UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

LEANDRO LISBOA PENZ

Coherence in distributed packet filters

Thesis presented in partial fulfillment
of the requirements for the degree of
Master of Computer Science

Prof. Dr. Raul Fernando Weber
Advisor

Porto Alegre, Porto Alegre, November 2008

CIP – CATALOGING-IN-PUBLICATION

Penz, Leandro Lisboa

Coherence in distributed packet filters / Leandro Lisboa Penz. – Porto Alegre: PPGC da UFRGS,

.

94 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS,

. Advisor: Raul Fernando Weber.

I. Weber, Raul Fernando. II. Título.

CONTENTS

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF FIGURES

ABSTRACT

Computer networks are under constant threat, even more when connected to the Internet. To decrease the risk of invasions and downtime, security devices such as the packet filter are deployed. As a first layer of security, the packet filter is responsible for blocking out unwanted traffic at key network locations. The packets dropped or forwarded by the filter are defined by a set of rules programmed by the network administrator. These rules are in the form of guarded commands, each with a condition and a decision section.

As the number of services and networks grow, the number of rules tend to grow as well. Beyond a certain threshold, the complexity of maintaining such a large and distributed set of rules becomes a burden for the network administrator. Mistakes can be easily made, compromising security.

This work develops the concept of "anomaly", each representing a potential problem, a contradiction or a superfluous rule in the rule set; i.e. a warning to the system administrator. There are 7 types of anomalies divided in two groups: single filter anomalies and networked anomalies. The single-filter anomalies warns the administrator about rules that contradict one another (the "conflict" anomaly) or have no effect ("invisibility" and "redundancy") in the analysed filter. The networked anomalies, on the other hand, analyse the filters in the context of the network topology and warn the administrator about filters that contradict one another ("disagreement"), filters that block desired traffic ("blocking"), rules that have no effect on the given network topology ("irrelevancy") and routers that are enabling unwanted traffic ("leaking"). Each type of anomaly is formally defined along with its algorithm.

The developed concepts were used to implement a tool — the *Packet Filter Checker* (PFC) — that reads a description of the rules and network topology in a simple custom language and reports all anomalies present. This tool is used to analyse and fix a fictional user case in several iterations of changing requirements. This shows the tool and the anomalies in the target context: where they help the network administrator.

Keywords: packet filter, firewall, security policy, policy conflict, rule coherence.

RESUMO

Redes de computadores estão sob constante ameaça, ainda mais quando conectadas à Internet. Para reduzir o risco, dispositivos de segurança como o filtro de pacotes são usados. Uma primeira camada de segurança, o filtro de pacotes é responsável pelo bloqueio do tráfego indesejado em posições chave da rede. Os pacotes que devem ser permitidos ou bloqueados pelo filtro são definidos através de um conjunto de regras programadas pelo administrador da rede. Essas regras tem duas partes: a seleção e a ação.

Conforme cresce a rede e o número de serviços, a quantidade de regras tende a aumentar. Passado certo limite, a complexidade de manter uma quantidade grande de regras se torna um fardo para o administrador. Isso aumenta a probabilidade de enganos que podem prejudicar a segurança da rede.

Este trabalho desenvolve o conceito de "anomalia", cada qual representa um problema em potencial, uma contradição ou uma regra supérflua dentro do conjunto de regras; ou seja, cada anomalia alerta o administrador da rede para determinada situação. Há 7 tipos de anomalias, que podem ser divididos em dois grupos: anomalias de filtro único e anomalias em rede. As anomalias de filtro único alertam o administrador sobre regras que se contradizem ("bloqueio") ou que não possuem efeito no filtro ("invisibilidade" e "redundância"). As anomalias em rede, por sua vez, alertam o administrador sobre filtros que se contradizem ("discordância"), filtros que bloqueiam tráfego desejado ("bloqueio"), regras que não se aplicam a nenhum pacote que passe pelo filtro onde estão ("irrelevância") e roteadores que permitem a passagem de tráfego indesejado ("vazamento"). Cada um desses tipos de anomalia é definido formalmente e apresentado junto com um algoritmo que a encontra.

As anomalias e seus algoritmos foram usados para implementar uma ferramenta, o *Packet Filter Checker* (PFC), que lê as regras e a descrição da topologia da rede e cria um relatório com todas as anomalias presentes. Este trabalho apresenta um caso de uso fictício que é analisado e corrigido com base nos resultados apresentados pela ferramenta. O caso de uso é apresentado em diversas iterações, cada uma representando alterações nos requisitos da rede. Este caso mostra a ferramenta e os conceitos no contexto-alvo: na ajuda ao administrador da rede.

Palavras-chave: filtro de pacotes, firewall, política de segurança, conflito, coerência de regras.

1 INTRODUCTION

Communication and interoperability between computers and other devices is of utmost importance to individuals and corporations. Applications such as electronic commerce, electronic mail, web browsing and others require the interconnection of computers to work. On the Internet and on most private networks, this communication is performed using the *Internet Protocol* (IP). This protocol provides the means to identify (*IP address*) and locate resources on the network.

Most of the data in an IP network is transported using two other protocols: the *Transmission Control Protocol* (TCP) and the *User Datagram Protocol* (UDP). TCP provides a way to create and identify error-free communication channels with delivery and ordering guaranteed between two end systems. TCP is the transport protocol used in the Web, for instance. TCP enables many users to access a service, and also a single user to access many instances of a service. UDP, on the other hand, does not have the concept of communication channel. It allows a service to be accessed by multiple users, but it does not guarantee delivery, order or correctness. That also means that it has less overhead over the transported data.

Through the use of these protocols by others of higher level, the Internet provides all its variety of services. And it is this large set of available services that makes the Internet hard to secure.

It is risky to be connected to the Internet. This risk comes not only from vulnerabilities on the software used to access the services (from operating systems to browsers), but also from the software that provides the services. In addition to that, there are malicious users on the network that are ready to exploit these vulnerabilities. To defend a network from these users, many software and hardware solutions are developed and deployed.

A first layer of protection can be found in firewalls, which are devices that can be deployed in key network locations to monitor and filter connections. Firewalls can be oriented to various aspects: application, traffic, service, etc. A very common orientation, and the one this work deals with, is the packet filter. The packet filter is a device used to control the data traffic between "network zones" with different security requirements. This control is configured by the use of rules. Each rule has two parts: the *match* and the *target* (WELTE; AYUSO, 2007). The match selects which packets the rule should be applied to, and the target is the action to be taken. The most common targets, which are available in every packet filter, are *accept* and *deny*. "Accept" let the selected packets travel through the device, while "deny" makes the filter drop the packets. Many filter implementations have other targets, to log or redirect packets, but "accept" and "deny" (and their equivalent) are the basic ones.

Even though filters are completely defined with such a simple unit (set of tuples match→target), filters can become quite complex and difficult to understand as the number of rules increase. The limitations in the semantics of matches and the complexity of the security policy are the main reasons for the high number of rules. In addition, these rules can be spread along many security devices, making the maintenance of the security policies and the network topology a difficult and error-prone task.

In a single filter, errors come mostly from the order of the rules. When a packet arrives, the filter searches the rule list from top to bottom until it finds a matching rule. If a new rule is placed too low in the list, it might not match all the intended packets; if a new rule is placed to high, it might match more packets than it should. Both situations can make the behaviour of the filter diverge from the security policy.

For single filters, this work defines 3 types of anomalies - potential errors that can come from the wrong ordering of the rules: the invisibility - when a rule never matches any packet; the conflict - when the order between two rules is neither trivial nor irrelevant; and the redundancy - when a rule can be removed without any impact on the filter. All these situations are undesired, and all can be easily fixed by the addition, removal or target changing of the rules.

In a system with multiple networked filters there are situations that come from the interaction of the rules of different filters. These situations form a whole new set of problems, as packets from one point to the other may be filtered by different rule sets as nodes fail and even under normal conditions. With the rules spread out between the filters, analysing and guaranteeing the overall behaviour becomes very hard.

For networked filters, we provide four other types of anomalies - in this case, when different paths in the network provide different accessibility for packets: the disagreement - when two filters define different targets for directly connected networks; the block - when a filter deny packets that should be accepted; the irrelevancy - when a filter has rules that deal with packets that don't pass through it; and the leak - when a set of routers provide a path for packets that should be dropped.

This work provides, for each anomaly, detailed definitions, examples and algorithms. The *Packet Filter Checker* (PFC) is the application developed that analyses networks and filters and generates many of the figures in this work. PFC is also where the algorithms presented were validated, and can thus find anomalies in filters that the user describes.

This work is organized as follows. In chapter 2, an introduction to computer networks and protocols is presented along with packet filters. Chapter 3 explores existing works that deal with rule coherence and positions this work among them. In chapter 4, the scheme used for the verification of rules in an isolated packet filter is explained. Chapter 5 extends this scheme to deal with a set of networked filters. Chapter 6 presents a case study and shows the use of the application developed. At last, the conclusion is drawn, followed by the references.

2 COMPUTER NETWORKS AND PACKET FILTERS

2.1 The Internet Protocol

The basic information on IP is taken from RFC791 (POSTEL, 1981a), even though a number of RFC's update the protocol (ALMQUIST, 1992; NICHOLS et al., 1998; RAMAKRISHNAN; FLOYD; BLACK, 2001). These updates are not relevant for this work, as they deal mostly with the field *Type of Service* that will not be used.

According to the OSI model (ZIMMERMANN, 1980), the purpose of the network layer is to provide the means to exchange data units between two transport endpoints.

In practice, and according to RFC791, the scope of IP is to deliver a limited sequence of bits (internet datagram) from a source to a destination in a system of interconnected networks. The protocol has no flow control and does not guarantee datagram ordering, delivery or integrity. What it provides is an abstraction over the links used by the source, destination and intermediate nodes.

An IP datagram has two parts: the header and the payload. The header has the information that is used by the protocol itself, while the payload has the data being transmitted.

The IP header has a field denominated **Protocol** that identifies the payload format. That is, it indicates if the payload is an ICMP control message, a TCP packet or an UDP datagram, among other options.

The other two important fields of the IP header are the **Source Address** and the **Destination Address**. These fields have the IP address of the source and destination device, respectively.

2.1.1 IP addressing and routing

The **IP address** is four octets that identify a network interface. The address, with the exception of some ranges, has global scope. For a specific service to be available on the internet, the host of the service must have an interface with a globally valid and unique IP address.

RFC791 specified that the number of nodes in a network was fixed and given by the class of the IP address of the network. This approach was replaced by a hierarchical scheme, where each network can be seen as a unique block from outside, and be divided internally as smaller networks. This new approach is called **Classless Inter-Domain Routing** (CIDR) (REKHTER; LI, 1993; FULLER et al., 1993). CIDR uses, in addition to the network address of the interface, a **network mask**. This mask has also four octets and identifies which bits of the address define the network and which define the host. For instance: an address 205.100.27.2 with the mask 255.255.255.0 belongs to a device on the network 205.100.27.0. As network masks have all "on" bits before the first "off" bit, they are usually represented by the number of "on" bits following a slash in the interface address. This representation is called the **network prefix** of the network. The network prefix of the example above is 205.100.27.2/24.

To get to its destination, an IP datagram that is not addressed to the local network must go through a set of routers (BAKER, 1995). **Routers** are network devices that connect different networks and forward datagrams from one interface to the other based on the datagram's destination address. The decision process is called **routing**.

Routers know which networks their interfaces are connected to, and also about other routers and their networks. This information can be provided statically by the network administrator or obtained dynamically through the use of routing protocols, such as *RIP* (MALKIN, 1993) and *OSPF* (MOY, 1998). It is also common to have a default route to use if a datagram is destined to an unknown network, specially in the presence of a connection to the Internet.

All considerations made above are valid inside an autonomous system (AS) (HAWKINSON; BATES, 1996). The macro structure of the Internet is composed by several autonomic systems that represent different corporative and administrative domains. The requirements and implementation of inter-AS routing are different from those presented above.

2.1.2 ICMP

The **Internet Control Message Protocol** (POSTEL, 1981b) is used for the exchange of control messages between two IP entities. These messages travel inside the payload of the IP datagram, but are part of the protocol and not an upper layer.

An ICMP message is identified with the value `1` in the Protocol field of the IP header. After the header, the type of message is identified by an octet, and the rest of the datagram contains the body of the message. Some message types are more common than others, and some are even obsolete (ZWICKY; COOPER; CHAPMAN, 2000). Some types available are:

echo request and echo reply These messages are used by utilities such as ping and traceroute. An IP entity send an echo request with some data to another entity, that has to reply with the same data.

destination unreachable Can be sent by a gateway upon discovering that the destination of an IP datagram can't be reached.

source quench Sent by a IP entity when it is receiving datagrams from a source too fast. Can be described as a "please slow down" request.

time to live exceeded This message means that a datagram has exceeded its Time To Live (TTL). Can signal routing problems, for instance.

parameter problem This message is sent to the source of an invalid datagram that had to be discarded.

redirect This message is sent to a host when a datagram should be sent through another path. The datagram is also forwarded to the correct path.

router announcement and router selection Can be used to request and announce routers in a network.

Even though ICMP messages are useful, they also represent a risk if allowed to flow freely in a network.

Destination unreachable and *time to live exceeded* are used by traceroute and allow the discovery of the topology of a network. *Destination unreachable* can be forged in such a way as to close a connection between two hosts.

Redirect, *router announcement* and *router selection* are supposed to modify the routing table of a host and can be used in a denial of service attack.

It is recommended (ZWICKY; COOPER; CHAPMAN, 2000) that administrators allow only the ICMP messages that are used and deny all others. This policy denies also several ICMP messages that are not cited above and that can be as dangerous.

2.2 UDP

RFC768 (POSTEL, 1980) defines the **User Datagram Protocol** (UDP) as a mean to exchange datagrams over IP.

As in other IP protocols, a UDP datagram has a header and a payload, that are both carried inside the payload of the IP datagram. The header identifies the source and destination ports.

Source port is an optional UDP field. It can be used to indicate the port that should be used for an answer.

The **destination port**, on the other hand, only has meaning when used with the destination IP. Together, they identify the destination process that is supposed to receive the datagram.

UDP, by itself, does not define anything else. That means that there are no datagram arrival, integrity or ordering guarantees.

UDP can be seen as IP plus ports, and it is usually used by the applications with transport requirements different from those provided by TCP.

2.3 TCP

The **Transmission Control Protocol** (TCP), defined in RFC793 (POSTEL, 1981c), provides a reliable communication channel between two processes in different hosts in a networked environment. TCP is connection oriented, and corresponds to the transport layer of the OSI model. The reliability of the channel implies that the data will arrive at its destination in the same order it was sent, without loss or duplication.

For the scope of this work, there are three fields of interest in the TCP header:

source port Identifies the sending process.

destination port Identifies the destination process.

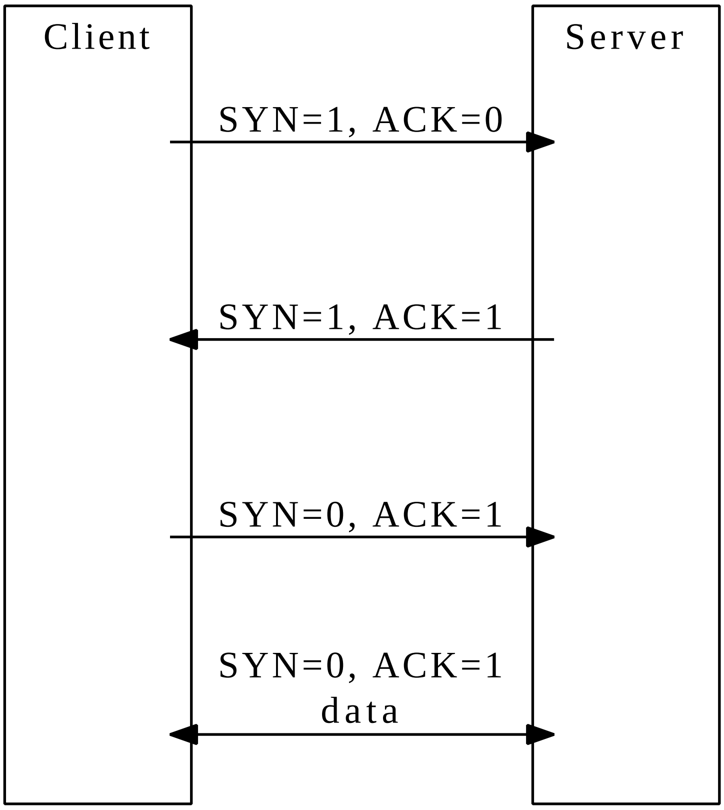


Figure 2.1: Starting a TCP connection

flags Flags are used in special packets, to establish a connection, for instance. The flags will be explained later on.

A single connection is identified globally by the set { $destination_{IP}$, $destination_{port}$, $source_{IP}$, $source_{port}$ }. That is the reason a host can access the service in several instances, as long as its source port is unique. TCP is bidirectional, requiring no extra connection for answers.

The most used TCP flags and their meaning (ZWICKY; COOPER; CHAPMAN, 2000):

ACK Acknowledgement. Should be “no” in every packed of a connection except the first one.

RST Reset. Aborts a connection.

SYN Synchronize. Used in connection establishment to synchronize the sequence numbers.

FIN Finish. Used when gracefully closing a connection.

Figure 2.1 shows that the first packet exchanged in the establishment of a TCP connection has the ACK flag off. That makes it easy to block connection initiation.

The RST flag is used to abort a connection. A packet with this flag on can be sent from any side at any moment when a problem is detected in the connection or an unexpected packet arrives. RST is sent, for example, when a connection start request is made to a port with no process attached.

The SYN flag makes it possible to check if a port has a process without completely establishing a connection. This is called **half open port scan** (HEMNI, 2006), and is performed by sending a packet with the SYN flag on and then checking the answer. If it is a packet with the RST flag, there is no process; if it is a packet with ACK on, then, there is one. As most systems do not log connections that are not fully open, it is possible to scan the processes of a host without alerting the system administrator.

Similar techniques can be used to discover the operating system of a host (VASKOVICH, 2008), as different implementations of IP behave slightly different in the presence of unusual datagrams.

Even though it is impossible to eliminate all threats with packet filters, they can be at least reduced. For instance, it is possible to block invalid packets in the filter, preventing some OS scan methods.

Another important use of packet filters is of centralizing and protecting against security failures in services that can be open by default or without the user's knowledge. Some operating systems have services on by default to make the user's life easier, even though that creates attack vectors. With packet filters, it is possible to prevent access to services that are not explicitly offered according to the security policy.

2.4 Packet filters

As mentioned, the **packet filter** is the component of a firewall responsible for blocking undesired datagrams. Packet filters are usually placed between networks in order to control the flow of data between zones with different accessibility requirements. In this aspect, they also act as routers.

A packet filter is an entity uniquely defined by:

- The IP address and network prefix length of its interfaces;
- The set of rules of the filter.

There are several models for the exact format of the rules, but every model studied shared a common subset:

- Every rule has two parts, one responsible for the selection of datagrams (*match*) and other that defines the action to be taken for the datagrams (*target*);
- The set of rules is analysed in a sequence by the filter for every packet, until a rule selects the packet and defines the action to be taken;
- The selection is made based on the fields of the header of IP, TCP, UDP, and ICMP;

- At least two targets are always implemented: one that allows the datagram to be forwarded (*accept*) and another that drops the datagram (*deny*).

From this common core, implementations develop different extensions for both the match and the target.

2.4.1 Match

The part of a rule that selects a packet is called **match**. The match uses one of the relations of *exact comparison*, *prefix comparison*, or *range comparison* of the fields of a header with a specified parameter (BABOESCU; VARGHESE, 2003; SRINIVASAN et al., 1998). These types of comparisons are mentioned in order of growing generality. That means that an exact comparison can be represented by a prefix comparison, and a prefix comparison can be represented by a range comparison, with no addition of rules.

The fields used are generally the flags of the headers of the protocols, the source and destination addresses, and the source and destination ports. Specifically, the flags are more properly selected with an exact comparison; the addresses with a prefix comparison; and the ports with a range comparison.

Meanwhile, the majority of works limit the semantics of the match in order to simplify the implementation or provide better performance. There are works (EPPSTEIN; MUTHUKRISHNAN, 2001; SU, 2000; HARI; SURI; PARULKAR, 2000) that take only the source and destination addresses. Some (EPPSTEIN; MUTHUKRISHNAN, 2001; SU, 2000) do it in order to be able to use 2D geometric structures to improve match performance.

In Woo (WOO, 2000), we see an algorithm that focuses on the prefix comparison of the source and destination addresses. Later, the authors extend this algorithm to match on five fields: source and destination addresses, source and destination ports and protocol.

Baboescu and Varghese (BABOESCU; VARGHESE, 2003), on the other hand, developed an algorithm for the detection of conflicts on two fields based on prefixes, and then extended it to five fields. Baboescu's work approached each field as a set of bits that can be matched by prefix.

Most of the aforementioned works tackle the problem of efficient and fast selection of datagrams and packets. Only a few of them deal with conflicts in the rules. Taylor has published a survey (TAYLOR, 2005) where the mentioned works can be seen in context.

2.4.2 Target

The **target** of a rule is the action that the filter takes when a datagram is selected by the rule. Most packet filter implementations have a variety of targets available. Those targets can be classified in two groups: terminating and non-terminating. Terminating targets are those that stop rule traversal when the corresponding rule matches, i.e. *deny*. Non-terminating targets are those that can be performed with no disturbance to rule traversal, i.e. *log*. Linux's *iptables* (WELTE; AYUSO, 2007) is a packet filter implementation with several terminating and non-terminating targets.

Even with this target variety, most of the academic works studied (EPPSTEIN; MUTHUKRISHNAN, 2001; SU, 2000; HARI; SURI; PARULKAR, 2000; EPPSTEIN; MUTHUKRISHNAN, 2001; SU, 2000) deal only with the *accept* and *deny* targets, both terminating. That can be justified by the fact that these two targets are the ones that provide filtering, and are thus the most important ones. They are also the ones that implement the behaviour of the filter for an external observer. As these two targets are symmetrically opposites, their relation defines what is a conflict in a set of rules.

2.4.3 Stateless and stateful filters

Packet filters can be classified in two groups: stateless and stateful.

Stateless filters are those that have no notion of connection state, and provide match features based only on the fields of a datagram.

Stateful filters, on the other hand, keep track of connection state and allow the user to make rules based on it. With stateful filters, it is possible to allow connections to be started in only one direction without matching on TCP flags explicitly. Stateful filters also tend to have a reduced number of rules, as it is not necessary to inset rules to match both flow directions: there is usually a rule that accepts traffic from already initiated connections, and the rest of the rules deal only with which connections can be initiated.

