Auto-configuration of Critical Network Infrastructure

Dinesha Ranathunga

April 15, 2017

Thesis submitted for the degree of
Doctor of Philosophy
in
Applied Mathematics
at The University of Adelaide
Faculty of Engineering, Computer and Mathematical Sciences
School of Mathematical Sciences

THE UNIVERSITY
of ADELAIDE

Contents

List of Tables

List of Figures

Signed Statement

I certify that this work contains no material which has been accepted for the award of any other degree or diploma in my name, in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text. In addition, I certify that no part of this work will, in the future, be used in a submission in my name, for any other degree or diploma in any university or other tertiary institution without the prior approval of the University of Adelaide and where applicable, any partner institution responsible for the joint-award of this degree.

I give consent to this copy of my thesis, when deposited in the University Library, being made available for loan and photocopying, subject to the provisions of the Copyright Act 1968.

I also give permission for the digital version of my thesis to be made available on the web, via the Universitys digital research repository, the Library Search and also through web search engines, unless permission has been granted by the University to restrict access for a period of time.

Signed: ..... Date: .....

Acknowledgements

Many thanks to my supervisors Prof. Matthew Roughan, Mr. Phil Kernick and Dr. Nick Falkner for their continuous support of my Ph.D, for their patience, motivation and vast knowledge. They are examples of the kind of researcher I can hope to be one day.

My gratitude goes out to the Australian Government, the Australian Research Council, the University of Adelaide and CQR Consulting for their financial support of this Ph.D. I hope the fruits of our research will prove beneficial in funding similar research ventures in the future.

Thanks to my colleagues and my collaborators whom I've enjoyed working with immensely. I hope we can continue working together in the future.

I would also like to thank my family: my wife and my parents for their great moral support throughout my Ph.D and life in general.

Last but not least, thanks to my beloved daughter Dilini, for simply being you.

Abstract

Until the turn of the millennia, many electricity, water and gas supply plant operators used analogue serial cabling to communicate between highly customised systems to control and manage their plants. Since then, cost reductions and increased flexibility have become possible through the use of COTS (Commodity-Off-The-Shelf) equipment. These have radically changed communication between critical infrastructure devices, but have also introduced risks to the domain; one example being the major incident at a German steel mill in 2014 [14]. Donna F. Dodson, Chief of CyberSecurity at NIST has stated that "There's an increase in free tools available focusing on industrial control systems. And greater hacker interest."

A common strategy to mitigate these risks is the extensive use of firewalls. Firewalls are not as simple as they appear. Efficient and reliable firewall security requires expertise in arcane, vendor-dependent configuration languages [15] and even then, configuration errors are common [97, 128, 129]. It is easy to complain about short-term thinking in firewall designers, but, at a deeper level the problem is that current approaches conflate multiple concerns: *i.e.,* they incorporate network, protocol and hardware dependent details into security policy, in an unsystematised manner. In this thesis we tackle this problem.

We begin by building a mathematically rigorous foundation for the design of security policies that separates divergent concerns. The formal foundations allow security administrators to reason about their network security; for instance to (i) show that certain types of traffic flows are impossible; and (ii) compare their security to industry best practices to check it complies and so on.

In particular, we design our policy framework with Supervisory Control And Data Acquisition (SCADA) networks in mind; these networks control the distributed assets of many critical infrastructure plants. In doing so, we consider the requirements of a security policy specification that are general in nature as well as specific to a SCADA network context. An example requirement is verifiability: a property that increases transparency in the framework and provides security administrators assurance of expected security outcome. Lack of verifiability in current firewall configuration platforms contribute to the *broken-by-design* networks found in practice. Moreover, we apply design principles derived from real

SCADA case studies [97] and industry best-practices [21, 117], to develop simple policy specification features that are easy to administer correctly.

We demonstrate the use of these specification features through a prototype implementation that creates *secure-by-design* networks. In enabling security by design, we (i) prevent policy emergence: *i.e.,* implicit definition of policy as a result of many small decisions with complex interactions; and (ii) support rigorous verification: from policy consistency and best-practice compliance checks to pre-deployment verification, we only allow deploying policies that deliver the expected security outcome; and (iii) protect proactively: security can't be purely reactive, placing pre-verified security controls prior to a cyber attack can prevent significant, expensive damage to system infrastructure.

Chapter 1

Introduction

A network facilitating the control of a power station requires a degree of reliability orders of magnitude higher than an enterprise IT network. Momentary communication outages have negligible effect on most enterprise networks, but they can cause potentially life threatening problems in a power plant. It is not surprising therefore, that Supervisory Control and Data Acquisition (SCADA) networks, which control the distributed assets of many critical infrastructure plants, are designed with high-availability and safety in mind.

High-availability in these networks is needed to control delay-sensitive industrial processes such as electricity generation reliably without disruption. Scheduled SCADA-network outages must be planned in advance and rebooting network devices is often impractical because it disrupts the continuous industrial process.

The lack of built-in security in SCADA networks was not historically considered a design compromise because these networks were intended to be air-gapped from all other networks. Connectivity to insecure enterprise networks and the Internet was not considered in their design. But air-gaps are impractical given the need to reduce costs and improve efficiency [117] (Byres calls the idea a myth to emphasise how poor a solution it is [20]).

SCADA networks are in practice becoming increasingly interconnected and their lack of built-in security exposes them to a multitude of external cyber threats. From thrill seeking script-kiddies to organised and well-funded cyber-criminals, various attackers continue to attempt to weaken or sabotage critical infrastructure. Recent examples include the hacking of traffic signs to display hoax messages in San Francisco [23], the hacking of a German steel mill that destroyed its blast furnace [14], the phishing attack which targeted the CEO of an Austrian Aerospace manufacturer (FACC) [100] and caused financial losses of over USD 50 million, the attack and compromise of key energy and control system related websites by the DragonFly group [123] and Stuxnet: an alleged state sponsored attack that damaged Iran's Uranium enrichment plant [41].

Firewalls, programmable network filters, play a key role in protecting the data and resources in a SCADA network from unauthorised access. They do so by filtering out unwanted traffic [27, 117]. As Rubin and Greer note below [103], it is therefore, vital that these firewalls are configured correctly.

> "The single most important factor of your firewall's security is how you configure it."
>
> *Rubin and Greer [103]*

Misconfiguration of firewalls in a SCADA network can lead to security breaches that cause significant physical and environmental damage, financial loss or worse, the loss of human lives. Even so, past surveys on firewall configurations have found multiple, serious errors to be a common occurrence [128, 129]. But these past studies were on enterprise networks. We conducted our own studies on the firewalls protecting critical systems and found similar problems [97].

Complex, proprietary configuration platforms offered by firewall vendors are a key contributor to these firewall misconfigurations [128]. These platforms require security policies to be specified manually, box-by-box, using large volumes of network minutiae such as IP addresses. Doing so makes the configuration process tedious and error prone. Moreover, policies can also interact; both within a firewall and between firewalls, which can produce adverse effects. Lack of built-in verifiability in existing firewall configuration platforms means SCADA owners must rely on external cyber-security audits to detect flaws in their firewall policies (an audit that may occur once a year if that often).

To complicate matters, existing firewall configuration platforms lack conceptual integrity: *i.e.,* coherent design principles and relationships. The shortfall has produced complex and uncoordinated firewall features that are difficult to administer correctly.

Network installations are also commonly built from the *bottom-up*, *i.e.,* a network-device is purchased, and configurations are then written. A firewall policy is the result of the configuration, which is the consequence of a purchasing decision. So, the policy is implicitly defined as a result of many small decisions that interact in complex ways. Hence, the quote below from Bertrand Russell, is salient.

> "The advantages of implicit definition over construction are roughly those of theft over honest toil."
>
> *Bertrand Russell*

In essence, the current manual approach to firewall configuration is *broken-by-design*; it employs incoherent firewall features that cannot even verify if a specified policy is error free. Moreover, policies cannot be specified abstractly; their design is closely guided by network implementation details.

A sensible way to overcome these configuration complexities is to automatically derive firewall configurations from high-level policies that are free of network intricacies. Auto-configuration, in general, succeeds in a number of ways for improving network reliability and reducing management costs. It reduces some of the repetitive (yet critical) tasks in management, and ensures that 'fat fingering' does not cause unnecessary errors.

Auto-configuration approaches using SDN have been proposed [71, 115], but they remain unrealistic for SCADA networks, which are often composed of legacy equipment. SCADA network administrators cannot simply replace an entire legacy security installation in practice. They need tools to support incremental changes in security as power plants are insecure now [97]. SCADA networks need a solution that works using off-the-shelf technology. In this thesis, we propose such a solution: *ForestFirewalls*[1].

Our system enables firewall policies to be described abstractly and in detail. So, a non-IT specialist could specify firewall policies with ease. Unlike current firewall configuration platforms, *ForestFirewalls* promotes modularity; *i.e.,* it caters for both non-technical users (*e.g.,* business managers) and technical users (network engineers). Non-technical users need to define high-level policies to meet their business goals. Technical users may need to add support for new protocols. Through modularity, *ForestFirewalls* also simplifies network maintenance and support.

But developing such a specification isn't easy and there are many pre-requisites to making such a specification meaningful. For one, we need to develop appropriate security abstractions to remove network intricacies from high-level policy. For another, we must enforce coherent concepts and relationships to produce pragmatic specification features that are easily administrable.

Moreover, a reliable firewall specification requires strong mathematical underpinnings. The formal foundations allow security administrators to reason about their network security with rigour; for instance to (i) show that certain types of traffic flows are impossible; and (ii) compare their security to industry best practices to check it complies and so on. *ForestFirewalls* is built on formal foundations to support such rigorous verification and create SCADA networks that are *secure-by-design*.

1.1 Chapter summary

Chapter 2 is Background and related work. It explains the problem addressed in this thesis in more detail and summarises the relevant literature.

[1]The name derives from the analogy of not being able to see the forest for the trees.

Chapter 3 presents a series of SCADA firewall configuration case studies. These lead to a set of abstractions that are key to describing firewall policies at a high-level, independent of network intricacies. In particular, we evaluate the usefulness of an abstraction introduced by the American National Standards Institute (ANSI)/International Society for Automation (ISA), to mitigate threats in control networks [59]. We find that this abstraction is too flexible and lacks key aspects required for automation. The extensions we propose to address these missing pieces produce a refined standard that can describe real SCADA firewall policies at a high level. The case studies also allow us to understand in depth, the problems of current firewall configuration in a SCADA network context.

Chapter 4 presents the mathematical modelling of the abstract policies developed in Chapter 3. The models allow us to derive the formal properties and constraints that a pragmatic firewall policy specification must satisfy. However, there are many obstacles to developing such formal semantics. For one, the lack of standards for firewall policy languages makes policy semantics rule-order and firewall-implementation dependent, hindering their meaningful comparison. We address these challenges, and propose algebraic frameworks that allow (i) direct comparisons of firewall policies; and (ii) policy to be mapped to network-firewalls provably correctly.

