Evaluation of Binary Decision Diagrams for Redundancy, Shadowing, Generalisation and Correlation in an Information Sharing Model

Omair Uthmani, Prof William J Buchanan, Dr Lu Fan, Alistair Lawson

May 21, 2013

Abstract

This paper defines a structured methodology which is based on the foundational work of Al-Shaer et al. in [1] and that of Hamed and Al-Shaer in [2]. It defines a methodology for the declaration of policy field elements, through to the syntax, ontology and functional verification stages. In their works of [1] and [2] the authors concentrated on developing formal definitions of possible anomalies between rules in a network firewall rule set. Their work is considered as the foundation for further works on anomaly detection, including those of Fitzgerald et al. [3], Chen et al. [4], Hu et al. [5], among others. This paper extends this work by applying the methods to information sharing policies, and outlines the evaluation related to these.

1 Introduction

In an increasingly connected world, data is becoming a key asset, especially within a Big Data context, where data from different domains can be brought together to provide new in-sights. Most of the systems we have in-place, though, have been built to securely keep data behind highly secure environments, and have difficulty in integrating with other disparate systems. This is now a major barrier to using data in a wide range of applications. Along with this, information sharing has many regulatory constraints, which often disable information sharing across domains, but, with carefully managed information architectures, it is possible to overcome many of these problems. An important challenge is thus to support information sharing across different domains and groups across multiple information systems. In the context of this paper, a domain is defined as the governance (and possible ownership) of a set of data, which is exposed to others through well-managed services.

The problem of providing governance around trusted infrastructures is highlighted by Boris Evelson who outlines that:

“Big data is such a new area that nobody has developed governance procedures and policies, there are more questions than answers.”

This paper outlines a novel modelling method for information sharing policies using Binary Decision Diagrams using a syntax defined in [6] and which is currently being used in a range of information sharing applications in health and social care.

2 Literature Review

2.1 Introduction

Information-sharing between police and community partner organisations lies at the very centre of the Intelligence-Led Policing model. However, a number of key barriers need to be overcome to ensure that this sharing occurs within a legal framework and is proportionate to the goal of the sharing. This chapter outlines a perspective on the Intelligence-Led Policing model, from its historical origins to the present state, including recent developments and identification of areas of concern. A number of examples of information sharing frameworks and a brief outline of their characteristics is included in order to highlight some of the advantages of collaborative working. The chapter defines certain key issues affecting enterprise systems interoperability, a key barrier to information sharing, and proposed measures for addressing these.

2.2 Policies

Policies are sets of rules which govern the behaviour of a system. The correct definition, implementation and audit of policies is essential to ensure that the system is compliant with its governing rules. High-level security policies are typically defined by an organisation's upper management and usually describe the *what*, or goals, of the security of the organisation. Without this definition of security goals, it is difficult to use security mechanisms effectively [7]. The lower-level implementation, or technical policies, are then created from the overall high-level policy. This is the *how* of the organisation's security policy and it is used to enforce the security policy. The general term *policy* is used interchangeably in literature to describe both the high-level policies, as well as the low-level implementation rules. In this paper, the term *policy* is used explicitly to refer to high-level policies while the term *rules* is used to refer to lower-level implementation rules. The processes involved in security policy creation and implementation are illustrated in Figure 1. A useful definition, taken from The Site Security handbook - RFC2196 [7], of an overarching security policy is:

> 'a formal statement of the rules by which people who are given access to an organization's technology and information assets must abide'

The principles and guidelines outlined in RFC2196 provide a good reference for system administrators and policy decision makers when creating security policies. At its core is a five-step iterative process detailing the processes for the creation and maintenance of security policies. A key element of this is that the definition of a policy is an ongoing process, with regular reviews and auditing of mechanisms and providing feedback to improve it. This highlights a fundamental principle that the process of creation, development and implementation of a policy is an integral part of the design of a system and is also a continuously evolving process, rather than simply being the implementation of various security products [8, 9].

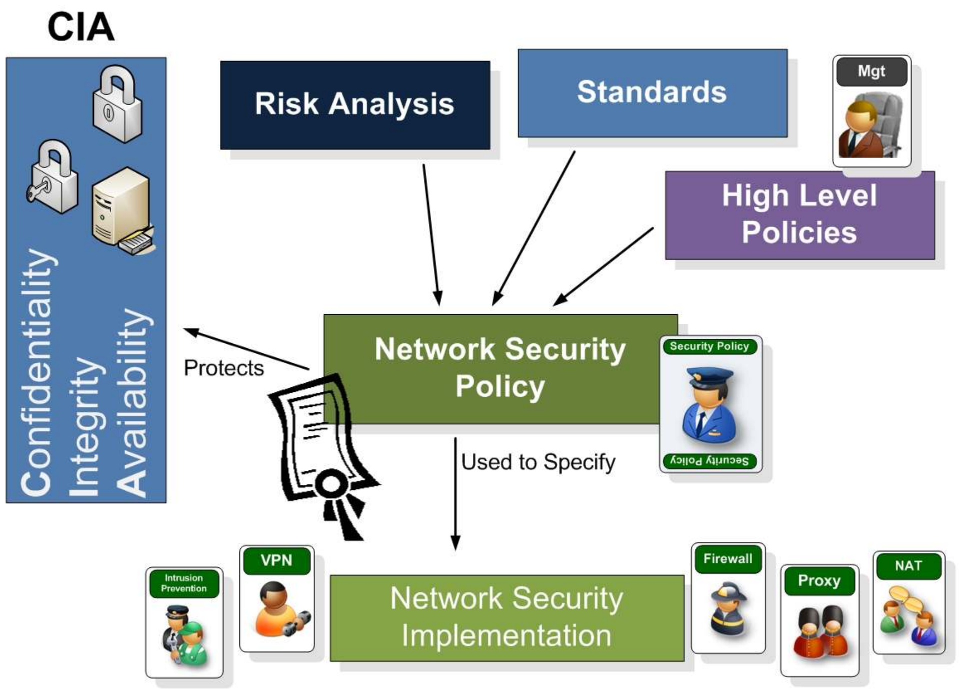


Figure 1: Policy creation and implementation processes.

2.3 Policy Enforcement

A number of industry standards and frameworks also highlight further principles of the policy creation process based on industry best practices. Two widely-used frameworks are Control Objectives for Information and Related Technology (COBIT) and the ISO27002 Code of Practice for Information Security Management (previously ISO 17799) which provide detailed industry standards in information security management and audit compliance [10]. A common theme, often emphasised in these guidelines, is the involvement of end-users in the implementation of policies, and their understanding of the goals the policies aim to enforce. End-users' awareness and comprehension of policies is often cited as a key factor in the policy's successful implementation. Danchev [11] refers to this as an awareness program which emphasises that even the most technologically advanced security measures, such as firewalls and intrusion detection and prevention systems, can be rendered useless by careless or misinformed end-users. As illustrated in Figure 1, defining security policies includes details on how the integrity, availability and confidentiality of assets belonging to an organisation are protected. In order to accomplish this, Sandhu et al. state in [12] that policy documentation should contain details of authentication, authorisation and auditing procedures, which are used to enforce higher policy goals and, often, should also specify the deployment of services including authentication systems, encryption and firewalls as part of the mechanisms used to enforce these policies. Samarati and de Vimercati [13] also show that, typically, the authorisation aspect of policy definition is usually of primary importance as it specifies *who* has access to *what* resource. Once this has been specified, the authentication policy is then used to reinforce this by defining what criteria a user needs to satisfy to ensure that they are who they claim to be. Finally, the auditing policy allows logs to be kept of any relevant action carried out by users on objects. This paper focuses on the authorisation aspect of policy definition as it forms the core of higher-level policy, and because both the authentication and auditing aspects of policy definition are inherently dependent on the authorisation conditions. Sandhu and Samarati refer to authorisation in [14] as the specification of what one party allows another to do with respect to the resources and objects mediated by the former. In terms of information sharing, this relates to ensuring that authorised individuals can access resources and that others cannot. It also means ensuring that requests to access a specific resource are only granted if the request is permitted in the policy definition. In terms of networks, the most commonly used access control mechanisms are firewalls and filtering routers which control access to resources by filtering network traffic, only allowing access that is specified by the security policy [15]. It is, however, difficult to ensure that lower-level implementation policies are always compliant with their higher-level security goals. This issue is frequently aggravated in multi-domain environments where policies need to be translated, interpreted and applied between the different domains involved, such as those of the police and various community partners. A possible solution to this, as described by Susan Hinrichs in [16], is to define the goals of a policy at a high-level and decouple these from the specifics of its implementation, which are defined at a lower-level. For example, Hinrichs states that a high-level policy statement may be in the form shown in Listing 1. Such a statement does not identify implementation details, such as which machines are part of the Engineering Department or on what port the server is listening. High-level policies, therefore, need to be translated into a format which is understandable by policy enforcement devices, such as routers or firewalls. Listing 2, for example, shows a firewall rule derived from the higher-level policy shown in Listing 1.

Listing 1: Example of a high-level policy.

```
Engineering Department should have access to the
    web server
```

Listing 2: Example of a firewall rule derived from Listing 1.

```
permit TCP traffic on port 80 from 192.168.56.0/24
    to 128.45.67.34/32
```

Abrams and Bailey also propose an approach in [17] which is similar to Hinrichs' top-down approach. They suggest the use of a layered concept, where a policy is first defined at a high-level and then abstracted at a number of lower-levels according to layers. Further, they suggest that users are grouped at different layers based on how the policy affects them. The higher-level policy can then be abstracted according to the layer at which the user resides. This approach allows the policy to be framed in a context that is relevant to the user and in terms with which the user is familiar. Abrams and Bailey suggest the following three-layered approach:

1. Top-level management.
2. Computer users.
3. Process level.

The management level is the highest level where the policy is defined in terms of its goals. In an enterprise, the management will, typically, view the policy as it affects the entire enterprise and define it in broad and general terms. At the computer user level, Abrams and Bailey suggest that the policy be abstracted based on the job functions performed by individual users. For example, certain users may require access to a system which allows them to update customer details, while other users may require access to payroll systems. Hence, the policy at this layer is abstracted so that it is defined in terms that are relevant to the specific user and the function they perform. The process layer forms the lowest level of this hierarchy, where the policy is abstracted in terms of how data is handled by individual systems, such as a customer management or payroll processing system. This would typically also include firewall and routing specifications detailing how data is filtered, routed and handled by the systems involved. The top-down approach suggested by Abrams and Bailey in [17] and Hinrichs in [16] has the advantage that policies are represented at a high-level using syntax and expressions which are close to natural language, such as written English. The motivation for this is to make the initial, high-level policy statements as clear and accessible to a human reader as possible. Such high-level abstractions also allow policy statements to be able to capture specific *nuances*, or intents of the administrator. A policy statement at a high-level, therefore, defines the policy in terms of its goal without being biased to its method of enforcement. This statement can then be interpreted by different applications and devices based on their own specific enforcement mechanisms. High-level policies, therefore, should only need to be defined once and can subsequently be translated to their specific enforcement points. Hence, a range of diverse policy enforcement mechanisms, including vendor-specific devices which use different syntaxes, may be used within a single domain as each device would derive its own rules from the same high-level policy. This mechanism ensures that although various policy enforcement mechanisms may express a specific policy differently, based on their own lower-level languages, they should be uniform in their policy interpretation and enforcement. A further benefit resulting from decoupling the mechanism for policy definition from the method for policy enforcement is that business rules, which change frequently and require continuous management, can be administered more efficiently. This is difficult to monitor in an environment where policy definition and policy enforcement are not decoupled. Fraser et al. outline a five-step iterative process in their Site Security Handbook [7] for policy creation and maintenance. One central component of their work is the definition of a security policy as a continuously evolving process. This implies that policies are audited and reviewed regularly to ensure that mechanisms for providing feedback and improvements are in place. The authors, therefore, see policy formalisation as an intrinsic aspect of the design of a system, an observation echoed by other researchers including security expert Bruce Schneier [8]. They also echo the observations of Cheswick et al. [9] that policy formalisation is an evolving process, requiring regular review, of which the actual implementation of security products is simply the consequence.

2.4 High-Level Policy Approaches

The development of effective methods for defining formal policies has been an active research area since the 1960s, which saw the development of strategies for securing mainframes and large-scale industrial computers. With increasing reliance on more complex systems, the area of policy-based management (PBM) [18] has seen significant interest not only from its traditional computer-security discipline, but also from the more diverse systems management community [19]. Damianou et al. identify a key driver for this in [20] as being the increased dependency of large-scale business processes on policy-based systems, where business goals, service-level agreements and trust relationships within or between enterprises, all have an impact on defining policies. However, as Susan Hinrichs identifies in [16], there is often a conceptual gap between high-level policy statements and their translation to respective lower-level enforcement configurations.

2.4.1 Role-Based Policy Definition

One area of exploration in addressing the gap between high-level policy statements and lower-level enforcement configurations is the development of policy specification languages based on formal logic. The attractiveness of logic-based languages is in their unambiguous mathematical formalism. This makes these languages amenable to modelling and analysis. Chen and Sandhu [21] describe the need for Role-Based Access Control (RBAC) constraints to be expressed in an unambiguous and precise manner and use mathematical logic notation. They describe RBAC Subjects, Privileges, Roles and Sessions in first order logic format. Hayton et al. [22] also use a logic-based language in the Role Definition Language (RDL). Their method uses roles as credentials and is based on the definite clause property of Horn clauses. In [23] Joshua Guttman proposes the Network Policy Tool (NPT). The NPT tool uses a high-level policy language, which is decoupled and independent of device configurations, to describe an enterprise's packet routing policy and defines where in an organisation's network specified packets can be routed. The tool describes the relationship between entities, such as network devices and users, using a description of the networks topology. These are specified using a Lisp-like policy specification language and modelled using a bipartite graph which represents network traffic routing devices, as well as the different areas of the network, using nodes. The undirected edges between the nodes represent the interfaces of the devices connected to the different areas. Each interface can have an associated *filtering posture* created, in both in-bound and out-bound directions. List 3 shows an example of NPT code specifying a corporate network. These postures are abstract representation of the bi-directional packet filtering that is enforced by routing and firewall devices. Although the NPT tool is useful as a reference mechanism for administrators creating lower-level specifications, it does not generate the device configurations itself.

Listing 3: Example of NPT-generated code specifying a corporate network [23].

```
(areas
;; name distinguished hosts
(external)
(periphery proxy-host)
(engineering eng-mail-server db-server)
(financial financial-mail-server)
(allied))
(connectivity
;; router name areas
(per/ext-router periphery external)
(per/eng/fin-router periphery engineering
financial)
(eng/allied-router engineering allied)
(allied/ext-router allied external))
(services
(telnet tcp 23)
(ftp tcp 21)
(ftp_data tcp 20) ...)
```

Guttman documents in later work [24] how aspects of the NPT tool can be improved upon. Specifically, he mentions how administrators find it difficult to use the abstract policy representations generated by NPT to model an existing network policy. He proposes the use of the Atomizer tool in [25] which allows administrators to generate NPT specifications directly from Cisco access control lists. One major advance of the Atomizer tool in achieving this is the use of Binary Decision Diagrams (BDD) [26]. Guttman uses BDDs to represent criteria which can then be used to classify network packets and, hence, be used to describe existing configuration files. BDDs are discussed in greater detail in Section 2.5.

2.4.2 Logic-Based Formal Policies

Another approach uses the Z-notation logic-based language which uses a combination of formal logic and set theory. Boswell [27] uses Z-notation in producing logic-based Mandatory Access Control (MAC) and Discretionary Access Control (DAC) formal security policy model for the NATO Air Command and Control System (ACCS). Hoagland et al. [28] take a different approach with the introduction of the Language for

Security Constraints on Objects (LaSCO), a formal policy language that expresses constraints based on graphs. The main advantage of this method is that it provides a visual representation of policies, which the author claims is easier to understand than complex formal logic statements. However, this approach is limited in that it is unable to handle obligation policies and often requires additional information, usually expressed as text in conjunction with graphical representation. While the graphical-textual hybrid representation makes LaSCO more expressive than the purely graphical version, it adds significant complexity. Listing 4 shows a sample of Boswell's Z-notation code which defines the basic Bell-LaPadula [29] security concept of *no read up*. This concept stipulates that no user is allowed to read from an object with a classification higher than the user's security clearance, which is a common requirement for classified information. Figure 2 illustrates a similar Bell-LaPadula concept in Hoagland et al.'s graphical representation.

Listing 4: Z-notation sample [27] defining Bell-LaPadula no read up concept.

```
SystemElements
∀ proc : ProcessId; obj : ObjectId; mode :
   AccessMode‖
(proc, obj, mode) ∈ cat ∧ mode ∈ ReadControlModes •
(clearanceOf proc) dominates (secLevelOf obj) ∨
(proc, obj, mode) ∈ activeTwoPersonRuleOps
```

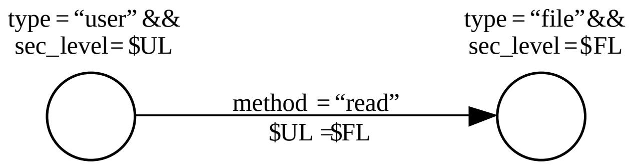


Figure 2: LaSCO policy graph [28] defining Bell-LaPadula no read up concept.

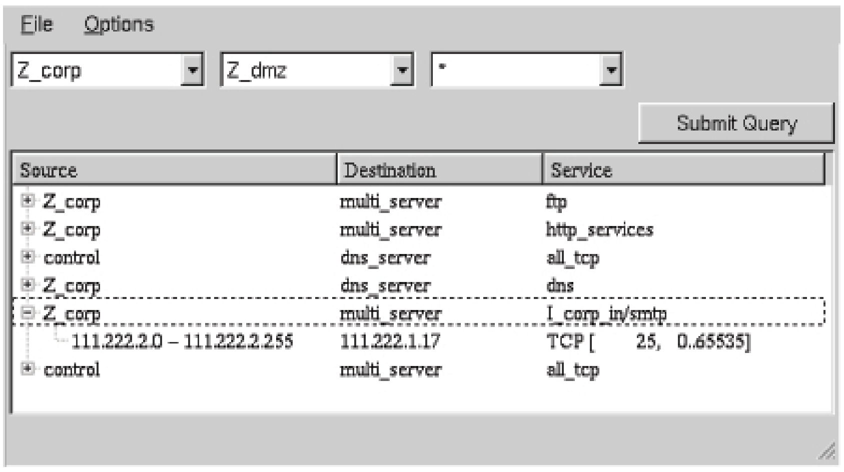
In [30], Bartal et al. propose another tool which takes a top-down approach called Firmato, a firewall management toolkit. Firmato uses a high-level policy language to create a vendor independent global policy, which can be compiled into individual vendor specific device configurations. The network topology is first defined in the high-level policy definition language, which is then translated into an entity-relationship model using a role-based model of the access policy and the relationship to the network topology. A core strength of the Firmato system, and an important improvement over Guttman's work in [24], is that it allows high-level policies to be defined independently of a network's topology. The advantage of this is that changes to the topology do not mean that the policy has to be reworked. Other motivations behind the system were to abstract the policy away from low-level languages, enabling vendor-independent management of firewall configurations, and to automatically generate configurations across multiple filtering devices from the high-level global policy. Firmato uses the Model Definition Language (MDL) as its high-level language. A sample of MDL, shown in Listing 5, defines network services and hosts.