As a disadvantage, stateful packet filters have a greater overhead in memory, as every connection state must be kept, and inferior performance when compared to the corresponding stateless filter (ZWICKY; COOPER; CHAPMAN, 2000).

3 EXISTING WORKS IN PACKET FILTER COHERENCE

Existing works related to packet filter coherence will be presented in this chapter.
The works were split in groups according to their approach and focus:

- rule pair analysis: works that use the relation between two rules and their order inside the filter to check their consistency;
- data structures: works that improve the performance of packet classification with some novel structure that is also used to check the consistency of rules;
- conflict resolution: works that not only present the definition of "conflict" as a sort of consistency problem, but also suggest resolution procedures;
- application-oriented: works that present applications that are complete implementation of some coherence scheme.

3.1 Rule pair analysis

Conflicts in the set of rules of packet filters can be defined by the relation between the rules of the filters, when analysed in pairs. That is the approach used by Al-Shaer and Hamed (AL-SHAER; HAMED, 2003a), their work is considered as a reference throughout this work.

3.1.1 Relations

R_x and R_y being two rules with fields $\{protocol$, $source_{IP}$, $source_{port}$, $destination_{IP}$, $destination_{port}$ $\}$, the relation between R_x and R_y can be one of:

disjoint If every field of R_x is completely disjoint in respect to the corresponding field of R_y. In other words: if the intersection of their fields is the empty set.

equal If every field of R_x is equal to the corresponding field of R_y.

inclusive If every field of R_x is equal or a subset of the corresponding field of R_y and the two rules are not equal.

R_x is said to be a **subset** of R_y, while R_y is said to be a superset of R_x.

partially disjoint If at least one field of R_x is a subset or a superset or equal to the corresponding field of R_y and there is at least one field of the two rules that is disjoint.

correlated If there is a field in R_x that is a subset or equal to the corresponding field of R_y and the other fields are a superset of the corresponding fields of R_y.

These relations are exclusive and complete. That means that for each pair of rules, there is always one, and only one, of those relations that apply (AL-SHAER; HAMED, 2002).

3.1.2 Anomalies

Anomalies are potential problems. A filter will work when anomalies are present, but it probably won't have the intended behaviour, or could have the same behaviour with a smaller set of rules.

This behaviour is not determined solely by the set of rules with their fields, but also by the order in which they are traversed by the filter. According to some models, the rule order defines an implicit priority that only appears when the filter is taken as a whole. As the network administrator does not define the order of rules explicitly, it is not considered a safe information.

From the relation between the rules and their order, the following anomalies are defined (AL-SHAER; HAMED, 2004a):

Shadowing If a rule R_x is a subset of a rule R_y that is checked before, then R_x will never select any packets.

This shows a rule that is unnecessary or in the wrong position.

Correlation When two rules with different targets are correlated. This shows a conflict where a change in the order of the two rules would result in a different filter behaviour.

Generalization When a rule R_x is a superset of a rule R_y that is checked before but with a different action. This makes R_y an exception of R_x. If the order of the two rules were exchanged, a shadowing anomaly would be reported.

Even though Al-Shaer and Hamed make this an anomaly, it can be quite common in a correct filter.

Redundancy When the removal of a rule would not cause any change on the filter's behaviour.

That means that R_x is a subset of R_y and they have the same target.

The anomalies of redundancy and shadowing represent real errors in rule configuration. On the other hand, correlation and generalization can occur in a correct filter, and are only anomalies because the behaviour of the filter is defined by the implicit rule order. That's why they are reported, so that the system administrator can confirm the order of the rules.

3.1.3 Criticism to rule pair analysis

Comparing every rule to every other rule is an approach that has some issues.

The worst issue is the lack of vision of the effect of the rules combined. It is possible to create a filter where a rule is not shadowed by any isolated rule, but is shadowed by a group of them. This lack of vision will result in false negatives.

Another big problem is the definition of the generalization and correlation anomalies. They can be present in a correct filter, and there is no way to prevent them from being reported. This leads to false positives, as there is no way to design an anomaly-free filter for some intended behaviour.

Another issue is the constant complexity, which always lies in $O(r^2)$, where r is the number of rules. That happens because every pair of rules must have its relation discovered in order to find the anomalies. That is a waste in a filter where all rules are disjoint, for example.

Even with these issues, Al-Shaer and Hamed's work is very important and its definition of rule relation and anomalies is used as the base for the evaluation of other works.

3.2 Data structures

In this section, works that deal mostly with different data structures and check some sort of coherence on the rules of a packet filter are exposed.

3.2.1 Tries and bit vectors

A **trie** is a tree in which the key of every node is given by its position in the tree (KNUTH, 1998).

Tries have their search time proportional to the size of the key instead of proportional to the number of entries. As such, tries can have faster lookup times than binary search trees. They also help with the problem of *longest prefix matching*. An example of trie can be seen in figure 3.1.

Tries are used on the problem of packet classification, mainly as part of a bigger structure, such as *grid-of-tries* (SRINIVASAN et al., 1998), *set prunnning trees* (SRINIVASAN et al., 1998), *extended grid-of-tries* (BABOESCU; SINGH; VARGHESE, 2003), *bit-vector* (LAKSHMAN; STILIADIS, 1998) or *aggregate bit vector* (BABOESCU; VARGHESE, 2001).

On the field of rule conflicts, tries are used in by Baboescu and Varghese (BABOESCU; VARGHESE, 2003) and Hari, Suri and Parulkar (HARI; SURI; PARULKAR, 2000).

Baboescu and Varghese (BABOESCU; VARGHESE, 2003) use bit vectors to find conflicts fast. The proposal of the article is to find every rule that has a potential conflict with a rule being added. Their work builds a trie for each field where each leaf has a bitmap of the rules that have either an exact equivalence or a prefix equivalence with the key. This structure is called a **bit vector** (example in table 3.1 and figure 3.2). To find potential conflicts with a new rule, every field traverse its corresponding trie, and the intersection of the bitmaps found for each field shows potentially conflicting rules.

Even though Baboescu's scheme finds potential conflicts between an old rules and a new one, it does not define what is supposed to happen with the conflicts found. The article does not

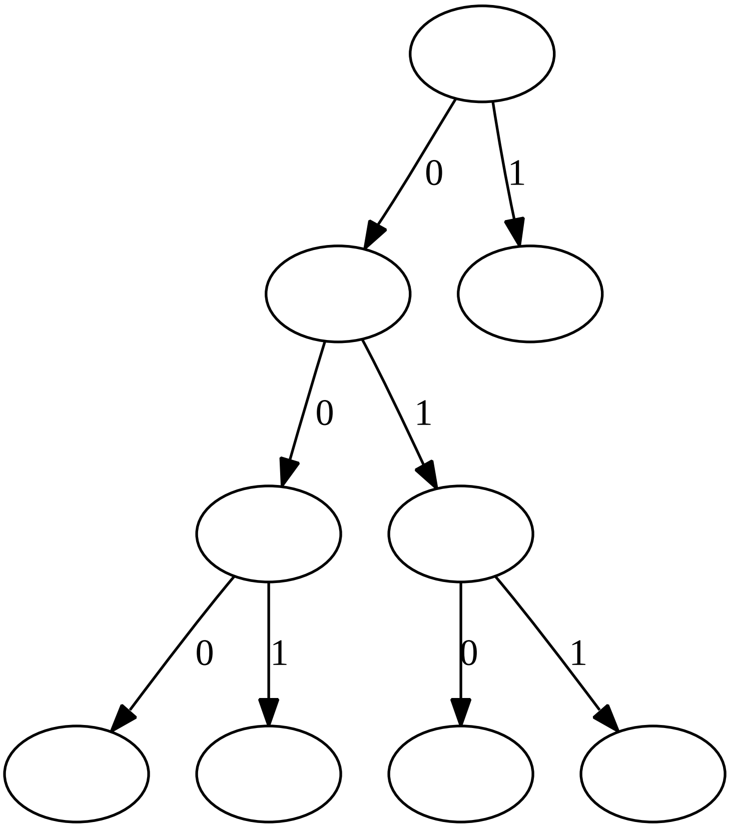


Figure 3.1: Example of trie for keys {000, 001, 010, 011, 100, 101, 110, 111 }.

Table 3.1: Table with a bit-vector example.

Rule	Field_1	Field_2
R_0	000000*	111001*
R_1	1001*	0000101*
R_2	10110*	111*
R_3	1111*	0000100*
R_4	00000100*	100010*
R_5	10111*	000000*
R_6	10*	1111*
R_7	0001010*	*
R_8	000111*	100011*
R_9	000000*	111*
R_{10}	*	*

mention rule order or even targets, and leaves this work to the system administrator.

3.2.2 Geometric structures

Some works (EPPSTEIN; MUTHUKRISHNAN, 2001; SU, 2000) use a geometric approach to tackle the problem of packet classification and conflict detection.

To enable that, those works reduce the packet to two fields, source and destination address, and use them as axis in a plane. This transforms the rules into rectangles in this plane, and the problem of packet classification becomes the problem of locating the highest-priority rectangle (rule) in the plane that contains the point (packet).

Eppstein and Muthukrishnan (EPPSTEIN; MUTHUKRISHNAN, 2001) use this approach also to detect conflicts. In this article, every rule has an explicit priority. A conflict arises when two rules with different targets and the same priority have an intersection in that is not inside another rectangle with higher priority.

This definition of conflict, and the fact that it is found without comparing pairs of rules makes this approach free of false positives and negatives. There are no false positives because it is always possible to get rid of the conflict in a filter by defining a rule with higher priority; there are no false negatives because the requirement of explicit priorities makes the conflict real.

Even though Eppstein's work requires a lot of restrictions on the definition of rules – only two fields and an explicit priority – it presents interesting results when compared to Al-Shaer and Hamed's.

3.3 Conflict resolution

Hari, Suri and Parulkar (HARI; SURI; PARULKAR, 2000) also develop an algorithm for fast conflict detection and suggest the resolution of these conflicts with rule addition.

The article assumes the rules have no priority, implicit or explicit. When there is a packet that two rules match, the one that is more specific should be used. A conflict is reported only if

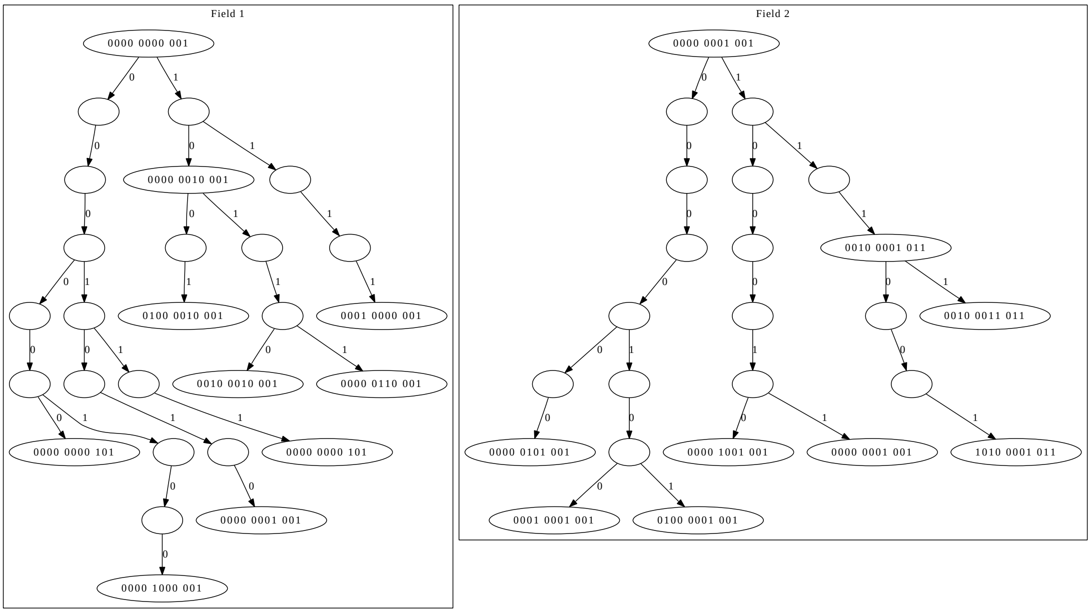


Figure 3.2: Bit-vector of the rules in 3.1

there is a tie. The conflict is then resolved by adding a rule that is more specific to the conflicting region.

Even though the article does not use any trie or geometric structure, it limits the fields in a packet to two in a first moment, and later extends the algorithm to five. The article also requires that the fields match only by prefix and not by arbitrary ranges.

3.4 Applications

Various works (MAYER; WOOL; ZISKIND, 2000; AL-SHAER; HAMED, 2003a; BARTAL et al., 2004; MAYER; WOOL; ZISKIND, 2006) develop applications that help the network administrator in checking the coherence of the rules of a packet filter. In this section these applications will be explored.

3.4.1 Firewall Policy Advisor

The **Firewall Policy Advisor** (FPA) (AL-SHAER; HAMED, 2003a), is an application with a graphical interface that discovers the relation between rules and shows the anomalies found. Anomalies were already discussed in 3.1.2, and need no further explanation.

The interface, by itself, allows rule entry and shows them in a tree structure defined in the same article. The anomalies are updated in real time and shown in the same window where the rules are edited, making it very easy to fix the rule set.

Al-Shaer's work analyses only isolated filters, and checks the rules by pairs.

3.4.2 Fang and Firewall Analyser

Mayer, Wool and Ziskind (MAYER; WOOL; ZISKIND, 2000) present **Fang** (*Firewall ANalysis enGine*): a graphical interface for permission query on a set of filters. The application must be configured with the network topology and filter rules. It provides answers to queries like "which services of host A can be accessed from host B", or "which hosts can access service C of host D".

Mayer's work is the first reference that tries to ease the administration of a set of filters in opposition to a single one. Its engine is built from a graph algorithm that processes the network topology and from a rule simulator that gives the answer to the queries made by the user.

Fang's greatest weakness is that it is only a query interface (MAYER; WOOL; ZISKIND, 2006). The researchers found out that users really don't know what to ask to check the security and efficiency of the set of rules.

For this reason, another application was developed: the **Firewall Analiser** (FA). FA requires from the user only the data input, and instead of providing a query interface, FA provides a report full of details about which service can be accessed from which host. FA also knows which zone is "external" (the Internet), and has a database of services. A big list of open services is expected to alert the network administrator about a misconfigured filter.

The presented applications can be divided in two groups: the ones that check rule coherence in an abstract form (FPA) and the ones that help the network administrator understand and check the functionality of a filter (Fang, FA). Only one application dealt with a set of filters (FA), and it did not check the set of rules for anomalies. That is, then, an open problem.

4 ISOLATED FILTER VERIFICATION

4.1 Introduction

After studying the existing packet filter[1] and coherence checkers, we can say that there is still room for improvement. In this chapter, a new checker for isolated packet filters is developed, along with a new set of anomalies. The goal is to improve on Al-Shaer and Hamed's approach, providing a more robust checker that deals with the rule set as a whole.

We begin by gathering the requirements and desirable features of the checker. The second step is to model datagrams, rules and filters. The model is based on sets, as that provides a familiar and flexible semantics for the whole system. The definition of what characterizes an anomaly follows. An anomaly is supposed to be anything "suspicious" in the filter, and that will be formally defined. As a last step, this chapter shows the algorithms and other artifacts involved.

4.2 Requirements

The system for the analysis of distributed filters that will be shown in a later chapter is an extension of the one shown in this chapter.

The analysis based on pairs of rules is to be avoided, as it limits the view of anomalies by not considering the effects of the rules combined. It also leads to a combinatory explosion if more than one filter is analysed.

The filter checker takes only the rules of the filter as input. Its output are the anomalies found, showing the rules involved and other available information.

The checker must present a complete report, and require no further analysis by the network administrator (in opposition to (MAYER; WOOL; ZISKIND, 2000)). It must be possible to eliminate false positives by the addition of rules (as in (HARI; SURI; PARULKAR, 2000), in opposition to (AL-SHAER; HAMED, 2003a)). It must consider at least five fields (protocol, source and destination addresses and ports) instead of two (as it is in (EPPSTEIN; MUTHUKRISHNAN, 2001; SU, 2000)), and should be extensible to more fields.

Anomalies should report probable mistakes by the system administrator. According to Al-Shaer and Hamed (AL-SHAER; HAMED, 2003a), most mistakes come from the wrong posi-

[1] A more correct term would be *datagram filter*, as the device acts also on IP and UDP datagrams and not only on TCP packets.

tioning of rules in the filter. So, the most suspicious property of a rule is its position, its implicit priority. Every time the position of a rule is neither irrelevant nor trivial, an anomaly must be reported.

Anomalies must also be reported when there are rules that have no effect on the filter, i.e. a rule could be removed and the behaviour of the filter would not change. That means that the rule is either superseded by another or that its target is wrong and the behaviour of the filter is not the one expected.

Other than that, rules are taken to be right, and it is not possible to report further probable errors with only the rules as input.

4.3 Model for datagrams, fields and rules

The model developed is based on sets, with a representation through ranges.

The fields of an IP datagram to be considered are: the source and destination addresses, the source and destination ports and the protocol. Every datagram is then represented as a 5-tuple, with a numeric value for each field. For the effects of analysis, every packed is uniquely and completely represented by the combination of these fields:

$$datagram = \begin{cases} protocol & \in Protocols \\ source_{address} & \in Addresses \\ source_{port} & \in Ports \\ destination_{address} & \in Addresses \\ destination_{port} & \in Ports \end{cases}$$

The Cartesian product of these fields forms the datagram space:

$$Datagram = Protocol \times Addresses \times Ports \times Addresses \times Ports$$

Each field domain is isomorphic to a contiguous subset of the natural numbers. Protocols are in the range $[0 - 2]$, each number representing one of ICMP, UDP or TCP. Ports are in the range $[0 - 65536]$.

A **region** is a subset of the datagram space that can be represented by a single range for each field. Each range is represented by its start and end values:

$$region = \begin{cases} protocol\ start & \in Protocols \\ protocol\ end & \in Protocols \\ source_{address}\ start & \in Addresses \\ source_{address}\ end & \in Addresses \\ source_{port}\ start & \in Ports \\ source_{port}\ end & \in Ports \\ destination_{address}\ start & \in Addresses \\ destination_{address}\ end & \in Addresses \\ destination_{port}\ start & \in Ports \\ destination_{port}\ end & \in Ports \end{cases}$$

As the ranges have the semantics of a set, the concepts of relations, operations and properties already present in set theory (DEVLIN, 1993) are readily available:

- Relations, given two ranges X and Y:
  - $\forall x \in X, \forall y \in Y : x \in Y, y \in X \Rightarrow X = Y$, two ranges are **equal** if their elements are the same.
  - $\forall x \in X, \exists y \in Y : x \in Y, y \notin X \Rightarrow X \subset Y, Y \supset X$, a range is a **superset** of another if it has all the elements of the other and at least one more. The latter is said to be a **subset** of the former.
  - $\forall x \in X : \exists x \in Y, X \not\subset Y, X \not\supset Y, X \neq Y \Rightarrow X \triangle Y$, if two sets have some elements in common and both have some unique elements in respect to the other, they are **correlated**.
  - $\forall x \in X, \forall y \in Y : \neg\exists x \in Y, \neg\exists y \in X \Rightarrow X \bowtie Y$, if two sets have no elements in common, they are **disjoints**.

  These relations are sufficient and necessary to describe every possible relation between two ranges.

  Proof: the relation between a value and an arbitrary range can be only $\in$ or $\notin$. If elements $x \notin Y$ are considered as being or not in X, elements $y \notin X$ considered as being or not in Y, and elements e that can be both in X and in Y, every possible relation can be mapped and seen in table 4.1. Figure 4.1 shows all relations, while figure 4.2 shows all possible transitions.

- binary operations:
  - **intersection**, $\cap$: from two ranges, creates a third that has only the elements that are in both ranges. This operation is not defined for disjoint ranges.

Table 4.1: Table of every possible relation between two ranges.

	$X = \{X\}$	$X = \{X, e\}$	$X = \{e\}$
$Y = \{Y\}$	$\bowtie$	$\bowtie$	$\bowtie$
$Y = \{Y, e\}$	$\bowtie$	$\triangle$	$\subset$
$Y = \{e\}$	$\bowtie$	$\supset$	$=$

Table 4.2: Number of elements of the result of an operation by relation of operands for ranges.

	Relation	max $\mid \cup \mid$	max $\mid - \mid$
A	$=$	1	0
B	$\subset$	1	0
C	$\supset$ with a common extreme	1	1
D	$\supset$ with different extremes	1	2
E	$\triangle$	1	1
F	$\bowtie$	2	1

- **union**, $\cup$: given two ranges, returns a *set of ranges* that has all elements present in them. The operation prevents repetition of elements in the ranges returned.
- **difference**, $-$: given two ranges, it creates the *set of ranges* that has all the elements in the first range that are not present in the second range.

The number of elements of the set of ranges generated by the operations of union and difference can be seen in table 4.2. Example of range operations are shown in table 4.3.

- properties: every range has a **modulus** ($|R|$) that is equal to the number of elements in it.

Ranges are represented as closed intervals containing the extremes, therefore it is not possible to represent a range with no elements. Operations that would return an empty range raise an error instead.

Regions are used to represent a restricted set of datagrams. A **regionset** is a set of regions, and is used to represent an arbitrary subset of datagrams.

$$regionset \subset Datagramas$$

Table 4.3: Example of range operations.