Chapter 5 then presents the design of our high-level firewall policy specification language: *ForestFirewalls*. We overcome the current complexities of firewall configuration by adhering to coherent design principles and key security-policy language requirements. One such requirement is verifiability: a property that is fundamental to making the policy framework transparent and providing security administrators assurance of expected security outcome. The design concepts and relationships we enforce promote conceptual integrity. The property leads to a simple policy specification that is easy to administer correctly.

Chapter 6 presents the implementation of *ForestFirewalls*. We describe here in detail the syntax and semantics of our high-level policy language and how policy is refined to the device level. We also outline the implementation of our algebras developed in Chapter 4 and the rigorous verification framework designed in Chapter 5.

Chapter 7 presents our application of *ForestFirewalls* to the real SCADA case studies in Chapter 3. By doing so, we demonstrate that our system significantly reduces the firewall configuration effort required and improves the baseline security policy deployed in a SCADA network. In addition, the automated policy refinement built into our system ensures that only efficient and concise firewall configurations are deployable.

Chapter 8 presents a framework for firewall reporting. These reports (*e.g.,* logs, traps and alarms) allow to continuously monitor firewall security, post deployment. Such a framework is necessary since existing standards for reporting and analysing firewall data are scant and vague. In particular, we lack clear direction on what firewalls should report.

Chapter 9 is the Conclusion.

1.2 Publication list

Parts of this thesis have been published

1. **Dinesha Ranathunga**, Matthew Roughan, Hung Nguyen, Phil Kernick and Nick Falkner, "Case Studies of SCADA Firewall Configurations and the Implications for Best Practices", accepted to IEEE Transactions on Network and Service Management, early access link *http://ieeexplore.ieee.org/document/7529047/*, 2016.

2. **Dinesha Ranathunga**, Matthew Roughan, Phil Kernick and Nick Falkner, "Malachite: Firewall policy comparison", IEEE Symposium on Computers and Communication (ISCC), pp. 310-317, June 2016, Messina, Italy.

3. **Dinesha Ranathunga**, Matthew Roughan, Phil Kernick and Nick Falkner, "The Mathematical Foundations for Mapping Policies to Network Devices", Proceedings of the 13th International Joint Conference on e-Business and Telecommunications, pp. 197-206, July 2016, Lisbon, Portugal.

4. **Dinesha Ranathunga**, Matthew Roughan, Phil Kernick, Nick Falkner, Hung Nguyen, Marian Mihailescu and Michelle McClintock, "Verifiable Policy-defined Networking for Security Management", Proceedings of the 13th International Joint Conference on e-Business and Telecommunications, pp. 344-351, July 2016, Lisbon, Portugal.

5. **Dinesha Ranathunga**, Matthew Roughan, Phil Kernick and Nick Falkner, "Towards standardising firewall reporting", 1st Workshop on the Security of Cyber Physical Systems (WOS-CPS), Springer LNCS, September 2015, Vienna, Austria.

6. **Dinesha Ranathunga**, Matthew Roughan, Phil Kernick, Nick Falkner and Hung Nguyen, "Identifying the missing aspects of the ANSI/ISA best practices for security policy", Proceedings of the 1st ACM Workshop on Cyber-Physical System Security (CPSS), pp. 37-48, April 2015, Singapore.

Chapter 2

Background and Related Work

Supervisory Control and Data Acquisition (SCADA) networks control the distributed assets of many critical infrastructures including power generation, water distribution, factory automation, sewage management and airport control systems.

SCADA networks differ to enterprise networks in that they are designed primarily for robustness and real-time performance. For instance, SCADA networks often incorporate devices such as Programmable Logic Controllers (PLCs) that control physical devices such as gas valves. PLCs must be robust to withstand the harsh environmental conditions they usually operator under. PLCs also need to perform in real-time to allow SCADA operators to closely monitor the critical process (*e.g.,* electricity generation) and control it.

At the same time, PLCs often have highly constrained memory and computational power but lack the sophisticated security functionality required to protect them from cyber attacks. This lack of built-in security features in SCADA devices means plant owners need to invest significant resources to improve the security of their critical networks.

A network controlling a power station understandably, requires a degree of reliability orders of magnitude higher than an enterprise IT network. Momentary outages have little effect on most enterprise networks, but, they can cause potentially life threatening problems in a power plant. To say the least, such disruptions are likely to cause significant monetary losses.

Thus, the security priorities in SCADA networks differ fundamentally to those in enterprise networks. We elaborate on these priority discrepancies below, but ultimately they set the scene for the work of this thesis.

2.1 Security priorities: SCADA vs Enterprise

Protecting Confidentiality (*i.e.,* privacy of data and commands sent over the network), is usually given the highest priority in enterprise networks because a breach may lead to regulatory noncompliance [55, 117]. Availability (*i.e.,* ability to send

Table 2.1: SCADA vs Enterprise security priorities, adapted from [117].

Priority	SCADA	Enterprise
High	Availability	Confidentiality
Medium	Integrity	Integrity
Low	Confidentiality	Availability

data and commands without disruption over the network), is usually given the lowest priority in these networks. Thus, network devices are often rebooted when troubleshooting problems and downtimes are often acceptable in these networks.

SCADA networks have these security priorities in reverse order as shown in Table 2.1. Availability has highest priority in these networks; the ability to control sensitive industrial processes such as electricity generation with reliability in real time is paramount [55, 117]. Hence, any outages need to be planned days or weeks in advance. Rebooting a network device is often not an option in SCADA networks as it can disrupt the continuous SCADA process.

Confidentiality is not the primary concern in SCADA networks which involve real-time systems. Once a control command has been issued and executed, it becomes relatively meaningless [55, 117].

Integrity defines the ability to send data and commands intact without unauthorised modifications. Integrity is rated as medium priority, relative to the other priorities in both enterprise and SCADA networks, but, is paramount for SCADA networks. A breach in people or process integrity can have a minor impact or be tolerable in an IT environment but be catastrophic in a SCADA environment.

2.1.1 Threat sources

SCADA networks are primarily designed for reliability and safety with minimum consideration for security [117]. This was not seen as a design compromise because historically these networks were expected to be air-gapped from insecure networks such as enterprise networks and the Internet. But air-gaps are impractical given the need to reduce costs and improve efficiency [19, 117] and Byres calls the idea a myth [20] to emphasise how poor a solution it is.

SCADA networks in practice, are becoming increasingly interconnected and hence exposed to a multitude of external threats. Typical threat sources include remote attackers (*i.e.,* thrill seekers that formulate attacks using online scripts and tools) [117], criminal groups (*i.e.,* organised, well funded crime groups that attack systems and conduct industrial espionage for monetary gain) and terrorists (*i.e.,* individuals and groups seeking to weaken or destroy critical infrastructure to threaten national security and cause casualties).

There are also many types of cyber attacks aimed at SCADA networks. For one, Denial of Service (DoS) attacks aim to prevent legitimate users from accessing a SCADA-network service or system. Examples of DoS attacks include SYN flooding and HTTP flooding [74]. A common counter-measure against a SYN flood is the enabling of SYN cookies in a perimeter firewall [39]. HTTP flooding can be mitigated through the use of a reverse proxy which is a gatekeeper that retrieves resources from its associated servers on behalf of a client, thereby reducing the possibility of the network being overwhelmed.

Another type of attack aimed at SCADA networks is Phishing which involves the fraudulent attempt via emails to steal identities, information, money or install malware [117]. A well known recent example is the email-based Phishing attack that targeted the CEO of an Austrian aerospace manufacturer (FACC) which resulted in the company losing over USD 50 million [100].

Phishing accounted for 17% of the incidents reported to ICS-CERT in 2015 [58]. Phishing databases are useful in detecting and mitigating these attacks through URL and email-content analysis. Encryption and two-factor authentication do not help mitigate Phishing attacks: victims are often logged into their email accounts already and will simply download the attack to their machines via the encrypted connection. Often, firewalls have been unable to block phishing emails effectively, but future versions [90] are seeking to bridge this gap. Industry best-practice guidelines recommend blocking inbound SMTP traffic to SCADA networks to prevent Phishing attacks at the outset.

Spyware-based attacks such as Zlob trojan [122] and Zango [119] attempt to capture and relay information about systems or individuals without consent.

Malware-based attacks such as the Duqu 2.0 worm [121] and the DRIDEX trojan [120] aim to disrupt system operation, gain access or control to computers without consent.

Both Spyware and Malware can propagate in to a SCADA network from infected servers in less-trusted networks. Basic forms of these worms and viruses may be detected and blocked using firewalls in general, but, sophisticated instances such as the Stuxnet worm that can still bypass many firewalls [41]. Extreme measures have therefore been taken in some cases to prevent cyber attacks and the spread of malware. For instance, since June 2016, the Singaporean government has commenced disabling Internet connectivity in public-service workstations [13] in an initiative to protect government networks from cyber threats.

Hack attacks are attempts by individuals to exploit vulnerabilities in network systems to accomplish their goals [117]. A recent example of this common type of attack is the hacking of a German steel mill [14] which resulted in its blast furnace being destroyed.

2.1.2 Security incidents

The Repository of Industrial Security Incidents (RISI) database records global security incidents that arise from cyber attacks on SCADA networks [117]. Analysis of the RISI data attributes 50% of the incidents to accidents [19], originating from negligent employees and third party contractors.

An example incident is the disabling of the Safety Parameter Display System and Plant Process Computers at Davis-Besse nuclear power plant due to a SQL Slammer infection [117]. The infection occurred as a result of a machine left unpatched in the critical network for over six months. Another example is the shutdown of operations of 13 Daimler Chrysler's manufacturing plants in the U.S due to a Zotob worm infection. Again, the infection was facilitated by systems that had been left unpatched for a week.

The RISI database analysis also reveals targeted (external and internal) attacks to be least frequent with only 20% of the incidents attributed to them [19]. But, these incidents are usually the *most damaging* as they often leverage detailed knowledge of SCADA systems. For instance, Vitek Boden: a disgruntled ex-employee, hacked into the computerised sewage system of Maroochy Shire Council in 2000 and released tonnes of raw sewage into public parklands and river systems [117]. Boden destroyed wild life and caused severe environmental damage. Another incident is the Stuxnet worm that targeted Siemens industrial software and equipment and damaged Iran's Uranium enrichment plant in 2010. In 2012, the network of an auto manufacturer was hacked and employee login details were stolen [110]. Between 2013 and 2014, a group of cyber attackers known as the DragonFly group attacked and compromised key energy and control system related websites [123]. They also targeted and compromised legitimate software packages made available online by ICS equipment manufacturers. Also in 2014, traffic signs on a street in San Francisco were hacked and left displaying "Godzilla Attack! Turn Back" [23]. In the same year, a German steel mill was hacked by cyber attackers, destroying its blast furnace [14].