Listing 5: A sample of Firmato Model Definition Language (MDL) [30].

```
<service-name> '='
<protocol-base>
```

```
'['<Dest-Port-No-Range> ','
<Src-Port-No-Range>']'
<host-name> '=' '[' <IP-Addr>
']' ':' <role-grp-name>
<host-grp-name> '=' '['
<IP-Range> ']' ':'
<role-grp-name>
```

Mayer, Wool, and Ziskind proposed FANG, firewall analysis engine [31], which improved on the earlier work by Bartal et al. and the Firmato system [30]. FANG is actually an analysis engine which runs on top of the same Firmato policy model. It can be used to test a policy before it is deployed and could also be use to audit a deployed policy. It has the functionality required to build the model from existing filtering configurations, and so it is classed as a bottom-up system. It can take Cisco router configuration files or Lucent firewall files as input to create the policy model and uses a separate parser module for each filtering device it supports, making the system scalable to multiple vendor platforms. The system works on multiple filtering devices, and so a global policy can be tested. A network topology has to be entered initially and this is done manually using the Model Development Language (MDL), as is the case with the Firmato system. The queries which can be performed on the FANG system are based around a triple of source host group (source network address range), a destination host group (destination network address range) and a network service. Queries can be created such as (*, web servers, http services) to find the answers to questions, such as 'which systems have access to the organisations web servers'. A GUI was created to perform queries, and drop down menus implement the query triples. An example, taken from Mayer et al. in [31], shows the result of a query asking 'which services can get from the internal network to the DMZ network' in Figure 3. Another strength of



Z_corp | Z_dmz | *

Source	Destination	Service
⊞ Z_corp	multi_server	ftp
⊞ Z_corp	multi_server	http_services
⊞ control	dns_server	all_tcp
⊞ Z_corp	dns_server	dns
⊟ Z_corp	multi_server	I_corp_in/smtp
111.222.2.0 – 111.222.2.255	111.222.1.17	TCP [25, 0..65535]
⊞ control	multi_server	all_tcp

Figure 3: FANG interface displaying results of a query [31].

the FANG system is that it can also be used to reverse engineer a model of a policy from firewall configurations. The administrator can then query the policy to become familiar with it before changes are made, or can check that the policy matches the overall security requirements. This auditing system has been improved on, by the creation of the Lumeta Firewall Analyser [32, 33]. This improved on the FANG system by automatically creating the queries needed to analyse the firewall policy model. Lumeta generates what the authors describe as the most interesting queries, and then displays the results of these queries. This attempts to highlight possible risks in the firewall policy and limits the need for user input to the system. The authors recognised the fact that one of the problems with their earlier system was that the administrator had to decide which queries to ask the system, and then create the queries manually. This is described as a significant usability issue with FANG. In the FANG system, the administrator has to enter the network topology description manually, using the Firmato MDL language. The requirement to use MDL, however, was identified as a considerable problem during beta testing and, in the new Lumeta system, the network topology can be created automatically using the network routing table. The graphical interface to FANG, shown in Figure 3, was also replaced as it was deemed difficult to use by testers. This has been replaced by a batch process which performs a comprehensive simulation of traffic through the firewall policy and reports on this. This is interesting as the administrator users found the original Firmato CLI interface easy to use, and yet it was replaced with a GUI. This shows the design was perhaps not tailored to the type of user correctly [34]. The output from the system is now a report in the form of web pages, with the ability to drill down into more detail if the user needs to. Using HTML to provide this type of flexible reporting is described as an ideal mechanism for security analysis tools. However, one significant problem with the FANG system is that it can only translate Lucent managed firewall, which does not have a large market share. The Lumeta system has added parser modules for CheckPoint firewall and Cisco access control list (ACL) configurations, so heterogeneous networks could be modelled, and therefore the product would be useful to a wider audience. The low-level configurations are abstracted to the Lucent managed firewall-based language used by Firmato and FANG, and the analysis query engine uses this as an input. The Lucent Managed Firewall language was used as it is contains high-level constructs and is easy to parse. The Lumeta architecture [33] is shown in Figure 4. A query-based system, similar

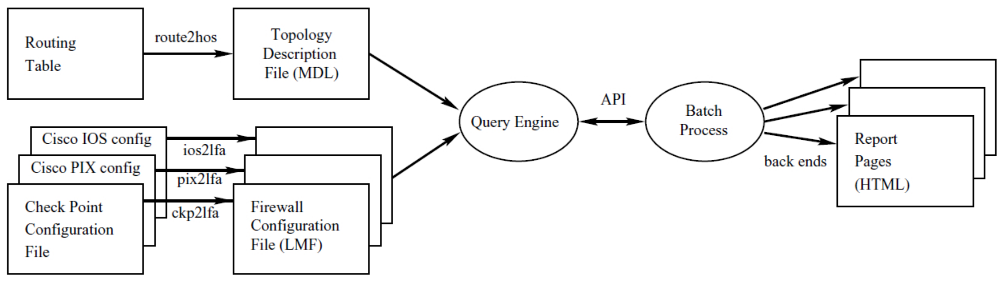
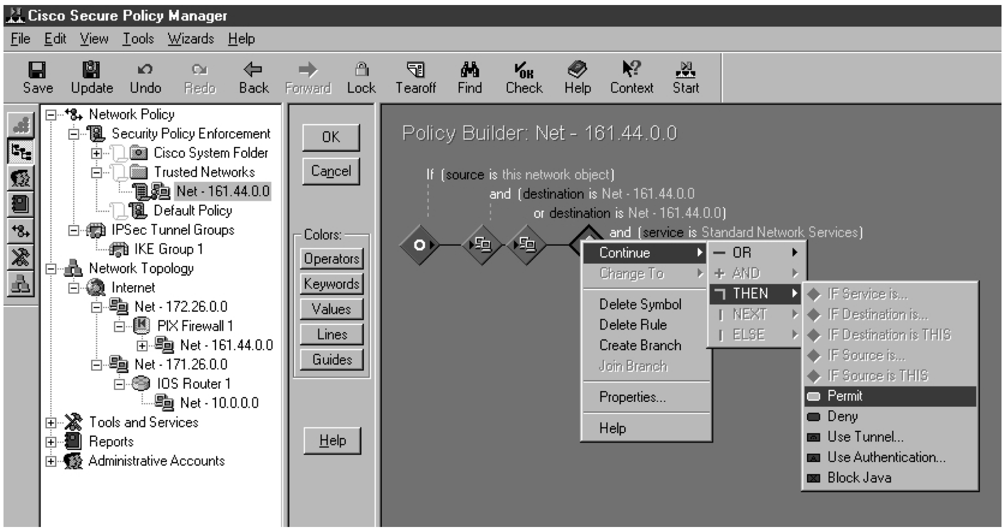


Figure 4: Lumeta architecture [33] illustrating data flow through the firewall analyser.

to the FANG system [31], was developed by Marmorstein and Kearns in [35] for the Linux iptables firewall. The ITVal system uses multi-way decision diagrams (MDDs) rather than binary decision diagrams (BDDs), but operates in much the same way. The authors' main motivation was to provide a simple query tool to aid in firewall configuration, so that an administrator could test a firewall configuration before it was deployed. The query language is designed to be simple and based on natural language. A single rule set can be read-in by the tool and an MDD model built. The queries are created in an English-based query language and return a simple textual answer, in a similar way as the FANG system does. However, a common problem with analysis systems using queries is that they have to be created by the administrator. The system administrator has to learn another query language, as well as knowing which queries to perform. The onus is on the administrator to work out what needs to be queried and when the query should be executed. Further, as a consequence of using MDDs, the number of possible decisions that can be made at any single node becomes variable. This is not a significant issue for the ITVal system, as it is designed for use in network environments where the rule syntax is strictly controlled and the number of bits required to represent a field does not change. Within the context of this paper, however, where information sharing policies can have multiple, user-defined fields which can, in turn, be of any length, using MDDs becomes unfeasible.

2.4.3 Graphical Approaches to Policy Formalisation

The Cisco Secure Policy Manager (CSPM) [36] is a policy management tool which incorporates aspects of the system defined by Hinrichs in [16]. The CSPM provides policy administrators with a graphical interface which uses a tree view to depict network topology information and configure policy. The topology tree represents the enforcement devices, and areas between devices, and contains a policy structure with the individual policies defined in an abstract policy language. These can then be applied to the enforcement devices in the topology tree to enforce policies between network zones. Figure 5, from [36], shows the interface to the CSPM tool. The Cisco CSPM



If (source is this network object)
and (destination is Net - 161.44.0.0
or destination is Net - 161.44.0.0)
and (service is Standard Network Services)

Figure 5: Cisco Secure Policy Manager (CSPM) interface [36].

tool generates configurations for routers and firewalls by first applying high-level policies to each specific device. This entails identification of rules which are applicable to an individual device and *pruning* rules which are not applicable. In this way, devices only receive rules which are relevant to them. This is a significant enhancement on previous systems which either apply all rules to all devices, or perform very minimal pruning. Secondly, the set of rules identified as being applicable to a specific device are evaluated against the physical resources available to that device. This evaluation would take into account factors such as processing power and memory available to the device, and whether these are adequate for the device to enforce the rules applied. Another factor which is part of this evaluation process is rule set anomaly analysis. This includes checking for conflicts and anomalies in the rule set. For example, if two rules specify the same source address, destination address and service, but different actions, the CSPM too would generate a notification to the administrator. The administrator can then decide to take corrective action, such as removing one of the two conflicting rules from the rule set. Once the administrator has taken the corrective action needed, the CSPM tool creates an intermediate, abstract, rule set for each device and stores it in a database. The database is used to store semantics of the device configurations to be created, but leaves the actual generation of specific configurations to an agent which reads the generic filter rules, and creates low-level rules in the device's native configuration language. This process allows different devices to be used while only requiring the final stage to be repeated to create configurations for the new devices. Although this is a Cisco-specific closed system, it could, in theory, be used to create other low-level vendor device configurations. Uribe and Cheung [37, 38] propose a similar mechanism to that proposed by Guttman [23], which generates low-level configurations from high-level policies. Their approach extends Guttman's approach of modelling and configuring firewalls to include intrusion detection systems as well. Enck et al. propose a configuration management system called PRESTO [39] which makes use of templates to create a hybrid scripting language, rather than introducing an entirely new language for high-level policy syntax definition. The templates are then used to generate low-level device configurations at run-time. Damianou et al. propose the multi-purpose policy specification language, Ponder [18]. Ponder is designed to meet multiple needs including providing a means of specifying security policies that map onto various access control implementation mechanisms, including firewalls, operating systems, databases and Java. Ponder has the capability to specify the entire high-level security policy, including access control policies, user authorisation policies, and traffic filtering policies. However, Ponder is more complex than other approaches mentioned and requires extensive training and familiarity to construct effective filtering policies. Another high-level policy modelling language, FLIP, is proposed by Zhang et al. in [40]. FLIP also allows high-level policies to be compiled into low-level device configurations. The scope of the FLIP system is global and can manage firewalls across an entire network. FLIP generates conflict free rules automatically, by performing conflict analysis as it generates the low-level device configurations. This improves on most of the systems described, which would need to be analysed separately for rule conflicts. Although this allows configurations to be generated more automatically, the integration of multiple functionalities together, as is proposed in FLIP, implies a reduction in its flexibility [34]. Listing 6 shows an example of a high-level policy which allows the use of '*Yahoo!*' instant messaging and Windows remote desktop service (tcp, port 3389) from the 140.192.* network, while blocking an on-line game on port 3724 and web proxy cache squid, which can be exploited by Trojans, on port 3128. In the example shown, the administrator has chosen to block all traffic on ports between 3100 and 3800.

Listing 6: An example of a high-level policy definition in FLIP [40].

```
service yahoo_msg = tcp.[port=5050],
torrent = tcp.[port >= 6881, port <= 6999];
policy_group student_policy
{
```

```
incoming:
yahoo_msg { deny any }
torrent { deny any }
outgoing:
http { allow any }
}
policy_group dom_std_policy extend student_policy
{
incoming:
yahoo_msg { allow any }
tcp.[port>=3100, port < 3800] { deny any }
tcp.[port=3389] { allow 140.192.* }
}
```

Pozo et al. propose another high-level policy language, AFPL in [41] and AFPL2 in [42], resulting from work at the University of Seville. This, again, intends to fill the gap between high-level network security policies and low-level firewall languages. It has been designed to be simpler than some of the preceding high-level languages while still retaining the functionality needed to describe filtering policies, and can also be automatically compiled into leading vendor firewall filtering languages. A similar proposal by Hinrichs et al. in [43] is the Flow-based Management Language (FML). FML is a formal, high-level network policy specification language designed to replace the low-level policy rules by controlling the *flows* of data within the network, regardless of the physical devices that they flow through. After specifying a policy, FML can be translated to low-level network hardware configuration rule sets automatically, using tools provided by the authors. This work contributes an extensible, adaptable and efficient mechanism for controlling network access. A number of policy verification initiatives are based on the Extensible Markup Language (XML). The Netconf protocol proposed by Bierman et al. in [44] arose out of recommendations made at a workshop of the Internet Engineering Task Force (IETF) in 2002. It was designed to be a simple mechanism through which device configurations could be managed using an XML-based system [45]. The entire, or partial, XML encoded configurations, of a Netconf enabled device can be retrieved, updated, and deployed back to the device by remote management applications. The protocols control messages are encoded in XML, as well as the data being sent. Major network device vendors, such as Cisco and Juniper Networks, now have XML-based agents in their latest products and are participating in Netconf standardisation [46]. Cisco Netconf configuration is detailed in [47] and Juniper in [48]. Although XML-based systems have many advantages over more traditional low-level languages, as highlighted by Munz et al. in [49], complex policies framed in XML are still not easily read and understood by non-expert human end users. From the perspective of information sharing between community partner organisations, this presents a significant hurdle to effective collaboration.

2.5 Policy Definition and Binary Decision Diagrams (BDDs)

A number of policy definition approaches are inspired by packet filtering techniques. Testing of packet filtering rule sets was explored by Hazelhurst et al. in the late 1990s [50]. The motivations for their work included the analysis of low-level rule sets to identify and understand their dependencies on the high-level policies that they implement. A further area of investigation, as explored by Chomsiri et al. in [51], centred on the performance advantages of using Binary Decision Diagrams (BDDs) instead of the traditional methods of processing rules sequentially. Their work developed a query-based system to analyse rule sets. BDDs were used to represent firewall and router access policies. Each rule in the rule set can be converted into a Boolean expression which is then combined in a BDD. Queries are used to pose questions about the rules, which can then be answered using the BDDs. An example of these 'what if?' queries might be 'which destination addresses can packets reach from a source address and a certain port?'. The user can analyse, and so test, a policy by querying the rule set in various ways. This can be used to validate and explore the policy before deploying the rule set onto filtering devices. The language used to specify the queries is a functional language (FL), and the output is a textual representation of the query answer. The FL query language is low-level and, often, difficult to use for the system administrator when creating queries. This was recognised, and the system was improved to include a graphical interface for easier querying of the policy in [52]. The authors also recognised that the best interface was one for visualising important information about the rule set, and for basic querying, but a textual interface was better suited for an advanced user to develop more powerful queries. The scope of the system described by Hazelhurst et al. [50] only extends to a single rule set, but later research expanded query-based systems to cover entire networks. The primary analysis mechanism is the manual query and answer system, but a basic rule set conflict analysis process was also developed. This automated the task of detecting redundant rules in the rules set prior to deployment and seems to be one of the first systems to perform conflict analysis within rule sets and is often cited by other research in the area. A further key finding of the work carried out by Hazelhurst et al. [50] and a key motivation for this paper, centres on the performance advantages offered by the use of binary decision diagrams (BDDs) over that of sequential processing of access rules, as used by most firewall and router packet filtering devices. The following sections provide an overview of packet filtering, as utilised by most firewall and routing systems, and identify areas where BDDs can be used to improve packet filtering system performance. Packet filtering devices, including routers and firewalls, operate at Layer 3, the network layer, of the Open Systems Interconnection (OSI) model, as illustrated in Figure 6. Devices operating at this level have no way of interpreting high-level security policies and usually only inspect the IP header of a packet to make filtering decisions. Security policies designed to be enforced by devices operating at this layer, therefore, need to be expressed in terms of the packet's protocol and source and destination IP addresses. Devices inspect packets passing through them and will attempt to match the relevant fields in a packet's header to specified rules. If a rule match is found, the devices undertake the action, either permit or deny, specified by the rule. If no rule match is found, the device denies the packet from passing through. Packet filtering devices are relatively inexpensive to implement [53] and have found widespread use in firewall and routing implementations. Devices using packet filtering have also seen better performance results [54] than application-level [55] or transport-level firewalls. The security rules which govern the behaviour of packet filtering devices are typically structured in the form of an access-control list, and are usually evaluated sequentially in a top-down fashion. Thus, whether the packet is permitted to pass through or denied depends on the action specified by first rule which matches the packet's relevant header fields. Although this structure allows for the uncomplicated representation of rules, it relies heavily on their correct ordering. Further, administrators need to be aware of the potential interaction between rules in

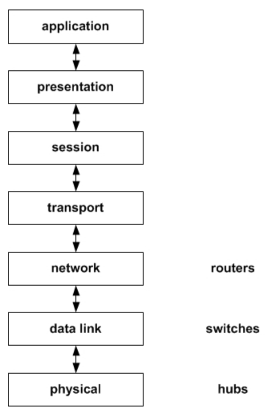


Figure 6: Open Systems Interconnection (OSI) reference model.

an access-control list and possible anomalies which can result from mis-configuration and can potentially cause devices to function incorrectly. As Adi Attar identifies in [56], devices represent access-control lists internally as a linear set of rules. The device's lookup process, which is the process used to determine whether or not a packet matches a rule, entails a sequential application of each individual rule specified in the access-control list to the packet. This process continues until a matching rule is found or until all rules in the list have been checked. If a matching rule is found, the action defined by that rule is taken. However, if no matching rule is found, the default rule applied is usually a deny passage rule. As Cheswick et al. identify in [9], the look-up process must be repeated sequentially for each packet, as no contextual information is retained by the inspecting device. Cheswick et al. in [9] and Attar in [56], both, highlight that the performance latency resulting from sequential lookup process is proportional to the size of the access-control list. A list with a large number of rules would, therefore, affect the performance of a system much more significantly than a list with a smaller number of rules. A method suggested by Oppliger in [53] and Cheswick et al. in [9] to improve overall performance and mitigate against latency effects is to re-order access-control list rules, such that the most often matched rules are located near the top of the list and the least matched rules located lower in the list. However, as Chapman notes in [57], re-ordering of rules in an access-control list is a complex and error-prone process since the order of rules, and potential conflict between rules, has a significant influence on the behaviour of the system. Other research has also been directed at development of methods that seek to address the performance issues resulting from sequential rule processing. In [56], Attar provides an overview of some of these proposed methods, broadly categorised as using either table-driven approaches or using specialised data structures. Table-driven methods include recursive flow classification (RFC) [58], cross producting [59] and range matching [60], while proposed specialised data structures approaches include the grid of tries [61], expression trees [62] and decision graphs [63].

2.5.1 Table-Driven Packet Filtering Approaches

Table-driven approaches to improve packet filtering classification mainly use algorithmic techniques which can represent access-control lists in tabular form. To achieve this, the algorithms rely on identification of specific patterns in the access list rules. For this reason, table-driven approaches tend to yield better performance results when applied to access-control lists which mainly contain similar rules but not to lists in which rules are dissimilar. Gupta and McKeown describe a table-driven approach called recursive flow classification (RFC) in [58]. The RFC approach essentially uses a multi-stage classification of rules that is able to filter one million packets per second in software and up to 30 million packets per second, in optimised hardware. Their proposed algorithm involves classification of rules which contain similar fields into classes and associating each class of rules with an identifier, *classID*. The algorithm then maps *S*-bits of the packet header to *T*-bits of the *classID*, where $T = logN$, $T << S$ and *N* is the number of rules. The authors suggest that a quick way of achieving this mapping would be to pre-emptively compute the value of *classID* for each of the 2^S different packet headers. Although this would achieve a mapping in one single memory access step, it would require very large amounts of memory. Hence, the authors suggest recursively performing the same mapping over several stages where each successive stage reduces the target rule set size to achieve a match. Due to the classification algorithm, as illustrated in Figure 7, relying on identifying similarities between fields, RFC tends to yield better performance results on access-control lists with a large number of similar rules which can be grouped into classes. However, where the target access lists contains dissimilar rules, an increase in the number of rules in the access list results in an exponential increase in the memory required to process it which, in turn, causes significant deterioration in performance. Srinivasan et al. also propose a similar approach in [59] where the rule's syntactical structure is used to mitigate against exponential memory requirements. Their method, similar to that proposed by Gupta and McKeown in [58], utilises a pre-emptive conversion of rules in an access-control list to generate cross-product table. Their method is based on a decomposition-based algorithm which utilises successive prefix matching of rules over a number of steps. This method relies on at least two steps where, initially, the longest prefix in the packet header field is matched and, secondly, using the cross-product of the results of the prefix match to determine the target rule. However, due to the multiplicative nature of the cross-product calculations, the table resulting from the algorithm is often extremely large and requires, either, very large memory space or long lookup times. In [60], Lakshman and Stiliadis attempt to address performance issues resulting from the sequential processing of rules by using bit-parallelism. Their algorithm defines a set of rules of size *N* in *K*-dimensional filter space. This process relies on a two-stage process where, initially, bit vectors of all rules for each field are calculated and, secondly, the rules most likely to complete the header are identified. The second stage is completed by calculating the bit vectors corresponding to each header field and then computing their common intersection. The first set bit in the resulting bit vector is used to locate the position of the applicable rule. Although use of the bit-parallelism technique yields better performance over sequential rule processing, it generally produces most significant improvements when used in conjunction with specialised parallel-processing hardware implementations. However, as with all approaches which make use of pre-processing, using bit-parallelism still suffers from poor update times.

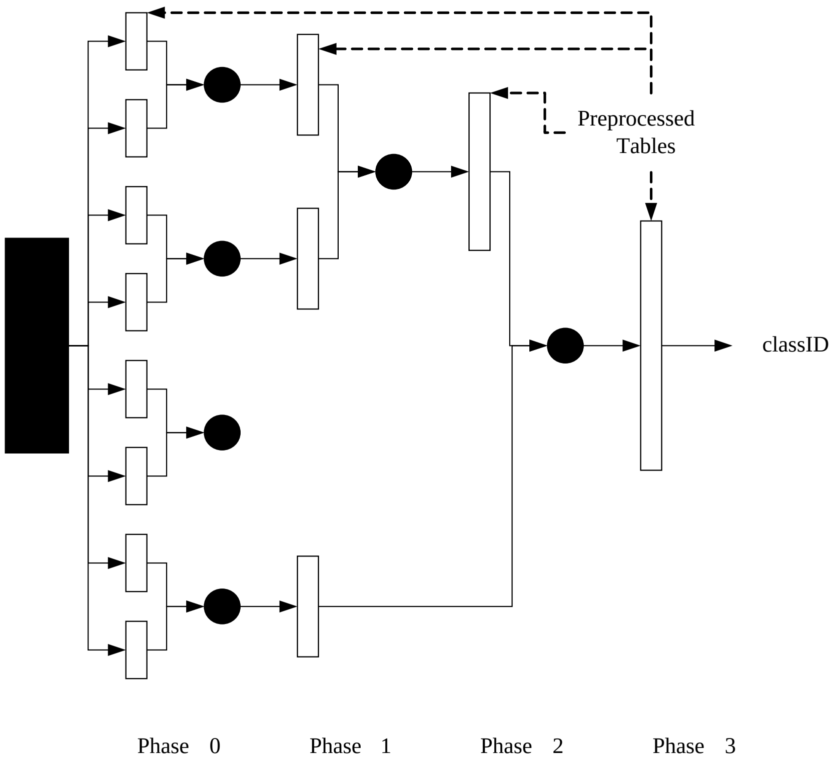


Figure 7: Illustration of packet flow in the RFC algorithm [58].