	Example	$\cap$	$\cup$	$-$
A	$[1-10], [1-10]$	$[1-10]$	$\{[1-10]\}$	$\emptyset$
B	$[3-7], [1-10]$	$[3-7]$	$\{[1-10]\}$	$\emptyset$
C	$[1-10], [4-10]$	$[4-10]$	$\{[1-10]\}$	$\{[1-3]\}$
D	$[1-10], [4-7]$	$[4-7]$	$\{[1-10]\}$	$\{[1-3], [8-10]\}$
E	$[1-7], [5-10]$	$[5-7]$	$\{[1-10]\}$	$\{[1-4]\}$
F	$[1-4], [7-10]$	N/A	$\{[1-4], [7-10]\}$	$\{[1-4]\}$

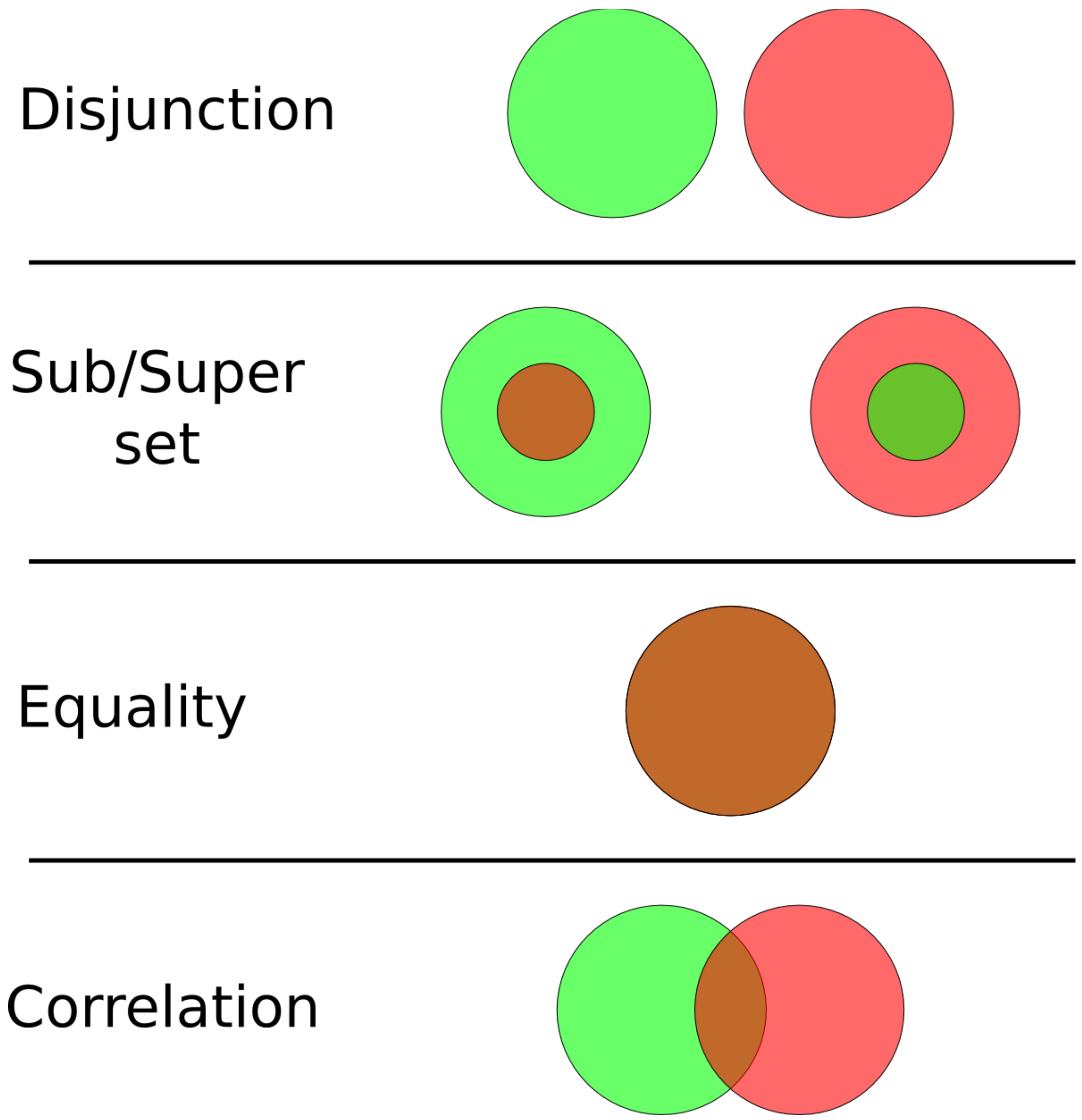


Figure 4.1: All possible set relations.

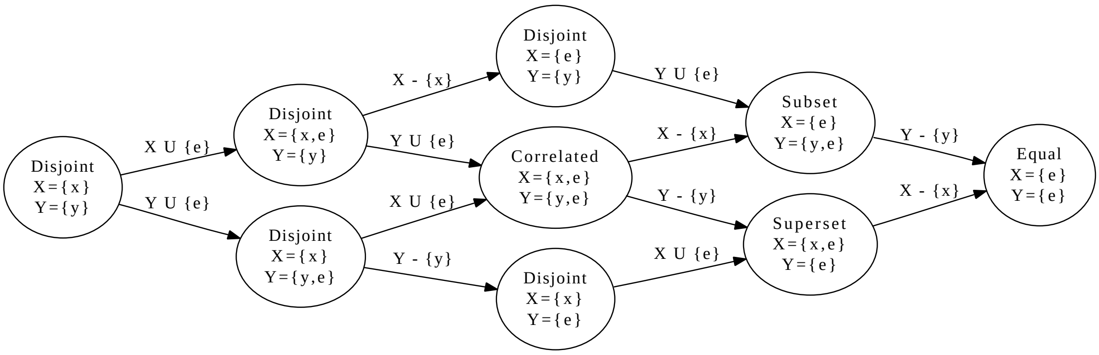


Figure 4.2: Every relation transition possible between X and Y. The direction of an edge can be inverted by inverting the operation.

Table 4.4: Relations and the maximum number of elements of the operations that create regionsets, in terms of the number n of fields.

Relation	max $\|\cup\|$	max $\|-\|$
$=$	1	0
$\subset$	1	0
$\supset$	1	$2n$
$\triangle$	$2n+1$	1
$\bowtie$	2	1

$$regionset = \begin{cases} region_1 & \in Regions \\ region_2 & \in Regions \\ region_3 & \in Regions \\ \ldots \end{cases}$$

As regions have the same semantics of sets, the relations, operations and properties that were discussed for ranges also apply. Operations that map to a set of ranges here map to a regionset, with the number of elements according to table 4.4.

The next entity that needs definition is the **rule**. A rule maps a region to an action, that can be either *accept* to let the datagrams pass, or *deny* to drop them. The region of a rule is called the **match** of the rule.

$$rule = \begin{cases} match & \in Regionsets \\ action & \in \{\text{accept}, \text{deny}\} \end{cases}$$

A filter, for the isolated checker, is a set of rules that define actions to every datagram in the domain:

$$filter = \Big\{ function : Datagrams \rightarrow \{\text{accept}, \text{deny}\}$$

4.3.1 Features of the model

The model based on sets has the following positive features:

Table 4.5: Example of a filter and its equivalents.

Region	Equivalent filter

Universe	`src = 10.0.1.0/24, ACCEPT`
	`src = 10.0.2.0/24, ACCEPT`
	`src = 0.0.0.0/0, DENY`

`src` ∈ 10.0.1.0/24	`src = 10.0.1.0/24, ACCEPT`
	`src = 0.0.0.0/0, DENY`

`src` ∈ 10.0.2.0/24	`src = 10.0.2.0/24, ACCEPT`
	`src = 0.0.0.0/0, DENY`

Other	`src = 0.0.0.0/0, DENY`

expressive As commented in 2.4.1, ranges are the most expressive representation for a field (others are exact match and prefix). The only restrictions imposed is that the values of the field must have total order and be bounded.

And by having the semantic of sets, the power of the relations and operations of set theory is readily available.

compact The representation of rules with regions keeps compactness in the presence of the set semantics. Every rule can be represented as a single region, even though the algorithms may use operations that multiply the number of regions present in memory.

extensible Although this work has a well-defined and limited the number of fields, the semantics and n-tuple behavior allows this number to be easily increased, without adding any special support.

4.4 Equivalent filters

When a subset of the universe of datagrams is considered, it is possible to eliminate the rules that do not apply to any datagram in the subset. The **equivalent filter** has exactly the same behaviour of the original filter for that subset.

By using equivalent filters, it is possible to analyse the rules of a filter in the context of their interaction. That makes it feasible to increase the number of rules being considered and to define anomalies in a broader view when compared to pair analysis.

The order of the rules in an equivalent filter is the same found in the original filter. So, the rule that defines the target for a datagram in the subset considered is the first rule of the equivalent filter that matches the datagram.

4.5 Anomalies

An anomaly is an evidence that the properties of consistency and minimalism do not hold. Both properties are highly desirable in packet filters, as the first prevents some kinds of mistakes

Table 4.6: R (greater priority) and S potential anomalies as a function of their relation.

Relation	Targets	Potential anomaly
$R = S$	any	Invisibility
$R \subset S$	same	Redundancy
$R \subset S$	different	-
$R \supset S$	any	Invisibility
$R \triangle S$	same	-
$R \triangle S$	different	Conflict
$R \bowtie S$	any	-

and the second potentially improves filter performance. These properties guide the development of the specific anomalies.

In a single filter, each anomaly represents a rule with no effect or a case where the order of the rules is neither *irrelevant* nor *trivial*.

The order of two rules is **irrelevant** if they are disjoint. The analysis based on equivalent filters avoids this case, as disjoint rules will never appear together in an equivalent filter.

On the other hand, the order of two rules is only **trivial** if the region of one rule is a subset of the region of the other. Every other relation makes the ordering ambiguous. If the order of two rules is trivial and they exchange places, an anomaly is reported (invisibility), as one of the rules has no effect.

This leads to three possible combination of rules (table 4.6):

- A rule that is a subset of another and has less priority: anomaly, the first rule can be removed or is in the wrong position;
- Two rules are correlated: if their targets are the same, no anomaly. If their targets are different, then their intersecting region must have a rule contained in both of them that defines the target and has a trivial order in respect to the previous rules. Otherwise, there is a conflict in the region, as any priority change will lead to a change in the target, and the priority is not trivial.
- A rule that is a subset of another, has greater priority and same target: anomaly, as the removal of this rule won't change the target for any datagram.

4.5.1 Invisibility

The first anomaly, **invisibility**, shows the rules that do not define the target for any datagram because of their priority. For all datagrams that these rules select, there is a rule with greater priority that already defined the datagram's target. The invisible rules are either out-of-place or completely unnecessary.

Invisible rules are not considered for the analysis of further anomalies, and so this anomaly must be the first one to be checked.

Figure 4.3 shows four diagrams with invisible rules. Each diagram represents a filter with only one field where each square represents a single field value, and each line represents a rule.

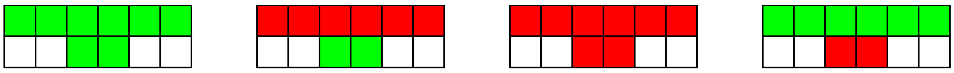


Figure 4.3: Invisible rules are not used.

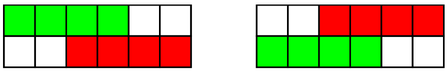


Figure 4.4: Conflicting rules have no obvious order.

The red rules have the deny target, and the green ones have the accept target. The rules with the highest priority are on top.

Formally:

$$\begin{aligned}
&\exists\, rule_1 \in Filter\\
&\neg\exists\, datagrama \in Datagramas\\
&rule(Filter, datagram) = rule_1\\
&\qquad \Rightarrow \text{Invisible}(rule_1)
\end{aligned}$$

Example:

- Filter F:

`Default`	from any	to any	→	Deny
`Invisible`	from 10.0.0.0/24	to any	→	Accept

PFC output (irrelevancy is another anomaly that will be presented later):

```
F:
  Invisible rule Invisible
Irrelevant rule Invisible in filter F
```

4.5.2 Conflict

The **conflict** arises when two rules define different targets for a set of datagrams and their order is not trivial. If the priority of the rules were exchanged, the behavior of the filter would change. The diagrams in figure 4.4 show two cases of conflicting rules for filters with a single field.

Even though conflicts can happen in correct filters, they rises an anomaly, and require a specific rule with greater priority to define the target for the set of datagrams pointed (figure 4.5).

With the third rule in place, the order of the two previous rules is irrelevant, and the order of the third is trivial.

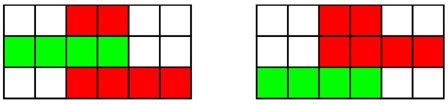


Figure 4.5: Conflicts can be fixed by adding a third rule with higher priority.

Formally:

$$
\begin{aligned}
&\exists\, datagrama \in Datagramas\\
&\exists\, rule_1 \in Filter\\
&\exists\, rule_2 \in Filter\\
&datagram \in match(rule_1)\\
&datagram \in match(rule_2)\\
&action(rule_1) \neq action(rule_2)\\
&\neg\exists rule_3 \in Filter\\
&datagram \in match(rule_3)\\
&match(rule_3) \subset match(rule_1)\\
&match(rule_3) \subset match(rule_2)\\
&\qquad \Rightarrow \text{Conflict}(rule_1, rule_2)
\end{aligned}
$$

Example:

• Filter F:	`r1`	from 10.0.0.1/32	to 10.1.0.0/24	→	Deny
	`r2`	from 10.0.0.0/24	to 10.1.0.5/32	→	Accept

PFC output:

```
F:
  Conflict rule r1 rule r2
    at [src 10.0.0.1/32 dst 10.1.0.5/32]
```

Fix:

• Filter F:	`r0`	from 10.0.0.1/32	to 10.1.0.5/32	→	Accept
	`r1`	from 10.0.0.1/32	to 10.1.0.0/24	→	Deny
	`r2`	from 10.0.0.0/24	to 10.1.0.5/32	→	Accept

Checker output:

```
No anomalies or errors found
```

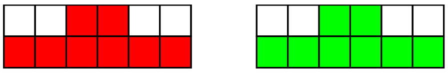


Figure 4.6: Redundant rules are not needed.

4.5.3 Redundancy

The **redundancy** points rules that, even selecting datagrams because of their priority, could be safely removed without causing any change in the behaviour of the filter. The diagrams in figure 4.6 show two examples.

The redundant rules are either not really necessary or have the wrong target. Changing their priority won't make rule necessary, but can make the rule invisible.

When taken together, the absence of conflicts verifies the **consistency** of a filter, while the absence of invisibilities and redundancies guarantees **minimalism** – that is, that all the rules are necessary.

Formally:

$$\begin{aligned}
&\neg\exists\, datagrama \in Datagramas\\
&\exists\, rule_1 \in Filter\\
&action(Filter, datagram) \neq action(Filter - rule_1, datagram)\\
&\qquad \Rightarrow \text{Redundant}(rule_1)
\end{aligned}$$

Example:

- Filter F:

`Redundant`	from 10.0.0.0/24	to any	→	Deny
`Default`	from any	to any	→	Deny

PFC output:

```
F:
  Redundant rule Redundant
```

4.6 Algorithms

This section shows the algorithms for finding the set of equivalent filters and the anomalies. The anomalies are further detailed, as to the process that is necessary to find them.

4.6.1 Equivalent filter set construction

For the construction of the equivalent filters, a **tree of rules** is used. Every inner node of this tree stores a rule, and the edges are identified as "in" and "out". Every leaf holds a list of regions. An example tree with 3 rules can be seen in figure 4.7.

The construction of the tree is incremental. The algorithm 1 details the process.

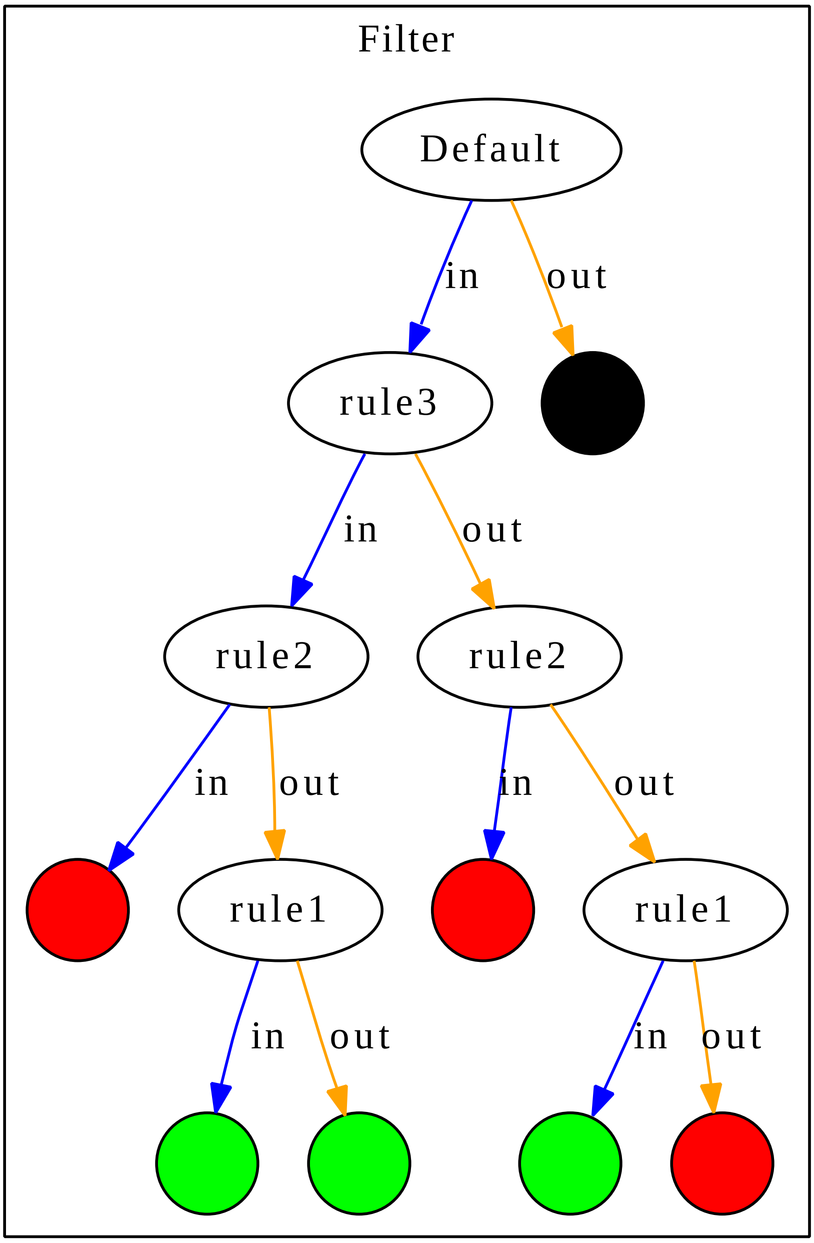


Figure 4.7: Example of a tree of rules.

After the tree is built, the equivalent filters are found by traversing the tree, storing the nodes where the edge taken was “in” and taking them only when the final leaf had at least one region. The details can be seen in the algorithm 2.

Algorithm 1 Tree of rules construction

```
function CREATETREE($\{rules\}$)
    $tree \leftarrow leaf$(total region)
    for all $rule$ in $rules$ do
        $tree \leftarrow$ INSERTRULE($tree, rule$)
    end for
    return $tree$
end function

function INSERTRULE($node, rule$)
    if $node$ is leaf then
        $regions_{\cap} \leftarrow node \cap rule$
        $regions_{-} \leftarrow node - rule$
        $node \leftarrow inner \begin{cases} \text{rule} \leftarrow rule \\ \text{in} \leftarrow leaf(regions_{\cap}) \\ \text{out} \leftarrow leaf(regions_{-}) \end{cases}$
    else                                                        ▷ Node is inner
        if exists $rules \cap node$ then
            $node_{in} \leftarrow$ INSERTRULE($node_{in}, rule$)
        end if
        if $|node - rule| > 0$ then
            $node_{out} \leftarrow$ INSERTRULE($node_{out}, rule$)
        end if
    end if
    return $node$
end function
```

4.6.1.1 Complexity, worst case

To calculate the complexity in memory and space for the worst case, the first step is to calculate the maximum number of equivalent filters. Then, the necessary tree of rules for this condition can be visualised. The complexity of such tree is calculated based on the number of inner nodes, leaves and number of regions stored on the leaves. The total complexity of the construction of the equivalent filters can then be calculated based on that figure.

The theoretical maximum number of equivalent filters generated from a filter with n rules is the sum of every possible combination of rules. Mathematically:

$$N_{equivalent\ filters} = \sum_{i=1}^{r} \binom{r}{i} \tag{4.1}$$

Algorithm 2 Equivalent filter set construction

```
function BUILDEQUIVALENTFILTERSET($\{rules\}$)
    $tree \leftarrow$ CREATETREE($\{rules\}$)
    return BUILDINNERNODE($tree$)
end function

function BUILDINNERNODE($node$)
    $filter \leftarrow \emptyset$
    if $node_{in}$ is leaf then
        if $|node_{in}| > 0$ then
            $filter \leftarrow filter \cup \{\{node_{rule}\}\}$
        end if
    else                                                        ▷ $node_{in}$ is inner
        for all $filter_{child}$ in BUILDINNERNODE($node_{in}$) do
            $filter \leftarrow filter \cup (filter_{child} + node_{rule})$
        end for
    end if
    if $node_{out}$ is leaf then
        if $|node_{out}| > 0$ then
            $filter \leftarrow filter \cup \{\emptyset\}$
        end if
    else                                                        ▷ $node_{out}$ is inner
        $filter \leftarrow filter \cup$ BUILDINNERNODE($node_{out}$)
    end if
    return $filter$
end function
```

Table 4.7: Example of worst case with 4 rules, numbered from 1 to 4.

# rules	# eq. filters	eq. filters
1	4	$\{1\},\{2\},\{3\},\{4\}$
2	6	$\{1,2\},\{1,3\},\{1,4\},\{2,3\},\{2,4\},\{3,4\}$
3	4	$\{1,2,3\},\{1,3,4\},\{1,2,4\},\{2,3,4\}$
4	1	$\{1,2,3,4\}$
$\sum$	15	

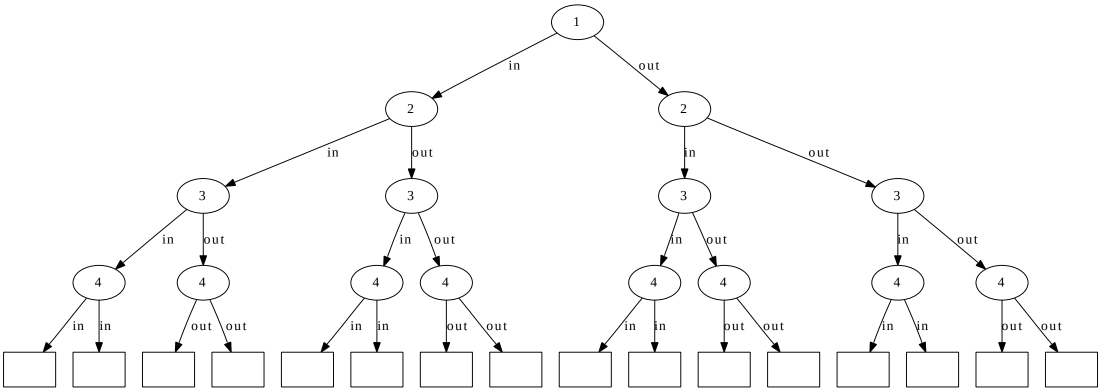


Figure 4.8: Tree of the worst case with 4 rules.