We described earlier, how an air gap was never a feasible approach to protect SCADA networks from cyber attacks. The only viable protection today is a firewall, or series of firewalls [21, 117] and so, we describe firewalls in detail next.

2.2 Firewalls

Firewalls are a standard mechanism for enforcing network security [21, 27, 117, 133]. They protect data and resources in a communications network from unauthorised access by filtering out unwanted network traffic.

Traditionally, firewalls are placed between an organisation and the outside

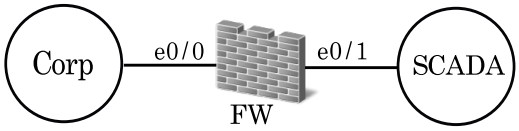


Figure 2.1: Example isolation of SCADA and Enterprise domains using a firewall.

world. But large organisations may require internal firewalls to isolate their security domains [133]. A security domain is a grouping of network devices under common administrative control. SCADA and Enterprise are two example security domains.

By isolating the security domains (*e.g.,* as in Figure 2.1), traffic from authorised users is also required to pass firewall inspection when crossing between domains. Otherwise, for instance, if an enterprise machine is compromised, the SCADA network could also be compromised with ease. Firewalls control this access and trust using filtering decisions that are based on a set of ordered rules defined to meet the security policy requirements of an organisation.

2.2.1 Evolution of firewalls

To illustrate the types of filtering available, we present an overview of the evolution of firewalls.

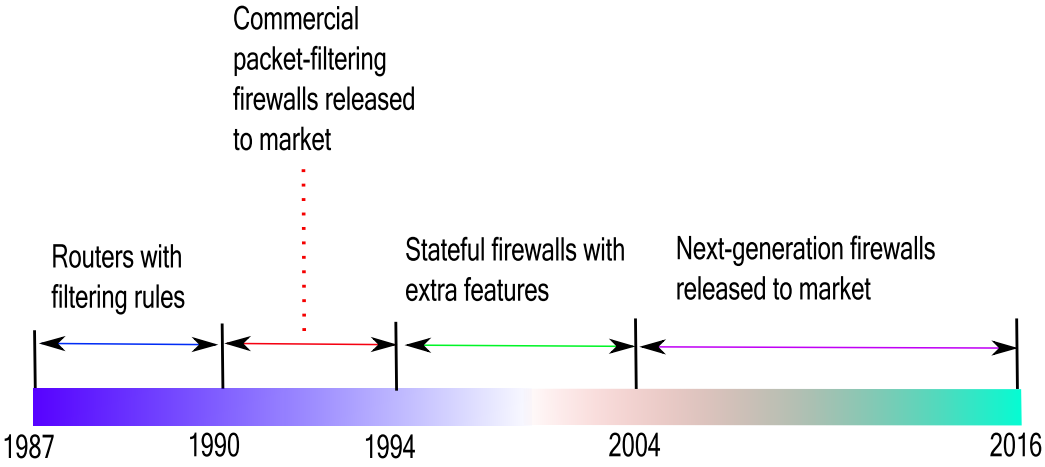


Figure 2.2: A firewall evolution timeline from the 1980s to date, adapted from [1].

As Figure 2.2 shows, the first firewalls used were in fact, routers with filtering rules in the late 1980s (we describe filtering rules below). These routers helped segment LANs and the filtering rules allowed to inspect incoming and outgoing traffic from each LAN segment.

Network-security companies soon realised the value of using dedicated firewalls to separate networks and filter traffic between them. As a result, the first commercial firewall was developed by Digital Equipment Corporation (DEC) in 1991 [1]. The device was a *packet-filtering (or static-filtering) firewall:* the simplest category of firewalls that employ a set of pre-defined rules to act upon individual traffic packets [133]. Rules are stored in Access Control Lists (ACLs) and the packet filtering criteria employed often involves the five tuple: source IP address, destination IP address, source port, destination port and protocol. We refer to these rules as filtering rules.

The simple packet-inspection criteria makes this type of traffic filtering low cost and very efficient [133]. The high cost of computing during this era, seriously limited the amount of complex processing conductible in a network firewall. Yet, static filtering was simple enough to allow these firewalls to be deployed in networks as external-facing perimeter firewalls (or commonly known as Internet-facing firewalls), for rapid filtering of malicious packets in large volumes of traffic.

Packet-filters however, cannot recognise packet-series relationships [133]. For instance, a DNS response is only valid if it matches a prior query. But, these simple firewalls cannot distinguish a 'spoofed' DNS response from a legitimate one. Hackers could leverage this weakness to attack and compromise systems behind the firewall.

Check Point addressed the shortfall in 1994 by launching the first *stateful firewall:* Firewall-1. Stateful firewalls extend packet-filtering firewalls and can also dynamically-track packet-series relationships. To do so, these firewalls record the history of accepted packets and current connection states to only allow *anticipated* traffic [133]. By blocking spoof attacks, these firewalls offer a higher level of security but are more expensive and vulnerable to a new class of attacks, *e.g.,* the SYN DoS attack.

Cost of computing began to reduce significantly in the mid 1990s, increasing the amount of processing conductible on a network firewall. So, between 1994 and 2004, firewall manufacturers began to incorporate additional features such as Network Address Translation (NAT), Virtual Private Network (VPN), Quality of Service (QoS) and Intrusion Prevention Systems (IPS) on their firewall offerings [1]. These hybrid firewalls dominated the market in this era.

During this period, large-scale cyber attacks on Web servers that include the PHF CGI exploit in 1996 [83], also led to the research and development of new security models to protect Web applications. Consequently, the first *Web Application Firewall (WAF)* known as Appshield was developed by Perfecto Software [1]. A WAF inspects Web – HTTP and HTTPS – traffic at the application layer to remove malicious code and objects embedded within [133].

The idea was generalised to Deep-Packet Inspection (DPI) firewalls. DPI firewalls are highly sophisticated firewalls that allow deep filtering into the applica-

tion layer, offering more granular security, but they incur a high CPU and memory cost. In addition to analysing the IP packet headers, these firewalls also inspect the packet payload using signature or regular expression matching [133]. Performing searches through the payload at wire speed for string patterns is computationally expensive and the number of searches that can be carried out is limited by the processing capabilities of the firewall hardware.

In fact, varying computational complexities exist for the different firewall traffic filtering classes (*e.g.,* static filtering has time complexity $O(log\ N)$ [102] while stateful filtering has time complexity $O(N^{O(log\ log\ N)})$ [45] where N is the ruleset length). Higher complexities can be mitigated by housing more computing resources: *e.g.,* powerful processors and denser memory, in a firewall. Progressively decreasing computing costs have made this accommodation pragmatic. But complex traffic filtering such as DPI can still reduce network performance significantly.

The concept of Next Generation Firewalls (NGFWs) was introduced by Gartner in 2003 combining the capabilities of traditional firewalls, Intrusion Prevention Systems (IPS) and deep-packet inspection [1]. A NGFW has the ability to understand the application-layer traffic traversing through the firewall in detail, allowing to mitigate attacks by blocking traffic that might exploit application protocol vulnerabilities [90].

NGFWs are capable of detecting application-layer threats with high precision, wide coverage and low error rates [90]. Commercial NGFWs were first rolled-out into the market by Palo Alto networks in 2008 [1]. Several other vendors including SonicWALL, Cisco, Huawei and Juniper, soon followed.

In contrast to using disparate technologies for IDS, IPS and firewalling, NGFWs don't leave gaps in their protection offering for attackers to exploit. The consolidation of these technologies into one system also reduces the operating costs and administrative complexity required.

So far, we have described how firewall technology has evolved and the capabilities they offer today. It is also worth understanding the security objectives of a firewall in a SCADA-network context and so we discuss these objectives next.

2.2.2 Overall security objectives of a SCADA firewall

The primary objective of a firewall protecting a SCADA subnet, is to eliminate direct connectivity between the *SCADA network* and the Internet [21, 117]. The Internet is the major source of threats. Direct connectivity to the greater Internet would encourage unsolicited inbound traffic to the SCADA network that could at the least congest it and block critical commands from reaching control devices such as PLCs.

Secondary objectives of a SCADA firewall include the physical and logical separation of SCADA traffic from enterprise traffic [21]. Enterprise networks commonly connect directly to the Internet, hence this separation of traffic reduces the exposure of a SCADA network to Internet-based attacks.

In addition, a SCADA firewall must restrict direct access from an enterprise network to the SCADA network and disallow control of SCADA devices from the enterprise network. Otherwise, a compromised host in the enterprise network could be leveraged to launch an attack on the SCADA network.

A SCADA firewall also helps facilitate remote support of SCADA systems (often performed by third-party vendors). Maintenance and troubleshooting of SCADA systems via remote access mean the connection used must be secured (*e.g.,* encrypted) and authorised (*e.g.,* using two-factor authentication) [21, 117].

A SCADA firewall also allows to restrict firewall management to a reserved set of stations over a secure connection [21]. This is of highest importance because if the management of the firewall is compromised, the whole system is compromised.

2.2.3 Firewall configuration

In order for a firewall to deliver the expected security outcome, it must be *configured* with a policy that meets the business and security goals of an organisation, prior to deployment. The components that constitute a firewall configuration is often diverse [24, 30, 64], for instance, they include interface configurations such as IP address and subnet masks, static routes that describe subnets and the gateways they can be reached from, ACLs (or rulesets) and implicit rules that enable firewall management services over and above ACLs.

Two example firewall configurations are shown in Listing 2.1 and Listing 2.2. The configuration in Listing 2.1 is from a Cisco ASA 5505 firewall and the configuration in Listing 2.2 is from a Linux IP-Tables firewall. For ease of comparison, the same security policy is implemented on both firewalls.

The configurations show how the firewall interfaces have been configured. For instance, the Ethernet0/0 interface of the Cisco ASA firewall points to the subnet 10.0.1/24 (Listing 2.1). The Cisco ASA also allows to assign a user-defined name to its interfaces (*e.g.,* Ethernet0/0 is named `corp`). These names enable easy reference of the interfaces within the configuration.

Both configurations have access-control rules defined to permit traffic flow through the firewalls. The Cisco ASA uses extended ACLs to hold these rules [30]. These ACLs are then applied to the firewall interfaces using the Cisco command 'access-group' to achieve the desired outcome (*e.g.,* access-list `scada_in` is defined and applied inbound to the `scada` firewall interface in Listing 2.1).

