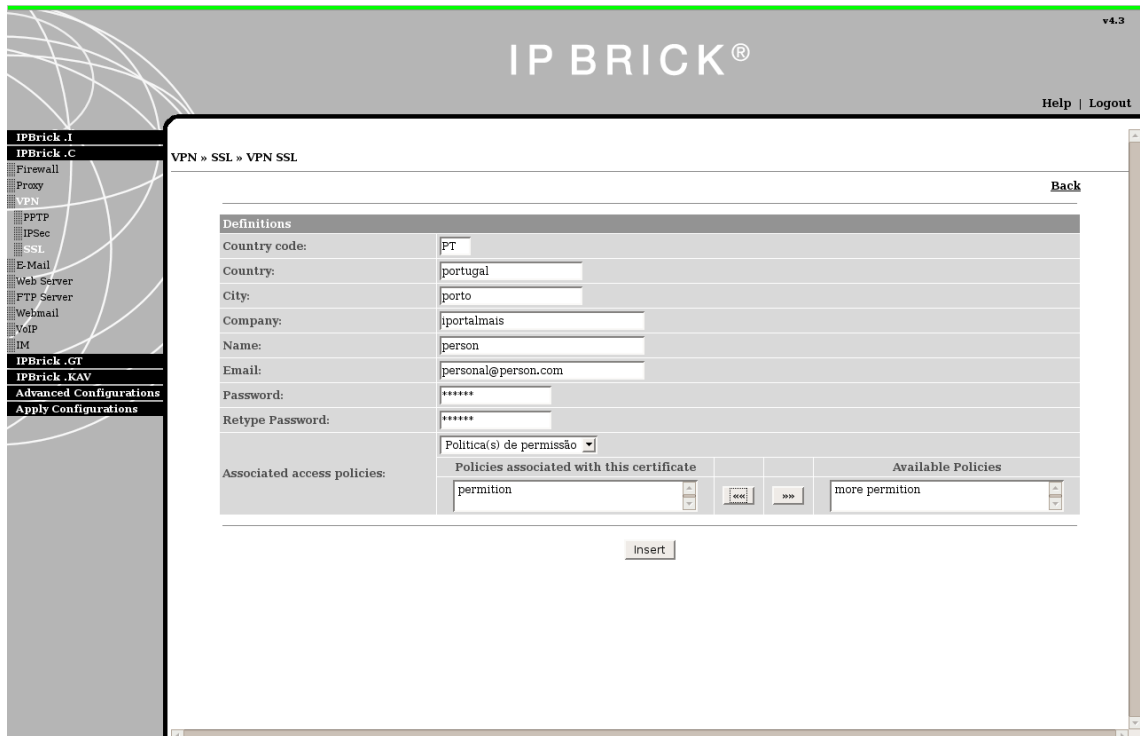


Figure 4.32: Interface for creating a VPN SSL certificate

Chapter 5

5. Tests and Results

5.1. Testbed

IPBrick is a software that not only offers efficiency and reliability, but also aims to ensure fast throughput.

Being this project integrant part of this software, CPU load and code optimization, as it was described before, were two concepts always taken in consideration during the development of this project.

To verify that this development meets the standards of all IPBrick's development, it was used as test server a SOHO appliance, with the following modest characteristics:

- Intel Celeron processor at 1.7 MHz
- 256 MB SDRAM memory
- 4x Ethernet 10/100 Mbps
- 2x USB ports

With the latest version of IPBrick installed, and with the new developments that are part of this project, this machine was the gateway for 14 development workstations of iPortalMais during the tests. IPBrick was run having it's default services running.

Figure 5.1 is the graphical representation of the scheme of this testbed. Although it appears two computers connected to the internet, these were only used in the tests for the access monitoring development and for the VPN SSL access policies. For the active connections development tests it was only used one computer connected to the internet.