2.5.2 Packet Filtering Approaches Using Specialised Data Structures

Packet filtering approaches which seek to address sequential rule processing performance issues tend to use graph-based techniques to represent access-control lists. Hence, they are less sensitive than table-driven methods to a device's internal representation of the access list. In [61], for example, Cheung and McCanne, propose the *Grid of Tries* solution. The define a *trie* as a tree-based data structure, based on Edward Fredkin's work in [64]. Each node in a *trie* has one or more children and edges to each one correspond to different bit sequences. Tracing a path from the root of the *trie* to any specific node would, therefore, yield a distinctive sequence of bits for that node. In their implementation, Cheung and McCanne extend the *trie* structure to two fields with positive results. However, their solution does is not easily extensible to multi-dimensional filters with greater number of fields. The work of Mogul et al. in [62] forms seminal contribution to packet filtering using the technique of representing access-control list rules as expression trees. The authors describe the technique as kernel resident, protocol-independent packet demultiplexing which functions as a programmable abstraction of a Boolean predicate function. This process can then be applied to a packet stream in order to filter a defined subset of the stream. Their original packet filter was known as known as the CMU/Stanford packet filter or CSPF and was based on a stack-oriented virtual machine, where selected packet contents could be pushed on a stack and Boolean and arithmetic operations could be performed over these stack operands. A drawback of this method is that repeated computations may be needed to compute the Boolean expression used to represent the packet header. This expression must then be matched to the expression tree representing the list of rules in order to locate the desired target rule, if one exists. However, this process causes a linear growth in lookup times as the number of rules in the access-control list grows. In [63], Decasper et al. seek to improve rule lookup performance times through the use of directed acyclic graphs (DAGs). Their approach uses tables of filters, where each filter represents a field in the packet header. A key advantage of this approach is that the size of the filter table is equivalent to the number of fields being checked. Hence, lookup times should be dependent on the syntax of rules but, theoretically, independent of the number of rules in an access-control list. Although the authors list fast packet classification times achieved through that this approach, they do not mention the memory required for this process. Due to the lack of comparative testing and the scarcity of analytical results of approaches to improving on the performance issues inherent in sequential rule processing, it is difficult to make definitive comparisons. However, performance results indicate that those approaches that make use of specialised data structures tend to yield better performance results than those approaches that are table-driven. For this reason, this research uses binary decision diagrams (BDDs) which is also a specialised data structure approach and is closely related to the *Grid of Tries* method proposed in [61]. However, as noted by Attar in [56], the BDD approach has the advantage that it provides more control over the order in which bits in the packet are inspected which could be defined with a specific field order in mind and, hence, provide faster lookup times while avoiding redundant checks. The BDD approach also has the advantage over the *Grid of Tries* method in that it is not restricted in the number of fields that it can filter.

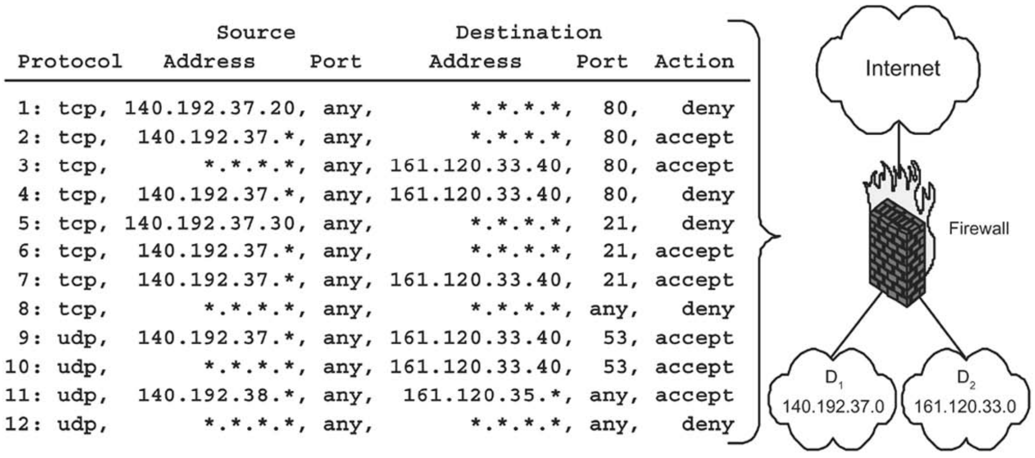
2.6 Firewall Rule Set Management Tools

The work by Ehab Al-Shaer and Hazem Hamed at DePaul University in Chicago has contributed greatly to research in the areas of policy modelling and analysis. Al-Shaer and Hamed have been the main contributors with over a dozen publications between them. Their first research into policy analysis focused on auditing legacy firewall policies to automatically discover conflicts and anomalies in firewall policies. This anomaly checking can also assist administrators when editing the deployed policies [65, 66]. The rule set conflict detection aims to highlight possible problems in a rule set, based on the order of the rules and the dependencies between rules resulting from rule ordering. For example, a more general rule before a more specific rule in a rule set would mean the more specific rule would never be reached. The more specific rule is classified as *shadowed* by the first rule if the filtering actions taken are different (pass and drop), or *redundant* if the actions are the same (for example, both rules pass the packet). These are classed, by the authors, as rule set *anomalies* [66]. The firewall policy advisor (FPA) tool was created based around a formal model of the firewall rules and the relationships between them. Modelling of the filtering policies is done using binary decision diagrams (BDDs) and algorithms have been created to detect anomalies in the rule set model. To prove the concept, the authors demonstrate a five tuple filtering syntax, using *protocol type*, *source IP address*, *source port*, *destination IP address* and *destination port*, to describe the filtering rules used as input to the system. These five tuples, along with the additional *rule order* and *action* fields, map directly to current low-level filtering languages such as Cisco access control lists (ACLs). The format of the five tuple filtering rule is shown in Listing 7. The literature only shows examples of these commonly used filtering fields, but the authors state this could easily be extended to include any other filtering fields from low-level languages [65]. This could be extended to include the filtering options available in modern low-level filtering languages, such as Cisco ACLs or Linux IP Tables. Note that the filters used in the examples only use classful ranges of IP addresses, and classless ranges would need a more sophisticated wildcard specification. Al-Shaer and Hamed define all the possible relationships between rules, which are then proved mathematically to be the union of all possible relations. Details of how anomalies within rule sets are defined are covered in Section 2.6.1.

Listing 7: Example of a high-level policy.

```
<order> <protocol> <src_ip> <src_port> <dst_ip> <
    dst_port>
<action>
```

In [1] Al-Shaer and Hamed describe an example policy rule set, as shown in Figure 8. This rule set and the relations within it are then modelled as a BDD. This is then represented as a policy tree, with nodes on the tree representing filtering fields, and branches being the values. Each path through the tree represents a rule in the input rule set. This model was chosen, as the rule set and the anomalies can be visualised by the users. An example of this type of tree, taken from [1], is shown in Figure 9, which illustrates the four anomaly types of redundancy, shadowing, generalisation and correlation. Section 2.6.1 offers a further detailed discussion on these anomaly types. Algorithms used to detect any anomalies within the rule set are then run, and can be displayed to the user in the FPA tool interface. The system provides a graphical interface which can show the policy tree and any rule set anomalies. The interface is shown in Figure 10, which



	Source		Destination		
Protocol	Address	Port	Address	Port	Action
1: tcp,	140.192.37.20,	any,	*.*.*.*,	80,	deny
2: tcp,	140.192.37.*,	any,	*.*.*.*,	80,	accept
3: tcp,	*.*.*.*,	any,	161.120.33.40,	80,	accept
4: tcp,	140.192.37.*,	any,	161.120.33.40,	80,	deny
5: tcp,	140.192.37.30,	any,	*.*.*.*,	21,	deny
6: tcp,	140.192.37.*,	any,	*.*.*.*,	21,	accept
7: tcp,	140.192.37.*,	any,	161.120.33.40,	21,	accept
8: tcp,	*.*.*.*,	any,	*.*.*.*,	any,	deny
9: udp,	140.192.37.*,	any,	161.120.33.40,	53,	accept
10: udp,	*.*.*.*,	any,	161.120.33.40,	53,	accept
11: udp,	140.192.38.*,	any,	161.120.35.*,	any,	accept
12: udp,	*.*.*.*,	any,	*.*.*.*,	any,	deny

Figure 8: Example of policy rule set [1].

is taken from [65]. The policy tree is shown in the left hand pane, and the discovered anomalies in the right hand pane. The rule set is shown in the bottom pane with the conflicts highlighted. The FPA tool also allows the user to maintain the rule set, by inserting, modifying and deleting rules. As the user edits the rules, the tool provides feedback on any conflicts that may be introduced by these actions. Although Al-Shaer and Hamed describe the Firewall Policy Advisor system in [66] as able to analyse legacy firewall policies, no automatic importing of these policies is described. The system seems to need an administrator to manually translate the low-level rule set into the five tuple syntax. This would classify the systems top-down, however, the translation of deployed policies is not difficult as it is a direct mapping from most firewall configuration languages. Fixed format configurations from devices can be easily parsed using a scripting language, such as Perl, as used by Avishai Wool in [32]. The FPA system does, however, provide advancement over the query-based analysis systems such as FANG [31] and Lumeta [32], in that it does not require as much effort to redefine deployed rules sets into the high-level policy specification languages used by these systems, as the FPA system can analyse the rule sets directly. Although the initial scope of the FPA system was as a single device and rule set, this was extended to anomaly detection across multiple firewalls in further work by Al-Shaer and Hamed in [67] and [2]. Application of the FPA to include other policy enforcement devices by Al-Shaer et al. in [68], such as intrusion-detection systems (IDS) and gateways, extended the system even further and allowed inter-device anomaly analysis across the global policy. The FIREMAN Firewall Analysis system, proposed by Yuan et al. in [69], can perform analysis of firewall filtering including detecting anomalies resulting from redundant and conflicting rules, which may cause errors in security configurations. Again, binary decision diagrams (BDDs) are used to model one or more interconnected firewalls. The analysis can discover policy anomalies, similar to the anomalies defined in the Firewall Policy Advisor system in [66], as well as detecting certain violations in the policy. The authors define policy violations as based on a blacklist or whitelist, which is defined by the administrator, as well as a general policy for all rule sets based on the twelve common firewall configuration errors identified by Avishai Wool in [70]. The model

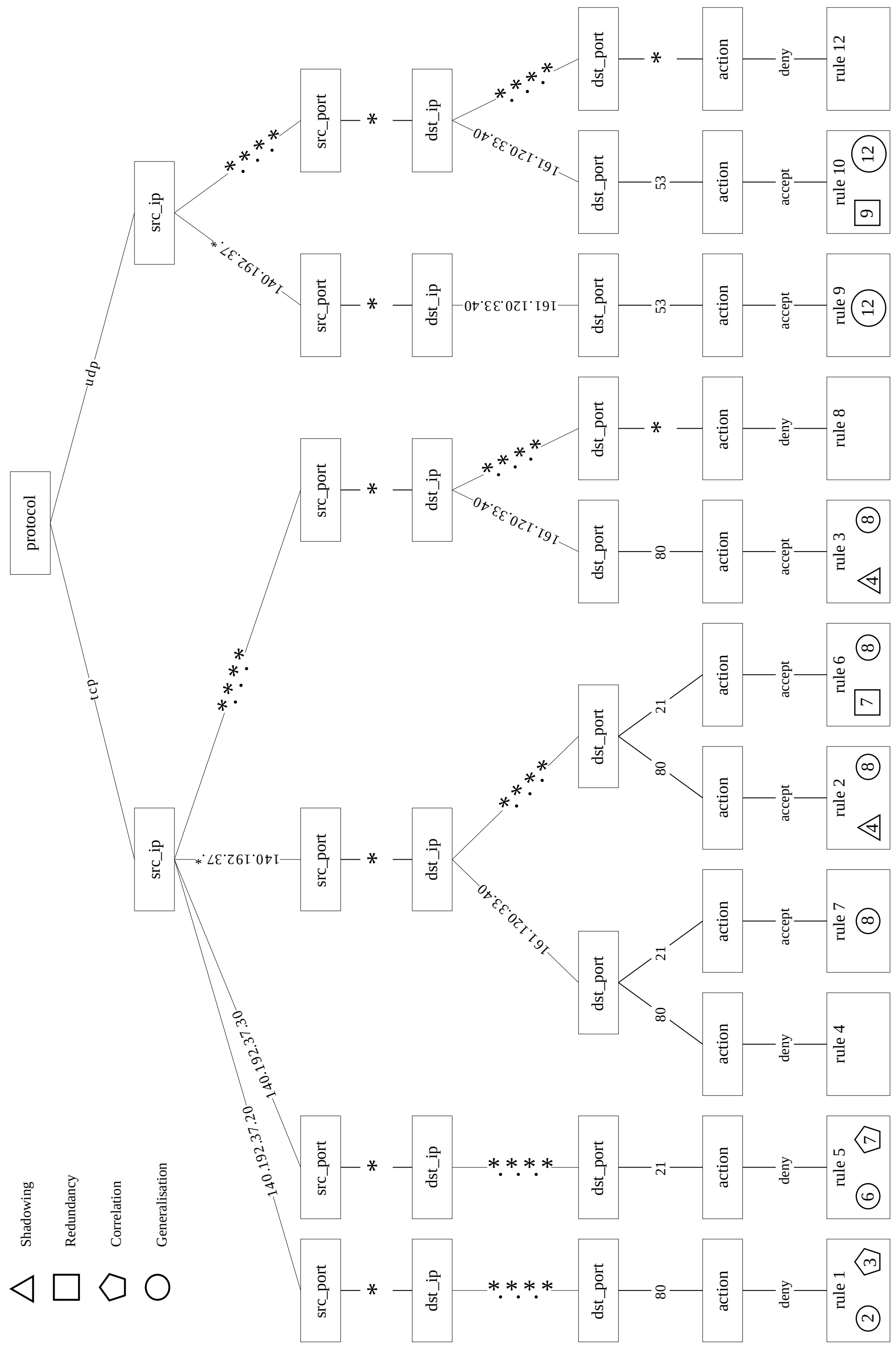
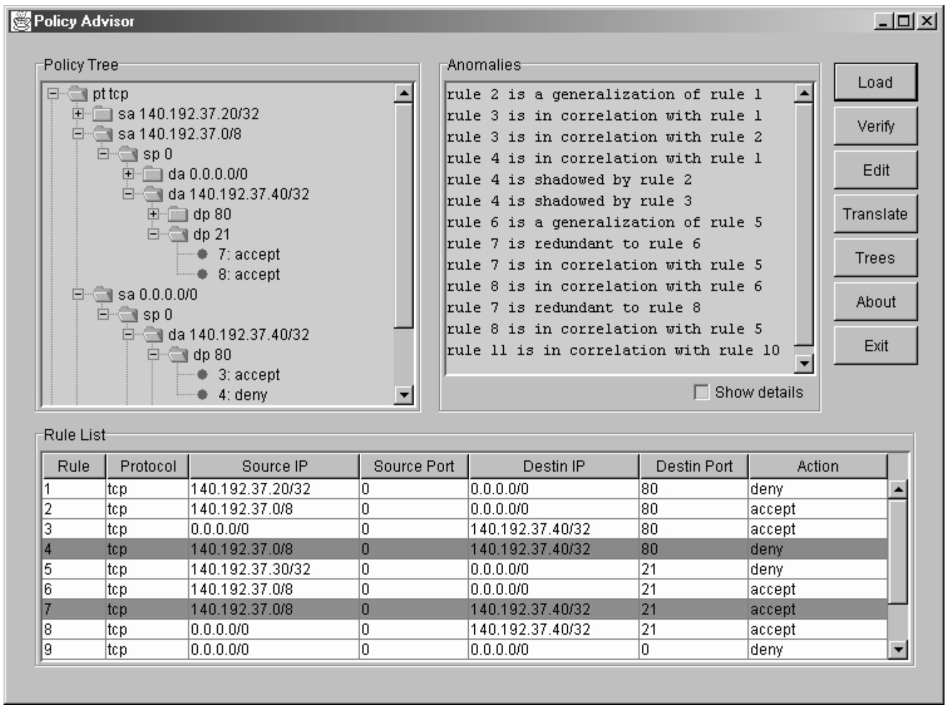


Figure 9: Policy tree derived from policy rule set in Figure 8.



```
pt tcp
  sa 140.192.37.20/32
  sa 140.192.37.0/8
    sp 0
      da 0.0.0.0/0
      da 140.192.37.40/32
        dp 80
        dp 21
          7: accept
          8: accept
  sa 0.0.0.0/0
    sp 0
      da 140.192.37.40/32
        dp 80
          3: accept
          4: deny
```

```
rule 2 is a generalization of rule 1
rule 3 is in correlation with rule 1
rule 3 is in correlation with rule 2
rule 4 is in correlation with rule 1
rule 4 is shadowed by rule 2
rule 4 is shadowed by rule 3
rule 6 is a generalization of rule 5
rule 7 is redundant to rule 6
rule 7 is in correlation with rule 5
rule 8 is in correlation with rule 6
rule 7 is redundant to rule 8
rule 8 is in correlation with rule 5
rule 11 is in correlation with rule 10
```

Rule	Protocol	Source IP	Source Port	Destin IP	Destin Port	Action
1	tcp	140.192.37.20/32	0	0.0.0.0/0	80	deny
2	tcp	140.192.37.0/8	0	0.0.0.0/0	80	accept
3	tcp	0.0.0.0/0	0	140.192.37.40/32	80	accept
4	tcp	140.192.37.0/8	0	140.192.37.40/32	80	deny
5	tcp	140.192.37.30/32	0	0.0.0.0/0	21	deny
6	tcp	140.192.37.0/8	0	0.0.0.0/0	21	accept
7	tcp	140.192.37.0/8	0	140.192.37.40/32	21	accept
8	tcp	0.0.0.0/0	0	140.192.37.40/32	21	accept
9	tcp	0.0.0.0/0	0	0.0.0.0/0	0	deny

Figure 10: Policy Advisor anomaly detection user interface [65].

of FIREMAN, however, is created with a bottom-up approach by parsing device configuration files. An evaluation of the system was carried out by creating artificial rule sets of up to 800 rules and performance evaluation was carried out based on timing metrics. The experimental conditions used were rule sets of incrementing size. The results indicated that the FIREMAN system can analyse up to 800 rules in less than three seconds. Research at AT&T Labs by Caldwell et al. in 2003, produced a configuration auditing and management system called EDGE [71]. This system creates an abstract model of the networks routing information by reading in and processing entire device configuration files. The network is modelled using an entity-relationship model stored in a database of network services and described in the configuration files. EDGE can provide off-line analysis of the network configuration and generate reports to identify inconsistent routing configurations on devices. The Administrator can then decide to amend the device configurations as required. The motivation for this system is similar to many of the other policy management systems discussed earlier. However, the system proposed by Caldwell et al. is unique in that it emphasises a bottom-up approach to identifying high-level policies from lower-level network configurations. A significant limitation with the EDGE system, which also affects the other approaches discussed earlier, is that administrators seeking to develop proficiency in the languages used to configure these systems face a considerable challenge due to their inherent complexity. Caldwell et al. propose automatic provisioning of configurations as a possible solution to this problem. In the case of the EDGE system, this means working with a model of the current network and using various visualisation techniques to help the administrator to understand the policies being implemented. This system has been successfully commercialised, and was effective in the management of complex, enterprise networks.

However, as identified by Zhang et al. [72], with increasing provisioning of enterprise-class cloud computing solutions, systems such as EDGE, which were not designed with cloud services in mind, are finding this limitation progressively more restrictive. Another limitation with systems which convert to, and from, vendor-specific configurations is that the configuration languages tend to change rapidly. For example, Cisco's access control list (ACL) filtering language has had many additional features added to it since the first version in the mid 1990s. This means that the parser modules have to be continually reviewed to keep them up-to-date with the latest syntax changes. As part of another configuration management system developed by Caldwell et al. in [73], some interesting research into Cisco router parsing and modelling was carried out at AT&T Labs. In [73], Caldwell et al. describe a learning system and adaptive parser. The overall system will be able to process and extract information from existing network configurations, much as described in their previous work with the EDGE system in [71]. However, the advance on the EDGE system is that the parser can adapt to changes in the configuration language being parsed. The parser is automatically generated from valid configurations, which are fed into the learning system. Hence, the system would not need manual changes to be made whenever the configuration language version was modified. This would overcome one of the major problems with the bottom-up approach to any type of policy modelling, especially when dealing with rapidly changing configuration languages such as Cisco device operating systems.