Listing 2.1: A Cisco ASA 5505 firewall configuration snippet from [30].

```
1  !
2  interface Ethernet0/0
3   speed 100
4   duplex full
5   nameif corp
6   security-level 60
7   ip address 10.0.1.1 255.255.255.0
8   ospf cost 10
9  !
10 interface Ethernet0/1
11  nameif scada
12  security-level 100
13  ip address 10.0.2.1 255.255.255.0
14  ospf cost 10
15 !
16
17 access-list scada_in remark Allow ping
18 access-list scada_in extended permit icmp any any
19 access-list scada_in remark email
20 access-list scada_in extended permit tcp host 10.0.2.4 host 10.0.1.7 eq
       smtp
21 access-list scada_in extended deny ip any any
22
23 access-list corp_in remark enable HTTPS
24 access-list corp_in extended permit tcp 10.0.1.1 255.255.255.0 host
       10.0.2.8 eq https
25 access-list corp_in extended deny ip any any
26
27 access-group scada_in in interface scada
28 access-group corp_in in interface corp
```

In contrast, the IP-Tables firewall uses 3 separate IP chains: INPUT, OUTPUT and FORWARD, to hold the access-control rules [93]. These chains allow or deny incoming, outgoing and pass-through traffic respectively.

A Cisco ASA extended ACL includes an explicit default rule: *deny ip any any*, to block all unwanted traffic. For instance, the ACL `corp_in` defined in Listing 2.1 permits HTTPS traffic from subnet 10.0.1/24 to host 10.0.2.8 and uses this default rule to block all other traffic. However, to achieve the same effect, the IP-Tables firewall employs 3 default rules (lines 26-28 in Listing 2.2).

By inspection, we can see that both firewall configurations contain a significant volume of minute detail such as IP addresses and port numbers. Each of these must be configured correctly to achieve the expected security outcome.

The configurations also indicate the presence of vendor-specific intricacies, for

Listing 2.2: A Linux IP-Tables firewall configuration snippet from [93].

```
1  .# interface configuration
2  DEVICE=eth0
3  BOOTPROTO=static
4  BROADCAST=10.0.1.255
5  HWADDR=00:13:72:3E:55:72
6  IPADDR=10.0.1.1
7  NETMASK=255.255.255.0
8  NETWORK=10.0.1.0
9  ONBOOT=yes
10 TYPE=Ethernet
11
12 DEVICE=eth1
13 BOOTPROTO=static
14 BROADCAST=10.0.2.255
15 HWADDR=EC:35:86:54:0E:B2
16 IPADDR=10.0.2.1
17 NETMASK=255.255.255.0
18 NETWORK=10.0.2.0
19 ONBOOT=yes
20 TYPE=Ethernet
21
22 .# rule configuration
23 -A FORWARD -i eth0 -p tcp -s 10.0.1.1/24 -d 10.0.2.8 --dport 443 -m
       state --state NEW,ESTABLISHED -j ACCEPT
24 -A FORWARD -i eth1 -p icmp --icmp-type 8 -j ACCEPT
25 -A FORWARD -i eth1 -p tcp -s 10.0.2.4 -d 10.0.1.7 --dport 25 -m state
       --state NEW,ESTABLISHED -j ACCEPT
26 -P INPUT DROP
27 -P OUTPUT DROP
28 -P FORWARD DROP
```

instance, the Cisco ASA allows assigning *security-levels* to firewall interfaces to define the level of trust bestowed on the network connected to that interface [30]. A security-level of 100 assigned to an interface defines the subnet attached to that interface as *highest trust* while a level of 0 assigned defines the subnet as *lowest trust*. In the absence of an ACL assigned to such an interface, certain traffic flows are permitted by default from an interface with a high security level to one with a lower security level. But, the feature is not available in IP-Tables firewalls.

We can also see how corresponding configuration components such as interface configurations and ACLs have distinct syntax across the two vendors. Configuring these require the use of proprietary and device specific platforms. Being fluent in one platform does not guarantee fluency in the other. Firewall administrators therefore, require specialised training on each platform, which can be expensive.

As the above configuration snippets depict, there is a lack of common standards or languages for firewall policy specification. This shortfall contributes to flawed firewall policies often being deployed to production networks [96, 128]. In the next section, we describe the common types of firewall errors and their causes.

2.2.4 Firewall errors

Surveys carried out in the past on firewall configuration of major corporations revealed multiple, serious errors present in each [97, 128, 129]. A serious firewall error (or misconfiguration) significantly increases the vulnerability of a network to cyber attack.

Wool studied 74 real enterprise firewall configurations [128] and found 80% of the firewalls were badly configured. Over 70% of the configurations he studied allowed access to the firewall over insecure (*i.e.,* unencrypted and poorly authenticated) protocols such as Telnet. Over 60% allowed management of the firewall via machines outside the perimeter of the network, without employing a VPN to securely access the firewall from the enterprise network. These are grave errors because if the management of the firewall is compromised, the whole system is compromised.

He also found that over 60% of the configurations included rules that were too broad and admitted more services than necessary. An example is a rule that permits all TCP traffic between two hosts [128]. Almost half of the configurations also allowed potentially unsafe protocols like DNS and FTP in imprudently. Thus, traffic pertaining to these protocols were not destined to legitimate servers.

Wool further identified that over 50% of the configurations allowed insecure NetBIOS protocols through the firewalls [128] These protocols are used by Microsoft Windows operating systems to facilitate file and printer sharing but are often leveraged to carry out attacks on networks.

The firewall errors discussed above can create security holes that allow malicious traffic to enter a protected network or block legitimate traffic and disrupt operations. The errors can have devastating consequences, in particular in a SCADA network. And there have been major incidents such as the hacking of traffic signs in San Francisco, as a result.

2.2.5 Causes of misconfiguration

A key contributor to the firewall errors found in practice is the large amount of minute details involved in the current approach to firewall configuration. We illustrated earlier, using example configurations from a Cisco ASA firewall and an IP-Tables firewall, how they often include network-centric IP addresses, port numbers *etc*. All of this low-level detail need to be configured precisely. Otherwise,

the enforced security will be different from that intended or at worse it could open a potentially dangerous security hole.

Firewall rulesets can also be lengthy, spanning hundreds of rules [96, 128] and need to be configured by hand using proprietary and device specific platforms. We observed the variation in the configuration syntax and semantics across the firewall vendors earlier. Firewall administrators require specialised training in each platform to be fluent in them.

To add to these complexities, rules within an ACL can also interact. Referred to as intra-ACL inconsistencies, these are caused by different rules having overlapping packet matching criteria. Depending on the extent of rule overlap, an intra-ACL inconsistency can be a redundancy or a conflict [4]. Moreover, because ACLs are often evaluated using a *first-match criteria* [30, 64, 93], these interactions mean that the rule order matters. So, even if the correct rule is entered, placing it in the wrong location in the ruleset, can render the wrong outcome. We will describe intra-ACL inconsistencies in detail in Subsection 3.1.6.

The presence of network-centric intricacies in firewall policies imply a close coupling between policy and the network. But networks change often occur in response to new business needs, upgrades and service demands. So, current firewall policies must also change as often with the network. This inability to maintain a low firewall policy complexity, relative to the network, further contributes to the firewall misconfigurations.

With the current approach to firewall configuration, administrators also cannot construct a single high-level firewall policy (*i.e.,* source of truth) for a network. For instance, the potentially heterogeneous firewalls in a network engage vendor-specific policy syntax and semantics which hinder maintaining a single concise ruleset. Thus, administrators and security managers alike cannot reliably look for who gets in and who doesn't [56].

In SCADA networks, firewall misconfigurations may also stem due to cultural differences; IT specialists usually configure the SCADA firewall(s), but lack the knowledge of industrial engineering requirements to do so correctly. Lack of co-operation between the IT and control teams can cause these specialists to configure the SCADA firewall(s) with an IT mindset. The resulting firewall configurations are unlikely to be compliant with SCADA best practices.

An important outcome of Wool's study of enterprise firewall configurations (discussed earlier) is that the use of later versions of firewall software did not correlate to fewer errors [128]. Newer software versions were unable to assist users to write more accurate firewall rulesets. What firewall vendors have concentrated on is introducing new and impressive features to create systems with as much or more complexity as the base firewall configurations.

The problem is exacerbated by the lack of built-in tools to automatically detect inconsistencies within firewall rulesets. These rulesets represent the policy of a firewall, so we shall next survey the tools and methods available for firewall-policy verification.

2.2.6 Firewall-policy verification

We stated earlier how inconsistencies such as conflicts and redundancies can occur in a firewall policy. Conflicts indicate an *ill-defined* policy and redundancies imply an *inefficient* one. Various tools have therefore, been proposed to conduct *bottom-up* analysis of firewall ACLs (or rulesets) to assist with *debugging* and *troubleshooting* [6, 44, 77, 111, 127, 131].

Lumeta [127] and Fang [77] are firewall-analysis tools that allow users to run queries against ACL rules to check firewall-configuration behaviour. Algosec Firewall Analyzer is a commercial closed-source tool based on the Lumeta and Fang engines. It allows administration of a large number of firewalls (up to a thousand) [6]. FireMon [44] and Skybox Firewall Assurance [111] are two other commercial firewall-maintenance tools that have audit capabilities. Skybox also automates firewall-rule lifecycle management which helps keep firewall rulesets clean and optimised. Some non-commercial tools [4, 46] have also focused on maintaining concise firewall configurations by detecting and removing ACL redundancies and conflicts.

The ability to debug policy errors is highly desirable in a firewall configuration platform. But, this debugging should support abstract queries, free of network-centric intricacies. The related works above analyse ACLs which contain network and vendor intricacies. So, the queries they support must change with ACLs even when the *policy intent* remains unchanged.

Another approach [76] detects policy anomalies by grouping hosts whose packets are treated identically by the firewall into equivalence classes. Doing so, suggests hosts in distinct equivalence classes have different policies, when in reality, they may not. For instance, hosts in the same security zone, in actuality, have identical policies in the absence of firewalls separating them.

A tool has also been developed to help check consistency between the policies and implementations [16] built on the Role-Based Access Control (RBAC) abstraction and the implementation encodes details of the physical system and traffic control mechanisms. Rysavy *et al.* [104] have also suggested checking ACLs against a security policy using Satisfiability Modulo Theory (SMT) verification. The policies are specified using the Security Policy Specification Language (SPSL) which operate at a low IP flow-level. The checking is performed on all possible flows, identified by source and destination IP addresses and port numbers. A better alternative is to check consistency between high-level versions of the specified and implemented firewall policies.

Checking a policy for inconsistencies includes checking its syntactic correctness. A syntactically correct policy must also be semantically correct for it to be *well designed*. For instance, it may be necessary to check a security policy's semantics against industry-recommended practices: *e.g.,* ANSI/ISA- 62443-1-1, for compliance. Security best-practice compliance is critical in SCADA networks where more restrictive practices are required to prevent catastrophic outcomes including harm or serious injury of people, or even death!