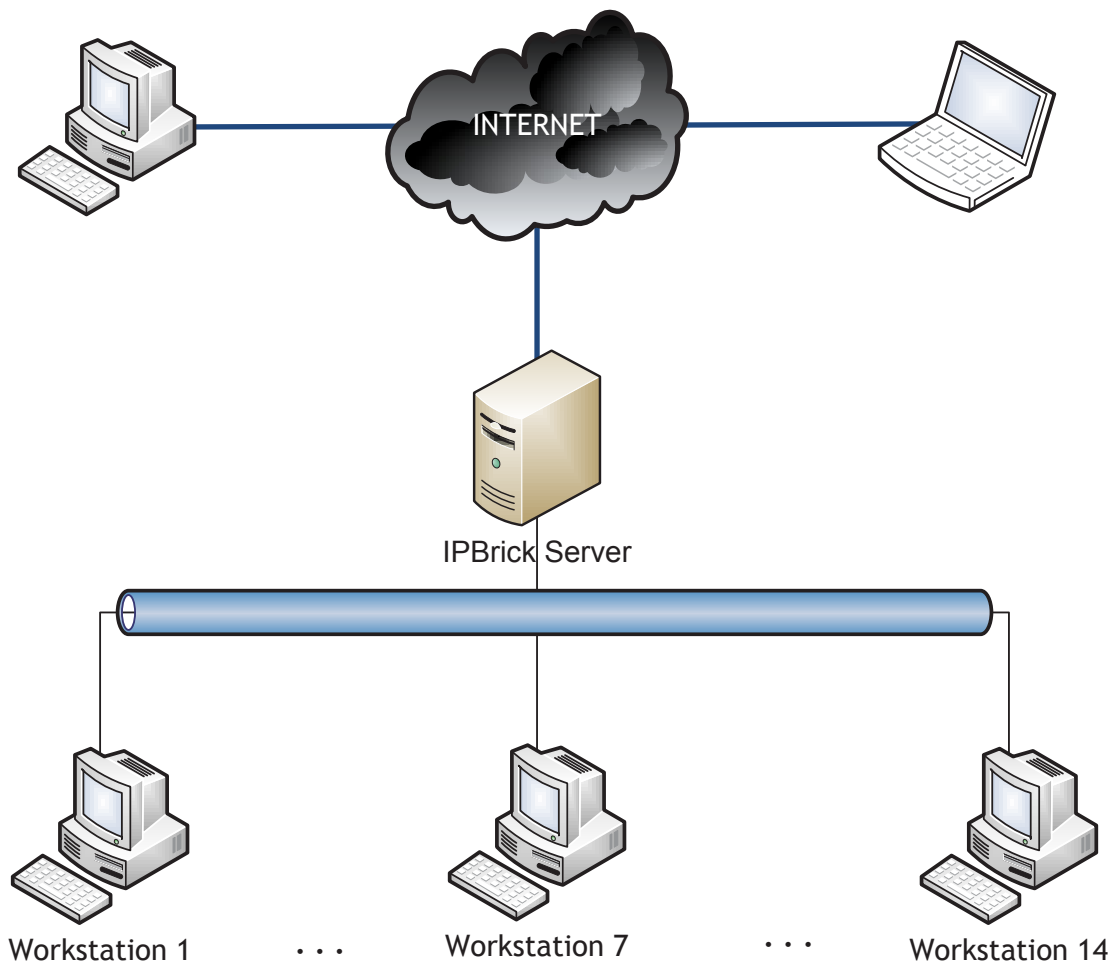


Figure 5.1: Testbed schema

5.2. Active connections tests

Tested during development and in a posterior test phase, it was possible to assert that all the connections tested were blocked by this new IPBrick feature. SSH sessions, VPN tunnels, FTP sessions, among others where part of this successful test.

Although it was created a tool to virtually block any type of connection, the goal of this part of the project was to successfully block, on the fly, the TCP/IP protocol connection type. As so, it will be described one of the tests made to acknowledge that, in fact, it can block TCP/IP protocol connections.

1. In IPBrick, it was permitted the SSH access through the public interface.
2. In the machine connected to the internet, it was started two SSHs session in the IPBrick server.

3. In the remote machine it was issued the command `top -d.1` in both shells, to allow a visible refresh of information from the server.
4. In IPBrick's interface, one of the connections with the IP of the remote machine and with destination port 22 (default port for SSH), was set to be blocked.
5. Roughly after one second, one of the shells froze.
6. After setting the second port 22 connection to be blocked, both shell, in the internet connected machine, are frozen.
7. Using `ngrep`, in IPBrick, it can be seen that no connections with the source port 22 is being sent to the remote machine.
8. After clicking the unblock link in IPBrick's interface, first in one, then in both connections, it was seen that each remote SSH shell started again to refresh.

5.3. Access monitoring tests

This development was run against three different sets of tests. One, it was tested the speed and CPU consume of the import script. Other was to assess if it correctly displayed the information that it imported from the syslog file. The last test to this development was the generation of the PDF report.

During several days it was made all four kinds of accesses to IPBrick, being some of them manually recorded in a separate log file, so to compare it with the displayed information for appraising.

This way it was able to be confirmed the correctness of the imported log information.

From the log files generated from the previous testing, it was created worst case scenario log files, i.e., each log file had the maximum size before it would be rotated, corresponding to 1 Megabyte and all the lines of the log files were fitted to be matched and parsed while IPBrick was configured to monitor all the four access types.

During different times of a day the import script was made to run on that machine, to always analyze the same test files.

It took an average of proximally 3.45s to complete the script execution in a machine that had a CPU load average of 0.8 at tests time.

5.4. Firewall rules ordering tests

The most presentable result of this development is the interface itself (Figure 4.27), which successfully allows the System Administrator to order the firewall rules in a much more comfortable and practical way.