Using the binomial theorem (KNUTH, 1997):

$$(x+y)^r = \sum_{i=0}^{r} \binom{r}{i} x^i y^{r-i} \tag{4.2}$$

Replacing $x = y = 1$ in 4.2 and removing the element with $i = 0$ (eliminates the empty filter), the maximum number of equivalent filters becomes:

$$N_{equivalent\ filters} = \sum_{i=1}^{r} \binom{r}{i} - 1 = 2^r - 1 \tag{4.3}$$

Statement 4.6.1. *The maximum number of equivalent filters for a filter with r rules is $2^r - 1$ in the worst case.*

This hypothetical filter would generate a tree of rules with each rules appearing in and out of every previous rules. Every rule would be correlated to every other rule.

As such, the addition of a new rule of index i generates 2^i new nodes on the tree at level i. At every new rule, the number of nodes on the tree doubles. An example case with 4 rules is shown in table 4.7, with a graphical representation in figure 4.8.

$$\begin{aligned} &N_{inner\ nodes} = 2^r - 1 \\ &N_{leaves} = 2^r \\ &N_{nodes} = N_{inner\ nodes} + N_{leaves} = 2^{r+1} - 1 \end{aligned} \tag{4.4}$$

The number of stored regions in a leaf depends on the path taken. Every time an "in" edge is taken, an intersection is made and the number of regions is kept. On the other hand, every time an "out" edge is taken, the number of regions is increased, as a subtraction is performed.

Algorithm 3 Subtraction of two regions

```
function SUBTRACTRANGE($region1, region2$)
    if $region1$ and $region2$ are disjoint then
        return $region1$
    else
        $range1 \leftarrow region1[1]$
        $range2 \leftarrow region2[1]$
        if $range1 \cap range2 = range1$ then
            return $range1 : d|d \leftarrow$ SUBTRACTRANGE($region1[2..], region2[2..]$)
        else
            $cuts \leftarrow d : region1[2..]|d \leftarrow$ SUBTRACTRANGE($range1, range2$)
            $intersec \leftarrow (range1 \cap range2) : d|d \leftarrow$ SUBTRACTRANGE($region1[2..], region2[2..]$)
            return $cuts \cup intersec$
        end if
    end if
end function
```

The worst case of a subtraction (algorithm 3) is when the second region is a subset from the first, with no shared limits. In this case, two cuts are made, for the parts of the first range that are not intersected. This dimension is then taken apart and the process repeated. This leads to statement 4.6.2.

Statement 4.6.2. *In a subtraction of two regions, the maximum number of regions generated is the number of fields considered doubled:* $|A - B| = 2c$.

This leads to the fact that the leaf that is "out" of every rule will have at most $(2c)^r$ regions, and the leaf "in" every rule will have only one region in the worst case.

There is also a combination here, as the exponent over $2c$ will appear $\left(\begin{array}{c} r \\ i \end{array}\right)$ times, for $0 < i < r$.

The total number of regions in the leaves becomes:

$$N_{regions\ on\ leaves} = \sum_{i=0}^{r} \left(\begin{array}{c} r \\ i \end{array}\right) (2c)^i \tag{4.5}$$

Replacing $y = 1$ and $x = 2c$ in the binomial theorem (equation 4.2) gives:

$$N_{regions\ on\ leaves} = (2c+1)^r = \sum_{i=0}^{r} \left(\begin{array}{c} r \\ i \end{array}\right) (2c)^i \tag{4.6}$$

Equations 4.4 and 4.6 are summarized in statement 4.6.3.

Statement 4.6.3. *The total number of inner nodes in the tree of rules is $2^r - 1$, with at most 2^r leaves, summing up to $2^{r+1} - 1$ nodes in the worst case.*

The total number of regions in the leaves of the tree of rules is $(2c + 1)^r$ in the worst case. They make the set of all the disjoint regions for the filter where different rules apply.

To add a new rule in the tree it is necessary to process all nodes and all regions of all leaves. To get the complexity of such process, it is necessary to multiply the number of nodes and regions by the number of rules ($2c + 1 > 2$ given):

$$complexity_{time} = r(2^r - 1 + (2c + 1)^r) \in O(r(2c + 1)^r) \tag{4.7}$$

The memory is in the order of the size of the tree (nodes plus regions) and the depth of the tree to hold the stack:

$$complexity_{space} = (2c + 1)^r + 2^{r+1} \in O((2c + 1)^r) \tag{4.8}$$

To build the equivalent filter set from the tree, the complexity in time is the number of nodes, and in space is the size of the resulting set plus the depth of the tree (stack). This leads to the statement 4.6.4.

Statement 4.6.4. . *The set of equivalent filter construction complexity is $O(r(2c)^r)$ in time and $O((2c + 1)^r)$ in space.*

4.6.1.2 Complexity, average case

The worst case analysed happens when every rule is correlated with every other, and none is invisible. That is unreal for the following reasons:

- The IP addresses are usually defined through network masks, and that makes it impossible to define two correlated ranges of addresses.
- Rules that use ports usually do not use ranges.

With that in mind, it is worth doing an analysis of an “average case”, that considers the worst case when the relation of correlation is not allowed, thus yielding a more realistic result. This case happens when all rules are nested and none is invisible. Depending on the targets of the rules, this filter may even have no anomalies. This case suggests the addition of rules in the opposite order that they are defined in the filter, as that prevents the duplication of rules, as every rule added will be “in” every previous rule. An example of a simple average case filter can be seen in figure 4.9.

The number of inner nodes of the tree is $r + 1$, and the number of leaves is also $r + 1$. So, the total number of nodes is $2r + 2$.

From statement 4.6.2, it is known that the number of regions created in the subtraction of two regions is $2c$. In this case, every leaf has only one “out” edge, and that is in the last edge. This way, the total number of regions in the leaves is $2cr + 1$.

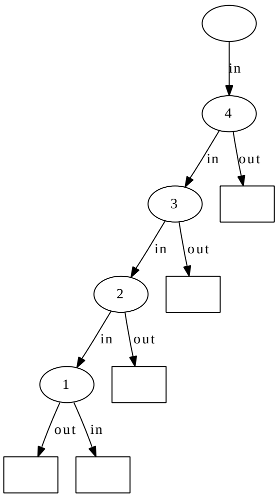


Figure 4.9: Tree of rules of the average case with 4 rules.

Statement 4.6.5. *The number of equivalent filters for the average case is equal to the number of rules.*

Again, to add a new rule it is necessary to process every previous rule. The complexity of such operation is in the order of r multiplied by the number of rules to add, that is, r^2.

In memory, the complexity is the size of the tree ($r + 1$ internal nodes plus $2cr + 1$ regions on the leaves) and the stack. The stack has the size in the order of the number r of rules in the tree.

Statement 4.6.6. *The construction of the set of equivalent filters has complexity $O(r^2)$ on time and $O(r)$ on space for the average case.*

4.6.2 Invisibility

The analysis of invisibility is simple. The checker collects every rule on top of every equivalent filter, and each rule appearing on the original filter and not appearing in the collection is marked as invisible. Algorithm 4 details the process.

Using a binary tree as the data structure for sets ($O(\log_2 n)$ insertion and $O(\log_2 n)$ search), the order of the second loop becomes insignificant and the complexity is dominated by the first.

Statement 4.6.7. *The algorithm of invisibility analysis has, for the worst case, $O(N_{equivalent\ filters} \log_2 r)$ in time and $O(r)$ in space. This happens when there are no invisible rules.*

The worst case for invisibility analysis is very common, in contrast with the worst case for equivalent filters. This must be the first analysis performed, as its output are used as inputs for the following.

4.6.3 Conflict

For the conflict analysis, it is enough to check the rule on top of every equivalent filter against the ones below it. That is the same as checking every rule against the rules that have lower

Algorithm 4 Invisibility analysis

```
function InvisibilityAnomaly($originalFilter, equivalentFilters$)
    $visible \leftarrow \emptyset$
    $invisible \leftarrow \emptyset$
    for all $filter$ in $equivalentFilters$ do
        $visible \leftarrow visible \cup \{filter[0]\}$
    end for
    for all $rule$ in $originalFilter$ do
        if $rule \notin visible$ then
            $invisible \leftarrow invisible \cup \{rule\}$
        end if
    end for
    return $invisible$
end function
```

priority, are not completely disjoint and do not have a rule as a subset defining the target for a conflicting region. Details can be seen in algorithm 5.

The external loop of the algorithm is run for every equivalent filter. The inner loop can be run at most $r - 1$ times. The maximum number of conflicts in a filter is the number of pairs of rules:

$$N_{conflicts} = \binom{r}{2} = \frac{r(r-1)}{2} \in O(r^2) \tag{4.9}$$

Taking the implementation of the set $conflicts$ as a chained list that has $O(1)$ for union (holds a pointer to the last element), the complexity of the algorithm in time becomes $O(N_{equivalent\ filters}(r-1))$. If a post-processing is done to avoid duplicate results, this post-processing will dominate and the algorithm will be $O(c \log_2 c) = O(r^2 \log_2 r)$ in time. In space, the algorithm holds only the set of results, and as such as complexity $O(r)$.

4.6.4 Redundancy

The redundancy is the last anomaly to be checked. Like the invisibility anomaly, the first step is to find the rules that are necessary in the filter. Any rule that is not necessary is reported as redundant.

For every equivalent filter, the first rule is tested. It is put in the necessary set if its removal changes the target of the equivalent filter or generates a conflict.

After all equivalent filters are processed, the rules that are not necessary raise redundancies.

The details can be seen in algorithm 6.

This algorithm makes, for every equivalent filter, either an analysis of conflicts or the processing of the filter rules. Both have the same complexity, so the algorithm has $O(N_{equivalent\ filters}(r-1))$ in time. In space, the algorithm holds the set of defining rules for each filter. That can be, potentially, the whole filter. That results in $O(N_{equivalent\ filters}r)$ in space.

Algorithm 5 Conflict analysis

```
function CONFLICTANOMALY($equivalentFilters, invisibles$)
    $conflicts \leftarrow \emptyset$
    for all $filter$ in $equivalentFilters$ do
        $filter \leftarrow filter - invisibles$
        $conflicts \leftarrow conflicts \cup$ EQUIVALENTFILTERCONFLICTS($filter$)
    end for
    return $conflicts$
end function

function EQUIVALENTFILTERCONFLICTS($filter$)
    $conflicts \leftarrow \emptyset$
    $top \leftarrow filter[0]$
    for all $rule$ in $filter[1..]$ do
        if $top_{target} \neq rule_{target} \wedge top \not\subset rule$ then
            $conflicts \leftarrow conflicts \cup \{(top, rule)\}$
        end if
    end for
    return $conflicts$
end function
```

Algorithm 6 Redundancy analysis

```
function RedundancyAnomaly($allRules, equivalentFilters, invisibles$)
    $necessary \leftarrow \emptyset$
    for all $filter$ in $equivalentFilters$ do
        $filter \leftarrow filter - invisibles$
        if RuleTopIsNecessary($filter$) then
            $necessary \leftarrow necessary \cup \{filter[0]\}$
        end if
    end for
    $redundant \leftarrow \emptyset$
    for all $rule$ in $allRules$ do
        if $rule \notin necessary$ then
            $redundant \leftarrow redundant \cup \{rule\}$
        end if
    end for
    return $redundant$
end function

function RuleTopIsNecessary($filter$)
    if $filter[0]_{target} \neq filter[1]_{target}$ then
        return True
    end if
    if |EquivalentFilterConflicts($filter[1..]$)| $> 0$ then
        return True
    end if
    return False
end function
```

5 DISTRIBUTED FILTERS VERIFICATION

5.1 Introduction

This chapter shows the distributed component of PFC. As the chapter that described the isolated analysis, it is structured as follows: first the requirements are gathered, then the model is developed along with the definition of anomalies and, at last, the algorithms are presented.

5.2 Requirements

The distributed checker is an extension of the isolated checker, and can use data provided by it, such as the *tree of rules*.

To avoid redundancies, the distributed checker verifies only anomalies that are related in some way to the network topology; the other anomalies are supposed to be reported by the isolated checker. This filter sees the filters as “flat”: it does not consider the relations of the rules inside a single filter, nor their order, only their effect as it appears to an external viewer.

As input, this checker gets the set of filters with their rules and the network topology, along with all the information that the isolated checker can provide. As output, it provides the report of anomalies. Anomalies are designed to show the administrator the rules that either contradict a rule in another filter or are unnecessary, verifying consistency and minimalism.

The algorithms should, whenever possible, avoid costs that are a function of the number of networks, trying to keep them bound by the number of routing elements in the system. The rationale for this is that the number of routers tend to grow at a lower rate than the number of networks, as they have a direct monetary cost.

5.3 Model

For the distributed model, three new entities are considered:

networks The word *network* is used to describe the atomic entity represented by a network prefix that exists on the system.

A network can be either **inner** when it connects two routers, or **terminal** otherwise.

$$network = \begin{cases} address & \in Addresses \\ prefix\ length & \in [0 - 32] \end{cases}$$

plain routers Plain routers are devices with no filter capabilities that connect two or more networks. They are modeled as a set of interfaces. Each interface is composed of an IP address and the connected network.

$$interface = \begin{cases} address & \in Addresses \\ network & \in Networks \end{cases}$$

$$router = \begin{cases} interface_1 & \in Interfaces \\ interface_2 & \in Interfaces \\ interface_3 & \in Interfaces \\ \ldots \end{cases}$$

filters are routers that forward packets selectively. Filters are programmed with rules and checked for anomalies, while plain routers are not. For the distributed checker, filters also have the set of interfaces in their model.

$$filter = \begin{cases} function : Datagrams \rightarrow \{\text{accept}, \text{deny}\} \\ interface_1 \in Interfaces \\ interface_2 \in Interfaces \\ interface_3 \in Interfaces \\ \ldots \end{cases}$$

routers are used to connect networks. They can be either *filters* or *plain routers*.

$$routers = plain\ router \vee filter$$

The system as a whole can be seen as a graph in which the networks (nodes) are connected by routers (edges). Cycles in this graph are possible, and happen when the network provides alternative routes. This is usually done by the administrator to have redundancy and/or load balancing. Any path can be used by a packet in the presence of failures, and for this reason, every path between two networks is a possible path.

The region algebra of the previous chapter is used as the base for the algorithms presented here.

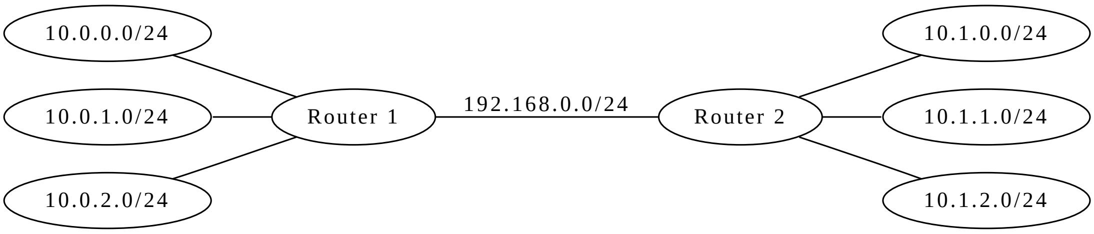


Figure 5.1: Example network

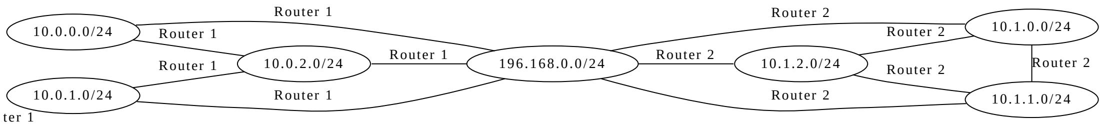


Figure 5.2: Graph of the network in figure 5.1 with networks as nodes and routers as edges

5.4 Network graph

As presented above, a system of networks can be seen as a graph where the nodes are the networks and the edges are the routers.

On the other hand, the number of routers tend to be much smaller than the number of networks, for the reasons mentioned in section 5.2. And as a router can have many terminal networks attached to it, this approach leads to a graph that has more edges and nodes than necessary. Figure 5.1 shows an example network, while figure 5.2 shows the resulting graph of this approach.

To avert this situation, the model used builds the graph with routers as nodes, and the connecting networks as edges. Terminal networks are attached to the node of the router they are connected. Figure 5.3 shows the result of this improved approach.

The number of terminal networks is expected to be greater than the number of inner networks. This expectation is reasonable from a security point of view, as it is desirable to isolate users and facilities in terminal networks.

If the number of terminal networks is greater than the number of inner networks, the number of paths in the graph can be expected to be less than the number of pairs of networks, as the number of paths depend only on the number of inner networks, while the number of pairs of networks grows also with the number of terminal networks. These assumptions influence some

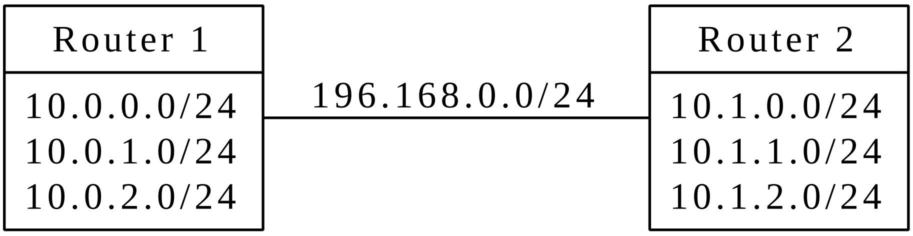


Figure 5.3: Graph of the network in figure 5.1 with routers as nodes and inner networks as edges

Table 5.1: Summary of the properties provided by the absence of each anomaly:

	Consistency	Minimalism
Isolated	conflict	invisibility, redundancy
Distributed	disagreement, block, leak	irrelevancy

choices about the algorithm structure, and are summarized in equation 5.1 below.

$$|\text{routers}| \ll |\text{inner nets}| < |\text{terminal nets}| \Rightarrow |\text{paths}| < |\text{nets}|^2 \tag{5.1}$$

5.5 Accessibility profile

A packet travels the network graph through the routers from its source network to its destination network. Each packet can be either forwarded or dropped by each filter, depending on the fields of the packet. Every pair of networks has, then, an **accessibility profile** for each filter, that shows the results of the filter for each packet that goes from the first network to the second. In other words, it is the equivalent filter that considers the region where the source address is the first network and the destination address is the second network.

The accessibility profile is the main tool used to check the filters for inter-filter consistency, as every path between two networks must present the same resulting accessibility profile. Even though the “accept” regions of the resulting accessibility profile of a path must be “accept” in every filter traversed, the “deny” regions can be defined by any filter of the path. Nevertheless, it is more convenient to have all the intended deny regions for a network in the nearest filter, as that makes the configuration more local, more resilient to topology changes and avoids rule duplication.

For that reason, the **correct accessibility profile** is assumed to be the one lifted from the first filter found by traversing the graph from one network to the other, as the first filter is the one that should have all the accept and deny regions configured correctly.

5.6 Anomalies

The first step of the distributed checker is to find all the isolated anomalies of all filters. Each conflict found is then checked to see if their match is inside a single interface. If it is, the conflict anomaly is dropped. This is only possible in the distributed setting because knowledge of the topology is needed.

After that, the distributed anomalies are found. Each anomaly is explained next.

5.6.1 Disagreement anomaly

When a packet travels from one network to the other, it goes through one or more filters. The first filter the packet finds is considered the provider of the correct accessibility profile.

But, if the network is not a terminal network connected to a filter, it may be possible that there are multiple filters that could be the first filter for a packet, as there can be more than one path connecting the two networks. This happens not only with inner networks, but also with terminal networks connected to routers.

If the accessibility profiles lifted from each of the first filters are not equal, the correct profile for the two networks cannot be determined, and a disagreement anomaly is reported.

A disagreement anomaly is also reported if the correct accessibility profile is not present in the last filter that a packet can find. This design decision was made because there are rules that make more sense when coded in the last filter.

Formally:

$$
\begin{aligned}
&\exists\, path_1 \in Paths \\
&\exists\, path_2 \in Paths \\
&path_1 \neq path_2 \\
&source_{address}(path_1) = source_{address}(path_2) \\
&destination_{address}(path_1) = destination_{address}(path_2) \\
&\exists\, datagram \in Datagrams \\
&source_{address}(datagram) = source_{address}(path_1) \\
&destination_{address}(datagram) = destination_{address}(path_1) \\
&action(filter_1(path_1), datagram) \neq action(filter_1(path_2), datagram) \\
&\qquad \Rightarrow \text{Disagreement}(datagram, filter_1(path_1), filter_1(path_2)) \\
&action(filter_1(path_1), datagram) \neq action(filter_n(path_1), datagram) \\
&\qquad \Rightarrow \text{Disagreement}(datagram, filter_1(path_1), filter_n(path_1))
\end{aligned}
$$

Example:

- Filter F1: | `r1` from 10.0.0.0/24 to 10.1.0.0/24 → Accept
- Filter F2: | `r2` from 10.0.0.0/24 to 10.1.0.0/24 → Deny

PFC output:

```
Disagreement
    of   filter F1, rule r1
    with filter F2, rule r2
    at   [src 10.0.0.0/24 dst 10.1.0.0/24]
```

Figure 5.4 shows the network topology of this example. Figure 5.5 shows a graphical representation of the global accessibility profile.