But current *bottom-up* approaches used to check policy correctness allow policy differences to occur even when *policy intent* remains unchanged. And best-practice compliance checks are only meaningful if we compare policy intent not implementation. Otherwise comparing a SCADA security policy with a generic ISA-recommended policy is impossible when either is inter-twined with implementation details.

A better and less tedious alternative is a *top-down* approach to check for policy inconsistencies. To do so, a single network-wide policy needs to be maintained, so that security administrators can easily determine who gets in and who doesn't [56]. These network-wide policies need to be free of minutiae, so changes to the policy's intent can be distinguished from changes to the network with clarity. Current top-down solutions allow creation of network-wide policies [9,12,49] and comprehensive checking of their syntactic correctness, but lack ability to determine policy semantic correctness.

Current firewall policy verification approaches do not leverage network-wide high-level firewall policies. The added complexity due to the presence of network and vendor intricacies hinders verification of high-level goals such as *no HTTP traffic allowed inbound to SCADA network*.

These verification approaches also often overlook checking that a firewall policy is compatible with the underlying network and its technology. This can lead to an artificial sense of security. For instance, the target network topology may be different to that perceived by a firewall-policy creator. The creator may consider two zones to be distinct in the policy specification, but in the absence of one or more firewalls enforcing a real separation between the zones, only a single zone may actually exist.

Current verification approaches also do not verify the expected security outcome of a firewall policy, prior to deployment. This capability is essential for security administrators to locate policy oversights, such as when firewall management is disabled, and be confident that their policies only allow the expected traffic services through their firewalls.

2.3 Network auto-configuration

We have described so far, how the current manual and device-centric approach to firewall configuration result in the serious errors found in the majority of the firewalls studied. This configuration approach is analogous to the design of circuit boards in the 1980s, a slow and error-prone manual process that required the operator to be a trained expert. But, the process has since been revolutionised through automation and circuit boards are now designed accurately and reliably by software. Similarly, to overcome the complexities of current firewall configuration, a sensible choice is to automate.

Auto-configurators in general, succeed in a number of critical tasks for improving network reliability and reducing management costs. They reduce some of the repetitive (yet critical) tasks in management, and ensure that 'fat fingering' does not cause unnecessary errors. Auto-configuration has been successfully used in many aspects of networking including ACL-rule placement [118], network-address assignment [37], routing [82, 99] and emulated-network generation [87].

The basic idea behind network auto-configuration is to use a Network Inventory Database (NIDB) and a set of rules (*i.e.,* policy) to generate device configurations that are then pushed into the devices of a working network [15]. Building a system to achieve this functionality, however, requires tackling several challenges.

For instance, how should we describe policy in network auto-configuration? We could use formal methods involving language design or a simple template-based approach [15]. With the language design approach, current practices resemble using Assembler. Thus, we need a high-level policy language and a compiler to create network configurations from more easily written code. But, taking the route of language design would not solve real problems quickly because often it requires operators (who are perennially busy, and pre-occupied with firefighting) to learn an entirely new language.

On the other hand, using templates is a more practical short-term solution. Templates represent parameterised configurations and the parameter values are assigned at run time from the NIDB [15]. This approach seems to have been more successful (to date) in real deployments. For instance, Chen *et al.* stated in 2007 [26] that 'The state of the art in network configuration management tends to be template-driven and device-centric.' But, the device-centric (*i.e.,* bottom-up) nature of template-driven design prohibits associating high-level goals with specific templates. A language based approach here would present stronger long-term potential.

Ideally, we need to strike a balance between the two imperatives: *i.e.,* adopt a hybrid that uses language at the high-level, but a template-driven implementation of policies to provide flexible and extensible, multi-vendor implementations. By doing so, we can obtain a long-term solution to high-level abstraction of network

policies, and a short-term solution that helps operators without requiring them to learn an entirely new language. This approach is highly desirable for developing a firewall auto-configuration system that can be fast deployed to SCADA networks.

Given the need for a firewall-policy language at the high-level, we describe the current state of these languages next.

2.4 Firewall-policy languages

Business and security goals are often described using natural language. These goals are often ambiguous [51, 73] and require human-translation to a machine interpretable technical specification. Thus, policy languages have been developed for various disciplines; some to suit specific domains like network- or security-management and others for general purpose: *e.g.,* Ponder [125]. Languages for network management such as Merlin and NetKAT help allocate resources such as bandwidth [9, 115]. Security-management languages include firewall-policy languages such as Firmato that assist with access-control of network traffic to enforce security [12].

Firewall-policy languages can be device-specific (*i.e.,* low-level) or abstract policy from the network (*i.e.,* high-level). Low-level languages such as the Cisco CLI are not user friendly: administrators cannot specify high-level goals such as *no single point of failure*, using them. These languages are often tightly coupled with the underlying network [30]. Using these low-level languages is similar to programming with Assembler in the 1970s, where programmers need to specify, for instance, hardware addresses in order to communicate with devices. Errors are a common occurrence in Assembler programs [40].

High-level firewall-policy languages such as Firmato, Filtering Postures, FLIP or Cisco Secure Policy Manager try to overcome the shortfall, by capturing policy intent, which allows to specify abstract goals [12,49,54,132]. The high-level policies can then be mapped to device-level filtering rules. High-level languages are analogous to present-day programming languages used by software developers. They support features such as modularisation and other constructs that automate common tasks. These languages also hand off many subtasks to the operating system or their interpreter (for instance memory management) and avoid wasted programming time in repetitive and error-prone tasks.

High-level policy languages may try to abstract policy from the network, but often, intricacies such as hostnames are still required. For example, Firmato [12] decouples security policies from network firewalls and increases language modularity and reusability. But, in order to refine the policy down to device-level filtering rules, input of host IP addresses in-policy is still required.

Cisco has also introduced security policy management tools (*e.g.,* VNMC for

VSG policy management) to cater for complexities introduced in network virtualisation [34]. For scalability, the tools allow operating systems (or VMs) to be allocated to zones and policies to be defined per zone. But, each VM still needs to be defined using low-level terms like hostnames.

SDN programming languages have been proposed [71, 92, 115] to derive network configurations from high-level policy, but they remain relatively unrealistic for SCADA networks, where TCP is still a recent innovation. And recent case studies [97] show that power plants are highly exposed to cyber attack. SCADA networks need a solution that works now, using off-the-shelf technology.

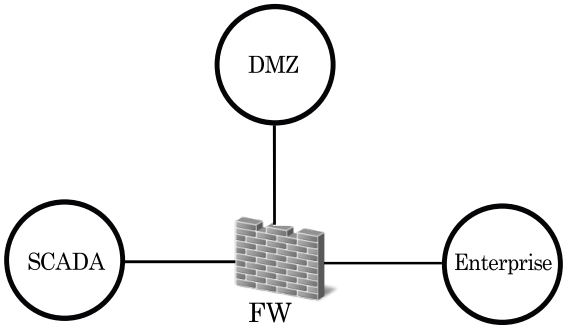
Current firewall-policy languages do not propose a high-level description that intuitively decouples policy from topology. Often, the network topology needs to be explicitly mapped to policy through the specification per host/subnet basis [12]. Doing so, hinders description of firewall policies using high-level requirements independent of vendor and network intricacies.

These policy languages also lack built-in automated verification capabilities. So, for instance, it is difficult to validate a specified policy is compatible with the underlying network and its technology. Existing firewall-policy languages also do not check the expected security outcome prior to deployment. These important features are required in a policy specification language to allow firewall configuration to become a commodity skill rather than a specialisation. The features would allow business managers and plant engineers alike to manage their SCADA firewalls. In this thesis we propose such a solution: *ForestFirewalls*.

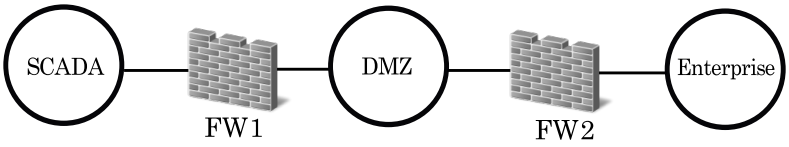
Recommended Firewall architectures: The best practices also recommend firewall architectures that employ three-zone systems for secure segregation of SCADA networks [21, 117]. An example architecture is a firewall with Demilitarised Zones (DMZs) which uses a three or more port firewall (Figure 2.3(a)) to enable a perimeter network (or DMZ) between SCADA and enterprise networks. Shared servers in the likes of data historians are located in the DMZ. The presence of a DMZ prohibits direct communication between the SCADA network and the enterprise network, but regular patching and/or updating of the shared servers is still required to reduce their vulnerability to cyber attack.

Another recommended architecture is paired-firewalls (*i.e.,* two serial firewalls) with a DMZ between them (Figure 2.3(b)) [21]. One firewall blocks arbitrary traffic inbound to the SCADA network while the other firewall blocks harmful traffic from a compromised server in the DMZ.

In recommending these architectures, management complexity and scalability are also considered, supplementary to their security capabilities. Both the above firewall designs offer high security and scalability and have moderate management complexity [21]. We will see in Chapter 3 that these two architectures are commonly deployed in practice to secure SCADA networks as a result.



(a) Single firewall with a DMZ that hosts shared enterprise or SCADA servers (*e.g.,* Historian).



(b) Paired firewalls with a DMZ that hosts shared servers.

Figure 2.3: Example best-practice firewall architectures adapted from [21].

Firewall-policy best practices: General firewall-policy best practices for SCADA networks include protocol, traffic-flow and source/destination recommendations [21, 117]. Recommended protocols are restricted to routable protocols (*e.g.,* TCP, UDP) for obvious reasons. Use of 'disjoint' protocols are also encouraged, so, protocols allowed in the SCADA network should be disallowed in the enterprise network and vice versa [21]. Doing so, reduces the risk of malware propagating into the SCADA network by requiring it to deploy multiple exploits over different protocols in order to succeed.

The best practices also prohibit direct traffic flow between the SCADA network and the Internet. The guidelines further recommend terminating traffic in the DMZs and preventing direct traffic flow between enterprise and SCADA networks [21]. Doing so reduces the risk of a successful cyber attack on the SCADA network from the less-secure enterprise network.

The source/destination recommendations in the best practices state that all 'permit' rules need to be IP address or port specific [21]. Adhering to them, prevents the use of inherently broad rules that apply for instance, to 'any' source or destination address.

These best-practice guidelines help define the allowed traffic services in an industrial control environment with clarity and accuracy. However, they do not distinguish high-level policy from low-level firewall rules. For example the recom-

Table 2.2: SCADA guidelines for service specific rule-sets from [21].