2.6.1 Anomalies in Firewall Rule Sets

In their cornerstone work of [1], Al-Shaer et al. define the fundamental principles of conflict-detection within firewall rule sets. This work, which also extends to a distributed scenario in [74] and led to formal anomaly definitions in [2], defines access-control list rules as having the structure as shown in Listing 8 and consisting of three components: condition, permission and order.

- Condition: The condition component of a rule is made up of a number of sub-fields. Although, these may vary depending on the proprietary syntax used by the device manufacturer, they usually contain protocol, source and destination IP address and source and destination port numbers. The first rule in an access-control list which matches the condition is applied to the packet.
- Permission: The permission component of a rule can either be permit or deny and defines whether a packet which matches the condition is permitted or denied passage through the device.
- Order: The order component of a rule identifies where, from top to bottom, the rule occurs in the access-control list.

Listing 8: Structure of access-control list rules as defined in [74]

```
Order ; Permission ; Condition
```

Al-Shaer et al. classify possible anomalies in access-control lists as redundancy, shadowing, generalisation and correlation anomalies, as shown in Figure 11. Following is a detailed description of these anomaly types. The description is based on two example rules, shown in Listing 9, which are derived from the structure shown in Listing 8.

Listing 9: Example of rules based on the rule structure shown in Listing 8

```
Rule X ($R_x$) has the structure
$R_x$ = Order ($O_x$) ; Permission ($P_x$) ; Condition ($C_x$)
Rule Y ($R_y$) has the structure
$R_y$ = Order ($O_y$) ; Permission ($P_y$) ; Condition ($C_y$)
```

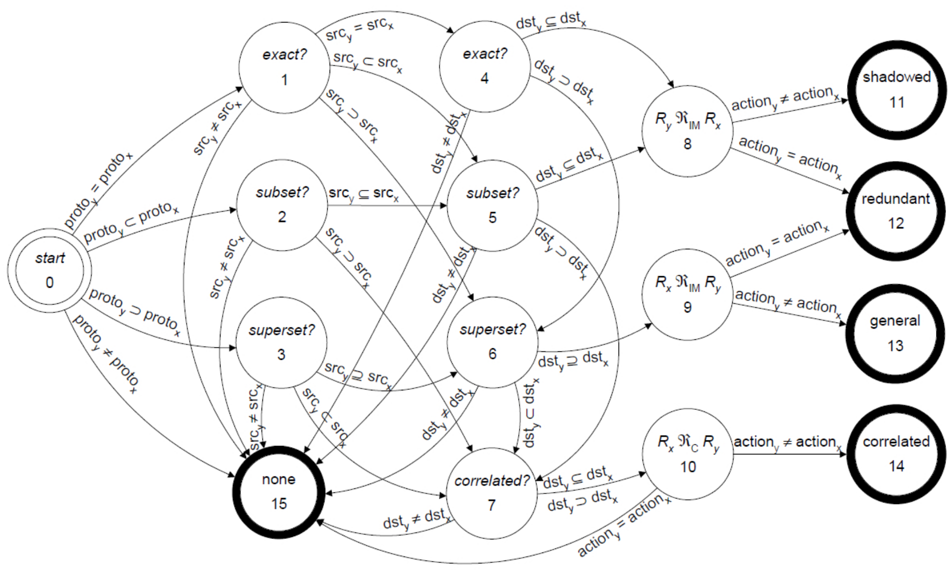


Figure 11: Anomaly state diagram from [66] for rules R_x and R_y, where R_x precedes R_y in the rules list.

Redundancy Anomaly A redundant rule has the same permission and the same conditions as another rule, such that if the redundant rule is removed, the behaviour of the rules list will not be affected. Redundancy can be determined by the logical criteria shown in Listing 10.

Listing 10: Logical criteria for the redundancy anomaly type

```
if the permissions of $R_x$ and $R_y$ are the same,
    ($P_x = P_y$),
then $R_y$ is redundant to $R_x$ if every field of $R_y$ is
    a subset or equal to the corresponding field in
    $R_x$, ($\forall i : R_y[i] \subseteq R_x[i]$)
or $R_x$ is redundant to $R_y$ if every field of $R_x$ is a
    subset or equal to the corresponding field in $R_y$
    ,
($\forall i : R_x[i] \subseteq R_y[i]$), where $i \in$ {protocol, s_ip, s_port,
    d_ip, d_port}
```

Shadowing Anomaly A rule is shadowed when a previous rule, higher up in the set of rules, matches all the conditions that match this rule, such that the shadowed rule will never be activated. Listing 11 illustrates the case where Rule Y (R_y) is shadowed by Rule X (R_x).

Listing 11: Logical criteria for the shadowing anomaly type

```
Rule Y $(R_y)$ is shadowed by Rule X $(R_x)$
if $(R_x)$ precedes $(R_y)$ in the rule set order, $(O_x < O_y)$
    ,
and the permissions of $R_x$ and $R_y$ are different,
    $(P_x \neq P_y)$,
and every field in the condition of $R_y$ is a subset
    or equal to the corresponding field in the
    condition of $R_x$,
$(\forall i : R_y[i] \subseteq R_x[i])$, where $i \in$ {protocol, s_ip, s_port,
    d_ip, d_port}
```

Generalisation Anomaly A rule is generalisation of another preceding rule if it matches all the packets of the preceding rule. Listing 12 illustrates the case where Rule Y (R_y) is a generalisation by Rule X (R_x).

Listing 12: Logical criteria for the generalisation anomaly type

```
Rule Y $(R_y)$ is a generalisation of Rule X $(R_x)$
if $(R_x)$ precedes $(R_y)$ in the rule set order, $(O_x < O_y)$
    ,
and the permissions of $R_x$ and $R_y$ are different,
    $(P_x \neq P_y)$,
and every field in the condition of $R_x$ is a subset
    or equal to the corresponding field in the
    condition of $R_y$,
$(\forall i : R_x[i] \subseteq R_y[i])$, where $i \in$ {protocol, s_ip, s_port,
    d_ip, d_port}
```

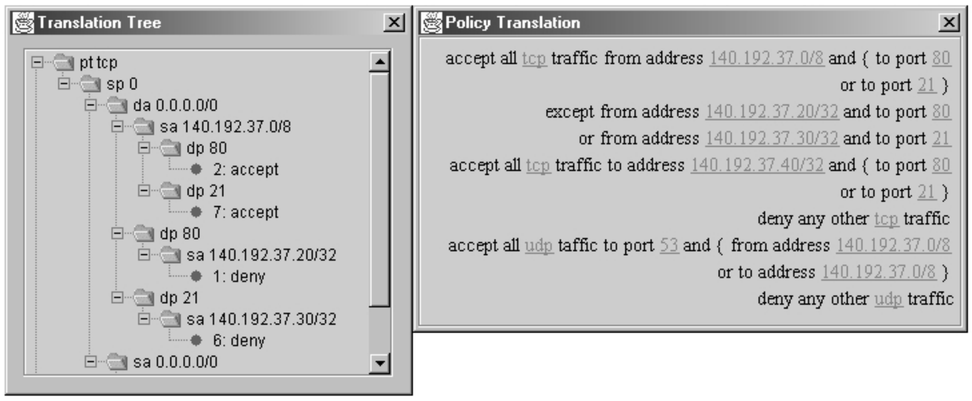
Correlation Anomaly Two rules are correlated if the first rule in order matches some of the fields of the condition of the second rule and the second rule matches some of the fields of the condition of the first rule. If some fields of the condition of Rule X (R_x) are subsets or equal to the corresponding fields of the condition of Rule Y (R_y), and the remaining fields of the condition of rule Rule X (R_x) are supersets to the corresponding fields of the condition of Rule Y (R_y), and the permissions of the two rules are different, then R_x is in correlation with R_y. Listing 13 illustrates the case where Rule X (R_x) is in correlation with Rule Y (R_y).

Listing 13: Logical criteria for the correlation anomaly type

```
Rule X $(R_x)$ is in correlation with Rule Y $(R_y)$
if the permissions of $R_x$ and $R_y$ are different,
    $(P_x \neq P_y)$,
and some fields in the condition of $R_x$ are subsets
    or equal to the corresponding fields in the
    condition of $R_y$ and the rest of the fields in
    the condition of $R_x$ are supersets of the
    corresponding fields in the condition of $R_y$,
($\forall i : R_x[i] \triangleright\triangleleft R_y[i]$, $\exists i, j$ such that $R_x[i] \subseteq R_y[i]$ and
    $R_y[j] \subset R_x[j]$ and $i \neq j$), where $i, j \in$ {protocol, s_ip,
     s_port, d_ip, d_port}
```

2.7 Reverse-Engineering Policies

Research by Al-Shaer and Hamed in [65] and [75] produced the first research into reverse engineering of existing device configurations to high-level, natural language policies. The authors aggregate network services together, using their Firewall Policy Advisor (FPA) [66] system as a base model, and produce a basic text based abstract policy description from a filtering rule set. They then use the binary decision diagram (BDD) produced by the FPA based on the rule set and generate another BDD based on network services. Services can then be aggregated together, changing it into a form which can be presented to the user in a natural language. The interface for the tool uses a graphical interface showing the network services based tree and the textual representation of the policy. Figure 12 from [65] illustrates the network service-based tree and the high-level policy which has been inferred. The creation of an abstract high-level



```
pt tcp
  sp 0
    da 0.0.0.0/0
      sa 140.192.37.0/8
        dp 80
          2: accept
        dp 21
          7: accept
      dp 80
        sa 140.192.37.20/32
          1: deny
      dp 21
        sa 140.192.37.30/32
          6: deny
    sa 0.0.0.0/0
```

```
accept all tcp traffic from address 140.192.37.0/8 and { to port 80
                                                        or to port 21 }
                  except from address 140.192.37.20/32 and to port 80
                       or from address 140.192.37.30/32 and to port 21
accept all tcp traffic to address 140.192.37.40/32 and { to port 80
                                                        or to port 21 }
                                              deny any other tcp traffic
accept all udp taffic to port 53 and { from address 140.192.37.0/8
                                         or to address 140.192.37.0/8 }
                                              deny any other udp traffic
```

Figure 12: Translation tree and policy translation interface [65].

policy from low-level device configurations has also been researched by Tongaonkar et al. in [76] and describes a technique to extract high-level policies from analysis of the low-level representations of rule sets. Their system attempts to flatten the rule set by aggregating overlapping rules together, thus eliminating the dependency on the order of the rules. This also reduces the size and complexity of the rule set, giving the administrator a better understanding of the policy it represents. This technique is similar to that used by Al-Shaer and Hamed in [65]. In [77], Bishop and Peisert propose a system of reverse engineering policies from Linux file system access control configurations. Although this does not involve firewall policies but user access policies from Linux password files, the reverse engineering process is relevant. Around the same time Golnabi et al. have also worked on the problem of inferring high-level policies in [78], but their approach is based around an active system and data mining of device logs, and not by static analysis of the device configurations. Similarly, Abedin et al. describe techniques in [79] which are used to mine firewall log files to regenerate effective firewall rules. Their method uses algorithms to reduce the data set through mining of firewall network traffic logs using packet frequencies. These are then used to calculate the occurrence of each attribute of a log record in a firewall log file, thus generating primitive rules. Actual firewall rules are then regenerated from these primitive rules using aggregation and a set of heuristics. Anomalies in the original rule set and defects in the firewall implementation are then identified through a comparison of the regenerated rules with the original, manually-defined rules.

3 Framework Implementation

3.1 Introduction

This paper builds upon the information sharing policy verification framework. Specifically, it extends the work of Al-Shaer et al. in [1] and that of Hamed and Al-Shaer in [2]. In these, the authors concentrated on developing formal definitions of possible anomalies between rules in a network firewall rule set.

3.2 Policy Verification Process

This section describes the mode of operation of the policy verification framework as illustrated in Figure 13. The process used to verify a proposed policy against possible anomalies uses:

- Definition of policy syntax structure and declaration of policy field elements.
- Syntactic verification of the proposed policy.
- Ontological verification of the proposed policy.
- Functional verification of the proposed policy.

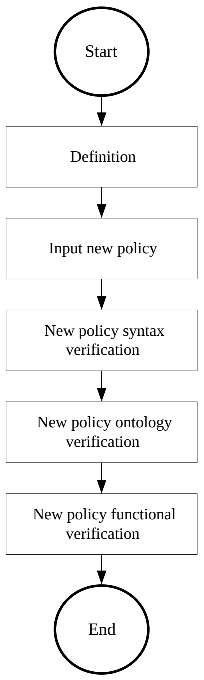

Start
Definition
Input new policy
New policy syntax verification
New policy ontology verification
New policy functional verification
End

Figure 13: Mode of operation of the policy verification framework.

3.2.1 Definition

The definition stage comprises a two-step process where the first step requires the definition of the syntax to be used to describe information sharing policies, and the second step requires a declaration of all possible elements which can occur as part of the fields of a policy. Thus, the definition stage defines the *universe* of all possible policy fields, as well as the possible elements of each field, and forms the basis of any subsequent policy verification processes. The following example illustrates the definition process.

Policy Definition Example This example assumes the scenario that two organisations, Police Force A and Child Protection Agency B, have initiated an information sharing agreement, a policy from which is shown in Listing 14.

Listing 14: Example of a policy in an information sharing agreement.

```
The Records Admin from the Records Unit of
   Child Protection Agency B permits a Sergeant
   from the Domestic Violence Unit of
   Police Force A to read the Unique Identifier of
   a Child, whilst complying with the
   Human Rights Act, 1998, as long as it is for an
   abuse investigation and the Sergeant is the
   Investigating Officer.
```

The first step of the definition stage requires the specification of the information sharing policy syntax. For the purposes of this example, information sharing policies are defined as having a nine-field syntax, where each field is enclosed within square brackets, '[' and ']', as illustrated in Listing 15.

Listing 15: Nine-field syntax used to define information sharing policies.

```
[permission] [requester] [relationship] [action] [
   attribute]   [object] [context] [owner] [
   compliance]
```

Once the information sharing policy syntax has been specified, it can be used to represent polices from the information sharing agreement. For the purposes of this example, the `requester` and `owner` fields are defined as hierarchical, while the remainder of the fields are non-hierarchical. The `requester` and `owner` fields are, both, subdivided into `domain`, `organisation`, `unit` and `role`, with a full stop (‘`.`’) used to delineate between each field sub-division. Listing 16 shows the example information sharing policy from Listing 14 using the syntax from Listing 15.

Listing 16: Information sharing policy from 14 espressed in syntax from 15.

```
[Permit] [Police.Police_Force_A.
   Domestic_Violence_Unit.Sergeant] with [
   Investigating_Officer] relationship [R]    [
   Unique_Identifier] of [Child] with [
   Abuse_Investigation] context from [Social_Care.
   Child_Protection_Agency_B.Records_Unit.
   Records_Admin] with Compliance               [
   Human_Rights_Act_1998]
```

The second step of the definition stage requires a declaration of all possible elements which can occur within the fields of a policy. Table 1 illustrates all possible elements from an example information sharing agreement. In fact, elements used to define the example policy shown in Listing 16 have all been selected from Table 1. It should be noted that a policy field can also be defined using the ‘*’ wildcard, which symbolises that an element has not been declared for a specific field in a policy. Further, as illustrated in Table 1, the elements of the hierarchical `requester` and `owner` fields are declared with respect to their specific higher-level fields. This means that the elements of the highest-level field, `domain`, would be declared by themselves. For example, in Table 1, the domains `Police` and `Social_Care` would be declared by themselves. The elements of the `organisation` field, however, are declared in relation to their respective domains. For example, the organisation `Police_Force_A` is declared in relation to its specific domain, as `Police.Police_Force_A`. Similarly, the organisation `Child_Protection_Agency_B` is declared in relation to its specific domain, as `Social_Care.Child_Protection_Agency_B`. The same principle applies to the subsequent lower-level fields of `unit` and `role`.

3.2.2 Syntax Verification

Syntax verification is the initial stage of the policy verification process. During this stage, a proposed policy is checked to verify that it satisfies the defined syntax criteria for information sharing policies, as specified previously during the definition stage, illustrated in Listing 15. The following example illustrates the syntax verification process. If the proposed policy does not comply with this syntax structure, the testing process is terminated, as other tests only need to be carried out if a policy meets the required syntax criteria.

Table 1: Possible elements of example information sharing policy.

Policy Field	Declared Elements
Permission	Permit Deny
Domain	Social_Care Police
Organisation	(Social_Care) + Child_Protection_Agency_B (Police) + Police_Force_A
Unit	(Social_Care.Child_Protection_Agency_B) + Records_Unit (Police.Police_Force_A) + Domestic_Violence_Unit
Role	(Social_Care.Child_Protection_Agency_B.Records_Unit) + Records_Admin (Police.Police_Force_A.Domestic_Violence_Unit) + Sergeant
Relationship	Investigating_Officer
Action	Read
Attribute	Health_Record Unique_Identifier
Object	Child
Context	Abuse_Investigation
Compliance	Data_Protection_Act

Syntax Verification Example This example assumes an information sharing policy, shown in Listing 17, is proposed to be added to an existing set of policies.

Listing 17: Example policy used for syntax verification example.

```
[Permit] [Police.Police_Force_A.*.Sergeant] with
    [*] relationship [R] [Unique_Identifier] of [
    Child] with       [Abuse_Investigation] context
    from [Social_Care.Child_Protection_Agency_B.
    Records_Unit.Records_Admin] with Compliance [
    Human_Rights_Act_1998]
```

The syntax verification stage checks that the number of fields specified in the proposed policy, as well as the syntax structure of the proposed policy, correspond with the syntax and number of fields defined earlier in the definition stage, as shown in Listing 15. In this example, the policy shown in Listing 17 is parsed to extract its constituent fields, as enclosed within square brackets, '[' and ']'. This process checks that the number of fields in the proposed policy corresponds correctly with the number of fields defined earlier in the definition stage. Further, the `requester` and `owner` fields from the proposed policy, that is fields two and eight, respectively, are checked to ensure that they correspond correctly with the hierarchical structure defined for these fields. This entails ensuring that fields two and eight contain four sub-divisions which are delineated using full stops, ('`.`'). Since the example policy shown in Listing 17 is expressed correctly using the defined syntax, the syntax verification is successful and the next stage of policy verification commences. If the proposed policy had not complied with the defined syntax structure, the testing process would be terminated, as other tests only need to be carried out if a policy meets the required syntax criteria.

3.2.3 Ontology Verification

Ontology verification is the second stage of the policy verification process, following syntax verification. During this stage, a proposed policy is checked to verify that each field of the policy statement comprises valid elements. An element is designated as valid if it has been previously declared in the definition stage. The following example illustrates this process.

Ontology Verification Example This example assumes a scenario where policy field elements, as shown in Table 2, are specified as part of the definition stage in an information sharing agreement.

Four information sharing policies, R_w, R_x, R_y and R_z, are proposed to be added to an existing set of policies. Listings 18, 19, 20 and 21 show policies R_w, R_x, R_y and R_z, respectively.

Table 2: Defined policy field elements for Ontology Verification Example.

Policy Field	Declared Elements
Permission	Permit Deny
Domain	Social_Care Police
Organisation	(Social_Care) + Child_Protection_Agency_B (Police) + Police_Force_A
Unit	(Social_Care.Child_Protection_Agency_B) + Records_Unit (Police.Police_Force_A) + Domestic_Violence_Unit
Role	(Social_Care.Child_Protection_Agency_B.Records_Unit) + Records_Admin (Police.Police_Force_A.Domestic_Violence_Unit) + Sergeant
Relationship	Investigating_Officer
Action	Read (R) Create (C) Update (U) Delete (D)
Attribute	Health_Record Unique_Identifier
Object	Child
Context	Abuse_Investigation
Compliance	Data_Protection_Act

Listing 18: Policy R_w used for ontology verification example.

```
[Permit] [Police.Police_Force_A.
   Domestic_violence_Unit.Sergeant] with [*]
   relationship [R] [Unique_Identifier] of [Child]
   with [Abuse_Investigation] context from
        [Social_Care.Child_Protection_Agency_B.
   Records_Unit.Records_Admin] with Compliance [
   Human_Rights_Act_1998]
```

Listing 19: Policy R_x used for ontology verification example.

```
[Permit] [Police.Police_Force_A.
   Domestic_violence_Unit.*] with [*] relationship
   [R] [Unique_Identifier] of [Child] with  [
   Abuse_Investigation] context from [Social_Care.
   Child_Protection_Agency_B.Records_Unit.
   Records_Admin] with Compliance [
   Human_Rights_Act_1998]
```

Listing 20: Policy R_y used for ontology verification example.

```
[Permit] [Police.Police_Force_A.
   Domestic_violence_Unit.Constable] with [*]
   relationship [R] [Unique_Identifier] of [Child]
   with [Abuse_Investigation] context from
        [Social_Care.Child_Protection_Agency_B.
   Records_Unit.Records_Admin] with Compliance [
   Human_Rights_Act_1998]
```

Listing 21: Policy R_z used for ontology verification example.

```
[Permit] [Police.Police_Force_A.
   Domestic_violence_Unit.Records_Admin] with [*]
   relationship [R]                     [
   Unique_Identifier] of [Child] with [
   Abuse_Investigation] context from [Social_Care.
   Child_Protection_Agency_B.Records_Unit.
   Records_Admin] with Compliance                [
   Human_Rights_Act_1998]
```

The initial step of the ontology verification stage comprises parsing of the proposed policies in order to extract their constituent field elements, as enclosed within square brackets, '[' and ']'. Table 3 shows field elements extracted from the example proposed policies.