5.5. VPN SSL access policies tests

Because of the extended integration that this development was subjected to, this was the longest and most complex test made.

At first two policies were created, one restriction policy and one permission policy, with a random set of policy members. After this, was created two client certificates, for the two machines connected directly to the internet, having one certificate the permission policy associated with it and the other the restriction policy.

The two machines successfully started a VPN SSL tunnel to the IPBrick server.

After this connection being established it was performed a series of tests to evaluate if all the functionalities of this development were working correctly. The following configuration changes were performed while the machines were connected to the server, as the policies are implemented through IPBrick's firewall; like this, one is able to change the policies associated to the certificates without having to interrupt the active connection. After introducing the configuration changes it was check if they made effect by pinging and establishing a SSH connection to the affected workstations from the VPN SSL client machines.

1. It was randomly changed each type of policy members, both from the restriction and from the permission policies.
2. It was removed and added the policies to the certificates, to test if without policies the VPN SSL client as access to the entire network.
3. It was created new policies and added to the certificates, in order to test the certificates with multiple policies associated.
4. To one of the certificates was removed all members, and then added the member that was the IP of the server itself. To the other it was removed all members and then added an IP subnet that was the one of the server. This was done in order to test if the code regarding the checking of the server IP in the policy members was correctly working.
5. It was associated with a certificate a policy that only had one member. This member was an IPBrick registered machine group. In all the interface pages that perform action upon the VPN SSL access policies (see Table 3.5), except for the

network interface management page, it was performed a change of IP and/or a deletion of an IPBrick registered machine that belonged to the machine group defined on the certificate. This was done both to assess if the functions were well implanted in this web pages of the services, as well to check if the part of the code that checks which policies are affected by these changes, and calls the function to reconstruct the VPN SSL firewall rules, is working correctly.

These tests were all performed with a positive outcome, confirming that, in all the above described situations, this development is working correctly.

There were more three situations needed to be tested, these ones or implying that the VPN SSL clients would not be connected at the time of these changes in IPBrick's configuration or not being directly related to the connected VPN SSL clients. These tests were:

1. The IP for the private network interface of IPBrick was changed. This was done to check if the firewall rules were updated accordingly with this change.
2. It was tried to set the VPN SSL network to have a mask of 30, and to have a mask of 28 when there were 5 active certificates. Like this, it was tested the implemented protection against erroneous configuration of the VPN SSL network value.
3. Having the VPN SSL network a mask of 28, it was tried to create a 4th certificate. The protection for the creation of more certificates than the defined subnet VPN SSL network mask should not allow this.

Again, all the above described tests made resulted in a positive appraisal of the integrity of this development.

After performing these various tests during development and in a posterior test phase it was possible to confirm that the functionalities of this development enable a System Administrator to add another "Ring of Security" to its VPN SSL gateway.

Chapter 6

6. Conclusion and Future Work

It was studied in this work ways to meet the strict demands of iPortalMais's client for a security sub-system for IPBrick.

Various questions were approached and from the various studied choices it was implemented four developments were implemented and integrated in IPBrick: a “on the fly” connection blocker using an integration of the `conntack-tools` program with IPBrick's firewall; it was added to IPBrick the capability to do monitoring of accesses, using a PHP script completely developed from scratch; the firewall rules ordering interface improvement and associating of access policies to the VPN SSL client certificates.

Therefore it was with success that the customer wishes were fulfilled. In the case of the demand to block connections the expectations were even surpassed, as the implemented solution is able to block “on the fly” virtually any type of connection/protocol that is passing through the IPBrick machine. Also with access monitoring, apart from the required information to display, it was possible to display some notes about the why of the disconnection of the access.

Even so, the possibilities of the expansion of the functionalities are huge. Using the “on-the-fly” connection blocker together with an Intrusion Detection System we could create a adaptive firewall. When the Intrusion Detection System would detect an attempt of intrusion we could set a trigger that would update the respective connection tracking entry and virtually the connection would be terminated immediately.

Also in the case of the access monitoring, this work could be the starting point of a extensive monitoring system of almost every services that IPBrick has, such as the Samba Server, Apache Server, etc.

This solution has the particularity of being inserted in IPBrick which is a Portuguese software directed to the easiness of use but with service quality always in mind.

With this IPBrick is even more feature reach and ready to take a stand on the competitive IT market of nowadays.

References

Andreasson, Oskar. *Iptables Tutorial 1.2.2*. 2006. <http://iptables-tutorial.frozentux.net/iptables-tutorial.html> (accessed Junho 2007).

Ayuso, Pablo Neira. 2008. <http://contrack-tools.netfilter.org/index.html> (accessed March 2008).

—. "Test Case." *contrack-tools: Connection tracking userspace tools for Linux*. 2007. (accessed March 2008).