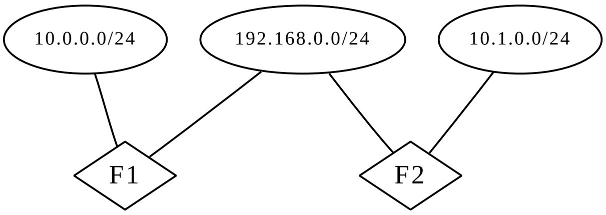
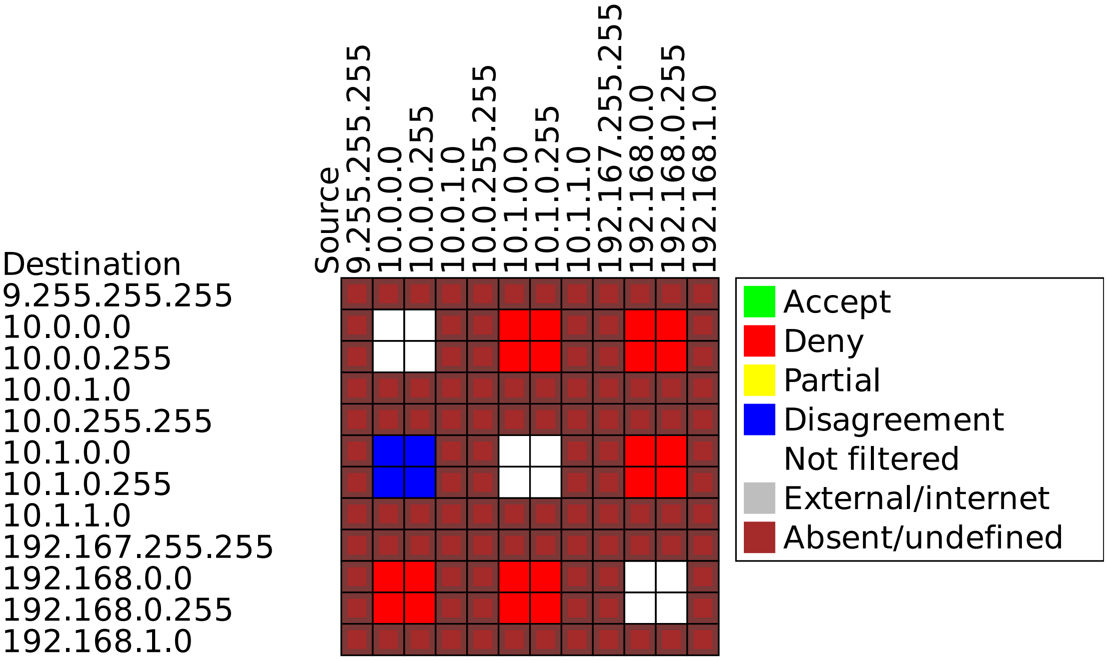


Figure 5.4: Topology of the disagreement example.



Destination \ Source	9.255.255.255	10.0.0.0	10.0.0.255	10.0.1.0	10.0.255.255	10.1.0.0	10.1.0.255	10.1.1.0	192.167.255.255	192.168.0.0	192.168.0.255	192.168.1.0
9.255.255.255	🟫	🟫	🟫	🟫	🟫	🟫	🟫	🟫	🟫	🟫	🟫	🟫
10.0.0.0	🟫	⬜	⬜	🟫	🟫	🟥	🟥	🟫	🟫	🟥	🟥	🟫
10.0.0.255	🟫	⬜	⬜	🟫	🟫	🟥	🟥	🟫	🟫	🟥	🟥	🟫
10.0.1.0	🟫	🟫	🟫	🟫	🟫	🟫	🟫	🟫	🟫	🟫	🟫	🟫
10.0.255.255	🟫	🟫	🟫	🟫	🟫	🟫	🟫	🟫	🟫	🟫	🟫	🟫
10.1.0.0	🟫	🟦	🟦	🟫	🟫	⬜	⬜	🟫	🟫	🟥	🟥	🟫
10.1.0.255	🟫	🟦	🟦	🟫	🟫	⬜	⬜	🟫	🟫	🟥	🟥	🟫
10.1.1.0	🟫	🟫	🟫	🟫	🟫	🟫	🟫	🟫	🟫	🟫	🟫	🟫
192.167.255.255	🟫	🟫	🟫	🟫	🟫	🟫	🟫	🟫	🟫	🟫	🟫	🟫
192.168.0.0	🟫	🟥	🟥	🟫	🟫	🟥	🟥	🟫	🟫	⬜	⬜	🟫
192.168.0.255	🟫	🟥	🟥	🟫	🟫	🟥	🟥	🟫	🟫	⬜	⬜	🟫
192.168.1.0	🟫	🟫	🟫	🟫	🟫	🟫	🟫	🟫	🟫	🟫	🟫	🟫

🟩 Accept
🟥 Deny
🟨 Partial
🟦 Disagreement
Not filtered
▨ External/internet
🟫 Absent/undefined

Figure 5.5: Global profile of the disagreement example.

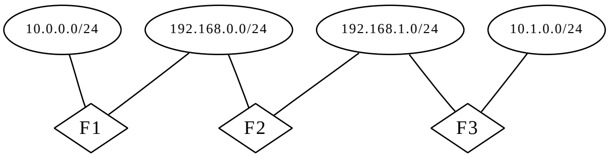


Figure 5.6: Topology of the blocking example.

5.6.2 Blocking anomaly

The accessibility profile is composed by the accept and deny regions of all addresses inside both network prefixes and ports.

The absence of disagreements guarantees that all the “accept” regions are accepted by every filter that can be the first or last filter to be found by a packet traveling from one network to the other. But, in order for a packet to effectively reach the destination network, the other filters in the path must also accept its passage.

So, if an intermediate filter blocks a subregion of an accept region found in the correct accessibility profile, a blocking anomaly is reported.

Formally:

$$\begin{aligned}
&\exists\, path \in Paths\\
&\exists\, filter \in path\\
&\exists\, datagram \in Datagrams\\
&source_{address}(datagram) = source_{address}(path)\\
&destination_{address}(datagram) = destination_{address}(path)\\
&action(filter_1(path), datagram) = accept\ \wedge\ action(filter, datagram) = deny\\
&\qquad \Rightarrow \text{Block}(datagram, filter)
\end{aligned}$$

Example:

- Filter F1: | `r1` from 10.0.0.0/24 to 10.1.0.0/24 → Accept
- Filter F2: | `default` from any to any → Deny
- Filter F3: | `r3` from 10.0.0.0/24 to 10.1.0.0/24 → Accept

PFC output:

```
Block in filter F2
    at [src 10.0.0.0/24 dst 10.1.0.0/24]
```

Figure 5.6 shows the network topology of this example.

5.6.3 Leaking anomaly

To accept a packet, all filters in the path must accept it. On the other hand, to deny a packet, a single denial is enough. If the correct accessibility profile denies a packet and there are no disagreements, then it is guaranteed that the packet is denied by the first filter it encounters.

In fact, that only happens if there is no path where all connections are provided by plain routers. If there is one, then this path provides a filter-free way for packets that should not be accepted according to the correct accessibility profile.

For this reason, if there is a blocked region in the accessibility profile and if there is also a path made only by plain routers for any network in the region, a leaking anomaly is reported.

Formally:

$$
\begin{aligned}
&\exists\, path_1 \in Paths\\
&\exists\, path_2 \in Paths\\
&path_1 \neq path_2\\
&source_{address}(path_1) = source_{address}(path_2)\\
&destination_{address}(path_1) = destination_{address}(path_2)\\
&\exists\, datagram \in Datagrams\\
&source_{address}(datagram) = source_{address}(path_1)\\
&destination_{address}(datagram) = destination_{address}(path_1)\\
&action(filter_1(path_1, datagram) = deny \;\wedge\; \neg\exists\, filter \in path_2\\
&\qquad \Rightarrow \text{Leak}(datagram, path)
\end{aligned}
$$

Example:

- Filter F1: | `r1` from 10.0.0.0/24 to 10.1.0.0/24 → Deny

PFC output:

```
Leak between {10.0.0.0/24} and {10.1.0.0/24} in path [router R1]
```

Figure 5.7 shows the network topology of this example.

5.6.4 Irrelevancy anomaly

The irrelevancy anomaly is reported for every rule that is not necessary in a given network topology. It is checked by keeping a set of rules that are candidates to irrelevancy, and removing from the set the rules that are found to be relevant.

Relevant rules are the rules that either appear on the correct accessibility profile or that prevent the blocking anomaly in an intermediate filter. That means that after checking for disagreements and blocks, all relevant rules are already removed from the candidate set. Therefore, after these verifications, all rules left in the candidate set are reported as irrelevant.

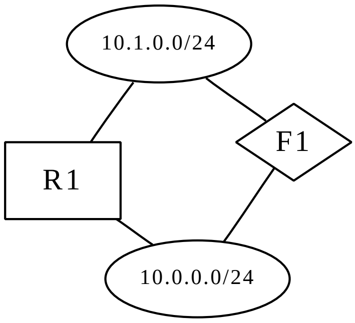


Figure 5.7: Topology of the leaking example.

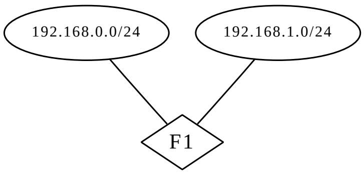


Figure 5.8: Topology of the irrelevancy example.

That implies that a rule that only blocks packets that are already blocked is irrelevant, as is a rule that accepts packets that are blocked.

Formally:

$$\begin{aligned}
&\exists\, path \in Paths\\
&\exists\, filter \in path\\
&\exists\, rule \in filter\\
&\forall\, datagram \in Datagrams\\
&source_{address}(datagram) = source_{address}(path)\\
&destination_{address}(datagram) = destination_{address}(path)\\
&action(filter, datagram) = action((filter - rule), datagram)\\
&\qquad \Rightarrow \text{Irrelevant}(rule)
\end{aligned}$$

Example:

- Filter F1:

`r1`	from 10.0.0.0/24	to 10.1.0.0/24	→	Accept
`default`	from any	to any	→	Deny

PFC output:

```
Irrelevant rule r1 in filter F1
```

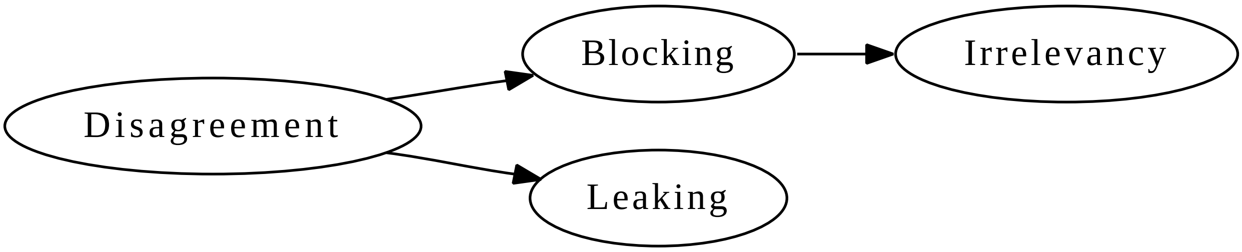


Figure 5.9: Hierarchy of the distributed anomalies

Figure 5.8 shows the network topology of this example.

It is interesting to note that there is a hierarchy of distributed anomalies. If there is a disagreement anomaly, the correct accessibility profile is undefined and the other anomalies can't be checked. If there is a blocking anomaly, the irrelevancy checking will not be complete and can give false positives.

5.7 Algorithms

The main loop of the algorithm iterates on all possible unique paths that can be found in the graph.

Statement 5.7.1. *The maximum number of paths happens when every router is connected to every other router by a dedicated network. By using the binomial theorem, we have that the maximum number of paths is* $|Paths| = 2^{|Routers|}$.

The longest path has all the routers, and thus length $|Routers|$.

Statement 5.7.2. *Every router being connected with every other is also the condition where the maximum number of inner networks is found:* $|Networks_{inner}| = |Routers|(|Routers| - 1)$

The condition required by these theoretical values is not likely to happen in practice for a system with a high number of routers, as every new router added to the network would require a connection to every other, and router ports are limited.

On the other hand, even though the number of terminal networks is bounded by the number of router ports, this bound is an external factor and changes as the router market changes. So, in this work, the maximum number of terminal networks is not assumed to be bounded by the number of routers, it is only assumed to be much greater than it.

With that in mind, there is a matter of balance to be decided. The algorithm has to analyse all paths as well as every pair of networks. This raises two options:

- iterate on all pairs of networks; for each pair, find all connecting paths;
- iterate on all paths; for each path, store information for every network connected by the path.

Equation 5.1 says that the number of paths is expected to be less than the number of pairs of networks. This leads to the choice to favor the second option, that sacrifices memory to save time. The algorithm, then:

1. builds the list of paths;
2. for all paths, lifts the accessibility profile for every connected network, checks disagreements and blocks and remove the used rules from the irrelevant candidates list;
3. all rules left in the list of irrelevant candidates are reported as irrelevant;
4. for all paths made only by plain routers, uses the global accessibility profile lifted from the second step to check for leaks.

Details on every step and their cost are presented next.

5.7.1 List of paths construction

The algorithm used to build the list of paths is a simple depth-first traversal for each node. No optimization reduces the order of this algorithm.

From (CORMEN et al., 2001), the cost of depth-first traversal is $O(|V| \times |E|)$. As the model has the routers as vertices and the inner networks as edges, and knowing that the maximum number of inner networks is $O(|Routers|^2)$ from statement 5.7.2, statement 5.7.3 summarizes the cost of the process.

Statement 5.7.3. *The cost of building the list of paths is the product of the number of nodes multiplied by the cost of the depth-first traversal:*
cost(DFT) $\in O(|Routers| \times |V| \times |E|) = O(|Routers| \times |Routers| \times |Routers|^2)$
cost(DFT) $\in O(|Routers|^4)$

5.7.2 Disagreements, blocks and leaks

After building the list of paths, the first three anomalies can be checked.

Algorithms 7, 8, 9 and 10 form a high-level representation of the checking. The algorithms lack features such as proper report of anomalies with the rules involved, skipping of already reported anomalies and language-specific optimizations.

5.7.2.1 Complexity

The cost of building the profile (algorithm 8) unfortunately depends on the number of networks attached to the router, and that is unavoidable. The profile has all the intersecting regions of the first filter's rule with the networks present in the first router as source and the networks present in the last routers as destination.

It is known from statement 4.6.3 that the maximum number of disjoints regions in a filter is $(2c + 1)^r$ in the worst case. It is enough to take the intersection from each network with each of these regions. Therefore, the number of regions in the correct accessibility profile for a single path is in $O(|\text{networks in first router}| \times |\text{networks in last router}| \times (2c+1)^r)$ in the worst case, where r is the number of rules in the first filter.

If every router has the same number of ports and the ports are all in use (worst case), then the number of networks in each router is the same number n. Theorem 5.7.4 summarizes the conclusion reached with this assumption:

Statement 5.7.4. *The number of region in the correct accessibility profile for each path is in*

$$O\left(n^2 \times (2c+1)^r\right)$$

That is also the time and space cost of building the profile.

The cost of checking one path is in $O(|\text{routers in path}| \times \text{profile size of path})$. In the worst case, every router is connected to every other, and the number of routers in each path is equal to the number of routers, leading to statement 5.7.5:

Statement 5.7.5. *The cost of checking one path is, in the worst case*

$$O\left(|\textit{Routers}| \times n^2 \times (2c+1)^r\right)$$

The process has no space requirements of its own, though, requiring only the profile to be present.

Another implication of having every router connected to every other router is that the number of paths is in $O\left(2^{|\text{Routers}|}\right)$. Joining this fact with statements 5.7.1, 5.7.4, and 5.7.5 we get to statement 5.7.6:

Statement 5.7.6. *The cost in time of checking all paths can be determined to be, in the worst case:*

$$O\left(|Routers| \times 2^{|Routers|} n^2 (2c+1)^r\right)$$

The cost in memory is the cost of the list of paths plus the cost of one profile, and that is $O\left(2^{|Routers|} + n^2(2c+1)^r\right)$.

The cost in memory will be dominated by the second parcel most of the time, as the tendency is to have a few routers with many rules and networks because of the monetary cost of routers.

5.7.3 Irrelevancies

As all relevant rules are removed from the list of candidates while checking disagreements and blocks, the irrelevancy checks in fact only reports the rules that are still in the list. This has minimal memory and time cost, if the number of irrelevant rules are small.

The absence of irrelevant rules testify the minimality of the rules in the filters.

Algorithm 7 Find distributed anomalies

```
function CHECKDISTRIBUTEDANOMALIES
    $paths \leftarrow$ list of paths
    $profile_{global} \leftarrow \emptyset$
    $rules_{leak\ candidates} \leftarrow$ all rules of all filters
    for all $path$ in $paths$ do
        BUILDPROFILEPATH($path$)
    end for
    for all $rule$ in $rules_{leak\ candidates}$ do
        Report irrelevant rule
    end for
    for all $path$ in $paths$ do
        FINDLEAKSINPATH($paths$)
    end for
end function
```

Algorithm 8 Building the profile for a path

```
function BUILDPROFILEFORPATH($path$)
    $router_{first} \leftarrow$ first router of $path$
    $router_{last} \leftarrow$ last router of $path$
    $filter_{first} \leftarrow$ first filter of $path$
    $filter_{last} \leftarrow$ last filter of $path$
    for all $network_1$ in $router_{first}$ do
        for all $network_2$ in $router_{last}$ do
            $networks \leftarrow (network_1, network_2)$
            Remove all rules in $filter_{first} \cap networks$ from $rules_{leak\ candidates}$
            Remove all rules in $filter_{last} \cap networks$ from $rules_{leak\ candidates}$
            $profile \leftarrow profile_{global} \cap networks$
            if $profile = \emptyset$ then
                $profile \leftarrow filter_{first} \cap networks$
                $profile_{global} \leftarrow profile_{global} \cup profile$
            end if
            BUILDPROFILEFORPATHNETWORKS($path, networks, profile$)
        end for
    end for
end function
```

Algorithm 9 Builds the profile for a pair of networks

```
function buildProfileForPathNetworks($path, networks, profile$)
    for all $filter$ in { first and last filters of path } do
        if $filter \cap networks \neq profile$ then
            Report disagreement
        end if
    end for
    for all $profile_{region}$ in $profile$ do
        if target of $profile_{region}$ is accept then
            for all $filter$ in $path$ do
                $filter_{region} \leftarrow filter \cap profile_{region}$
                Remove all rules in $filter_{region}$ from $rules_{leak\ candidates}$
                if target of $filter_{region}$ is not accept then
                    Report block
                end if
            end for
        end if
    end for
end function
```

Algorithm 10 Leak finding

```
function findLeaksInPath($path, profile$)
    if all routers of $path$ are plain then
        if there is a deny region in $profile$ then
            Report leak
        end if
    end if
end function
```

5.8 The Internet

There is a special case in the filter analysis that corresponds to the internet.

When the system administrator defines a network interface of a router as connected to the internet, the checker does two things: first, it considers every network that is not present anywhere else in the topology as connected to that interface. Secondly, even if there are many internet connections, the checker does not consider any path that has such interface as an edge. That is justified by the fact that the network administrator will probably never have internal packets travel from one edge of his network to the other using an external connection. The internet is modelled as a network interface that is connected to every network that is not present in the system.

These considerations are not present in the shown algorithms because they are made mostly as a pre-processing stage and as a verification in the list of paths construction, that is also not shown.

6 ASSERTIONS

6.1 Introduction

Up to this point, PFC is able to detect contradictions and other problems in an algebraic way, considering the set of rules implemented and the current network topology.

It is interesting to provide a mechanism to perform a functional verification of the filter, that is, to test if the filter in fact conforms to its specification.

This is done with *assertions*.

6.2 Requirements

The set of assertions will also be provided as input to PFC, in addition to the set of rules and the network topology.

Assertions will be simple verifications of the overall filter behaviour. They are checked after the algebraic verifications, and their violation is reported with the anomalies.

6.3 Model

Each assertion has the same format of a rule, being composed of a match and a target.

$$assertion = \begin{cases} match & \in Regionsets \\ action & \in \{\text{accept}, \text{deny}\} \end{cases}$$

Unlike rules, though, assertions must be globally true, all of them, simultaneously. Assertions must hold in the global accessibility profile, as that guarantees that assertions hold globally when there are no anomalies.

6.4 Errors

As assertions are not part of the implemented topology, they do not generate anomalies, but **assertions errors**.

Assertions are first checked among themselves. As all assertions must hold globally, the error of **contradiction** is reported for assertions that contradict one another.

If an assertion does not hold, an assertion **violation** error is reported.

6.5 Algorithm

Assertion verification is performed the same way rule disagreement is checked. See algorithm 9 for the high-level reference.

6.6 Conclusion

Assertions are a simple mechanism for filter conformance checking.

They can also help evaluate the impact of a specification change, help with its implementation and help in regression testing.

Assertions can be compared to unit testing for packet filter configuration.

7 CASE STUDY

7.1 Introduction

A case study is presented in this chapter.

The case study is an artificial example. As the most interesting and non-obvious problems of a network arise in the presence of changes, the case study is divided in iterations. Each iteration changes the network and the requirements before implementing a naïve solution that will be checked with PFC and then fixed.

Assertions are used along a security policy to verify the implementation's conformance.

7.2 Case 1: single filter

Misc. Inc. started out as a small company. At the beginning, we had only a small handful of computers connected to a switch. As the company grew, security became an issue. After a virus wiped out our customer's information, the company decided to specialize roles, split the network and develop a security policy.

The network got divided by functionality and access requirements. Servers were put in two networks: one for the IT servers (mail, file, proxy, web server) and one for the main servers. Users were put in a third network. The topology can be seen in figure 7.1.

The security policy stated that:

- every network has access to the mail server (IT);
- internet web access is filtered by a proxy (IT) that can access the internet;
- the web server (IT) can be accessed from the internet;
- the other servers should not be able to initiate connections;
- users have complete access to the main servers and the file server;
- users can only access the internet through the proxy.

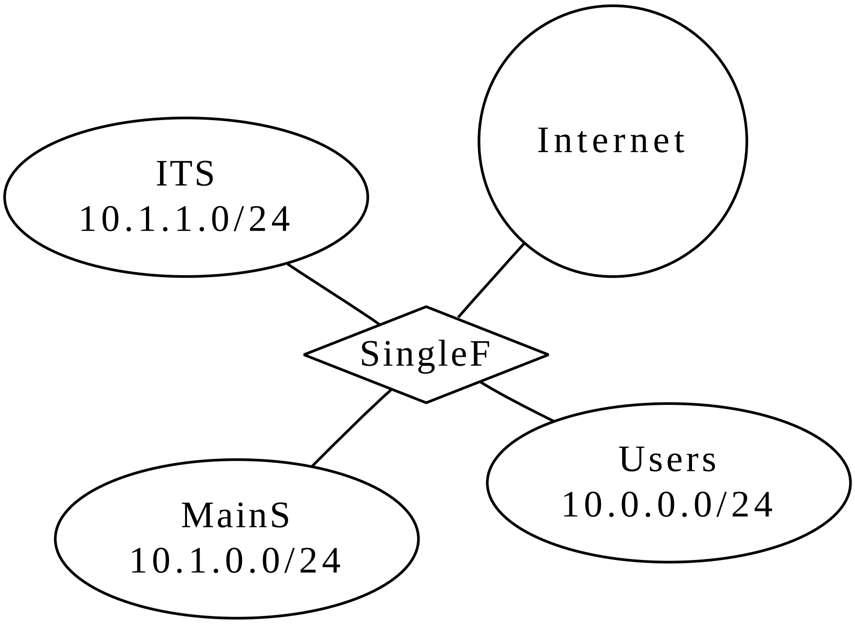


Figure 7.1: Topology of case 1

7.2.1 Naïve solution

The IP address allocation follows. Networks have their first letter capitalized; server networks and hosts end in an `S`; filters end in `F` and routers in `R`.