Service	Inbound to *SCADA*	Outbound from *SCADA*
DNS	Block all requests	Allow requests to DMZ on a case by case basis
HTTP	Block all connections (use HTTPS instead), ideally block all HTTP traffic but if critical configure HTTP proxy on firewall to block scripts and Java applications	Use HTTPS instead or HTTP proxy to screen traffic
FTP	Block all traffic unless secured with encrypted tunnel and two-factor authentication	Allow all traffic
Telnet	Block commands from enterprise network unless over secure tunnel	Allow traffic over encrypted tunnel only
SMTP	Block all traffic	Allow all traffic
SNMP	Block all traffic unless over (separate) secure management network	Block all traffic unless over secure network
DCOM	Allow traffic only from DMZ	Allow traffic only to DMZ
Industrial protocols	Allow traffic only from DMZ	Allow traffic only to DMZ

mendation: 'all permits rules need to be IP address/Port specific', applies within a firewall implementation context but not in a high-level policy context. A clear separation would have otherwise outlined unambiguous goals for both management level policy makers and network engineers.

A summary of service-specific firewall policy best practices adapted from [21], are shown in Table 2.2. These guidelines are however, specific to a simple security topology involving three zones: SCADA, Enterprise and DMZ. In practice many others can exist.

For example, the best practices specify disabling HTTP inbound to the SCADA zone because the protocol is known to carry worms and attacks across networks. If absolutely necessary, they recommend using HTTPS as a secure alternative.

Similarly, inbound and outbound traffic restrictions apply to many protocols including DNS and FTP. DNS queries should not be allowed to cross the SCADA firewall as the protocol is unsafe and is commonly used to carry out Distributed DoS (DDoS) attacks [66]. Industry therefore, recommends the use of a local DNS or host files instead of an external DNS server in the SCADA zone [21].

FTP is another unsecured protocol that sends all session data including user login information, un-encrypted across the network [8]. FTP-based communica-

tion is therefore, vulnerable to address-spoofing attacks that can be leveraged by hackers to steal sensitive files from an internal server.

The cyber-security standards above also suggest several security-policy abstractions to describe firewall policies. These abstractions provide the foundations for a high-level firewall policy description for auto-configuration. Hence, we discuss these abstractions in detail next.

2.4.1 Security abstractions

Role Based Access Control (RBAC) is a security abstraction commonly used in industry to decouple policy from the underlying enforcement mechanism. The abstraction allows assigning permissions to users via roles. A role summarises users that will share a job function [43].

But, with firewall policies our aim is to assign permissions (*i.e.,* services) to IP addresses (*i.e.,* hosts) in a network consisting of heterogenous devices. Therefore, some extensions are required to apply the RBAC abstraction to network security. For instance, assigning a role to a host is insufficient, permission must be given to a pair of hosts (*i.e.,* source and destination) instead [12].

The RBAC model is also user-centric and does not consider, for instance, the relationship between the user (traffic source) and the resource (traffic destination). The Attribute Based Access Control (ABAC) abstraction aims to address this shortfall by using attributes (or data fields) associated with a user, resource or action to help determine if some user can access a particular resource in a given way [57]. But as with RBAC, extensions are required to apply the generic ABAC model usefully to network security. For instance, firewalls often employ a first-match strategy to resolve rule-conflicts [30, 64, 93]. So, an ABAC-based policy language would need to support such conflict-resolution mechanisms.

A key drawback with both RBAC and ABAC abstractions are, they *do not intuitively map* policy to network topology. So, a firewall-policy language built on either abstraction would require intricacies like IP addresses to be input through the policy specification when implementing policy on a network instance [12, 124]. Explicit specification of such minutiae in-policy prohibits concise description of policies and increases policy length. For instance, XACML is an XML based security-policy language built on the ABAC model [81]. Even simple XACML access-control policies can span hundreds of lines of code or more [51].

RBAC and ABAC models may appear to offer more flexibility but the in-policy network intricacies and the syntax and semantics of the underlying policy structure used (*e.g.,* XML) often makes these policies human in-comprehensible [78]. An example is XACML where policy authors find it difficult to know with precision, if the firewall policy blocks potentially-unsafe HTTP traffic from entering a protected SCADA Zone by simple inspection of the policy.

In a useful high-level policy specification, the underlying abstraction must intuitively decouple policy from the network implementation, separating the *what* from the *how*. We describe an abstraction suitable to achieving this goal in detail below.

Zone-Conduit model

Lack of internal network segmentation is a significant contributor to the quick spread of security threats and attacks in SCADA networks [19, 59, 117]. The ANSI/ISA standards introduce the concepts of *zones* and *conduits* as a way of segmenting and isolating the various sub-systems in a control system [59]. The *Zone-Conduit model* is a very useful starting point for a high-level description of security policy, and so we describe it in detail here.

A *zone* is a logical or physical grouping of an organisation's systems with similar security requirements based on criticality and consequence [59]. By grouping systems in this manner, a *single security policy* can be defined for all members of a zone. For example, three security zones can be defined to accommodate low, medium and high-risk systems, with each device assigned to its respective zone based on their security level needed. A low-risk system can be accommodated within a medium or high security zone without compromising security, but not vice versa.

A *single zone-policy* implies that selected subsystems in a zone, such as a server, should not have their own separate policies (*i.e.,* no exceptions). Allowing *exceptions* would impart a false sense of security to those systems. These systems are only as secure as the zone itself, in the absence of any firewalls enforcing a real separation [59]. The restriction also implies that the resulting security policies should be independent of network intricacies such as IP addresses and hostnames.

A *conduit* provides the secure communication path between two zones, enforcing the policy between them [59]. Security mitigation mechanisms (*e.g.,* firewalls) are implemented within a conduit. A conduit could consist of multiple links and firewalls, but is logically a single connector. Conduits abstract *how* a policy is enforced from *what* is enforced. So, the resulting security policies are also independent of vendor and device intricacies.

The Zone-Conduit model decouples policy from topology in an intuitive manner, and allows a user to specify policies at a zone-level (*i.e.,* high-level), making them concise and easily human comprehensible. Figure 2.4 shows two typical zones in a SCADA network; the *SCADA-Zone* and the *Corporate-Zone*, linked by a conduit.

Zone and conduit concepts are intended as a platform for high-level security policy description. Before using these concepts in policy specification for firewalls, it is best to evaluate their usefulness. In particular, how well they cater for

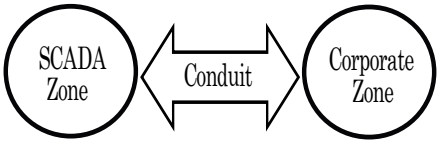


Figure 2.4: Example Zone-Conduit model, adapted from [19].

security architectures used in practice in real networks (*e.g.,* back-to-back firewalls). So, in the next chapter we will employ case studies of real-world SCADA networks to identify any missing pieces in the standard that hinder achieving this objective. We can then propose solutions to overcome the shortfalls and ensure the refined standard can meet policy specification requirements found in practice.

Chapter 3

Case Studies of SCADA Firewall Configurations

In the previous chapter, we described how a high-level security policy description is the starting point for firewall auto-configuration. The ANSI/ISA Zone-Conduit concepts provide a basis for developing such a policy description, but, before using these concepts in automation, it is prudent to determine how suitable they are for describing real-world SCADA firewall policies. In this chapter, we describe our approach to evaluating the usefulness of these concepts in the high-level specification of firewall policies through a series of real SCADA case studies.

An end goal of ours is to build a *top-down* firewall policy description for SCADA firewalls. A pre-requisite to achieving this goal is to employ a *bottom-up* approach that parses real SCADA firewall configurations to identify the missing pieces of the Zone-Conduit model. These case studies also help audit the firewall configurations and evaluate

- how well real-world SCADA firewalls are configured, *e.g.,* are they compliant with industry best practices in [21, 117]?
- how efficient real-world SCADA firewall configurations are, *e.g.,* do they exclude redundant or obsolete rules?

Related past works have conducted *bottom-up* analysis of firewall rulesets (*i.e.,* ACLs) to assist with *debugging* and *troubleshooting* [4, 7, 25, 46, 76, 77, 127]. For instance, Chen *et al.* have proposed an automated method to detect firewall misconfigurations [25]. The solution first translates ACLs into desired filtering actions for a set of test packets. These test packets are then used to detect and diagnose misconfigurations.

Another approach [76] detects policy anomalies by grouping hosts whose packets are treated identically by the firewall into equivalence classes. Doing so, can show hosts in distinct equivalence classes having different policies, when in

reality, they should not. For instance, hosts in the same zone, in actuality, have the same policy in the absence of firewalls separating them. We address this shortfall by identifying the disjoint zones in the network (*i.e.,* zones separated by firewalls). We then look for hosts within a single zone that have different policies.

In Chapter 2, we described several commercial firewall analysis and audit tools: Lumeta [127], Fang [77], Algosec Firewall Analyzer [7], FireMon [44] and Skybox Firewall Assurance [111]. These tools also analyse ACLs to check firewall configuration behaviour and maintain clean and optimised ACLs. Some non-commercial tools [4, 46] have also focused on maintaining concise firewall configurations by detecting and removing ACL redundancies and conflicts.

Past works have also analysed firewall rulesets to help with policy design and check correct policy implementation [16]. The access-control policies supported by this tool are built on the RBAC abstraction. The implementation encodes details of the physical system and traffic control mechanisms. The tool concentrates on checking consistency between the policies and implementations and does not deal with auto-configuration.

Rysavy *et al.* [104] have likewise proposed the verification of ACLs against a security policy using Satisfiability Modulo Theory (SMT). The policies are specified using the Security Policy Specification Language (SPSL), which operate at the low IP flow-level. The checking is performed on all possible flows, identified by source and destination IP addresses and port numbers. In contrast, our aim is high-level policy specification, in terms of zones, not IP addresses.

To the best of our knowledge, our work is the first to attempt to identify high-level inconsistencies over multiple firewalls on a whole network. We build our firewall policy description on the ANSI/ISA Zone-Conduit abstraction. The model decouples network topology and security policy, so, the reduced policy complexity allows these policies to be understood by humans with ease.

The related works described above also target non-SCADA domains in general. But, the observations made through parsing firewall configurations (*e.g.,* misconfigurations) in these domains also occur in SCADA environments. For instance, Wool analysed 74 real Enterprise firewall configurations [128, 129] and found 80% of the firewalls contained serious misconfigurations. He found over 70% of the configurations allowed access to the firewall over insecure (*i.e.,* unencrypted and poorly authenticated) protocols such as Telnet. Over 60% of the configurations included rules that were too broad and admitted far too many services than necessary. An example is a rule that permits all TCP traffic between two hosts.