Barnum, Sean, and Amit Sethi. *Attack Patterns as a Knowledge Resource for Building Secure Software*. Cigital, Inc., 2007.

Barrett, Daniel J., Richard E. Silverman, and Robert G. Byrnes. *Linux Security Cookbook*. 1st edition. California: O'Reilly, 2003.

Black Hat. "Routing & Tunneling Protocols Attacks." *BlackHat Briefings*. Amsterdam: Black Hat, 2001.

Bozan, Mihai. *The DHTML / JavaScript Calendar*. 2008. <http://www.dynarch.com/projects/calendar/> (accessed April 2008).

Friedl, Jeffrey E. F. *Mastering Regular Expressions*. Sebastopol, California: O'Reilly & Associates, 1997.

Fuchs, Thomas. *script.aculo.us it's about the interface, baby!* <http://script.aculo.us/> (accessed April 2008).

IANA, Jon Postel, David Johnson, Tom Markson, Bill Simpson, and Zaw-Sing Su. "ICMP TYPE NUMBERS." *iana: Internet Assigned Numbers Authority*. February 13, 2008. <http://www.iana.org/assignments/icmp-parameters> (accessed June 2008).

IBM ISS. "IBM X-Force 2007 Trend Statistics Report." *IBM Internet Security Systems*. 2008. http://www.iss.net/documents/literature/x-force_2007_trend_statistics_report.pdf (accessed May 2008).

iPortalMais - Soluções de Engenharia para Internet e Redes, Lda. *IPBrick - Manual de Referencia*. Porto: iPortalMais - Soluções de Engenharia para Internet e Redes, Lda., 2006.

—. *IPBrick Unified Communications over IP*. 2008. <http://www.ipbrick.com/> (accessed June 2008).

Lowth, Chris. *Introducion "Cutter 1.03"*. April 2005. <http://www.lowth.com/cutter/> (accessed March 2008).

Mochi Media, Inc. *MochiKit*. April 2006. <http://mochikit.com> (accessed April 2008).

Netfilter Core Team. "The netfilter.org "iptables" project." 2008. <http://www.netfilter.org/projects/iptables/index.html> (accessed March 2008).

Netfilter core team. *The netfilter.org "libnetfilter_contrack" project*. 2007. http://www.netfilter.org/projects/libnetfilter_contrack/index.html (accessed February 2008).

—. *The netfilter.org "libnfnetlink" project*. 2007. <http://www.netfilter.org/projects/libnfnetlink/index.html> (accessed February 2008).

Neustaetter, Greg. *Scriptaculous Lists with PHP*. August 2006. <http://www.gregphoto.net/sortable/> (accessed April 2008).

Plathey, Olivier. *FDPF Library*. December 2004. <http://www.fpdf.org/> (accessed May 2008).

Porteneuve, Christophe. *Prototype and script.aculo.us*. USA: The Pragmatic Programmers LLC., 2007.

Prototype Core Team. *Prototype JavaScript framework*. 2007. <http://prototypejs.org/> (accessed April 2008).

Rash, Michael. *Linux firewalls : attack detection and response with iptables, psad, and fwsnort*. No Starch Press, inc., 2007.

Richardson, Roberts. "CSI Survey 2007." *Computer Security Institute*. 2007. <http://i.cmpnet.com/v2.gocsi.com/pdf/CSISurvey2007.pdf> (accessed June 2008).

Telethra, Inc. 2008. <http://openvpn.net/> (accessed April 2008).

The PHP Group. *PHP Manual*. 2001-2008. <http://www.php.net/manual/en/index.php> (accessed March 2008).

Toxen, Bob. *Real world Linux security*. 2nd Edition. New Jersey: Prentice Hall PTR, 2003.

Turnbull, James. *Hardening Linux*. Apress, 2005.

Appendix A

ip_contrack_tcp.h

```
#ifndef _IP_CONNTRACK_TCP_H
#define _IP_CONNTRACK_TCP_H
/* TCP tracking. */

enum tcp_contrack {
    TCP_CONNTRACK_NONE,
    TCP_CONNTRACK_ESTABLISHED,
    TCP_CONNTRACK_SYN_SENT,
    TCP_CONNTRACK_SYN_RECV,
    TCP_CONNTRACK_FIN_WAIT,
    TCP_CONNTRACK_TIME_WAIT,
    TCP_CONNTRACK_CLOSE,
    TCP_CONNTRACK_CLOSE_WAIT,
    TCP_CONNTRACK_LAST_ACK,
    TCP_CONNTRACK_LISTEN,
    TCP_CONNTRACK_MAX
};

struct ip_ct_tcp
{
    enum tcp_contrack state;

    /* Poor man's window tracking: sequence number of valid ACK
       handshake completion packet */
    u_int32_t handshake_ack;
};

#endif /* _IP_CONNTRACK_TCP_H */
```