It must be noted here that where the '*' wildcard is used instead of a field element, the corresponding field is not checked for against an entry in the definition. Comparison between elements of the proposed policies, as shown in Table 3, and the declared elements shown in Table 2, illustrates that each element in the proposed policy, R_w, exists as a valid declared element, in its respective declared field. Therefore, for policy R_w, the ontology verification process is successful, and the next stage of verification can commence. Similar to policy R_w, each element in the proposed policy, R_x, also

Table 3: Field elements of proposed information sharing policies R_w, R_x, R_y and R_z for Ontology Verification Example 3.2.3.

Policy Field	Elements of Policy R_w	Elements of Policy R_x	Elements of Policy R_y	Elements of Policy R_z
Permission	Permit	Permit	Permit	Permit
RD	Police	Police	Police	Police
RO	(Police)+ PFA	(Police) + PFA	(Police) + PFA	(Police) + PFA
RU	(Police)+ (PFA)+ DVU	(Police)+ (PFA)+ DVU	(Police)+ (PFA)+ DVU	(Police)+ (PFA)+ DVU
RR	(Police)+ (PFA)+ (DVU)+ Sergeant	(Police)+ (PFA)+ (DVU)+ *	(Police)+ (PFA)+ (DVU)+ Constable	(Police)+ (PFA)+ (DVU)+ RA
Relationship	*	*	*	*
Action	R	R	R	R
Attribute	Unique Identifier	Unique Identifier	Unique Identifier	Unique Identifier
Object	Child	Child	Child	Child
Context	Abuse Investigation	Abuse Investigation	Abuse Investigation	Abuse Investigation
OD	SC	SC	SC	SC
OO	(SC)+ CPAB	(SC)+ CPAB	(SC)+ CPAB	(SC)+ CPAB
OU	(SC)+ (CPAB)+ RU	(SC)+ (CPAB)+ RU	(SC)+ (CPAB)+ RU	(SC)+ (CPAB)+ RU
OR	(SC)+ (CPAB)+ (RU)+ RA	(SC)+ (CPAB)+ (RU)+ RA	(SC)+ (CPAB)+ (RU)+ RA	(SC)+ (CPAB)+ (RU)+ RA
Compliance	Data Protection Act	Data Protection Act	Data Protection Act	Data Protection Act

Table 4: *

Field or Element		Abbreviation
Requester/Owner Domain	:	RD/OD
Requester/Owner Organisation	:	RO/OO
Requester/Owner Unit	:	RU/OU
Requester/Owner Role	:	RR/OR
Police_Force_A	:	PFA
Domestic_Violence_Unit	:	DVU
Social_Care	:	SC
Child_Protection_Agency_B	:	CPAB
Records_Unit	:	RU
Records_Admin	:	RA
Data_Protection_Act	:	DPA

exists as a valid declared element, in its respective declared field. Since R_x shows the `Requester_Role` element as the wildcard '*', this field is not checked against the respective field in the definition, and, hence, the ontology verification process for policy R_x is also successful.

In the case of policy R_y, however, the element for the `Requester_Role` field is shown as `Constable`. Since `Constable` is not a defined element for the `Requester_Role` field, and, hence, does not appear as a defined element in the column in Table 2, policy R_y will fail the ontology verification stage. In the case of policy R_z, although the `Requester_Role` field, shown as `Records_Admin`, exists as a declared element in the definitions in Table 2, it does not belong to the `Police.Police_Force_A.Domestic_violence_Unit` hierarchy. Therefore, policy R_z will also fail the ontology verification stage. In the case where a policy fails the ontology verification stage, the testing process would be terminated as other tests only need to be carried out if a policy meets the required ontology criteria.

3.2.4 Functional Verification

The functional verification stage is the final stage of the policy verification process and identifies any potential anomalies which may exist between a proposed policy and those present in an existing set of policies. This stage uses the anomaly definitions of redundancy, shadowing, generalisation and correlation, as detailed in Section 2.6.1. Therefore, functional verification is split into four stages, each to check for a specific category of anomaly. The logical state diagram for anomaly classification is based on the work of Al-Shaer and Hamed in [66], illustrated in Figure 14.

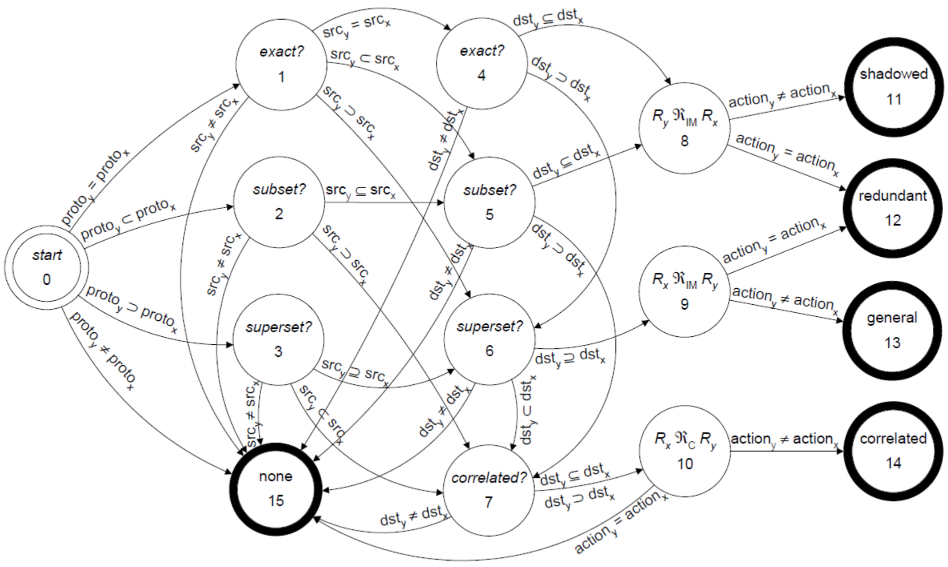


Figure 14: Anomaly state diagram from [66] for rules R_x and R_y, where R_x precedes R_y in the rules list.

A simplified version of the anomaly classification method, derived from the work of Al-Shaer and Hamed in [66], is shown in Figure 15. As indicated, anomalies are detected by logically comparing the permissions, P_x and P_y, and conditions, C_x and C_y, for any two policies, R_x and R_y.

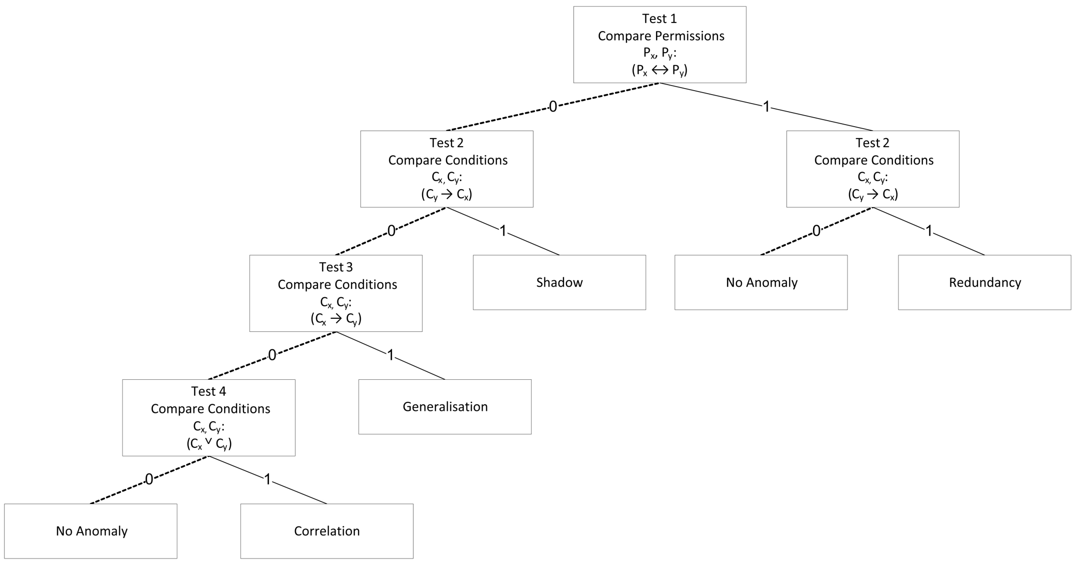


Figure 15: Simplified anomaly classification method for functional verification.

The comparison process, with respect to anomaly detection, entails identification of a subset, superset or equivalence relation between respective fields of the policies, R_x and R_y, which is achieved by carrying out bitwise logical operations between comparative fields. When operating on non-hierarchical fields, the operation simply involves using the entire field in the comparison. For hierarchical fields, however, this involves the definition of the fields as logical conjunctions ($\wedge$), of all of their sub-fields. The logical operation is then performed for the entire hierarchical field. For example, the `Requester` and `Owner` fields can both be defined as logical conjunctions of their constituent `Domain`, `Organisation`, `Unit` and `Role` sub-fields:

$$\begin{aligned} Requester_x, Owner_x &: \quad Domain_x \wedge Organisation_x \wedge Unit_x \wedge Role_x \\ Requester_y, Owner_y &: \quad Domain_y \wedge Organisation_y \wedge Unit_y \wedge Role_y \end{aligned}$$

In practice, the logical comparisons are carried out using Binary Decision Diagrams (BDDs), as implemented for firewall rules by Hazelhurst et al. in [52] and access list modelling by Hazelhurst in [80]. However, the logical 'AND' operation, or conjunction ($\wedge$), is used here to identify subset, superset or equivalence relations. This allows a generic method of illustrating logical relationships, which is independent of the internal computations of any specific Binary Decision Diagram (BDD) software package.

Since BDDs perform bitwise logical operations, the permissions, P_x and P_y, and conditions, C_x and C_y, of policies R_x and R_y must first be represented in binary bits. However, unlike modelling firewall rules and access lists, information sharing policies can have fields of varying lengths. This is due to the fact that there is no limit on the number of possible elements which may be declared as part of a field in an information sharing agreement. Hence, since the number of possible elements in a field can vary, where each field must be adjusted for the number of bits before any bitwise operations can be performed on it. The adjustment process involves assigning an element identifier, Element ID, to each element in a field and then representing the element identifier in binary form. Hence, the total number of binary variables needed to represent a field element is dependent on the total number of possible field elements declared during the definition stage. Table 5 illustrates the assignment of Element IDs to elements from Table 2, from the Ontology Verification Example, and their binary representations.

Table 5: Assignment of Element IDs to elements from Table 2 and their binary representation.

Policy Field	Elements from Definition	Element ID	Binary Representation
Permission	Deny	0	0
	Permit	1	1
Domain	Social Care	1	01
	Police	2	10
Organisation	Child Protection Agency B	1	01
	Police Force A	2	10
Unit	Records Unit	1	01
	Domestic Violence Unit	2	10
Role	Records Admin	1	01
	Sergeant	2	10
Relationship	Investigating Officer	1	1
Action	Read (R)	1	001
	Create (C)	2	010
	Update (U)	3	011
	Delete (D)	4	100
Attribute	Health Record	1	01
	Unique Identifier	2	10
Object	Child	1	1
Context	Abuse Investigation	1	1
Compliance	Data Protection Act	1	1

Listing 22 shows an example information sharing policy with field elements populated from elements defined in Table 5. Listing 23 illustrates the example policy from Listing 22 in its binary representation.

Listing 22: Example information sharing policy with field elements populated from elements defined in Table 5.

```
$R_x$:
[Permit] [Police.Police_Force_A.
   Domestic_Violence_Unit.Sergeant] with [
   Investigating_Officer] relationship [R]    [
   Health_Record] of [Child] with [
   Abuse_Investigation] context from [Social_Care.
   Child_Protection_Agency_B.Records_Unit.
   Records_Admin] with Compliance               [
   Data_Protection_Act]
```

Listing 23: Example information sharing policy from Listing 22 in binary representation.

```
Rx:
[1] [10.10.10.10] [1] [001] [01] [1] [1]
    [01.01.01.01] [1]
```

The binary representation in Listing 23 shows that an information sharing policy populated from elements defined in Table 5 will consist of 26 binary bits. If, however, the Role field is now updated to include two additional roles, Constable and Analyst, the number of bits required to represent the policy will change. Table 6 shows the Role field from Table 5 updated with the additional roles.

Table 6: Role field from Table 5 updated with Constable and Analyst roles.

Policy Field	Elements from Definition	Element ID	Binary Representation
Role	Records Admin	1	001
	Sergeant	2	010
	Constable	3	011
	Analyst	4	100

The same information sharing policy, R_x from Listing 22, now expressed using the updated Role field from Table 6, will have the binary form as shown in Listing 24. As shown, 28 binary bits are now required to represent R_x whereas previously only 26 binary bits were required. This illustrates the sensitivity of binary expressions and, hence, any binary calculations, to modifications made to the set of elements registered for an information sharing agreement.

Listing 24: Example information sharing policy from Listing 22 represented in binary form using updated Role field from Table 6.

```
Rx:
[1] [10.10.10.010] [1] [001] [01] [1] [1]
    [01.01.01.001] [1]
```

The following examples provide details on each stage of the functional verification process. The methods used for functional verification are as illustrated in Figure 15. Each example assumes an information sharing agreement between two organisations, *Police Force A* and *Child Protection Agency B*. Only the `requester` and `owner` fields are manipulated in the following anomaly verification examples for reasons of brevity.

3.2.5 Example of Redundancy Anomaly Verification

Elements from Table 5 are used to define two information sharing policies. Listing 25 shows R_x, an existing policy in the agreement and Listing 26 shows R_y, a proposed policy to be added. Listing 27 shows the binary representations of the two policies.

Listing 25: Existing policy set for redundancy anomaly functional verification example.

```
Rx :    [Permit] [Police.Police_Force_A.
   Domestic_Violence_Unit.*] with [*] relationship
   [R] [*] of [Child] with [*] context from [
   Social_Care.Child_Protection_Agency_B.
   Records_Unit.*] with Compliance [*]
```

Listing 26: Proposed policy for redundancy anomaly functional verification example.

```
Ry :    [Permit] [Police.Police_Force_A.
   Domestic_Violence_Unit.Sergeant] with [*]
   relationship [R] [*] of [Child] with [*] context
    from [Social_Care.Child_Protection_Agency_B.
   Records_Unit.*] with Compliance [*]
```

Listing 27: Binary representation of existing policy R_x from Listing 25 and proposed policy R_y from Listing 26 for redundancy anomaly functional verification example.

```
Rx :    [1] [10.10.10.*] with [*] relationship [R]
   [*] of [1] with [*] context from [01.01.01.*]
   with Compliance [*]
Ry :    [1] [10.10.10.10] with [*] relationship [R]
    [*] of [1] with [*] context from [01.01.01.*]
   with Compliance [*]
```

Test 1: As illustrated in Figure 15, the first comparison in the functional verification stage, Test 1, is to compare the permissions, P_x and P_y, from policies R_x and R_y to check if they are the same. This operation is illustrated in the computation below using the logical relationship that if the permissions P_x and P_y are the same, $(P_x \Leftrightarrow P_y)$ is TRUE, then the expression $((P_x \wedge P_y) \vee (\neg P_x \wedge \neg P_y))$ must also be TRUE.

$$\begin{array}{ccc} P_x & : & 1 \\ P_y & : & 1 \\ (P_x \Leftrightarrow P_y) & : & ((P_x \wedge P_y) \vee (\neg P_x \wedge \neg P_y)) \\ & : & ((1 \wedge 1) \vee (\neg 1 \wedge \neg 1)) \\ \therefore (P_x \Leftrightarrow P_y) & : & \text{TRUE} \end{array}$$

Test 2: Since P_x and P_y are the same, the next comparison in the functional verification stage, Test 2, is to compare the conditions, C_x and C_y, to check if $(C_y \Rightarrow C_x)$ is TRUE. This test, if TRUE, indicates that C_y is either equal to, or a subset of, C_x which means that policy R_y is redundant to policy R_x. This operation is illustrated in the computation below using the logical relationship that if condition C_y implies condition C_x, $(C_y \Rightarrow C_x)$ is TRUE, then the expression $(\neg C_y \vee C_x)$ must also be TRUE.

$$\begin{array}{rcc} C_x & : & \text{10.10.10.XX} \\ C_y & : & \text{10.10.10.10} \\ C_y \Rightarrow C_x & : & \neg C_y \vee C_x \\ & : & \text{01.01.01.01} \vee \text{10.10.10.XX} \\ \therefore C_y \Rightarrow C_x & : & \text{TRUE} \end{array}$$

Conclusion: As can be seen from Figure 15, if Test 1 and Test 2 result in TRUE, policy R_y is redundant to policy R_x.

3.2.6 Example of Shadow Anomaly Verification

Elements from Table 5 are used to define two information sharing policies. Listing 28 shows R_x, an existing policy in the agreement and Listing 29 shows R_y, a proposed policy to be added. Listing 30 shows the binary representations of the two policies.

Listing 28: Existing policy set for shadow anomaly functional verification example.

```
Rx :    [Deny] [Police.Police_Force_A.
   Domestic_Violence_Unit.*] with [*] relationship
   [R] [*] of [Child] with [*] context from [
   Social_Care.Child_Protection_Agency_B.
   Records_Unit.*] with Compliance [*]
```

Listing 29: Proposed policy for shadow anomaly functional verification example.

```
Ry :    [Permit] [Police.Police_Force_A.
   Domestic_Violence_Unit.Sergeant] with [*]
   relationship [R] [*] of [Child] with [*] context
    from [Social_Care.Child_Protection_Agency_B.
   Records_Unit.*] with Compliance [*]
```

Listing 30: Binary representation of existing policy R_x from Listing 28 and proposed policy R_y from Listing 29 for shadow anomaly functional verification example.

```
Rx :    [0] [10.10.10.*] with [*] relationship [R]
   [*] of [1] with [*] context from [01.01.01.*]
   with Compliance [*]
Ry :    [1] [10.10.10.10] with [*] relationship [R]
    [*] of [1] with [*] context from [01.01.01.*]
   with Compliance [*]
```

Test 1: As illustrated in Figure 15, the first comparison in the functional verification stage, Test 1, is to compare the permissions, P_x and P_y, from policies R_x and R_y to check if they are the same. This operation is illustrated in the computation below using the logical relationship that if the permissions P_x and P_y are the same, $(P_x \Leftrightarrow P_y)$ is TRUE, then the expression $((P_x \wedge P_y) \vee (\neg P_x \wedge \neg P_y))$ must also be TRUE.

$$\begin{array}{ccc} P_x & : & 0 \\ P_y & : & 1 \\ (P_x \Leftrightarrow P_y) & : & ((P_x \wedge P_y) \vee (\neg P_x \wedge \neg P_y)) \\ & : & ((0 \wedge 1) \vee (\neg 0 \wedge \neg 1)) \\ \therefore (P_x \Leftrightarrow P_y) & : & \text{FALSE} \end{array}$$

Test 2: Since P_x and P_y are different, the next comparison in the functional verification stage, Test 2, is to compare the conditions, C_x and C_y, to check if $(C_y \Rightarrow C_x)$ is TRUE. This test, if TRUE, indicates that C_y is either equal to, or a subset of, C_x which means that policy R_y is shadowed by policy R_x. This operation is illustrated in the computation below using the logical relationship that if condition C_y implies condition C_x, $(C_y \Rightarrow C_x)$ is TRUE, then the expression $(\neg C_y \vee C_x)$ must also be TRUE.

$$\begin{array}{ccc} C_x & : & 10.10.10.XX \\ C_y & : & 10.10.10.10 \\ C_y \Rightarrow C_x & : & \neg C_y \vee C_x \\ & : & 01.01.01.01 \vee 10.10.10.XX \\ \therefore C_y \Rightarrow C_x & : & \text{TRUE} \end{array}$$

Conclusion: As can be seen from Figure 15, if Test 1 results in FALSE and Test 2 results in TRUE, policy R_y is shadowed by policy R_x.

3.2.7 Example of Generalisation Anomaly Verification

Elements from Table 5 are used to define two information sharing policies. Listing 31 shows R_x, an existing policy in the agreement and Listing 32 shows R_y, a proposed policy to be added. Listing 33 shows the binary representations of the two policies.