Serious firewall misconfigurations also occur in mission-critical SCADA networks. In particular, our case studies reveal how incorrect use of firewall security features such as Cisco *security levels* (described in detail in Subsection 3.1.2) and *NAT-control* lead to a broad range of services being enabled by accident. Our case

studies also uncover the use of outdated firewall software in production SCADA environments. Such software contains publicly listed security vulnerabilities that make these firewalls (and the SCADA plants) easy targets for cyber attackers.

Our analysis of the firewall configurations is within a technical context of what the firewalls are supposed to do, how the networks around them are organised, and how the actual configurations deviate from the intent of the designers. This type of analysis is rare in the published literature, particularly in a SCADA network context, because it is rare outside of the SCADA domain to have standards to compare implementations against, or to have access to discuss policies with their designer due to the sensitivity of the data.

3.1 Methodology

Firewall configurations are long and complex. For example, the configurations we discuss have an average length of 769 lines each. So, we need to automate our analysis. Existing tools such as Fang [77] or Lumeta [127] do not support Zone-Conduit based high-level policies. So, we built an automated Parser: *FireP*, to parse our configurations using Zone-Conduit concepts.

It is worth describing *FireP* in detail here because it explains the use of Zone-Conduit concepts in the analysis of practical networks. The Parser is depicted in Figure 3.1. The details are described below:

Firewall Config: The input firewall configuration text-files containing interface configurations, static routes and ACLs. Multiple files can be input to build a network picture.
Interface and Route Processing: The processing of firewall interface configurations and static routes. This extracts interface names, subnet IP addresses, security levels, additional network and gateway IP addresses.
Rule Processing: The processing of ACLs assigned to firewall interfaces and any implicit rules. Implicit rules enable services through the firewall over and above ACLs. More details are discussed in Subsection 3.1.2.
Conduit definition: The definition of conduits that inter-connect the security zones in the SCADA network. Details are covered in Subsection 3.1.3.
Zone-Conduit Model: The abstract topology output of the SCADA network.
Interaction Filtering & Synthesis: The filtering of ACL rule interactions and synthesis with implicit rules. Details of this stage are covered in Subsection 3.1.6.
Service-flow views: The output traffic-flow views for the firewall. A service-flow view describes an enabled protocol through the firewalls between zones.

FireP can currently use the following firewall configurations as input: Cisco Adaptive Security Appliance (ASA), Cisco Private Internet eXchange (PIX), Cisco

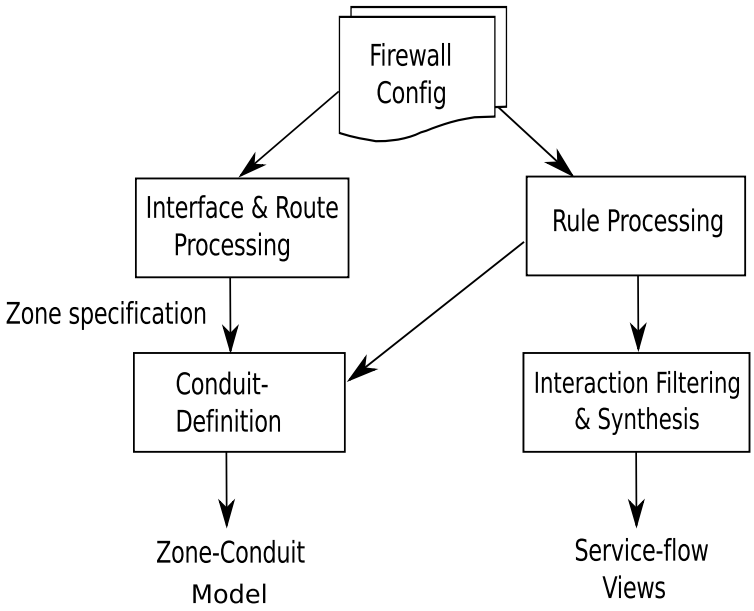


Figure 3.1: Firewall configuration parsing process.

IOS (IOS) or Cisco Firewall Services Module (FWSM). The focus on Cisco arose because these were the devices used in the real SCADA networks we studied, reflecting Cisco's dominance in the market [116]. However, note that there are substantial differences in the configurations across this range of devices. We describe the parsing process and the outputs generated in detail below.

3.1.1 Zone Construction

FireP constructs zones by analysing the interfaces and subnets defined in the firewall configurations. It first assumes that each firewall interface connects to a disjoint zone, and looks for indications that these potential zones should merge. For instance, if the assumed disjoint zones actually formed a single zone, traffic flow from each of these zones to a third zone must be equally controlled by the ACLs on the firewall interfaces connected to these zones. In the absence of any such evidence in the ACLs, we can deduce our original assumption holds and the zones must be disjoint. The original *Zone-Firewall model* is updated to reflect any merged zones identified.

The Parser also processes static routes; these contain IP address details of next-hop gateways and networks reachable via them. We extend the Zone-Firewall model by identifying and including these additional networks and gateways.

3.1.2 Implicit Rules

In a Cisco firewall, traffic flows can be enabled explicitly through ACLs or implicitly via several methods. One available method in ASA, PIX and FWSM firewalls is to assign *security levels* to the firewall interfaces [30]. An interface security level (from 0 to 100) is defined as a level of trust bestowed on the network connected to that firewall interface. In the absence of an ACL assigned to such an interface, certain traffic flows are permitted by default from an interface with a high security level to one with a lower security level [30].

A zone with its firewall interface assigned a security level of 100 is classified as a highest-trust zone [30]. Such a zone cannot be accessed by a lower-trust zone by default. Likewise a zone with its firewall interface assigned a security level of 0 is classified as a lowest-trust zone and is accessible by any zone with its interface assigned a security level greater than zero.

Cisco also facilitates the flow of traffic between firewall interfaces with the same security levels assigned. This flow is enabled via a special Cisco CLI command. Using the command, two zones with equal trust levels can access each other by default.

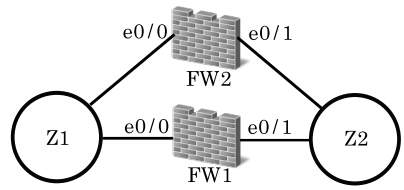
Special configuration commands with higher precedence than interface-assigned ACLs can also enable services implicitly in Cisco firewalls. For example, these commands can enable SSH or HTTP firewall management traffic into the firewall interfaces [30]. In Subsection 3.1.4, we discuss in detail how we use zones to accomodate this firewall management traffic.

Implicit rules provide quick and easy alternatives to ACLs in enabling services through the firewall. They may not map to clear policies but are convenient. However, auto-configuration relies on clear security policies to permit traffic through a firewall. Implicit rules may aim to provide this, but we will see that they confuse the situation in practice.

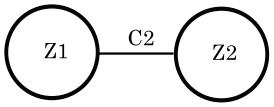
Implicit rules are a feature of some of Cisco's routers and firewalls. Such implicit commands are absent, for instance, in Check Point firewalls such as NGX [24]. We aim to develop a policy description that is firewall-vendor independent. However, due to Cisco's dominance [116] as a security appliance vendor in SCADA and enterprise environments, the networks we had access to used Cisco firewalls.

3.1.3 Zone-Conduit Model

FireP builds the Zone-Conduit model using *firewall-only paths* to define conduits. For instance, we can define a single conduit between two zones with a single-firewall path between them. This case is almost trivial, but the question of how to map a network to zones and conduits has more complex cases.

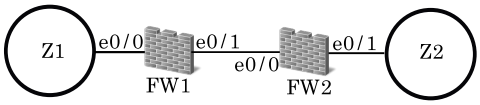


(a) Zone-Firewall model for two zones separated by parallel firewalls.



(b) Zone-Conduit model.

Figure 3.2: Parallel-firewall Conduit definition.



(a) Zone-Firewall model for two zones separated by serial firewalls.

(b) Zone-Conduit model.

Figure 3.3: Serial-firewall Conduit definitions.

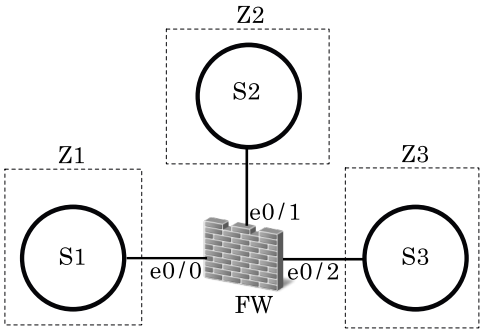
When two zones are connected by parallel links, the ANSI/ ISA standard allows them to be modelled as multiple conduits. But, multiple conduits imply multiple policies could exist between these zones, when only one is possible from the strict interpretation of a *zone*. So, we define a *single conduit* to implement the single policy relationship (*e.g.,* C2 in Figure 3.2). The resultant conduit policy $p_R = p_{Q \cup T}$, where Q, T are the packet sets allowed by the policies (or ACLs) of parallel firewalls $FW1$, $FW2$. Auto-configuration is simplified by this single-conduit representation, since any policy between the two zones is enforced by all firewalls creating conduit C2.

Firewall paths can also include firewalls in series (Figure 3.3(a)). This 'back-to-back' firewall architecture is one of the industry recommended security architectures [21] where defence-in-depth is achieved by using different vendors' devices. The ANSI/ISA guidelines lack clarity on how to define zones and conduits for precisely capturing the distinct policy requirements of these serial firewalls (for instance, FW2 may have logging enabled while FW1 may not).

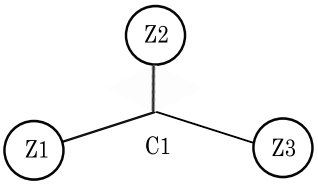
A single conduit containing both firewalls exists, if we dismiss the inter-firewall link. But, for automation, a single conduit hinders precise configuration of the distinct firewalls.

We treat this connecting link as a separate zone, to overcome the specification shortfall. It is referred to as an *Abstract-Zone* in the absence of any real network devices within it (Figure 3.3(b)). The approach creates two separate conduits (C1, C2), containing a firewall each.

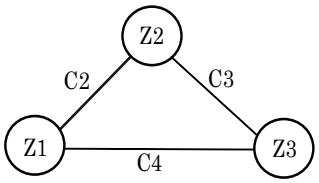
A conduit may also inter-connect more than two zones [59]. ANSI/ISA guide-



(a) Zone-Firewall model for three zones separated by a firewall.



(b) Hyper-graph Zone-Conduit model.