`Users`	10.0.0.0/24
`MainS`	10.1.0.0/24
`ITS`	10.1.1.0/24
`Internal`	10.0.0.0/8
`All`	0.0.0.0/0
`webS`	10.1.1.2
`mailS`	10.1.1.3
`fileS`	10.1.1.4
`proxyS`	10.1.1.4

The direct translation of the security policy results in the following description:

- Filter SingleF:

`Mail`	from any	to 10.1.1.3/32:23	→	Accept
`Web`	from any	to 10.1.1.2/32:80	→	Accept
`NoProxy2Internal`	from 10.1.1.4/32	to 10.0.0.0/8	→	Deny
`ProxyA`	from 10.1.1.4/32	to any	→	Accept
`NoServer`	from `MainS`	to any	→	Deny
`Users2Main`	from `Users`	to `MainS`	→	Accept
`Users2File`	from `Users`	to 10.1.1.4/32	→	Accept
`Users2Proxy`	from `Users`	to 10.1.1.4/32:3180	→	Accept
`Default`	from any	to any	→	Deny

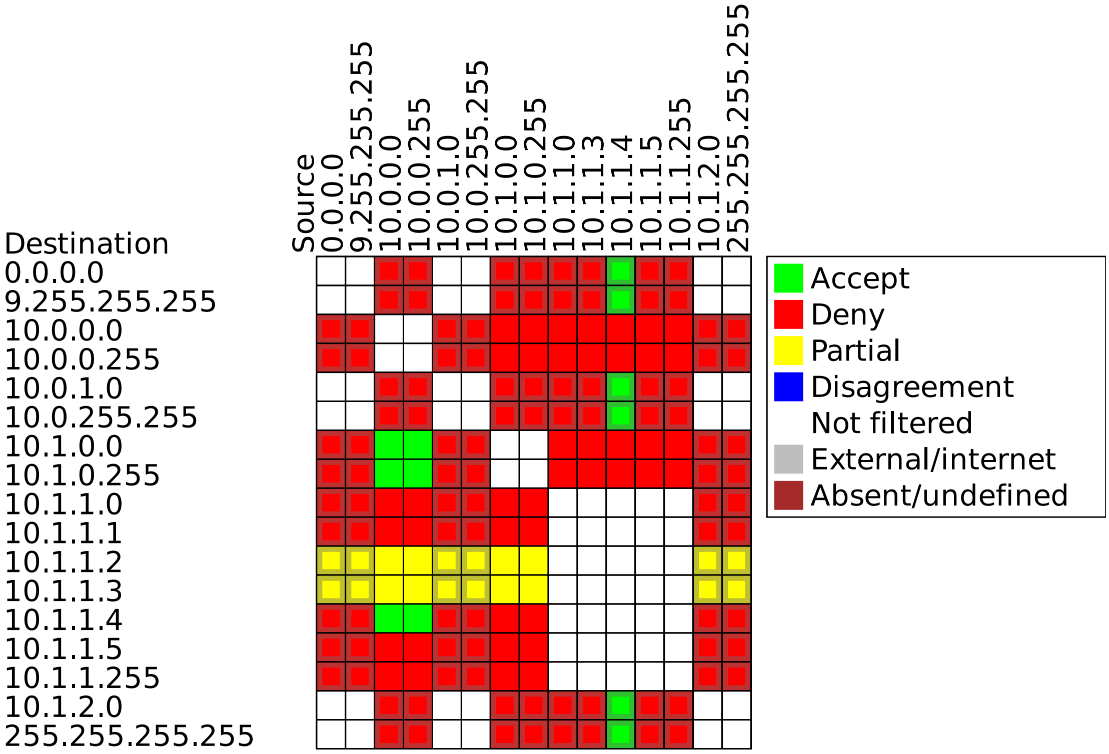


Figure 7.2: Profile of the naïve solution of case 1

Figure 7.2 shows the profile of the filter, figure 7.3 shows the tree of rules.

PFC found the following anomalies:

```
SingleF:
  Invisible rule Users2Proxy
  Conflict rule Web rule NoServer
    at [src 10.1.0.0/24 dst 10.1.1.2/32:80]
  Conflict rule Mail rule NoServer
    at [src 10.1.0.0/24 dst 10.1.1.3/32:23]
  Redundant rule NoServer
Irrelevant rule Users2Proxy in filter SingleF
```

It is easy to see that the invisibility of rule `Users2Proxy` and its consequent irrelevancy is due to the fact that the file server and proxy server are the same physical machine, and the more general rule (`Users2File`) comes first in the filter.

On the other hand, the conflicts with the `NoServer` rule happen because the rules `Mail` and `Web` allow the main servers to connect to the mail and web servers, and that is against the security policy.

At last, the redundancy of rule `NoServer` is pointed out because its effect is already provided by the `Default` rule.

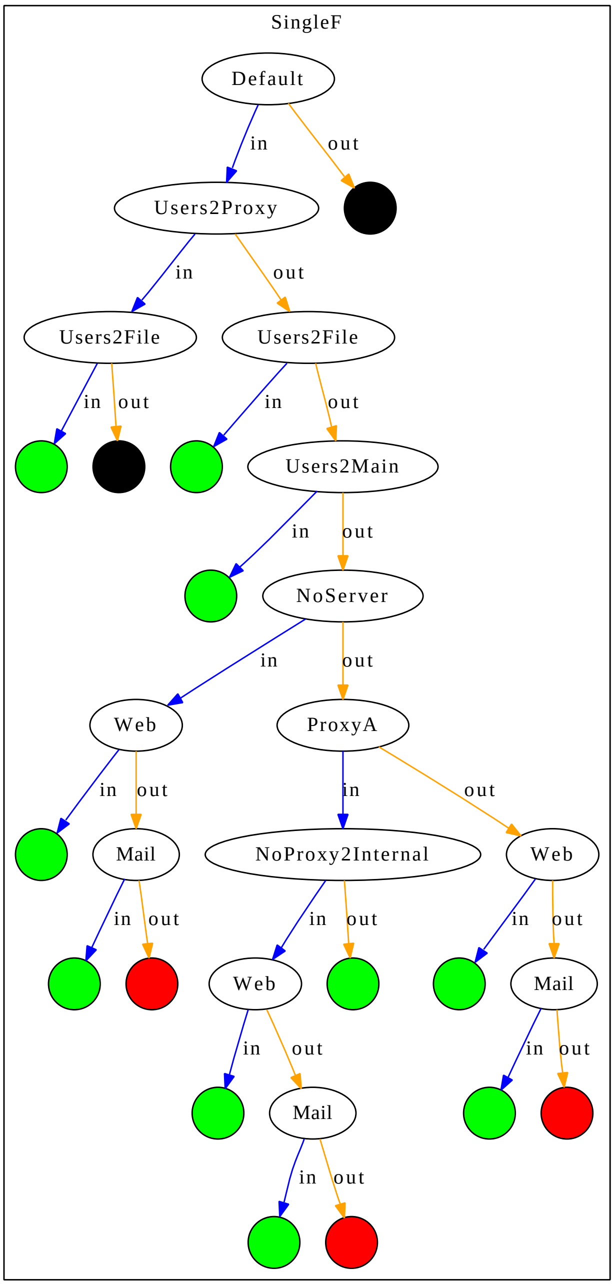


Figure 7.3: Tree of rules of the naïve solution of case 1

In the next section a proper solution is developed.

7.2.2 Proper solution

A simple fix for the problems found in the naïve solution is to remove the rules Users2Proxy and NoServer. Even though the first rule may be safely removed, the second rule points to a real conflict between what was implemented and the security policy.

A better approach is to transform both rules into *assertions*, as they are both correct and must be globally valid: the first rule is invisible only because the file server and the proxy server are accessed through the same IP address; the second rule is really redundant, but verifies the filter implementation.

After converting both rules to assertions, the assertion NoServer fails. To fix that, three rules are implemented: NoServer2Mail and NoServer2Web, to prevent all servers from accessing the mail and web server and to prevent the proxy server from accessing the main server. The output of PFC is then:

```
No anomalies or errors found
```

It's interesting to note that the isolated analysis of this filter would point out that the rule NoProxy2Server conflicts with the rules Mail and Web. However, these conflicts are filtered out as the conflicting datagrams are all within a single interface and would not go through the filter.

The final rule set is:

• Filter SingleF:	NoServer2Mail	from MainS	to 10.1.1.3/32:23	→	Deny
	NoServer2Web	from MainS	to 10.1.1.2/32:80	→	Deny
	Mail	from any	to 10.1.1.3/32:23	→	Accept
	Web	from any	to 10.1.1.2/32:80	→	Accept
	NoProxy2Internal	from 10.1.1.4/32	to 10.0.0.0/8	→	Deny
	ProxyA	from 10.1.1.4/32	to any	→	Accept
	Users2Main	from Users	to MainS	→	Accept
	Users2File	from Users	to 10.1.1.4/32	→	Accept
	Default	from any	to any	→	Deny

• Assertions:	Users2Proxy	from Users	to 10.1.1.4/32:3180	→	Accept
	NoServer	from MainS	to any	→	Deny

Figure 7.4 show the profile of this solution. The tree of rules of the proper solution ca be seen in figure 7.5.

7.3 Case 2: Multiple filters

That solution has worked very well until we had a critical failure on a component that had to be replaced. As we were in a position where network downtime was very expensive, we decided

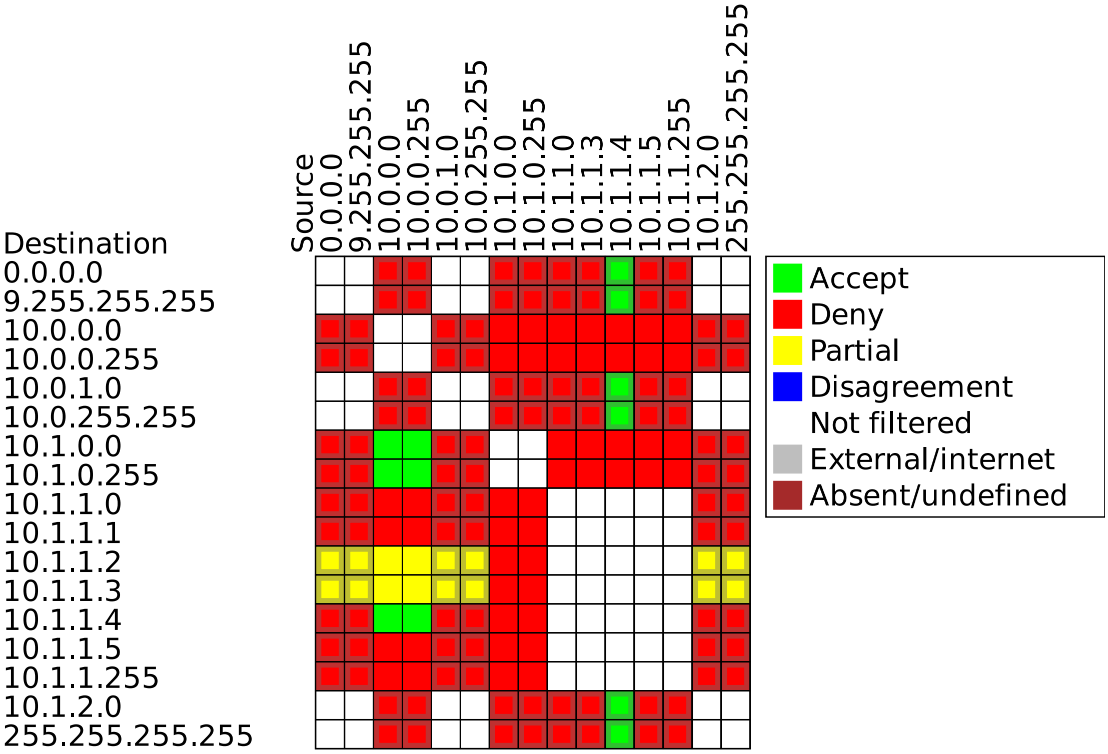


Figure 7.4: Profile of the proper solution of case 1

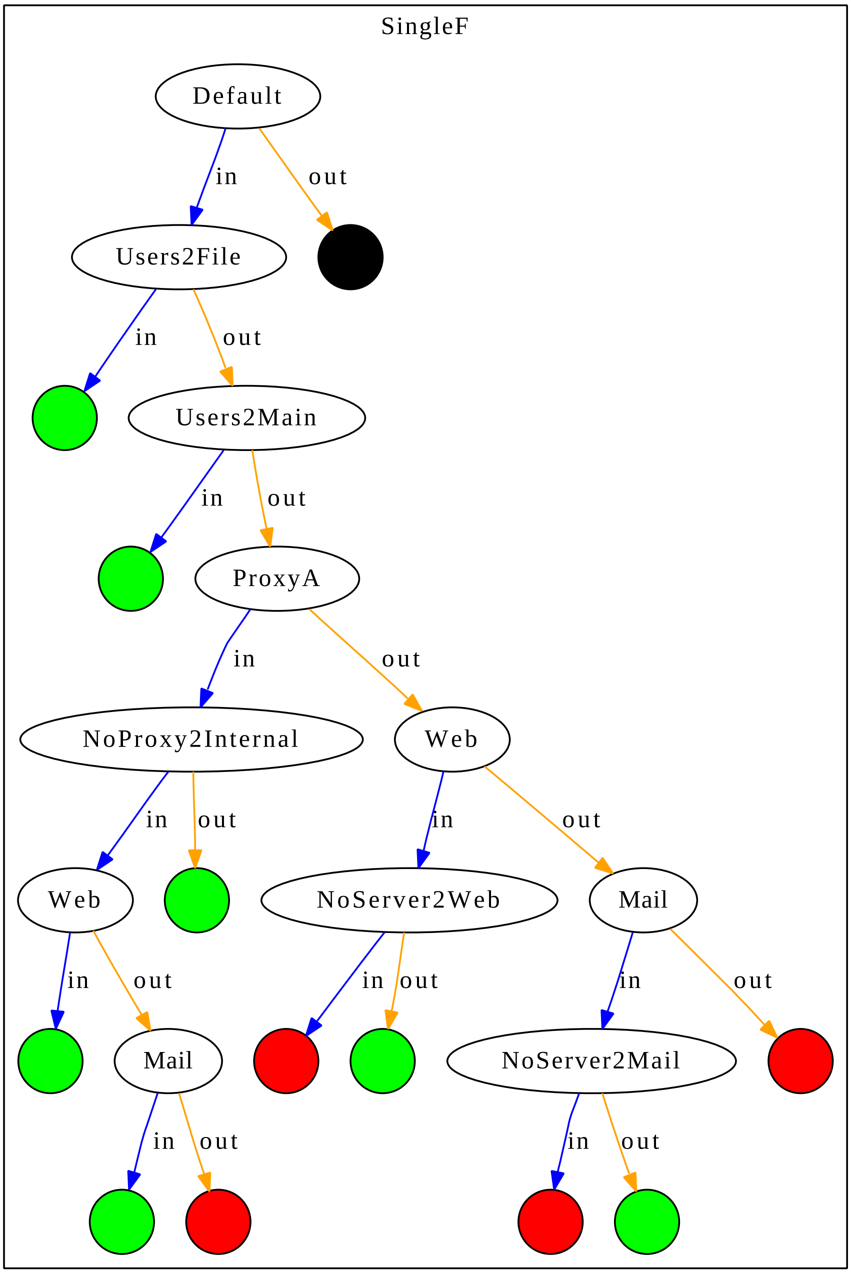


Figure 7.5: Tree of rules of the proper solution of case 1

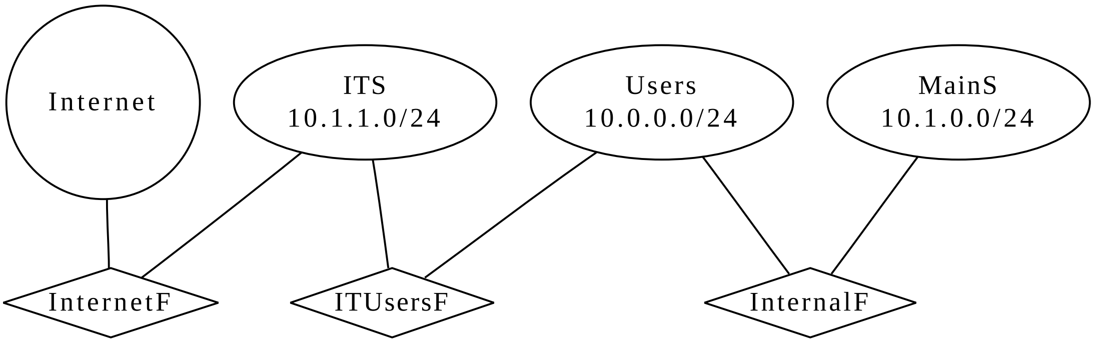


Figure 7.6: Topology of case 2

that the best course of action was to increase redundancy and to buy 3 filters. Those would be placed in a security-minded arrangement, with a replacement priority; that is, if the internal filter failed, it would be replaced with the internet filter while the faulty filter was fixed. The new topology can be seen in figure 7.6.

7.3.1 Naïve solution

To make a safe transition, the IT department decided to replicate the old filter's rules in all filters and remove the rules that were related to networks not directly connected. They also noticed that assertions were a good thing and coded some more. The initial setup can be seen below:

- Filter InternetF:

`Mail`	from any	to 10.1.1.3/32:23	→	Accept
`Web`	from any	to 10.1.1.2/32:80	→	Accept
`NoProxy2Internal`	from 10.1.1.4/32	to 10.0.0.0/8	→	Deny
`ProxyA`	from 10.1.1.4/32	to any	→	Accept
`Default`	from any	to any	→	Deny

- Filter ITUsersF:

`Mail`	from any	to 10.1.1.3/32:23	→	Accept
`Web`	from any	to 10.1.1.2/32:80	→	Accept
`NoProxy2Internal`	from 10.1.1.4/32	to 10.0.0.0/8	→	Deny
`ProxyA`	from 10.1.1.4/32	to any	→	Accept
`Users2File`	from `Users`	to 10.1.1.4/32	→	Accept
`Default`	from any	to any	→	Deny

- Filter InternalF:

NoServer2Mail	from MainS	to 10.1.1.3/32:23	→	Deny
NoServer2Web	from MainS	to 10.1.1.2/32:80	→	Deny
Mail	from any	to 10.1.1.3/32:23	→	Accept
Web	from any	to 10.1.1.2/32:80	→	Accept
NoProxy2Internal	from 10.1.1.4/32	to 10.0.0.0/8	→	Deny
ProxyA	from 10.1.1.4/32	to any	→	Accept
Users2Main	from Users	to MainS	→	Accept
Users2File	from Users	to 10.1.1.4/32	→	Accept
Default	from any	to any	→	Deny

- Assertions:

Users2Proxy	from Users	to 10.1.1.4/32:3180	→	Accept
Users2MainS	from Users	to MainS	→	Accept
Users2Mail	from Users	to 10.1.1.2/32:80	→	Accept
Users2Web	from Users	to 10.1.1.3/32:23	→	Accept
Users2File	from Users	to 10.1.1.4/32	→	Accept
NoMailStart	from 10.1.1.3/32	to any	→	Deny
NoWebStart	from 10.1.1.2/32	to any	→	Deny
NoFileStart	from 10.1.1.4/32	to any	→	Deny
NoProxyInternal	from 10.1.1.4/32	to 10.0.0.0/8	→	Deny
NoServerStart	from MainS	to any	→	Deny

PFC found the following problems:

```
Disagreement
    of   filter InternalF, rule NoServer2Mail
    with filter ITUsersF, rule Mail
    at   [src 10.1.0.0/24 dst 10.1.1.3/32:23]
Disagreement
    of   filter InternalF, rule NoServer2Web
    with filter ITUsersF, rule Web
    at   [src 10.1.0.0/24 dst 10.1.1.2/32:80]
Irrelevant rule ProxyA in filter ITUsersF
Irrelevant rule Mail in filter InternalF
Irrelevant rule Web in filter InternalF
Irrelevant rule ProxyA in filter InternalF
Irrelevant rule Users2File in filter InternalF
Assertion NoFileStart  violated at
    [src 10.1.1.4/32]
```

Although the solution implements the desired behaviour, it is nowhere near minimal, and has some other issues.

The first problems are the disagreements. They are reported because even though ITUsersF is not connected to MainS, it must agree with InternalF on whether the mail and web servers

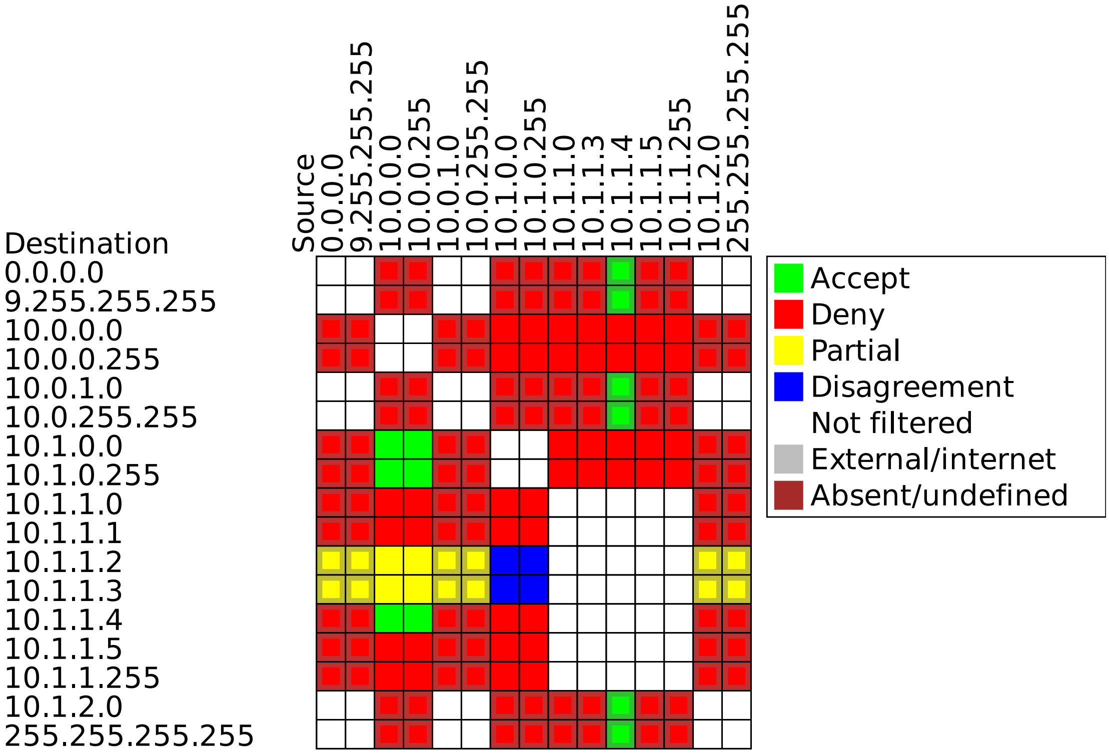


Figure 7.7: Profile of the naïve solution of case 2

can connect to the main servers. To fix it, we add NoServer2Mail and NoServer2Web to ITUsersF.