Listing 31: Existing policy set for generalisation anomaly functional verification example.

```
Rx :    [Deny] [Police.Police_Force_A.
   Domestic_Violence_Unit.Sergeant] with [*]
   relationship [R] [*] of [Child] with [*] context
    from [Social_Care.Child_Protection_Agency_B.
   Records_Unit.*] with Compliance [*]
```

Listing 32: Proposed policy for generalisation anomaly functional verification example.

```
Ry :    [Permit] [Police.Police_Force_A.
   Domestic_Violence_Unit.*] with [*] relationship
   [R] [*] of [Child] with [*] context from [
   Social_Care.Child_Protection_Agency_B.
   Records_Unit.*] with Compliance [*]
```

Listing 33: Binary representation of existing policy R_x from Listing 31 and proposed policy R_y from Listing 32 for generalisation anomaly functional verification example.

```
R_x :    [0] [10.10.10.10] with [*] relationship [R]
    [*] of [1] with [*] context from [01.01.01.*]
    with Compliance [*]
R_y :    [1] [10.10.10.*] with [*] relationship [R]
    [*] of [1] with [*] context from [01.01.01.*]
    with Compliance [*]
```

Test 1: As illustrated in Figure 15, the first comparison in the functional verification stage, Test 1, is to compare the permissions, P_x and P_y, from policies R_x and R_y to check if they are the same. This operation is illustrated in the computation below using the logical relationship that if the permissions P_x and P_y are the same, $(P_x \Leftrightarrow P_y)$ is TRUE, then the expression $((P_x \wedge P_y) \vee (\neg P_x \wedge \neg P_y))$ must also be TRUE.

$$\begin{array}{ccc}
P_x & : & 0 \\
P_y & : & 1 \\
(P_x \Leftrightarrow P_y) & : & ((P_x \wedge P_y) \vee (\neg P_x \wedge \neg P_y)) \\
 & : & ((0 \wedge 1) \vee (\neg 0 \wedge \neg 1)) \\
\therefore (P_x \Leftrightarrow P_y) & : & \text{FALSE}
\end{array}$$

Test 2: Since P_x and P_y are different, the next comparison in the functional verification stage, Test 2, is to compare the conditions, C_x and C_y, to check if $(C_y \Rightarrow C_x)$ is TRUE. This test, if TRUE, indicates that C_y is either equal to, or a subset of, C_x which means that policy R_y is shadowed by policy R_x. This operation is illustrated in the computation below using the logical relationship that if condition C_y implies condition C_x, $(C_y \Rightarrow C_x)$ is TRUE, then the expression $(\neg C_y \vee C_x)$ must also be TRUE.

$$\begin{array}{ccc}
C_x & : & 10.10.10.10 \\
C_y & : & 10.10.10.XX \\
C_y \Rightarrow C_x & : & \neg C_y \vee C_x \\
 & : & 01.01.01.00 \vee 10.10.10.10 \\
\therefore C_y \Rightarrow C_x & : & \text{FALSE}
\end{array}$$

Test 3: Since the result of Test 2 is FALSE, the next comparison in the functional verification stage, Test 3, is to compare the conditions, C_x and C_y, to check if $(C_x \Rightarrow C_y)$ is TRUE. This test, if TRUE, indicates that C_x is either equal to, or a subset of, C_y which means that policy R_y is a generalisation of policy R_x. This operation is illustrated in the computation below using the logical relationship that if condition C_x implies condition C_y, $(C_x \Rightarrow C_y)$ is TRUE, then the expression $(\neg C_x \vee C_y)$ must also be TRUE.

$$\begin{array}{ccc}
C_x & : & 10.10.10.10 \\
C_y & : & 10.10.10.XX \\
C_x \Rightarrow C_y & : & \neg C_x \vee C_y \\
 & : & 01.01.01.01 \vee 10.10.10.11 \\
\therefore C_y \Rightarrow C_x & : & \text{TRUE}
\end{array}$$

Conclusion: As can be seen from Figure 15, if Test 1 results in FALSE, Test 2 results in FALSE and Test 3 results in TRUE, policy R_y is a generalisation of policy R_x.

3.2.8 Example of Correlation Anomaly Verification

Elements from Table 5 are used to define two information sharing policies. Listing 34 shows R_x, an existing policy in the agreement and Listing 35 shows R_y, a proposed policy to be added. Listing 36 shows the binary representations of the two policies. It must be noted that, unlike in previous examples, in this example the conditions C_x and C_y of policies R_x and R_y have different `Requester` and `Owner` fields. Therefore, the conjunction of both these fields will be used in the tests below when comparing conditions.

Listing 34: Existing policy set for correlation anomaly functional verification example.

```
Rx :    [Deny] [Police.Police_Force_A.
   Domestic_Violence_Unit.Sergeant] with [*]
   relationship [R] [*] of [Child] with [*] context
    from [Social_Care.Child_Protection_Agency_B.
   Records_Unit.*] with Compliance [*]
```

Listing 35: Proposed policy for correlation anomaly functional verification example.

```
Ry :    [Permit] [Police.Police_Force_A.
   Domestic_Violence_Unit.*] with [*] relationship
   [R] [*] of [Child] with [*] context from [
   Social_Care.Child_Protection_Agency_B.
   Records_Unit.Records_Admin] with Compliance [*]
```

Listing 36: Binary representation of existing policy R_x from Listing 34 and proposed policy R_y from Listing 35 for correlation anomaly functional verification example.

```
Rx :    [0] [10.10.10.10] with [*] relationship [R]
    [*] of [1] with [*] context from [01.01.01.*]
   with Compliance [*]
Ry :    [1] [10.10.10.*] with [*] relationship [R]
   [*] of [1] with [*] context from [01.01.01.01]
   with Compliance [*]
```

Test 1: As illustrated in Figure 15, the first comparison in the functional verification stage, Test 1, is to compare the permissions, P_x and P_y, from policies R_x and R_y to check if they are the same. This operation is illustrated in the computation below using the logical relationship that if the permissions P_x and P_y are the same, $(P_x \Leftrightarrow P_y)$ is TRUE, then the expression $((P_x \wedge P_y) \vee (\neg P_x \wedge \neg P_y))$ must also be TRUE.

$$\begin{array}{rcc}
P_x & : & 0 \\
P_y & : & 1 \\
(P_x \Leftrightarrow P_y) & : & ((P_x \wedge P_y) \vee (\neg P_x \wedge \neg P_y)) \\
 & : & ((0 \wedge 1) \vee (\neg 0 \wedge \neg 1)) \\
\therefore (P_x \Leftrightarrow P_y) & : & \text{FALSE}
\end{array}$$

Test 2: Since P_x and P_y are different, the next comparison in the functional verification stage, Test 2, is to compare the conditions, C_x and C_y, to check if $(C_y \Rightarrow C_x)$ is TRUE. This test, if TRUE, indicates that C_y is either equal to, or a subset of, C_x which means that policy R_y is shadowed by policy R_x. This operation is illustrated in the computation below using the logical relationship that if condition C_y implies condition C_x, $(C_y \Rightarrow C_x)$ is TRUE, then the expression $(\neg C_y \vee C_x)$ must also be TRUE.

		Requester Field	Owner Field
C_x	:	10.10.10.10	01.01.01.XX
C_y	:	10.10.10.XX	01.01.01.01
$C_y \Rightarrow C_x$	:	$\neg C_y \vee C_x$	$\neg C_y \vee C_x$
	:	01.01.01.00 $\vee$ 10.10.10.10	10.10.10.10 $\vee$ 01.01.01.11
	:	FALSE	TRUE
$\therefore C_y \Rightarrow C_x$	:	FALSE	

Test 3: Since the result of Test 2 is FALSE, the next comparison in the functional verification stage, Test 3, is to compare the conditions, C_x and C_y, to check if $(C_x \Rightarrow C_y)$ is TRUE. This test, if TRUE, indicates that C_x is either equal to, or a subset of, C_y which means that policy R_y is a generalisation of policy R_x. This operation is illustrated in the computation below using the logical relationship that if condition C_x implies condition C_y, $(C_x \Rightarrow C_y)$ is TRUE, then the expression $(\neg C_x \vee C_y)$ must also be TRUE.

		Requester Field	Owner Field
C_x	:	10.10.10.10	01.01.01.XX
C_y	:	10.10.10.XX	01.01.01.01
$C_x \Rightarrow C_y$	:	$\neg C_x \vee C_y$	$\neg C_x \vee C_y$
	:	01.01.01.01 $\vee$ 10.10.10.11	10.10.10.00 $\vee$ 01.01.01.01
	:	TRUE	FALSE
$\therefore C_x \Rightarrow C_y$	:	FALSE	

Test 4: Since the result of Test 3 is FALSE, the next comparison in the functional verification stage, Test 4, is to compare the conditions, C_x and C_y, to check for correlated fields. For this example, this requires checking that the `Requester` and `Owner` fields of one policy are the same as, or subsets of, the corresponding fields of the other policy and that the remaining fields of the former policy are supersets of the corresponding fields of the latter policy. Formally, this means that for the fields being compared, if for certain fields $(C_x \Rightarrow C_y)$ is TRUE, then $(C_y \Rightarrow C_x)$ must be TRUE for the remaining fields being compared. This operation is illustrated in the computation below using the logical relationship that if condition C_x implies condition C_y, $(C_x \Rightarrow C_y)$ is TRUE, then the expression $(\neg C_x \vee C_y)$ must also be TRUE.

$$
\begin{array}{ccccc}
 & & \text{Requester Field} & & \text{Owner Field} \\
C_x & : & \text{10.10.10.10} & & \text{01.01.01.XX} \\
C_y & : & \text{10.10.10.XX} & & \text{01.01.01.01} \\
Requester_x \Rightarrow Requester_y & : & \neg Requester_x \vee Requester_y & & \\
 & : & \text{01.01.01.01} \vee \text{10.10.10.11} & & \\
 & : & \text{TRUE} & & \\
Owner_y \Rightarrow Owner_x & : & & & \neg Owner_y \vee Owner_x \\
 & : & & & \text{10.10.10.10} \vee \text{01.01.01.11} \\
 & : & & & \text{TRUE} \\
\therefore Requester_x \Rightarrow Requester_y & & & & \\
\wedge Owner_y \Rightarrow Owner_x & : & \multicolumn{3}{c}{\text{TRUE}}
\end{array}
$$

Conclusion: As can be seen from Figure 15, if Test 1 results in FALSE, Test 2 results in FALSE, Test 3 results in FALSE and Test 4 results in TRUE, policies R_x and R_y are correlated.

4 Framework Evaluation

4.1 Introduction

This section presents an analysis of the evaluation scenarios which are constructed in order to analyse the performance of the proposed information sharing framework, based on varying central processing unit (CPUs) and random-access memory (RAM) availability.

4.2 Total Processing-Time

This section offers an overview of the total processing times for the four scenarios under evaluation. As illustrated in Figures 16, 17, 18 and 19, the total processing times increase with respect to increasing policy set size for all four scenarios. This is expected, since the resource configurations do not change within a scenario, and only the policy set size varies. Hence, with the same available resource configuration, it is expected that the total processing time will increase proportionally to the size of the policy set. In fact, as illustrated by the graphs in the figures mentioned, the rate of change of the total processing time increases with increasing policy set size, indicating a polynomial relationship. A related observation from Figures 16, 17, 18

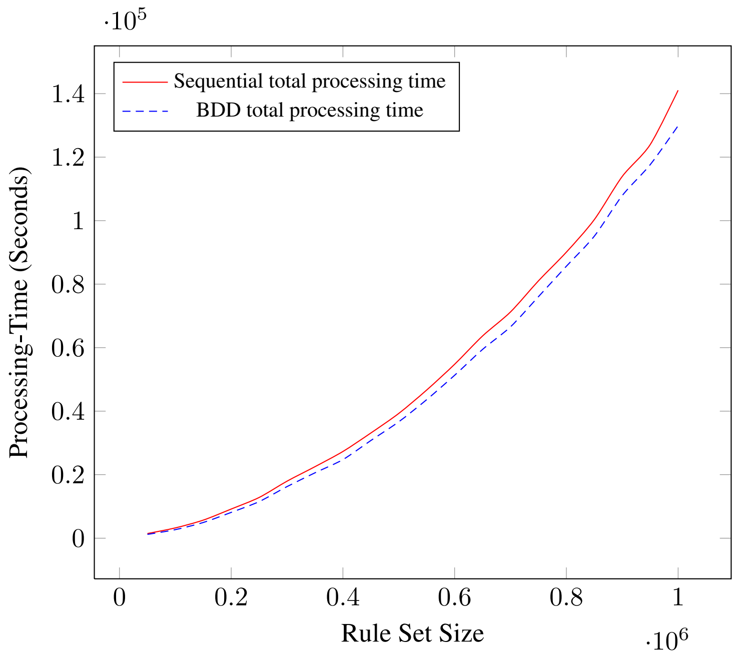


Figure 16: Scenario 1: Comparison of total processing times using sequential and BDD methods against increasing rule set size

and 19 is that the total processing times for the sequential method are higher than the total processing times for the method using Binary Decision Diagrams (BDDs), for all four evaluation scenarios. This result is expected, as the process using BDDs, due to

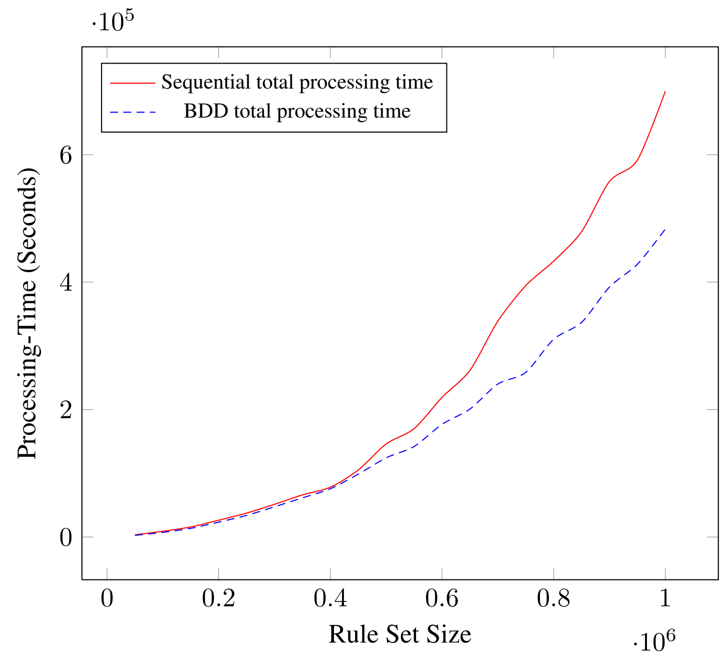


Figure 17: Scenario 2: Comparison of total processing times using sequential and BDD methods against increasing rule set size

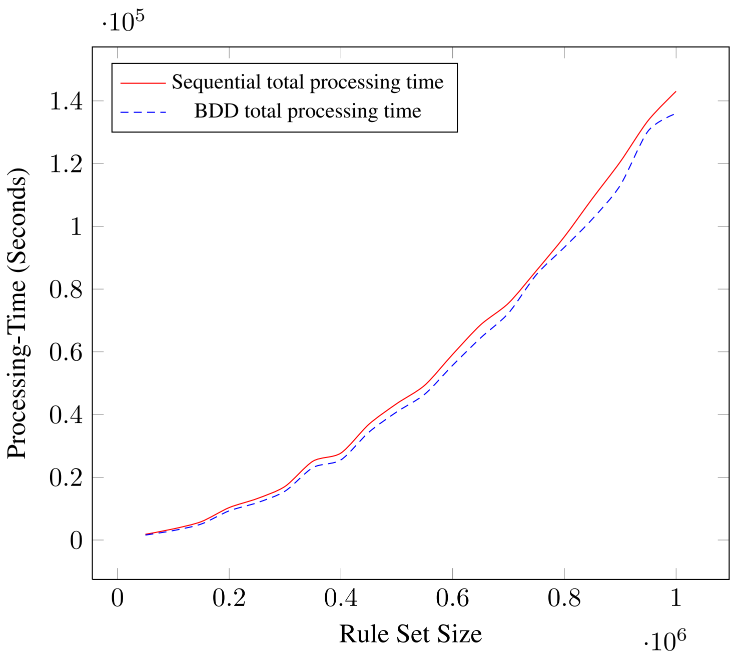


Figure 18: Scenario 3: Comparison of total processing times using sequential and BDD methods against increasing rule set size

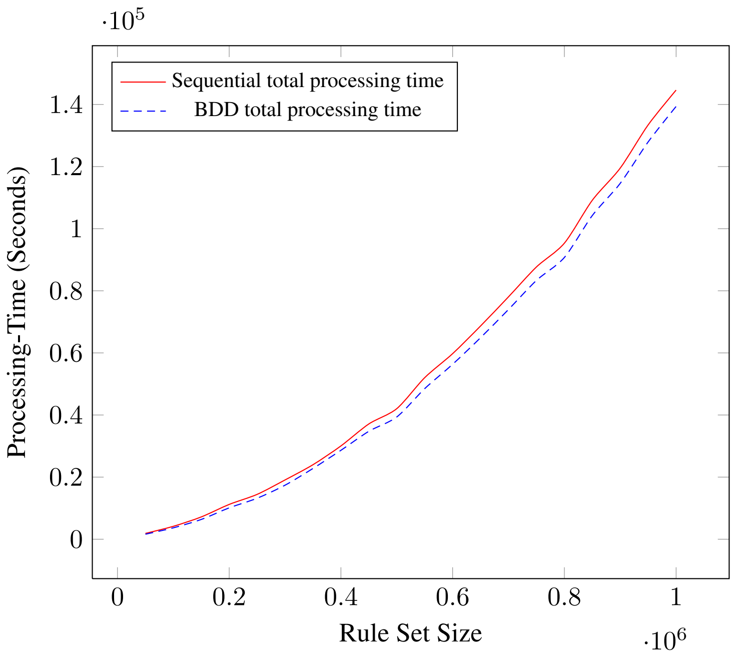


Figure 19: Scenario 4: Comparison of total processing times using sequential and BDD methods against increasing rule set size

their tree structure, involves fewer computations than sequential comparisons. Further, it should be noted that the total processing time for the sequential process increases at a greater rate than the total processing time for the process using BDDs. This is illustrated in Figures 20, 21, 22 and 23, where each figure highlights the increasing difference between sequential and BDD total processing times. The calculation for this is formalised as $f(x)$ in Function 1.

$$f(x) = (TotalProcessingTime)_{Sequential} - (TotalProcessingTime)_{BDD} \quad (1)$$

An important outcome from this observation is that, as policy set sizes increase, policy verification processes using BDDs offer increasingly better performance, in terms of faster total completion times, than verification processes using sequential matching. It is also important to note that, apart from total processing times, measurements were also made of the percentage processor utilisation over time and are shown in the results in the Appendices. These were taken for each stage of the policy verification process and for every policy set size (50,000 to 1,000,000 policies) under test, during each of the four evaluation scenarios. An analysis of these measurements indicates that the percent processor utilisation does not vary noticeably with either increasing policy set size or with changes in the resource configuration as defined by each evaluation scenario. However, it is interesting to note that the arithmetic mean value of the percent processor utilisation for the sequential method is consistently lower than that BDD-based method. This implies that, although there is no direct relationship between increased processor utilisation and increasing policy set sizes, the BDD-based method, registers lower processor utilisation than the sequential comparison method. Table 7 offers a comparison of the variation in total processing times against available resources, based on the four scenarios under evaluation. This involves computing the arithmetic mean

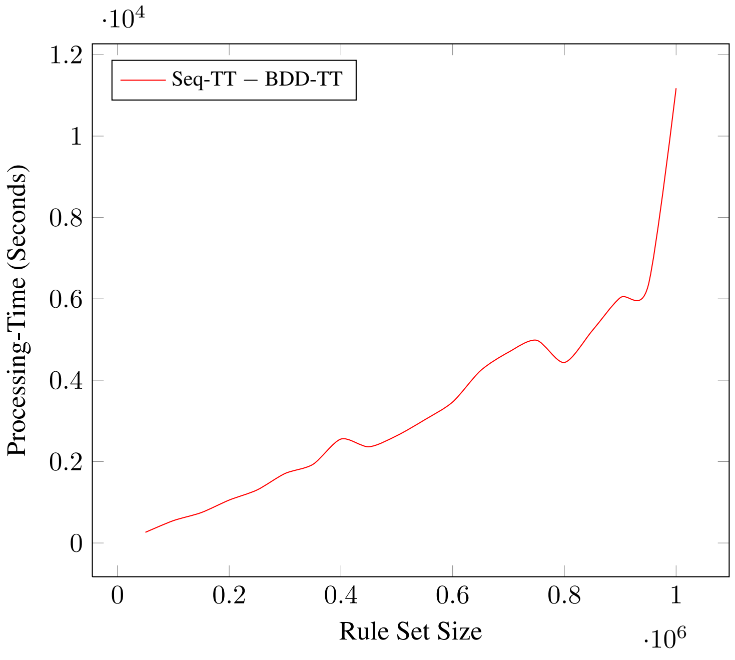


Figure 20: Scenario 1: Graph of difference in total processing times of sequential and BDD methods against increasing policy set size.