Following the disagreements, some irrelevant rules are pointed out. It is safe to remove the corresponding rules.

The last problem is the assertion failure. It happens because, as in the single filter case, the proxy and the file server are the same PC. This time it is better to split them for the sake of security than remove the assertion.

7.3.2 Proper solution

After removing the irrelevant rules, the following result is generated:

```
ITUsersF:
  Redundant rule NoProxy2Internal
InternalF:
  Redundant rule NoServer2Mail
  Redundant rule NoServer2Web
  Redundant rule NoProxy2Internal
Assertion Users2Proxy  violated at
    [src 10.0.0.0/24 dst 10.1.1.4/32:3180]
```

Besides the redundant rules, the failed assertions shows that now that the file server and proxy are not on the same machine, an explicit rule for the proxy is needed. We remove the redundant rules, and add the proxy rule to get:

```
No anomalies or errors found
```

The profile is displayed in figure 7.8. It is easy to see that the profile is equivalent to the final profile of the single filter case (figure 7.4), except for the proxy/file server split.

The final setup is below:

- Filter InternetF:

Mail	from any	to 10.1.1.3/32:23	→	Accept
Web	from any	to 10.1.1.2/32:80	→	Accept
NoProxy2Internal	from 10.1.1.4/32	to 10.0.0.0/8	→	Deny
ProxyA	from 10.1.1.4/32	to any	→	Accept
Default	from any	to any	→	Deny

- Filter ITUsersF:

NoServer2Mail	from MainS	to 10.1.1.3/32:23	→	Deny
NoServer2Web	from MainS	to 10.1.1.2/32:80	→	Deny
Mail	from any	to 10.1.1.3/32:23	→	Accept
Web	from any	to 10.1.1.2/32:80	→	Accept
Users2Proxy	from Users	to 10.1.1.4/32:3180	→	Accept
Users2File	from Users	to 10.1.1.5/32	→	Accept
Default	from any	to any	→	Deny

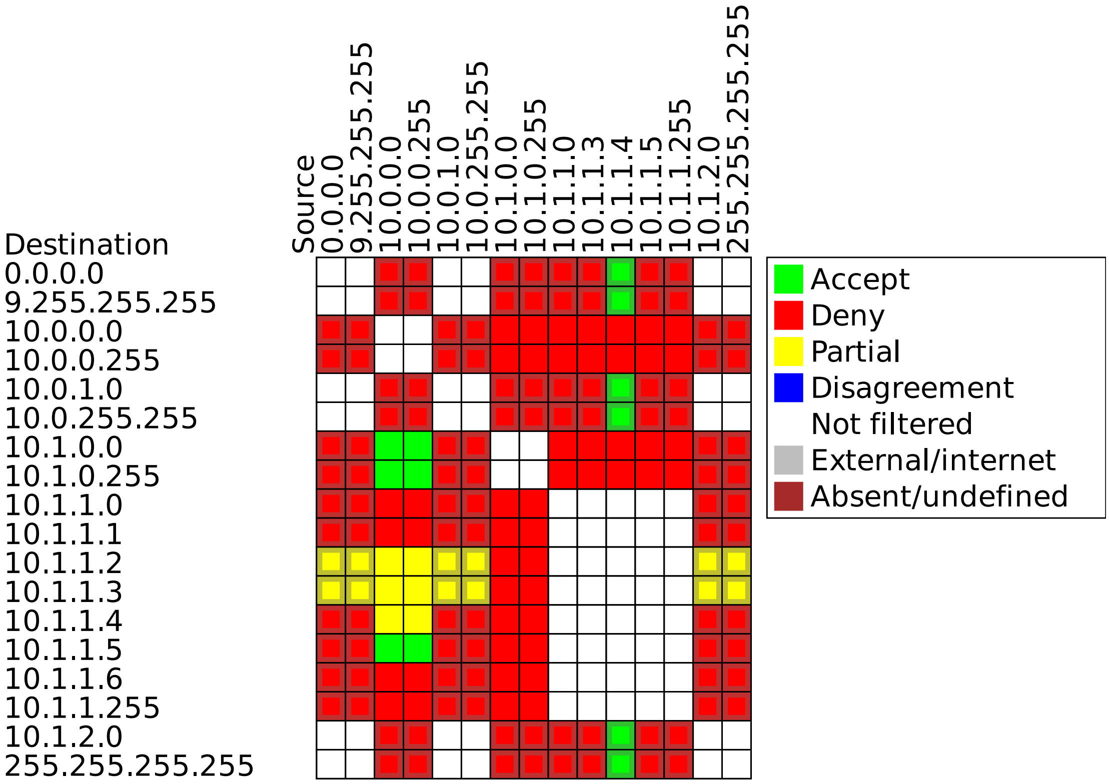


Figure 7.8: Profile of the proper solution of case 2

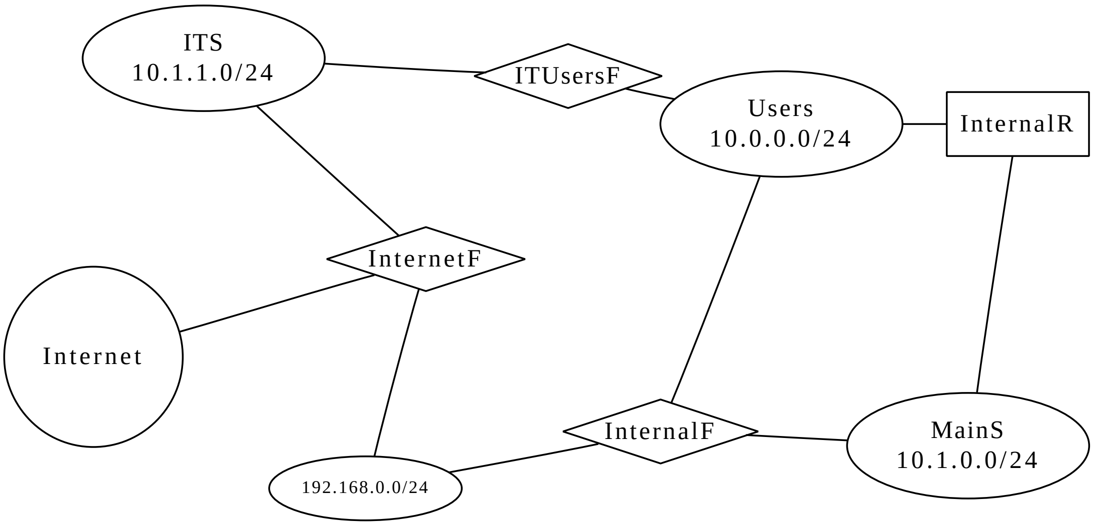


Figure 7.9: Topology of case 3

- Filter InternalF:

Users2Main	from Users	to MainS	→	Accept
Default	from any	to any	→	Deny

- Assertions:

Users2Proxy	from Users	to 10.1.1.4/32:3180	→	Accept
Users2MainS	from Users	to MainS	→	Accept
Users2Mail	from Users	to 10.1.1.2/32:80	→	Accept
Users2Web	from Users	to 10.1.1.3/32:23	→	Accept
Users2File	from Users	to 10.1.1.5/32	→	Accept
NoMailStart	from 10.1.1.3/32	to any	→	Deny
NoWebStart	from 10.1.1.2/32	to any	→	Deny
NoFileStart	from 10.1.1.5/32	to any	→	Deny
NoProxyInternal	from 10.1.1.4/32	to 10.0.0.0/8	→	Deny
NoServerStart	from MainS	to any	→	Deny

By increasing the number of filters with a carefully thought topology, we have not only increased security but also decreased the complexity of each filter, making maintenance easier.

7.4 Case 3: Even more redundancy

Even though the current setup allows a filter to fail without compromising the whole network, it was decided that even more redundancy was desired. The extreme filters InternetF and InternalF were then connected, and a router between Users and MainS was deployed. The resulting topology can be seen in figure 7.9.

This setup prevents the loss of connectivity between Users and the internet when the filter ITUsersF fails and between Users and MainS when the filter InternalF fails.

7.4.1 Naïve solution

Keeping the current rule set generates the profile in figure 7.10. PFC reports:

```
Disagreement
    of   filter ITUsersF, rule Default
    with filter InternetF, rule ProxyA
    at   [src 10.1.1.4/32 dst 0.0.0.0-9.255.255.255,
          src 10.1.1.4/32 dst 11.0.0.0-192.167.255.255,
          src 10.1.1.4/32 dst 192.168.0.0/24,
          src 10.1.1.4/32 dst 192.168.1.0-255.255.255.255]
Disagreement
    of   filter InternalF, rule Default
    with filter ITUsersF, rule Mail
    at   [src 10.0.0.0/24 dst 10.1.1.3/32:23,
          src 192.168.0.0/24 dst 10.1.1.3/32:23]
Disagreement
    of   filter InternalF, rule Default
    with filter ITUsersF, rule Users2File
    at   [src 10.0.0.0/24 dst 10.1.1.5/32]
Disagreement
    of   filter InternalF, rule Default
    with filter ITUsersF, rule Users2Proxy
    at   [src 10.0.0.0/24 dst 10.1.1.4/32:3180]
Disagreement
    of   filter InternalF, rule Default
    with filter ITUsersF, rule Web
    at   [src 10.0.0.0/24 dst 10.1.1.2/32:80,
          src 192.168.0.0/24 dst 10.1.1.2/32:80]
Disagreement
    of   filter InternalF, rule Default
    with filter InternetF, rule Mail
    at   [src 10.0.0.0/24 dst 10.1.1.3/32:23,
          src 10.1.0.0/24 dst 10.1.1.3/32:23,
          src 192.168.0.0/24 dst 10.1.1.3/32:23]
Disagreement
    of   filter InternalF, rule Default
    with filter InternetF, rule Web
    at   [src 10.0.0.0/24 dst 10.1.1.2/32:80,
          src 10.1.0.0/24 dst 10.1.1.2/32:80,
          src 192.168.0.0/24 dst 10.1.1.2/32:80]
Disagreement
    of   filter InternalF, rule Users2Main
    with filter ITUsersF, rule Default
    at   [src 10.0.0.0/24 dst 10.1.0.0/24]
Block in filter InternalF
    at [src 0.0.0.0-9.255.255.255 dst 10.1.1.2/32:80,
        src 0.0.0.0-9.255.255.255 dst 10.1.1.3/32:23,
        src 10.0.1.0-10.0.255.255 dst 10.1.1.2/32:80,
        src 10.0.1.0-10.0.255.255 dst 10.1.1.3/32:23,
        src 10.1.2.0-192.167.255.255 dst 10.1.1.2/32:80,
        src 10.1.2.0-192.167.255.255 dst 10.1.1.3/32:23,
        src 192.168.1.0-255.255.255.255 dst 10.1.1.2/32:80,
        src 192.168.1.0-255.255.255.255 dst 10.1.1.3/32:23]
Leak between MainS and Users in path [router InternalR]
Assertion Users2Proxy  violated at
    [src 10.0.0.0/24 dst 10.1.1.4/32:3180]
Assertion Users2Mail  violated at
    [src 10.0.0.0/24 dst 10.1.1.2/32:80]
Assertion Users2Web  violated at
    [src 10.0.0.0/24 dst 10.1.1.3/32:23]
Assertion Users2File  violated at
    [src 10.0.0.0/24 dst 10.1.1.5/32]
```

Now that InternalF is connected to InternetF, they must agree on the profile of the IT servers network. The last 7 disagreements are reported because of this. The other disagreement is reported because ITUsersF does not allow the proxy to access the internet. With the new connection, that is a possible path, and will be used if the port of InternetF that connects it to ITS fails.

Moving on to the blocks, it is easy to see that they are reported because the filters do now allow the traffic that previously would not go through them: InternalF must support IT and internet traffic; ITUsersF and InternetF must support the traffic to MainS.

On the other hand, the leak reported was caused by the presence of a router in parallel with InternalF. That filter did not allow traffic from the servers into the user network, but the router does. The only solution in this case is to replace the router with a filter or eliminate the router completely. We will do the former.

At last, the assertion problem was caused by the disagreements, that prevent proper assertion evaluation.

7.4.2 Proper solution

Now, there is so much redundancy, that almost every path is a possible one. The best course of action is to replicate the rules of the single filter case in every filter, including the new InternalF2 which replaced InternalR. That makes PFC give us:

```
Assertion Users2Proxy  violated at
    [src 10.0.0.0/24 dst 10.1.1.4/32:3180]
```

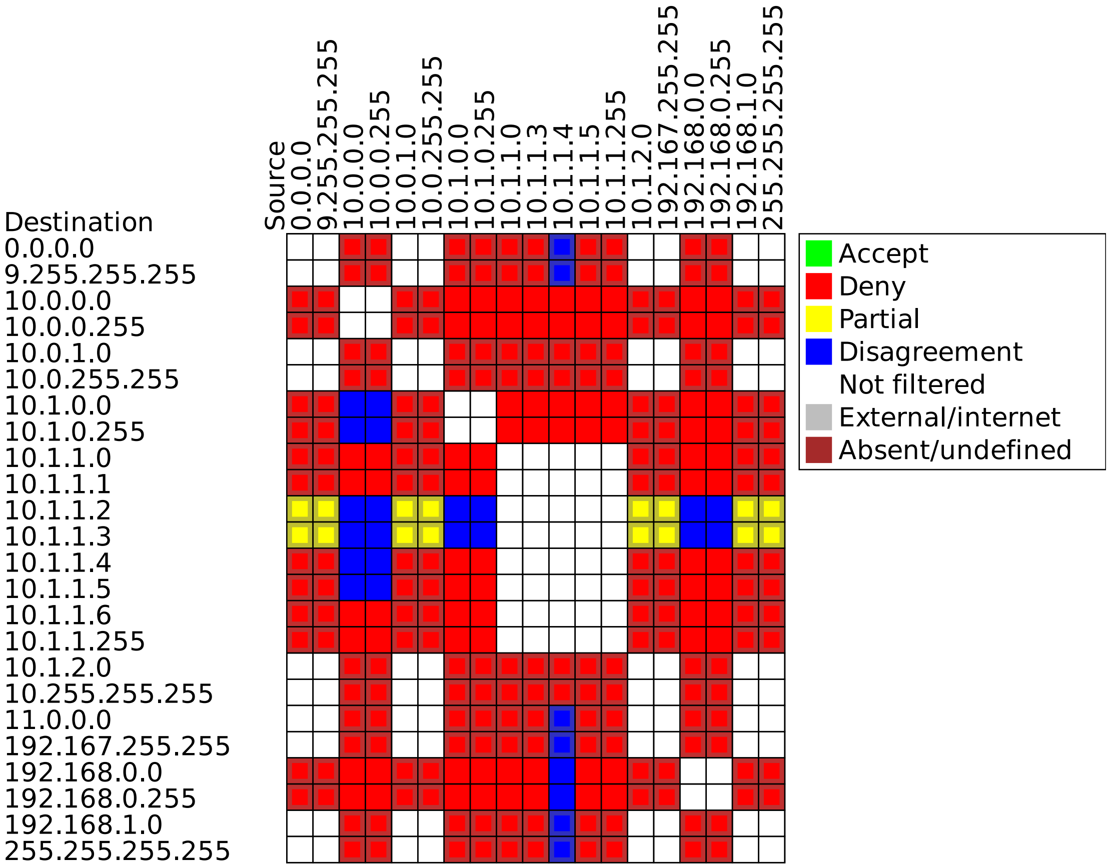


Figure 7.10: Profile of the naïve solution of case 3

That, of course, happened because the IP address of the proxy was changed in case 2. Adding the rule Users2Proxy to every filter, we get from PFC:

```
No anomalies or errors found
```

The profile can be seen in figure 7.11, the topology is in figure 7.12, while the complete setup is below.

- Filter InternetF:

NoServer2Mail	from MainS	to 10.1.1.3/32:23	→	Deny
NoServer2Web	from MainS	to 10.1.1.2/32:80	→	Deny
Mail	from any	to 10.1.1.3/32:23	→	Accept
Web	from any	to 10.1.1.2/32:80	→	Accept
NoProxy2Internal	from 10.1.1.4/32	to 10.0.0.0/8	→	Deny
ProxyA	from 10.1.1.4/32	to any	→	Accept
Users2Main	from Users	to MainS	→	Accept
Users2File	from Users	to 10.1.1.5/32	→	Accept
Users2Proxy	from Users	to 10.1.1.4/32:3180	→	Accept
Default	from any	to any	→	Deny

- Filter ITUsersF:

NoServer2Mail	from MainS	to 10.1.1.3/32:23	→	Deny
NoServer2Web	from MainS	to 10.1.1.2/32:80	→	Deny
Mail	from any	to 10.1.1.3/32:23	→	Accept
Web	from any	to 10.1.1.2/32:80	→	Accept
NoProxy2Internal	from 10.1.1.4/32	to 10.0.0.0/8	→	Deny
ProxyA	from 10.1.1.4/32	to any	→	Accept
Users2Main	from Users	to MainS	→	Accept
Users2File	from Users	to 10.1.1.5/32	→	Accept
Users2Proxy	from Users	to 10.1.1.4/32:3180	→	Accept
Default	from any	to any	→	Deny

- Filter InternalF:

NoServer2Mail	from MainS	to 10.1.1.3/32:23	→	Deny
NoServer2Web	from MainS	to 10.1.1.2/32:80	→	Deny
Mail	from any	to 10.1.1.3/32:23	→	Accept
Web	from any	to 10.1.1.2/32:80	→	Accept
NoProxy2Internal	from 10.1.1.4/32	to 10.0.0.0/8	→	Deny
ProxyA	from 10.1.1.4/32	to any	→	Accept
Users2Main	from Users	to MainS	→	Accept
Users2File	from Users	to 10.1.1.5/32	→	Accept
Users2Proxy	from Users	to 10.1.1.4/32:3180	→	Accept
Default	from any	to any	→	Deny

- Filter InternalF2:

`NoServer2Mail`	from `MainS`	to 10.1.1.3/32:23	→	Deny
`NoServer2Web`	from `MainS`	to 10.1.1.2/32:80	→	Deny
`Mail`	from any	to 10.1.1.3/32:23	→	Accept
`Web`	from any	to 10.1.1.2/32:80	→	Accept
`NoProxy2Internal`	from 10.1.1.4/32	to 10.0.0.0/8	→	Deny
`ProxyA`	from 10.1.1.4/32	to any	→	Accept
`Users2Main`	from `Users`	to `MainS`	→	Accept
`Users2File`	from `Users`	to 10.1.1.5/32	→	Accept
`Users2Proxy`	from `Users`	to 10.1.1.4/32:3180	→	Accept
`Default`	from any	to any	→	Deny

- Assertions:

`Users2Proxy`	from `Users`	to 10.1.1.4/32:3180	→	Accept
`Users2MainS`	from `Users`	to `MainS`	→	Accept
`Users2Mail`	from `Users`	to 10.1.1.2/32:80	→	Accept
`Users2Web`	from `Users`	to 10.1.1.3/32:23	→	Accept
`Users2File`	from `Users`	to 10.1.1.5/32	→	Accept
`NoMailStart`	from 10.1.1.3/32	to any	→	Deny
`NoWebStart`	from 10.1.1.2/32	to any	→	Deny
`NoFileStart`	from 10.1.1.5/32	to any	→	Deny
`NoProxyInternal`	from 10.1.1.4/32	to 10.0.0.0/8	→	Deny
`NoServerStart`	from `MainS`	to any	→	Deny

Source Destination	0.0.0.0	9.255.255.255	10.0.0.0	10.0.0.255	10.0.1.0	10.0.255.255	10.1.0.0	10.1.0.255	10.1.1.0	10.1.1.3	10.1.1.4	10.1.1.5	10.1.1.255	10.1.2.0	192.167.255.255	192.168.0.0	192.168.0.255	192.168.1.0	255.255.255.255
0.0.0.0			🟥	🟥			🟥	🟥	🟥	🟥	🟩	🟥	🟥			🟥	🟥		
9.255.255.255			🟥	🟥			🟥	🟥	🟥	🟥	🟩	🟥	🟥			🟥	🟥		
10.0.0.0	🟥	🟥			🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
10.0.0.255	🟥	🟥			🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
10.0.1.0			🟥	🟥			🟥	🟥	🟥	🟥	🟩	🟥	🟥			🟥	🟥		
10.0.255.255			🟥	🟥			🟥	🟥	🟥	🟥	🟩	🟥	🟥			🟥	🟥		
10.1.0.0	🟥	🟥	🟩	🟩	🟥	🟥			🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
10.1.0.255	🟥	🟥	🟩	🟩	🟥	🟥			🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
10.1.1.0	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥						🟥	🟥	🟥	🟥	🟥	🟥
10.1.1.1	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥						🟥	🟥	🟥	🟥	🟥	🟥
10.1.1.2	🟨	🟨	🟨	🟨	🟨	🟨	🟥	🟥						🟨	🟨	🟨	🟨	🟨	🟨
10.1.1.3	🟨	🟨	🟨	🟨	🟨	🟨	🟥	🟥						🟨	🟨	🟨	🟨	🟨	🟨
10.1.1.4	🟥	🟥	🟨	🟨	🟥	🟥	🟥	🟥						🟥	🟥	🟥	🟥	🟥	🟥
10.1.1.5	🟥	🟥	🟩	🟩	🟥	🟥	🟥	🟥						🟥	🟥	🟥	🟥	🟥	🟥
10.1.1.6	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥						🟥	🟥	🟥	🟥	🟥	🟥
10.1.1.255	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥						🟥	🟥	🟥	🟥	🟥	🟥
10.1.2.0			🟥	🟥			🟥	🟥	🟥	🟥	🟩	🟥	🟥			🟥	🟥		
192.167.255.255			🟥	🟥			🟥	🟥	🟥	🟥	🟩	🟥	🟥			🟥	🟥		
192.168.0.0	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟥	🟥	🟥	🟥			🟥	🟥
192.168.0.255	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟥	🟥	🟥	🟥			🟥	🟥
192.168.1.0			🟥	🟥			🟥	🟥	🟥	🟥	🟩	🟥	🟥			🟥	🟥		
255.255.255.255			🟥	🟥			🟥	🟥	🟥	🟥	🟩	🟥	🟥			🟥	🟥		

- 🟩 Accept
- 🟥 Deny
- 🟨 Partial
- 🟦 Disagreement
- Not filtered
- ⬜ External/internet
- 🟫 Absent/undefined

Figure 7.11: Profile of the proper solution of case 3

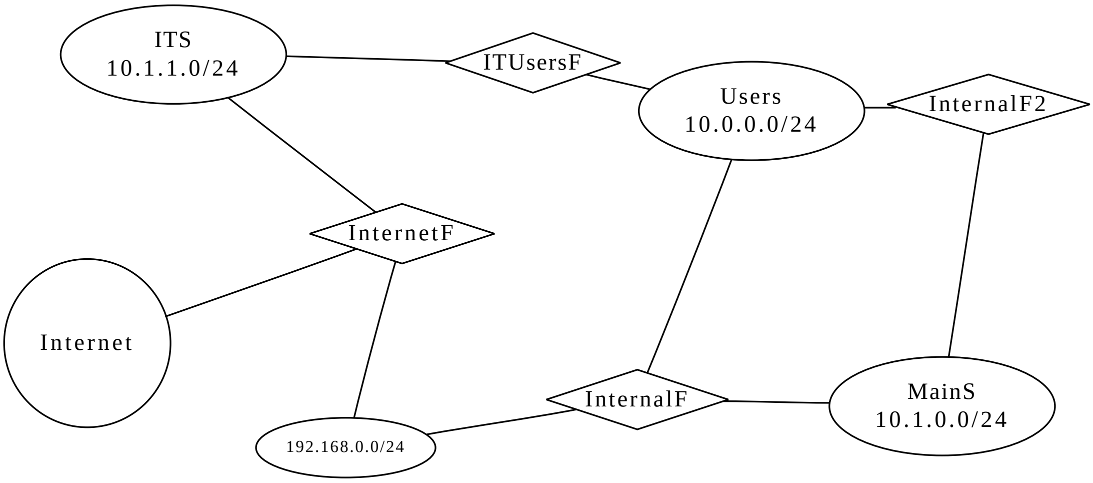


Figure 7.12: Topology of the proper solution of case 3

CONCLUSION

This master thesis presents the results of a research and work on coherence in packer filters. It also documents the design and demonstrates *PFC*, the Packet Filter Checker that was developed.