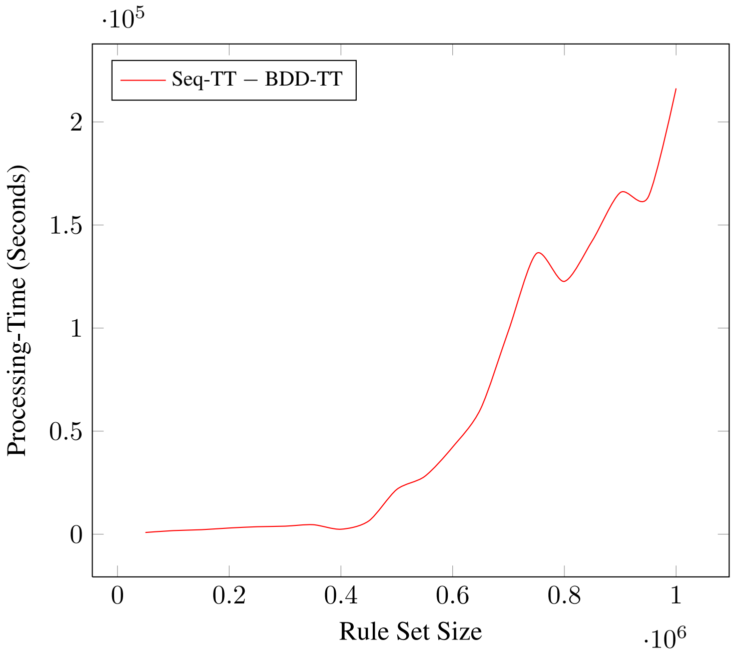


Figure 21: Scenario 2: Graph of difference in total processing times of sequential and BDD methods against increasing policy set size.

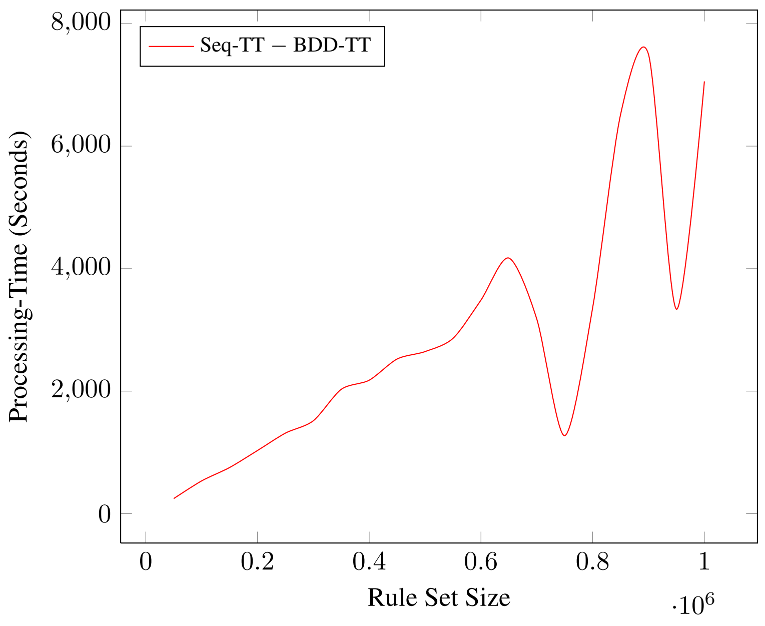


Figure 22: Scenario 3: Graph of difference in total processing times of sequential and BDD methods against increasing policy set size.

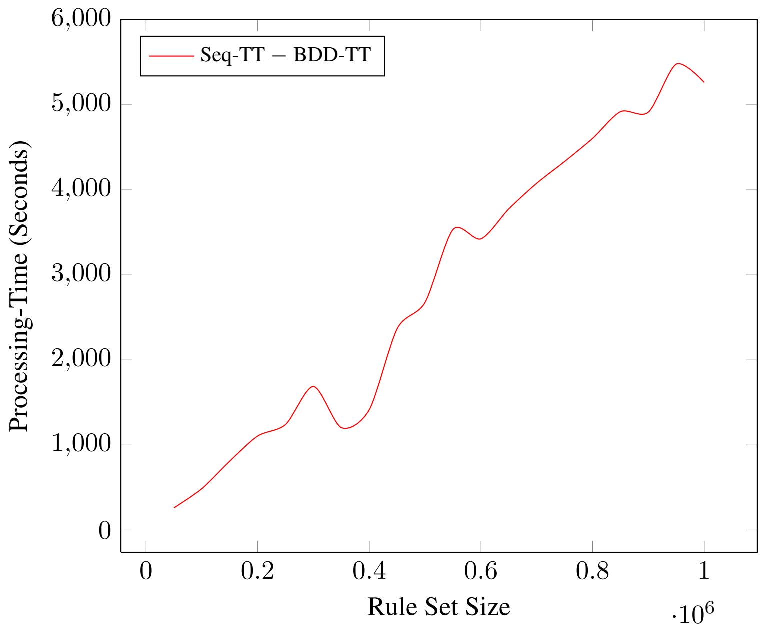


Figure 23: Scenario 4: Graph of difference in total processing times of sequential and BDD methods against increasing policy set size.

for the total processing time using the sequential method, (PTMean$_{Seq}$), and again for the BDD-based method, (PTMean$_{BDD}$), for each scenario. As can be seen from Table 7, there is minimal variation in the total processing times, for the sequential and BDD-based methods, for Scenarios 1, 3 and 4. The variation in PTMean$_{Seq}$ for Scenarios 1, 3 and 4 is 4004 seconds, while the variation in PTMean$_{BDD}$ is 4559 seconds. The results for Scenario 2, however, indicate significantly higher values, with a PTMean$_{Seq}$ time of 234263 seconds and a PTMean$_{BDD}$ time of 172911 seconds. These values are in excess of the range values for the other evaluation scenarios by a factor of 58.5 for PTMean$_{Seq}$ and 37.9 for PTMean$_{BDD}$. As the resource configuration for Scenario 2

Table 7: Mean total processing times by scenario.

Evaluation Scenario	PTMean$_{Seq}$ Seconds	PTMean$_{BDD}$ Seconds	PTMean$_{Seq}$ − PTMean$_{BDD}$ Seconds
Scenario 1	52966	49531	3435
Scenario 2	234263	172911	61352
Scenario 3	56324	53449	2875
Scenario 4	56970	54090	2880

has the lowest amount of random-access memory (RAM), this result implies a high dependence of the testing process, for both the sequential and BDD-based methods, on RAM availability. It could be argued that this result may be influenced by the availability of the number of central processing units (CPUs). However, this argument is discounted on the basis that Scenario 3, which includes the same number of CPUs as Scenario 2 but has double the available RAM, shows similar results to the other evaluation scenarios. Also, Scenario 1, which includes half the number of available CPUs as Scenario 2, but with double the available RAM, also shows similar results to the other evaluation scenarios. A comparison of these results indicates that varying the number of available CPUs has only a minimal effect on total processing times, while a decrease in the amount of available RAM may significantly affect processing times. The relatively negligible variation between Scenarios 1, 3 and 4 also imply that a minimum amount of RAM is required for optimal performance. Based on the results in Table 7, this amount seems to be 2048MB in the case of the selected evaluation scenarios, as an increase in this amount, in Scenario 4 for example, does not result in an improvement in total processing times. Further, as shown in Figure 17, Scenario 2 displays the largest difference between sequential and BDD-based total processing times. This is an indication that, under constrained RAM conditions, the sequential method is adversely affected to a much higher degree than the BDD-based method. Table 8 offers a more detailed overview of total processing time by providing a breakdown of the mean total processing times of each stage of the policy verification process. The mean processing times for both the sequential, PTMean$_{Seq}$, and BDD-based, PTMean$_{BDD}$, methods are expressed as percentages of the total mean processing time. As indicated in Table 8, the majority of the processing time is utilised at the functional verification stage and, secondly, by the ontology verification stage. The syntax verification utilises the minimum amount of processing time for each scenario. This holds true for both the sequential and BDD-based methods.

Table 8: Mean processing times by scenario and verification stage, expressed as percentage of PTMean$_{Seq}$ and PTMean$_{BDD}$.

Scenario	Verification Stage	Percentage of Mean Processing Time (Sequential)	Percentage of Mean Processing Time (BDD)
Scenario 1	Syntax	0.27	0.26
	Ontology	15.12	16.20
	Functional	84.61	83.54
Scenario 2	Syntax	0.05	0.96
	Ontology	5.50	6.50
	Functional	94.45	92.54
Scenario 3	Syntax	0.22	0.30
	Ontology	21.43	22.52
	Functional	78.35	77.18
Scenario 4	Syntax	0.24	0.28
	Ontology	23.14	24.34
	Functional	76.62	75.38

4.3 Random-Access Memory (RAM) Usage

As identified in Section 4.2, the performance of policy verification process is sensitive to the amount of available random-access memory (RAM). This section offers an analysis of the mean RAM usage with respect to each stage of the verification process.

4.3.1 Syntax Verification Stage

Figure 24 illustrates the mean random-access memory (RAM) usage for the syntax verification stage of each of the four evaluation scenarios. As indicated, the general trend of RAM usage for this stage is a logarithmic increase approaching a stable value. Table 9 lists these values, in megabytes (MB), for all four scenarios, along with the corresponding policy set sizes, where they occur. The results indicate that Scenario 1 approaches a stable mean RAM usage value of 11.47 MB, while Scenarios 2, 3 and 4 approach a value of 11.70 MB.

Table 9: Stable mean RAM usage values (S-MRU) for the syntax verification stage and corresponding policy set sizes.

Scenario	Stable RAM Value (Megabytes)	Policy Set Size
Scenario 1	11.47	600000
Scenario 2	11.70	350000
Scenario 3	11.70	400000
Scenario 4	11.70	550000

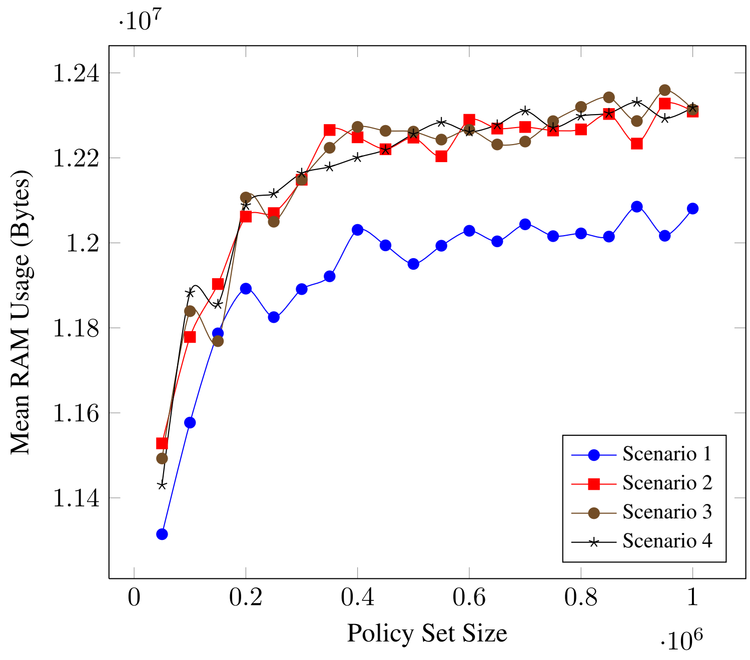


Figure 24: Graph of Mean RAM Usage (S-MRU) during the syntax verification stage against increasing policy set size.

4.3.2 Ontology Verification Stage

Figure 25 illustrates the mean random-access memory (RAM) usage for the ontology verification stage of each of the four evaluation scenarios. Table 10 lists these values, in megabytes (MB), for all four scenarios. The respective policy set sizes are not listed as the value fluctuates evenly around a central mean, and does not vary significantly with respect to increasing policy set sizes.

Table 10: Stable mean RAM usage values (O-MRU) for the ontology verification stage and corresponding policy set sizes.

Scenario	Stable RAM Value (Megabytes)
Scenario 1	21.14
Scenario 2	21.48
Scenario 3	21.47
Scenario 4	21.46

4.3.3 Functional Verification Stage

This section offers a comparison of the mean random-access memory (RAM) usage during the functional verification stage of the policy verification process. It focuses on

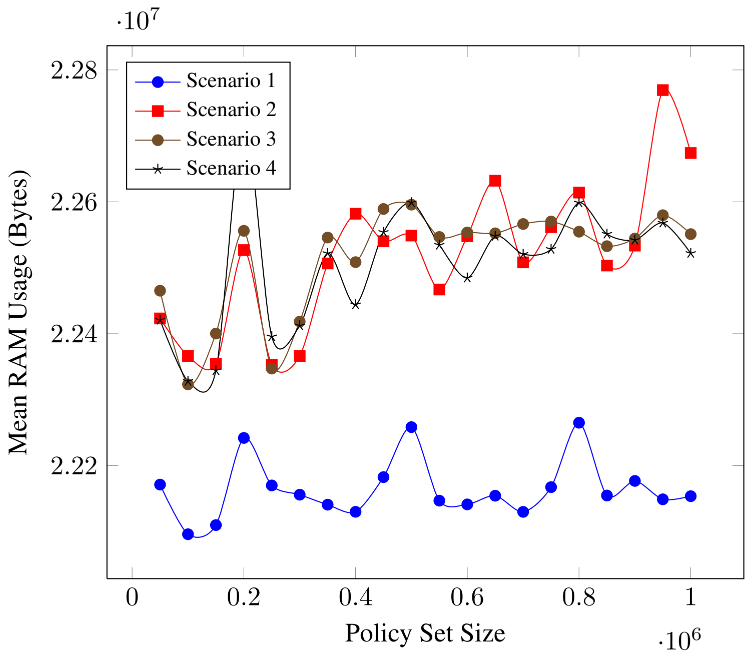


Figure 25: Graph of Mean RAM Usage (O-MRU) during the ontology verification stage against increasing policy set size.

the difference in average RAM usage between sequential and binary decision diagram (BDD) methods for anomaly detection. The average RAM usage value is defined here as the arithmetic mean of all RAM readings taken during the functional verification stage. Figures 26, 27, 28 and 29 offer a comparison of the average RAM usage using the sequential method, Seq-MRU, and the BDD method, BDD-MRU, against increasing rule set size for Scenarios 1, 2, 3 and 4, respectively. A clear indication from Figures 26, 27, 28 and 29 is that there is a linear relationship between the mean RAM usage and increasing policy set size, for both sequential and BDD-based functional verification methods. This holds true for all four evaluation scenarios. Further, all four scenarios also display a rate of increase for the sequential method that is much higher than that of the BDD-based method, for increasing policy set size. This implies that the BDD-based functional verification method is much more efficient, in terms of RAM usage per policy set size, than the sequential method. Another feature of the graphs in the figures mentioned is that, for all four scenarios, the initial RAM usage value for the sequential method is much lower than that of the BDD-based method. This is expected, since the BDD-based method requires an allocation of RAM for the initialisation of the BDD structure, while the sequential method does not. For sequential comparison, only the RAM amount required to load a policy set is allocated. With increasing sizes of policy sets, however, the BDD-based method requires progressively lower amounts of RAM allocation, per policy set size, compared to the sequential method. This is due to the specialised structure of the BDD decision tree which, once a policy set has been loaded into memory in the form of a binary string, carries out anomaly-detection logic functions more efficiently than sequential comparisons. For this reason, as policy set sizes increase, the sequential comparison method requires more RAM allocation than

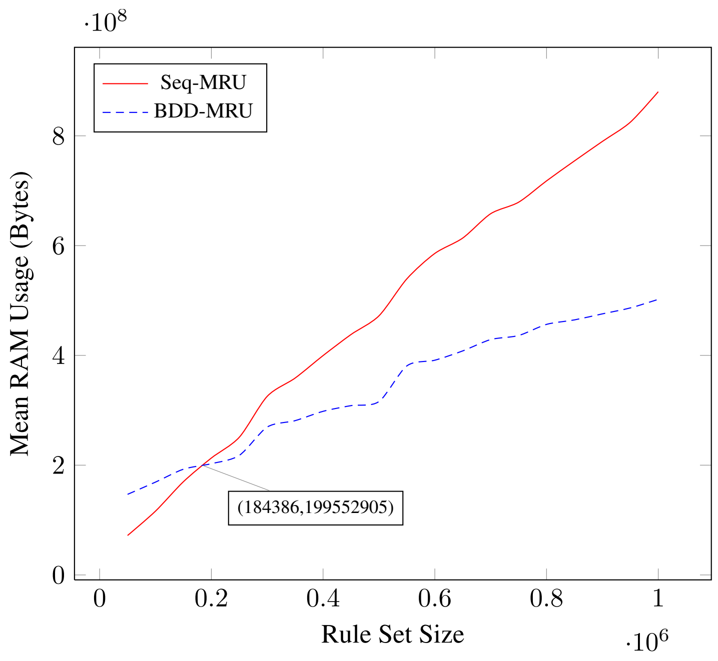


Figure 26: Scenario 1: Comparison of Sequential Mean RAM Usage (Seq-MRU) and BDD Mean RAM Usage (BDD-MRU) during the functional verification stage against increasing rule set size.

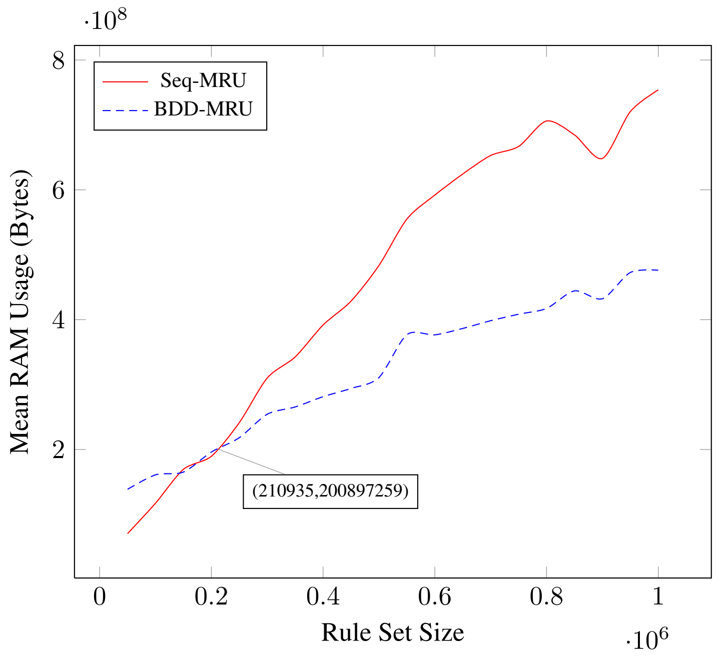


Figure 27: Scenario 2: Comparison of Sequential Mean RAM Usage (Seq-MRU) and BDD Mean RAM Usage (BDD-MRU) during the functional verification stage against increasing rule set size.

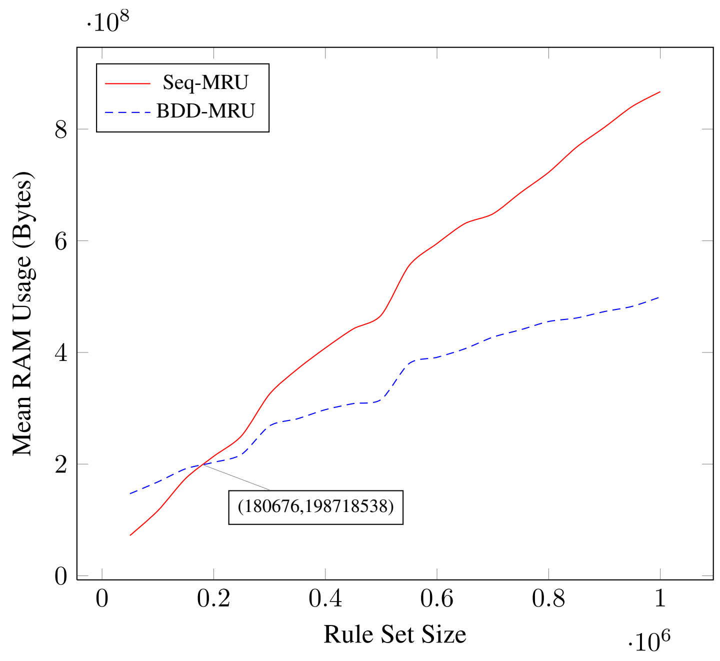


Figure 28: Scenario 3: Comparison of Sequential Mean RAM Usage (Seq-MRU) and BDD Mean RAM Usage (BDD-MRU) during the functional verification stage against increasing rule set size.

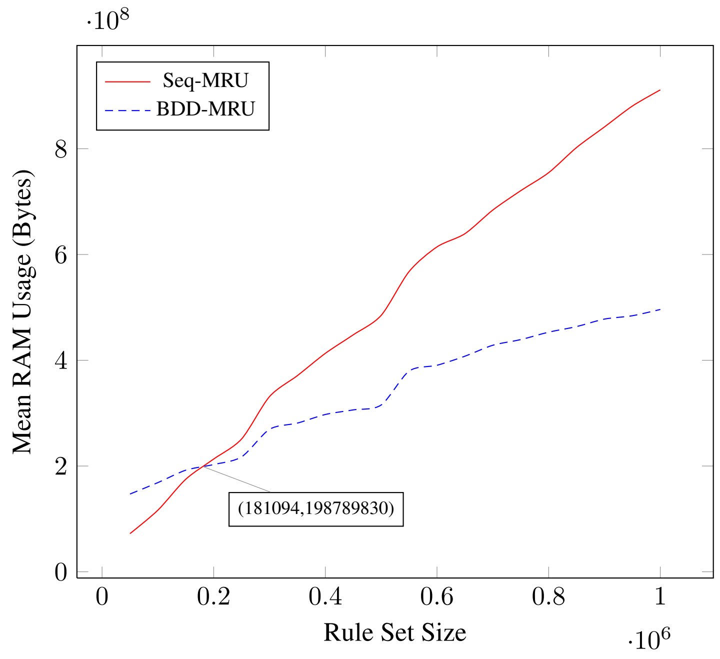


Figure 29: Scenario 4: Comparison of Sequential Mean RAM Usage (Seq-MRU) and BDD Mean RAM Usage (BDD-MRU) during the functional verification stage against increasing rule set size.

the BDD-based method for the same size of policy set. An interesting aspect of this trend is that, for each evaluation scenario, the sequential comparison method requires lower RAM allocation than the BDD-based method for certain policy set sizes. Table 11 lists the points at which the RAM usage for both methods is the same. The lowest RAM value where this occurs is 189.51MB for Scenario 3, while the highest is 191.59MB for Scenario 2. Interestingly, these points also form the boundaries for the range of policy set sizes where RAM usage is the same, which is 180676 policies for Scenario 3 and 210935 policies for Scenario 2. An important consequence of these

Table 11: Rule set sizes by scenario at which the sequential and BDD-based RAM usage values are equal.