From chapter 1 to chapter 3, a research of the environment is made. Concepts of how the network operates and existing work in packet filter coherence are presented. It is shown that there is room for improvement, as the existing works on algebraic verification of isolated filters have some limitations, and there is no checker for multiple filters.

In chapter 4, the first contribution is made: a new algebraic checker for isolated filters. By using a more robust model based on sets and different models for rule relation and anomalies, the checker is not susceptible to the problems of false positives and negatives that could be found in previous works. The model and the design of the anomalies are based on the assumption that the order of rules is the main source of errors in a filter. So, every time the order of the rules is plain wrong (a rule is never used), trivial (inversion would result in a wrong order), or irrelevant (inversion would not change the filter), an anomaly is reported. Even though the administrator may have to add rules that do nothing to the behaviour to the filter only to clear conflicts, an anomaly-free filter is much easier to maintain: the administrator can change the order of the rules without fearing a change in the behaviour of the filter, and new rules rise anomalies indicating the rules they interfere with when placed ambiguously.

In chapter 5, the algebraic checker for distributed filters is presented. This checker is the main contribution of this work. By using the set algebra of the previous chapter and a model of how the filters should behave, a set of distributed anomalies was developed, and the isolated checker extended into a distributed one. Once again, we base the anomalies on a single assumption: that every possible path for each datagram inside the network must present the same behaviour. That is a reasonable assumption, as the network should not accept more types of datagrams from a point to another when a filter is too loaded or fails. As with the isolated filter design, the administrator may have to add rules in some filters to clear all anomalies, but an anomaly-free network is easier to maintain and change – each new rule that needs to be in further filters will lead to an anomaly.

Chapter 6 tackles functional verification of distributed filters with assertions. The idea is simple, but very useful: the verification of some properties of the resulting accessibility profile. By using assertions, the administrator can code the expected behaviour of the filter in a higher level, and let PFC check if the implementation conforms. The value of this idea demonstrated on the next chapter, the case study.

The case study is made of a series of iterations on one fictional company. In each iteration, the requirements of the network are changed along with its topology. Even though the security policy doesn't change, it was possible to demonstrate every anomaly.

The examples of the case study as well as the examples of anomalies in the previous chapters were all analysed with PFC. PFC was also used to generate the figures of the profiles and the representations of the examples. For the figures of the topologies, PFC generated an output suitable for use with *Graphviz* (ELLSON; GANSNER, 2008). PFC was developed using Haskell (JONES, 2003), a purely functional programming language with non-strict semantics. Haskell made a difference in the development of the algebraic analyses with its strong, static polymorphic types. On the library front, Parsec (LEIJEN; MEIJER, 2001) provided an easy-to-use parser combinator and QuickCheck (CLAESSEN; HUGHES, 2000) was used to create automatic property tests.

It is in the interest of the network administrator to clear its network of all anomalies, as their absence testify the consistency (conflict; disagreement, blocking, and leaking) and minimalism (invisibility, redundancy; irrelevancy) of the setup. There is one issue, though: the administrator might have to add unnecessary rules to filters only to get them anomaly-free. That can be the case when there are conflicts and disagreements, as these anomalies are report situations of ambiguity where PFC cannot determine the administrator's intention. A possible solution for this issue could be a better integration of the rule checker with the assertion checker: only report conflicts and disagreements when there is no assertion to define the desired behaviour for the analysed region, and take the target of the assertion as the correct one for the rest of the analyses.

Another lesser issue that might arise is the tool's reliance on the uniqueness of IP addresses. The internal representation of the model, and even the syntax of the network description file depend on it. That is an issue as it blocks the support for address translation (NAT). But as it is unusual for a company to have more than one translated network, this issue might not have an impact.

These issues aside, we have achieved our goal: the development and implementation of an algebraic checker for rules in single and networked filters. To our knowledge, there is no other algebraic rule checker that supports networked filter, that is a contribution to the field by itself. Other than that, our single-filter checker has two properties that are found in no other: every anomaly reported can be eliminated by the addition or removal of rules (no persistent false positives) and the rules are analysed in their context, as there are some problems that can only be detected that way. Our tests and the case study show that the solution developed is robust and presents comprehensible information for all the anomalies, and can potentially help the network administrator setup and maintain a secure network.

REFERENCES

AL-SHAER, E. S.; HAMED, H. H. **Design and implementation of firewall policy advisor tools**. [S.l.]: DePaul University, 2002. (CTI-TR-02-006).

AL-SHAER, E. S.; HAMED, H. H. Firewall policy advisor for anomaly detection and rule editing. In: IEEE/IFIP 8TH INT. SYMP. INTEGRATED NETWORK MANAGEMENT (IM 2003), 2003. **Proceedings...** [S.l.: s.n.], 2003. p.17–30.

AL-SHAER, E. S.; HAMED, H. H. Management and translation of filtering security policies. In: INT. CONF. COMMUNICATIONS (ICC 2003), 38., 2003. **Proceedings...** IEEE, 2003.

AL-SHAER, E. S.; HAMED, H. H. Discovery of policy anomalies in distributed firewalls. In: INFOCOM, 2004. **Anais...** IEEE, 2004.

AL-SHAER, E. S.; HAMED, H. H. Modeling and management of firewall policies. **IEEE Trans. Network and Service Management**, [S.l.], v.1, n.1, apr 2004.

ALMQUIST, P. **RFC 1349**: type of service in the Internet Protocol suite. 1992.

BABOESCU, F.; SINGH, S.; VARGHESE, G. Packet Classification for Core Routers: is there an alternative to CAMs? In: INFOCOM, 2003. **Anais...** [S.l.: s.n.], 2003.

BABOESCU, F.; VARGHESE, G. Scalable packet classification. In: SIGCOMM, 2001. **Anais...** ACM, 2001. p.199–210.

BABOESCU, F.; VARGHESE, G. Fast and scalable conflict detection for packet classifiers. **J. Computer & Telecommunications Networking**, [S.l.], v.42, n.6, p.717–735, aug 2003.

BAKER, F. **RFC 1812**: requirements for IP version 4 routers. 1995.

BARTAL, Y.; MAYER, A. J.; NISSIM, K.; WOOL, A. Firmato: a novel firewall management toolkit. **ACM Trans. Comput. Syst.**, [S.l.], v.22, n.4, p.381–420, 2004.

CLAESSEN, K.; HUGHES, J. QuickCheck: a lightweight tool for random testing of haskell programs. In: ICFP, 2000. **Anais...** ACM, 2000. p.268–279.

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. **Introduction to Algorithms, Second Edition**. [S.l.]: The MIT Press and McGraw-Hill Book Company, 2001.

DEVLIN, K. **The Joy of Sets**: fundamentals of contemporary set theory. [S.l.]: Springer Verlag, 1993.

ELLSON, J.; GANSNER, E. **Graphviz**. Available at `http://www.graphviz.org`. Accessed: 21 Feb. 2008.

EPPSTEIN, D.; MUTHUKRISHNAN, S. Internet packet filter management and rectangle geometry. In: SODA, 2001. **Anais...** ACM, 2001. p.827–835.

FULLER, V.; LI, T.; YU, J.; VARADHAN, K. **RFC 1519**: Classless Inter-Domain Routing (CIDR): an address assignment and aggregation strategy. 1993.

HARI, A.; SURI, S.; PARULKAR, G. M. Detecting and Resolving Packet Filter Conflicts. In: INFOCOM, 2000. **Anais...** IEEE, 2000. p.1203–1212.

HAWKINSON, J.; BATES, T. **RFC 1930**: guidelines for creation, selection, and registration of an autonomous system (AS). 1996.

HEMNI, A. (Ed.). **Firewall Policies and VPN Configurations**. Rockland, MA: Syngress Publishing Inc., 2006.

JONES, S. P. (Ed.). **Haskell 98 Language and Libraries – The Revised Report**. Cambridge, England: Cambridge University Press, 2003.

KNUTH, D. **The Art of Computer Programming, Volume 1**: fundamental algorithms. third.ed. [S.l.]: Addison-Wesley, 1997. v.1.

KNUTH, D. **The Art of Computer Programming, Volume 3**: sorting and searching. second.ed. [S.l.]: Addison-Wesley, 1998. v.3.

LAKSHMAN, T. V.; STILIADIS, D. High-Speed Policy-Based Packet Forwarding Using Efficient Multi-Dimensional Range Matching. In: SIGCOMM, 1998. **Anais...** ACM, 1998. p.203–214.

LEIJEN, D.; MEIJER, E. **Parsec**: direct style monadic parser combinators for the real world. 2001.

MALKIN, G. **RFC 1388**: RIP version 2 carrying additional information. 1993.

MAYER, A. J.; WOOL, A.; ZISKIND, E. Fang: a firewall analysis engine. In: IEEE SYMPOSIUM ON SECURITY AND PRIVACY, 2000. **Anais...** [S.l.: s.n.], 2000. p.177–187.

MAYER, A. J.; WOOL, A.; ZISKIND, E. Offline firewall analysis. **Int. J. Inf. Sec.**, [S.l.], v.5, n.3, p.125–144, jul 2006.

MOY, J. **RFC 2328**: OSPF version 2. 1998.

NICHOLS, K.; BLAKE, S.; BAKER, F.; BLACK, D. **RFC 2474**: definition of the Differentiated Services field (DS field) in the IPv4 and IPv6 headers. 1998.

POSTEL, J. **RFC 768**: User Datagram Protocol. 1980.

POSTEL, J. **RFC 791**: Internet Protocol. 1981.

POSTEL, J. **RFC 792**: Internet Control Message Protocol. 1981.

POSTEL, J. **RFC 793**: Transmission Control Protocol. 1981.

RAMAKRISHNAN, K.; FLOYD, S.; BLACK, D. **RFC 3168**: the addition of Explicit Congestion Notification (ECN) to IP. 2001.

REKHTER, Y.; LI, T. **RFC 1518**: an architecture for IP address allocation with CIDR. 1993.

SRINIVASAN, V.; VARGHESE, G.; SURI, S.; WALDVOGEL, M. Fast and Scalable Layer Four Switching. In: SIGCOMM, 1998. **Anais...** ACM, 1998. p.191–202.

SU, C.-F. High-speed packet classification using segment tree. In: IEEE GLOBAL TELECOMMUNICATIONS CONFERENCE, 2000. **Proceedings...** [S.l.: s.n.], 2000. v.21, p.560–571.

TAYLOR, D. E. Survey and taxonomy of packet classification techniques. **ACM Comput. Surv.**, [S.l.], v.37, n.3, p.238–275, 2005.

VASKOVICH, F. **Nmap Network Scanning**. Available at `http://nmap.org/book`. Accessed: 28 June 2008.

WELTE, H.; AYUSO, P. N. **netfilter/iptables project homepage**. Available at `http://www.netfilter.org`. Accessed: 20 Feb. 2008.

WOO, T. Y. C. A Modular Approach to Packet Classification: algorithms and results. In: INFOCOM (3), 2000. **Anais...** [S.l.: s.n.], 2000. p.1213–1222.

ZIMMERMANN, H. OSI Reference Modell - The ISO Model of Architecture for Open Systems Interconnection. **IEEE Transactions on communication**, [S.l.], v.28, p.425–432, Apr. 1980.

ZWICKY, E. D.; COOPER, S.; CHAPMAN, B. D. **Building Internet Firewalls**. 2.ed. [S.l.]: O'Reilly Associates, 2000.

APÊNDICE A RESUMO DOS PRINCIPAIS RESULTADOS

A.1 Introdução

Este capítulo apresenta um resumo deste trabalho, com foco nos resultados obtidos, em português.

O trabalho em si começa pela apresentação da modelo usado para a análise das regras de um filtro de pacotes. A partir desse modelo, é desenvolvido o conceito de anomalia isolada, que se trata de uma incoerência nas regras de um filtro. Até este ponto, o presente trabalho se assemelha com o trabalho de Al-Shaer e Hamed (AL-SHAER; HAMED, 2002, 2003a, 2004a, 2003b, 2004b). Em seguida, é apresentado o modelo desenvolvido para redes de computadores com filtros e roteadores. Com este modelo, são definidas as anomalias distribuídas, e esta é a principal contribuição deste trabalho. As anomalias distribuídas são as incoerências encontradas entre regras de filtros diferentes e outras incoerências da própria topologia da rede. Depois disso as asserções são apresentadas, que nada mais são do que testes de tráfego feitos sobre o modelo da rede. Em seguida é feito um estudo de caso e o trabalho é finalizado.

Segue um resumo dos principais pontos do trabalho.

A.2 Anomalias isoladas

O modelo de filtro de pacotes desenvolvido considera o filtro como uma função matemática cuja entrada é um datagrama IP e cuja saída é uma ação de duas disponíveis: aceite ou descarte. O filtro é programado pelo administrador da rede através de regras, cada uma fazendo o mapeamento de um conjunto de datagramas para uma ação. As regras são avaliadas sempre na mesma ordem, e a primeira que possuir o datagrama recebido em seu conjunto define a ação a ser tomada para esse.

Trabalhos anteriores (AL-SHAER; HAMED, 2003a) indicam que a maior parte dos erros na configuração de filtros vem da ordenação das regras. Por isso, cada caso no qual a ordenação relativa de duas regras em um filtro não é nem trivial nem irrelevante vai gerar uma anomalia. Além disso, o desempenho de um filtro é inversamente proporcional ao número de regras presente, e por isso vamos também gerar anomalias para regras desnecessárias.

A partir da relação entre os conjuntos de datagramas das regras (sub-conjunto, super-conjunto, igualdade ou disjunção), da sua ordem no filtro e da sua ação, as anomalias isoladas são definidas:

- Invisibilidade: quando todos os datagramas selecionados por uma regra já foram selecionados por regras anteriores - ou seja, a regra nunca vai selecionar nenhum datagrama na posição atual.

- Conflito: quando o conjunto de datagramas selecionados por duas regras com ações diferentes possui uma interseção não-vazia, a regra na posição superior não é um subconjunto da regra inferior, e essa interseção não está totalmente contida em uma terceira regra superior a ambas. Nessas condições, a troca da ordem dessas duas regras não geraria nenhuma anomalia e o comportamento do filtro seria alterado. Para efeito de comparação: se a regra superior for um subconjunto da regra inferior, a troca da ordem vai gerar uma anomalia de invisibilidade.

- Redundância: quando uma regra, mesmo selecionando pacotes devido à sua posição, poderia ser removida sem alteração no comportamento do filtro. Isso indica que todos os pacotes selecionados pela regra redundante são selecionados também por regras inferiores com a mesma ação. Normalmente, a regra é desnecessária mesmo ou há uma ação incorreta em alguma regra.

A anomalia de conflito aponta ambiguidades na ordenação das regras, enquanto que as anomalias de invisibilidade e redundância apontam regras desnecessárias. Em um filtro sem anomalias, qualquer troca na ordem de duas regras vai ou gerar uma anomalia ou não ter efeito no comportamento do filtro. Isso facilita muito a manutenção dessas regras pelo administrador de rede, que pode otimizar o filtro passando as regras mais importantes para o topo sem medo dos potenciais efeitos colaterais.

Com relação aos trabalhos de Al-Shaer e Hamed, que também definem anomalias para regras em filtros isolados, podemos apontar os seguintes pontos como vantagens das anomalias do presente trabalho:

- Visão das regras no contexto geral: a análise das regras é feita dentro do contexto do filtro, e não duas-a-duas. Com isso, não são apontadas anomalias que não aparecem no resultado do filtro. Por exemplo: se um conflito diz respeito a datagramas que são todos selecionados por uma regra superior as duas regras conflitantes, ele não é reportado. Isso evita falsos positivos.

- Possibilidade de remoção de todas as anomalias: as anomalias foram projetadas de forma a permitir sua remoção através da inserção, remoção e alteração da ordem das regras e de sua ação. Com isso, é possível "corrigir" qualquer filtro. No trabalho de Al-Shaer e Hamed, algumas anomalias são inerentes ao comportamento desejado para o filtro.

A.3 Anomalias distribuídas

Para a análise das anomalias distribuídas, é necessário ter a topologia completa da rede, além das regras de cada filtro. A rede é modelada como um grafo, no qual os nós são os filtros e roteadores, e as arestas são as redes que os ligam. Cada datagrama possui uma origem e um destino dentro deste grafo, mas muitas vezes há vários caminhos intermediários possíveis. Neste trabalho, consideramos que todos os caminhos devem apresentar as mesmas permissões para o datagrama: ou a passagem é permitida em todos os caminhos ou proibida. Com isso, a rede se torna mais robusta à falhas, já que a falha de um filtro não vai impedir um acesso que antes era possível ou pior, permitir um acesso anteriormente proibido. Anomalias distribuídas são, por isso, casos nos quais a permissão para determinados datagramas difere dependendo do caminho que este tomar:

- Discordância: quando o filtro mais próximo de uma rede possui uma ação para determinados datagramas que difere da ação do filtro mais próximo da rede de destino desses datagramas. Não há como saber qual dos dois filtros possui a ação correta, que deve ser verificada pelo administrador. Se mais de um filtro puder ser o primeiro (ou último) para determinados datagramas, todos estes filtros devem também concordar.

- Bloqueio: quando o filtro mais próximo da rede de origem e o filtro mais próximo da rede de destino de determinados datagramas concordam que estes devem ser aceitos, mas um filtro intermediário os descarta. Provavelmente a ação incorreta é a deste filtro intermediário.

- Vazamento: quando o filtro mais próximo da rede de origem e o filtro mais próximo da rede de destino de determinados datagramas concordam que estes devem ser descartados, mas há um caminho alternativo comporto apenas por roteadores - que são modelados como filtros que não descartam nenhum datagrama. Neste caso, alguns datagramas podem passar dependendo das condições da rede, por isso o “vazamento.”

- Irrelevância: dada a topologia da rede, só um conjunto limitado de datagramas vai passar por cada filtro. A irrelevância aponta regras que selecionam apenas datagramas que estão fora desse conjunto, e que são, por isso, desnecessárias. O contraste com a redundância aparece no fato da irrelevância usar o conhecimento da topologia da rede para analisar as regras.

As anomalias distribuídas, assim como as isoladas, sempre podem ser eliminadas através da manipulação das regras. Uma rede livre de anomalias garante que não haverá alteração na acessibilidade na presença de falhas ou tráfego intenso. Manter a rede sem anomalias também facilita a manutenção, já que a alteração de um filtro vai gerar anomalias que apontam para todos os outros filtros que precisam ser alterados para que a rede fique consistente novamente.

O modelo e a definição das anomalias distribuídas é a principal contribuição do presente trabalho. Até o momento, não havia nenhum outro trabalho na literatura que analisasse de forma algébrica as regras de uma rede inteira com diversos filtros.

A.4 Asserções e o protótipo PFC

As anomalias verificam a consistência e o minimalismo da rede e das regras dos filtros de pacotes. Contudo, elas não garantem que o comportamento implementado é o desejado.

Asserções são um mecanismo simples que podem ser usado para verificar o comportamento da rede. Possuem a mesma sintaxe de uma regra: um conjunto de pacotes e uma ação. Contudo, as asserções são verificadas sobre o comportamento global da rede. Ou seja, depois de todas anomalias reportadas, é reportado um erro para cada asserção que não corresponder ao implementado. Dessa forma, o administrador da rede pode implementar “testes de unidade” do comportamento da rede e ter mais garantias ainda quando precisar fazer uma manutenção.

Para verificar as asserções e as anomalias, foi criado um protótipo que implementa os algoritmos desenvolvidos. O *Packet Filter Checker* (PFC) recebe a topologia da rede e as regras de cada filtro, e gera um relatório com todas as anomalias e erros de asserção encontrados. O PFC tem também a capacidade de gerar diversos outros artefatos, como os gráficos usados no decorrer deste trabalho. Seu uso pode ser observado melhor no capítulo 7, que demonstra o caso de uso.

Informática UFRGS

PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

"Coherence in Distributed Packet Filters"

por

LEANDRO LISBOA PENZ

Dissertação apresentada aos Senhores:

Ana Cristina
____
Profa. Dra. Ana Cristina Benso da Silva
(PUCRS)

____
Prof. Dr. Lisandro Zambenedetti Granville

____
Prof. Dr. João Cesar Netto

Vista e permitida a impressão.
Porto Alegre, 25/11/2009

Raul F.
____
Prof. Dr. Raul Fernando Weber,
Orientador.

____