Scenario	No. of policies in Policy Set Size	RAM Usage Value (Megabytes)
Scenario 1	184386	190.31
Scenario 2	210935	191.59
Scenario 3	180676	189.51
Scenario 4	181094	189.58

values is that, for policy set sizes of approximately 180000 policies and less, the sequential comparison method is more efficient than the BDD-based method, in terms of RAM usage per policy set size, for all four evaluation scenarios. This means that, in order to successfully complete the functional verification process, the sequential method requires a much lower allocation of RAM than the BDD-based method, as long as the policy set size is lower than approximately 180000 policies. For all policy set sizes larger than 180000 policies, however, the BDD-based method displays a lower RAM allocation requirement. Further, the difference in RAM allocation amounts between the sequential and BDD-based methods is linear. The linear relation here demonstrates the higher rate of RAM usage against policy set size for the sequential method compared against the BDD-based method. In terms of a ratio of policy set size against RAM usage amount, the BDD-based method, therefore, offers a more efficient application of available RAM than the sequential method.

5 Conclusion

This paper outlined an evaluation and discussion on the performance of the information sharing framework, based on an analysis of its performance during each of the four evaluation scenarios. As expected, it was evident from the analysis that increasing policy set sizes required an increasing amount of resources, especially in terms of random-access memory (RAM), and resulted in longer processing times. Although this trend was true for all four evaluation scenarios, it was evident that the binary decision diagram (BDD) based method performed better than the sequential method. This result was verified in the analysis of total processing times and percent processor utilisation, where the BDD-based method consistently resulted in faster times and lower utilisation than the sequential method. Further, the difference in total processing times between the sequential and BDD-based methods increased significantly with increasing policy set sizes, as was shown in Table 7. Although this trend was observed for all four evaluation scenarios, that is the BDD-based method had faster total processing times than the sequential method for all four scenarios, it was particularly pronounced for Scenario 2. While the difference in total processing times was 3435, 2875 and 2880 seconds for Scenarios 1, 3 and 4, respectively, it was 61352 seconds for Scenario 2. Further analysis of the RAM usage values over time for Scenario 2, indicate anomalous values for sets consisting of greater than 500,000 policies. This implies a high dependence of the framework on available resources, as limited RAM availability in Scenario 2 may have contributed to the anomalous values recorded for larger policy sets. The analysis also verified that the performance of the policy verification process is dependent on the functional verification stage more than the syntax or ontology verification stages, irrespective of available resources. As indicated in Table 8, for example, over 75 percent of the mean processing time is taken up by the functional verification stage. This is a crucial argument in favour of the BDD-based method. This is not only because the total processing times for the BDD-based method are lower than those for the sequential comparison method, but also because the rate of increase in processing time is significantly higher for the sequential comparison method than for the BDD-based method. This observation holds true for all four evaluation scenarios and, therefore, highlights the increasing advantage of the BDD-based relative to increasing policy set sizes. As discussed earlier in Section 2, there is a one-to-many relationship between high-level policies and their lower-level implementations. Due to the large number of legal jurisdictions and organisations involved in collaborative frameworks, a single information sharing policy may require a large number of abstractions and interpretations, resulting in large policy sets. Hence, it is likely that the advantage offered by the BDD method will be of vital advantage in this context.

References

[1] E. Al-Shaer and H. Hamed, "Modeling and management of firewall policies," *IEEE Transactions on Network and Service Management*, vol. 1-1, pp. 2–10, 2004.

[2] H. Hamed and E. Al-Shaer, "Taxonomy of conflicts in network security policies," *Communications Magazine, IEEE*, vol. 44, no. 3, pp. 134 – 141, 2006.

[3] W. M. Fitzgerald, F. Turkmen, S. N. Foley, and B. O'Sullivan, "Anomaly analysis for physical access control security configuration," in *Risk and Security of Internet and Systems (CRiSIS), 2012 7th International Conference on*. IEEE, 2012, pp. 1–8.

[4] Z. Chen, S. Guo, and R. Duan, "Research on the anomaly discovering algorithm of the packet filtering rule sets," in *Pervasive Computing Signal Processing and Applications (PCSPA), 2010 First International Conference on*, 2010, pp. 362–366.

[5] H. Hu, G.-J. Ahn, and K. Kulkarni, "Detecting and resolving firewall policy anomalies," *Dependable and Secure Computing, IEEE Transactions on*, vol. 9, no. 3, pp. 318–331, 2012.

[6] L. Fan, W. Buchanan, C. Thuemmler, O. Lo, A. Khedim, O. Uthmani, A. Lawson, and D. Bell, "Dacar platform for ehealth services cloud," *The 4th International Conference on Cloud Computing (IEEE Cloud '11)*, In Press.

[7] B. Fraser, J. P. Aronson, N. Brownlee, and F. Byrum, "Site Security Handbook (rfc 2196)," Online, September 1997. [Online]. Available: http://tools.ietf.org/html/rfc2196

[8] B. Schneier, *Secrets and lies : digital security in a networked world*. New York: John Wiley, 2000.

[9] W. R. Cheswick, S. M. Bellovin, and A. D. Rubin, *Firewalls and Internet Security*. Boston: Addison-Wesley, 2003.

[10] Y. Bhaiji, *CCIE Professional Development - Network Security Technologies and Solutions*. Cisco Press, 2008.

[11] D. Danchev, "Building and implementing a successful information security policy," online at ddanchev.blogspot.com, 2011.

[12] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models, rbac," *Computer*, vol. 29, no. 2, pp. 38–47, 1996.

[13] P. Samarati and S. C. de Vimercati, "Access control: Policies, models, and mechanisms," *Lecture Notes in Computer Science*, vol. 2171, pp. 137–196, 2000.

[14] R. Sandhu and P. Samarati, "Authentication, access control, and audit," *ACM Computing Survey*, vol. 28, no. 1, pp. 241–243, 1996.

[15] T. Corbitt, "Protect your computer system with a security policy," *Management Services*, vol. 46(5), pp. 20–21, 2002.

[16] S. Hinrichs, “Policy-based management: bridging the gap,” in *Computer Security Applications Conference, 1999. (ACSAC ’99) Proceedings. 15th Annual*, 1999, pp. 209 –218.

[17] M. Abrams and D. Bailey, “Abstraction and refinement of layered security policy,” in *Information Security: An Integrated Collection of Essays*. IEEE Computer Society Press, 1995, pp. 126–136.

[18] N. Damianou, N. Dulay, E. Lupu, and M. Sloman, “The ponder policy specification language,” in *Policies for Distributed Systems and Networks*, ser. Lecture Notes in Computer Science, M. Sloman, E. Lupu, and J. Lobo, Eds. Springer Berlin / Heidelberg, 2001, vol. 1995, pp. 18–38.

[19] “Integrated network management: Moving from bits to business value,” in *Tenth IFIP/IEEE International Symposium on Integrated Network Management (IM 2007)*, A. Keller, E. Al-Shaer, and H.-G. Hegering, Eds., Munich, Germany, May 2007.

[20] N. C. Damianou, A. K. Bandara, M. S. Sloman, and E. C. Lupu, “A survey of policy specification approaches,” Tech. Rep., 2002.

[21] F. Chen and R. S. Sandhu, “Constraints for role-based access control,” in *RBAC ’95: Proceedings of the first ACM Workshop on Role-based access control*. New York, NY, USA: ACM, 1996, p. 14.

[22] R. Hayton, J. Bacon, and K. Moody, “Access control in an open distributed environment,” in *IEEE Symposium on Security and Privacy*, May 1998, pp. 3–14.

[23] J. Guttman, “Filtering posture: Local enforcement for global policies,” in *Proc. IEEE Symp. Security Privacy*, 1997.

[24] J. D. Guttman, “Rigorous automated network security management,” *International Journal of Information Security*, vol. 4, pp. 29–48, 2005.

[25] ——, “Security goals: Packet trajectories and strand spaces,” *Lecture Notes in Computer Science*, pp. 197–261, 2001.

[26] R. Bryant, “Graph-based algorithms for boolean function manipulation,” *Computers, IEEE Transactions on*, vol. C-35, no. 8, pp. 677–691, August 1986.

[27] A. Boswell, “Specification and validation of a security policy model,” *Software Engineering, IEEE Transactions on*, vol. 21, no. 2, pp. 63–68, Feb 1995.

[28] J. Hoagland, R. Pandey, and K. N. Levitt, “Security policy specification using a graphical approach,” Computer Science Department, University of California Davis, Tech. Rep., 1998.

[29] D. E. Bell and L. J. LaPadula, “Secure computer system: Unified exposition and multics interpretation,” Electronic Systems Division, Air Force Systems Command, Hanscom AFB, Bedford, MA 01731, Tech. Rep., March 1976.

[30] Y. Bartal, A. Mayer, K. Nissim, and A. Wool, “Firmato: a novel firewall management toolkit,” in *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on*, 1999, pp. 17–31.

[31] A. Mayer, A. Wool, and E. Ziskind, "Fang: a firewall analysis engine," in *Security and Privacy, 2000. S P 2000. Proceedings. 2000 IEEE Symposium on*, 2000, pp. 177–187.

[32] A. Wool, "Architecting the lumeta firewall analyzer," in *Proceedings of the 10th conference on USENIX Security Symposium - Volume 10*, ser. SSYM'01. Berkeley, CA, USA: USENIX Association, 2001, pp. 7–7.

[33] A. Mayer, A. Wool, and E. Ziskind, "Offline firewall analysis," *International Journal of Information Security*, vol. 5, pp. 125–144, July 2006.

[34] P. Jaferian, D. Botta, F. Raja, K. Hawkey, and K. Beznosov, "Guidelines for designing it security management tools," in *CHiMiT '08: Proceedings of the 2nd ACM Symposium on Computer Human Interaction for Management of Information Technology*. New York, NY, USA: ACM, 2008, pp. 1–10.

[35] R. Marmorstein and P. Kearns, "A tool for automated iptables firewall analysis," in *Proceedings of the annual conference on USENIX Annual Technical Conference (ATEC)*. Berkeley, CA, USA: USENIX Association, 2005, pp. 44–44.

[36] Cisco Systems, "Cisco secure policy manager." [Online]. Available: http://www.cisco.com/warp/public/cc/pd/sqsw/tech/csmp_rg.pdf

[37] T. E. Uribe and S. Cheung, "Automatic analysis of firewall and network intrusion detection system configurations," in *Proceedings of the 2004 ACM workshop on Formal methods in security engineering*, ser. FMSE '04. New York, NY, USA: ACM, 2004, pp. 66–74.

[38] T. Uribe and S. Cheung, "Automatic analysis of firewall and network intrusion detection system configurations," *Journal of Computer Security*, vol. 15, no. 6, pp. 691–715, 2007.

[39] W. Enck, P. McDaniel, S. Sen, P. Sebos, S. Spoerel, A. Greenberg, S. Rao, and W. Aiello, "Configuration management at massive scale: system design and experience," in *2007 USENIX Annual Technical Conference*. Berkeley, CA, USA: USENIX Association, 2007, pp. 1–14.

[40] B. Zhang, E. Al-Shaer, R. Jagadeesan, J. Riely, and C. Pitcher, "Specifications of a high-level conflict-free firewall policy language for multi-domain networks," in *SACMAT '07: Proceedings of the 12th ACM symposium on Access control models and technologies*. New York, NY, USA: ACM, 2007, pp. 185–194.

[41] S. Pozo, R. Ceballos, and R. M. Gasca, "Model-based development of firewall rule sets: Diagnosing model inconsistencies," *Information and Software Technology*, vol. 51, no. 5, pp. 894–915, 2009.

[42] S. Pozo, A. J. Varela-Vaca, and R. M. Gasca, "Afpl2, an abstract language for firewall acls with nat support," *International Conference on Dependability*, pp. 52–59, 2009.

[43] T. L. Hinrichs, N. S. Gude, M. Casado, J. C. Mitchell, and S. Shenker, "Practical declarative network management," in *Proceedings of the 1st ACM workshop on Research on enterprise networking*, ser. WREN '09. New York, NY, USA: ACM, 2009, pp. 1–10.

[44] A. Bierman, K. Crozier, R. Enns, T. Goddard, E. Lear, P. Shafer, S. Waldbusser, and M. Wasserman, “Netconf configuration protocol (rfc 4741),” December 2006. [Online]. Available: http://tools.ietf.org/html/rfc4741

[45] S. Halle, R. Deca, O. Cherkaoui, R. Villemaire, and D. Puche, “A formal validation model for the netconf protocol,” *15th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM 2004)*, pp. 147–158, 2004.

[46] M. Choi, H. Choi, J. Hong, H. Ju, and S. POSTECH, “Xml-based configuration management for ip network devices,” *Communications Magazine, IEEE*, vol. 42, no. 7, pp. 84–91, 2004.

[47] Cisco Systems, “Network configuration protocol,” Jun 2009. [Online]. Available: http://www.cisco.com/en/US/docs/ios/netmgmt/configuration/guide/nm_cns_netconf.pdf

[48] Juniper Networks, “Netconf api guide,” 2008. [Online]. Available: http://www.juniper.net/techpubs/software/junos/junos91/netconf-guide/netconf-guide.pdf

[49] G. Munz, A. Antony, F. Dressler, and G. Carle, “Using netconf for configuring monitoring probes,” in *IEEE/IFIP Network Operations & Management Symposium (IEEE/IFIP NOMS 2006), Poster Session, Vancouver, Canada, Apr*, 2006.

[50] S. Hazelhurst, A. Fatti, and A. Henwood, “Binary decision diagram representations of firewall and router access lists,” University of the Witwatersrand, Johannesburg, South Africa, Tech. Rep. TR-Wits-CS-1998-3, 1998.

[51] T. Chomsiri, X. He, and P. Nanda, “Limitation of listed-rule firewall and the design of tree-rule firewall,” in *Proceedings of the 5th international conference on Internet and Distributed Computing Systems*, ser. IDCS’12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 275–287. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-34883-9_22

[52] S. Hazelhurst, A. Attar, and R. Sinnappan, “Algorithms for improving the dependability of firewall and filter rule lists,” in *Proceedings of the 2000 International Conference on Dependable Systems and Networks (formerly FTCS-30 and DCCA-8)*, ser. DSN ’00. Washington, DC, USA: IEEE Computer Society, 2000, pp. 576–585. [Online]. Available: http://portal.acm.org/citation.cfm?id=647881.737947

[53] R. Oppliger, *Internet and Intranet Security*, ser. Artech House Computer Security Series. Artech House, 2001.

[54] M. Gonçalves, *Firewalls: a complete guide*, ser. Standards and Protocols Series. McGraw-Hill, 2000, no. v. 1.

[55] L. Medina, *The Weakest Security Link Series*. iUniverse, 2003.

[56] A. Attar, “Performance characteristics of bdd-based packet filters,” University of the Witwatersrand, Johannesburg, South Africa, Tech. Rep. TR-Wits-CS-2002-2, 2002.

[57] D. B. Chapman, “Network (in)security through ip packet filtering,” in *Proceedings of the Third Usenix Unix Security Symposium*, Baltimore, MD, September 1992, pp. 63–76.

[58] P. Gupta and N. McKeown, “Packet classification on multiple fields,” *SIGCOMM Comput. Commun. Rev.*, vol. 29, no. 4, pp. 147–160, August 1999.

[59] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel, “Fast and scalable layer four switching,” *SIGCOMM Comput. Commun. Rev.*, vol. 28, no. 4, pp. 191–202, October 1998.

[60] T. V. Lakshman and D. Stiliadis, “High-speed policy-based packet forwarding using efficient multi-dimensional range matching,” *SIGCOMM Comput. Commun. Rev.*, vol. 28, no. 4, pp. 203–214, October 1998.

[61] G. Cheung and S. McCanne, “Dynamic memory model based framework for optimization of ip address lookup algorithms,” in *Proceedings of the Seventh Annual International Conference on Network Protocols*, ser. ICNP ’99. Washington, DC, USA: IEEE Computer Society, 1999, pp. 11–.

[62] J. Mogul, R. Rashid, and M. Accetta, “The packer filter: an efficient mechanism for user-level network code,” *SIGOPS Oper. Syst. Rev.*, vol. 21, no. 5, pp. 39–51, November 1987.

[63] D. Decasper, Z. Dittia, G. Parulkar, and B. Plattner, “Router plugins: a software architecture for next-generation routers,” *IEEE/ACM Trans. Netw.*, vol. 8, no. 1, pp. 2–15, February 2000.

[64] E. Fredkin, “Trie memory,” *Commun. ACM*, vol. 3, no. 9, pp. 490–499, Sep. 1960. [Online]. Available: http://doi.acm.org/10.1145/367390.367400

[65] E. Al-Shaer and H. Hamed, “Design and implementation of firewall policy advisor tools,” DePaul University, CTI Technical Report, Tech. Rep. CTI-TR-02-006, August 2002.

[66] ——, “Firewall policy advisor for anomaly discovery and rule editing,” in *Integrated Network Management, 2003. IFIP/IEEE Eighth International Symposium on*, 2003, pp. 17 – 30.

[67] ——, “Discovery of policy anomalies in distributed firewalls,” in *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, vol. 4, 2004, pp. 2605 – 2616 vol.4.

[68] E. Al-Shaer, W. Marrero, A. El-Atawy, and K. ElBadawi, “Towards global verification and analysis of network access control configuration,” DePaul University, Chicago, IL, USA, Tech. Rep., 2008.

[69] L. Yuan, H. Chen, J. Mai, C.-N. Chuah, Z. Su, and P. Mohapatra, “Fireman: a toolkit for firewall modeling and analysis,” in *Security and Privacy, 2006 IEEE Symposium on*, May 2006, pp. 15 pp. –213.

[70] A. Wool, “A quantitative study of firewall configuration errors,” *Computer*, vol. 37, no. 6, pp. 62–67, june 2004.

[71] D. Caldwell, A. Gilbert, J. Gottlieb, A. Greenberg, G. Hjalmtysson, and J. Rexford, “The cutting edge of ip router configuration,” in *Proceedings of 2nd ACM Workshop on Hot Topics in Networks (Hotnets-II)*, 2003.

[72] M. Zhang, R. Ranjan, S. Nepal, M. Menzel, and A. Haller, “A declarative recommender system for cloud infrastructure services selection,” in *Proceedings of the 9th international conference on Economics of Grids, Clouds, Systems,*

and Services, ser. GECON'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 102–113. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-35194-5_8

[73] D. Caldwell, S. Lee, and Y. Mandelbaum, "Learning to talk cisco ios: Inferring the ios command language from router configuration data," AT&T, Tech. Rep., 2007.

[74] E. Al-Shaer, H. Hamed, R. Boutaba, and M. Hasan, "Conflict classification and analysis of distributed firewall policies," *Selected Areas in Communications, IEEE Journal on*, vol. 23, no. 10, pp. 2069 – 2084, 2005.

[75] E. Al-Shaer and H. Hamed, "Management and translation of filtering security policies," in *2003 IEEE International Conference on Communications*. IEEE Press, 2003.

[76] A. Tongaonkar, N. Inamdar, and R. Sekar, "Inferring higher level policies from firewall rules," in *Proceedings of the 21st conference on Large Installation System Administration Conference*. Berkeley, CA, USA: USENIX Association, 2007, pp. 2:1–2:10. [Online]. Available: http://portal.acm.org/citation.cfm?id=1349426.1349428

[77] M. Bishop and S. Peisert, "Your security policy is what," University of California at Davis, Tech. Rep., 2006.

[78] K. Golnabi, R. Min, L. Khan, and E. Al-Shaer, "Analysis of firewall policy rules using data mining techniques," in *Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP*, april 2006, pp. 305 –315.

[79] M. Abedin, S. Nessa, L. Khan, E. Al-Shaer, and M. Awad, "Analysis of firewall policy rules using traffic mining techniques," *International Journal of Internet Protocol Technology*, vol. 5, pp. 3–22, April 2010.

[80] S. Hazelhurst, "Algorithms for analysing firewall and router access lists," University of the Witwatersrand, Johannesburg, South Africa, Tech. Rep. TR-WitsCS -1999-5, 2000.