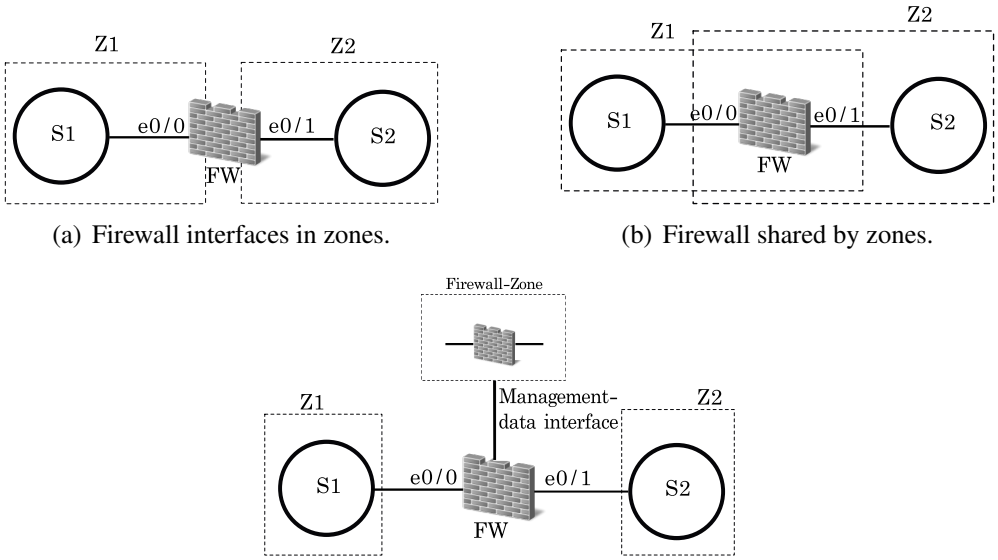


(c) Simple-graph Zone-Conduit model.

Figure 3.4: Conduit-definition alternatives.

lines are unclear on appropriate conduit definitions in such circumstance. As a consequence, the example Zone-Firewall model in Figure 3.4(a), could be modelled using a hyper-graph (Figure 3.4(b)). In this model, the firewall (FW) is located inside the hyper-edge conduit C1 which has *one-to-many zone-communication* paths. This complex conduit can implement multiple security policies; between Z1 and Z2, Z2 and Z3 and Z3 and Z1. Accommodating this complexity means the conduit must track the participating zones per policy. There is also no clear mapping of the ACL rules enforcing each policy to the firewall interfaces. Hence, the hyper-graph conduit model hinders auto-configuration.

To simplify the complexities of hyper-edge conduits, we propose a Zone-Conduit model that consists of only *one-to-one zone-communication* paths (Figure 3.4(c)). Each conduit now implements a single security policy between two zones. The simple design also requires each conduit to only contain the firewall interfaces attached to its connecting zones (*e.g.,* C2 contains e0/0 and e0/1). A conduit path now reveals the exact firewall interfaces and their layout with respect to the connecting zones, enabling easy placement of required ACL rules. The choice of simple-edge conduits, allows us to enforce a strict *1:1 mapping* between conduits and policies.



(a) Firewall interfaces in zones.

(b) Firewall shared by zones.

(c) Dedicated Firewall-Zone.

Figure 3.5: Firewall-Zone alternatives for two subnets S1 & S2 separated by a firewall.

This logical method of conduit-definition can lead to multiple conduits sharing the same firewall (*e.g.,* C2, C3, C4 share FW in Figure 3.4(c)).

In summary, a single conduit need not always map to a single firewall. In fact one-to-many and many-to-one mappings between conduits and firewalls are more useful for high-level security specification. However, for the mapping between conduits and policies, it is more beneficial if a conduit always implements a single relationship between only two zones. This restriction produces a unique (*i.e.,* canonical) interpretation that is compounded across implementations. Allowing multiple relationships otherwise, can also lead to specification of policies that breach the restrictions implied by a zone: *i.e.,* single zone policy.

3.1.4 Firewall Management Access Control

Firewalls need to have secure, authorised network management access to themselves, supplementary to offering mitigation capabilities to zones. The ANSI/ISA standard lack clear direction on how Zone-Conduit concepts should be used to capture firewall management traffic requirements. This is a critical shortfall, because if management of the firewall is compromised, the entire system is compromised.

There are several possible ways to address the issue, as illustrated in Figure 3.5.

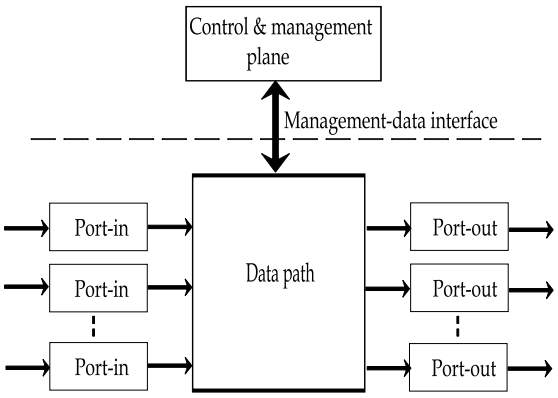


Figure 3.6: Logical firewall architecture adapted from [32], depicting the Firewall-Zone Management-data interface.

Firewall Partially Included in Zones: With this approach, each firewall *interface* belongs to the zone directly connected to that interface (Figure 3.5(a)). It implies that all-IP traffic to the firewall from zones Z1 and Z2 is allowed.

While simple, this approach has obvious problems. The design prevents restriction of firewall access by its connected zones; *e.g.,* disallowing HTTP access to the firewall by zone Z1 would be impossible with this type of a model.

Firewall Shared Between Zones: This model assigns a firewall interface to all connected zones (Figure 3.5(b)), also implying removal of any traffic restriction between hosts and subnets within each zone and the firewall by default. The outcome is similar to that of Figure 3.5(a), preventing placement of a required policy between a zone and the firewall.

Firewall in its Own Zone: Here, we exclude the firewall from belonging to any existing zone and place it in a new security zone on its own. This may seem more complex but it represents the real situation well. This new Firewall-Zone (FWZ) is connected to the firewall (Figure 3.5(c)) via the Management-Data Interface (MDI). The MDI is a logical interface that provides traffic packets to the firewall's control and management plane (Figure 3.6) from the data path [32]. This control plane is responsible for processing the firewall bound management traffic, while the data path handles the traffic forwarded through the firewall.

The Firewall-Zone allows restrictions to be placed on the firewall to regulate its management traffic. The model depicts the firewall's dual role precisely, and allows imposition of restrictions such as disallowing HTTP access to the firewall by zone Z1 (*e.g.,* by placing ACL rules on interface e0/0).

Our introduction of a dedicated Firewall-Zone simplifies management policy specification. It allows firewall-management and non-management traffic to be considered equally, but to be specified separately. This clean approach can fur-

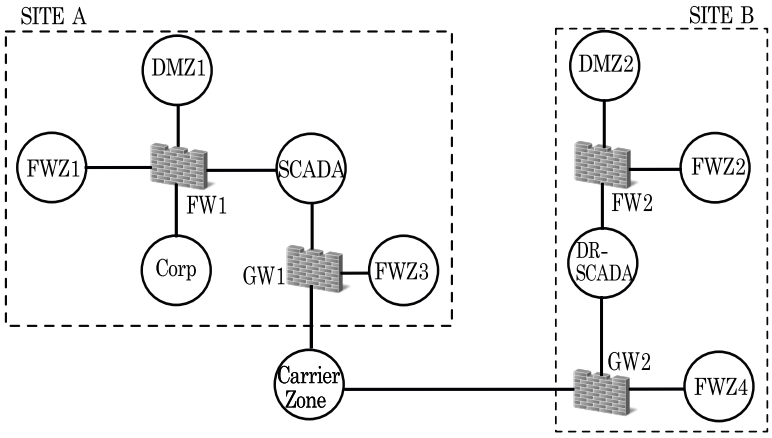


Figure 3.7: Carrier-Zone interconnecting geographically dispersed sites.

ther restrict the management-traffic type allowed, *e.g.,* block Telnet, as per best practices. Of course additional security mechanisms (*e.g.,* password access) are required, but these are outside the current scope of this analysis and thesis.

3.1.5 Carrier Network Abstraction

Real networks often utilise a Carrier Network (CN) provided by a telecommunication service provider to interconnect geographically dispersed sites. This is prevalent in SCADA networks which control distributed field-site equipment from a centralised control centre over, for example, a leased line Wide Area Network (WAN).

The traffic relayed via the CN is controlled by the security policies between the zones within the two interconnecting sites (Figure 3.7). We model the interconnectivity provided by a CN via a *Carrier-Zone* as shown in Figure 3.7. This zone provides connectivity, facilitates security policy specification between the sites and abstracts-away the underlying implementation details.

3.1.6 Service-flow Views

A service-flow view is a directed-graph of zones that can initiate and/or accept that service protocol. The Parser generates these for the protocols; IP, TCP, UDP and ICMP *etc.*, broken down by port and zone as applicable. The output views are graphical representations using GraphML [48], viewable using tools that support the format such as yEd [130].

We generate the *explicit service-flow views* by processing the firewall ACLs. In doing so, intra-ACL and inter-ACL interactions that stem from rule-overlaps need to be considered.

Algorithm 1 Process intra-ACL interactions.

input: An ACL consisting of potentially interacting rules.
output: A rule-set containing intra-ACL interaction free rules.

Step
1. For each rule in input ACL, find all rules that precede and overlap it. Two ACL rules r1, r2 overlap if every field in r1 forms either a subset, superset or is equal to the corresponding field in r2, *i.e.,* $\forall i; r1[i] \otimes r2[i]$ where $\otimes \in \{\subset, \supset, =\}$ and $i \in \{source\_ip, source\_port, dest\_ip, dest\_port, protocol\}$.
2. Derive the net effect of each rule in ACL (*e.g.,* r1), and its preceding and overlapping rules (*e.g.,* r2) as:
   (i) NULL; if r1 is shadowed by r2 (*i.e.,* $\forall i; r1[i] \subset r2[i]$).
   (ii) (r1-r2); if r1 is a generalisation of r2, (*i.e.,* $\forall i; r1[i] \supset r2[i]$).
   (iii) r1 \r2; if rules partially overlap.
   (iv) any of (i)-(iii) above; if rules conflict (*i.e.,* rules overlap but their actions mismatch), so, update r2's action to r1's and use Step 2 recursively to determine net effect.
3. Build a new rule-set consisting of the net rules.

The outcome of a pair of interacting rules depends on several factors: the rule order, the level of overlap (*i.e.,* partial, full overlap or subset) and the rule actions. Depending on the extent of rule overlap, an intra-ACL interaction can be any of the following.

Definition 1 (Intra-ACL interaction)**.** An intra-ACL interaction is a *generalisation*, a *shadowed-rule*, a *partial-overlap* or a *conflict* [2]. A *generalisation* occurs when a subset of the packets matched to a rule has been excluded by one or more preceding rules with an identical action. A *shadowed-rule* is the opposite; all packets applicable to such a rule have already been matched by a preceding rule with an identical action. A *partial-overlap* occurs when the set of packets matched to a rule intersect in part with a preceding rule with a similar action. In a *conflict*, the current rule intersects with preceding rules but specifies a different action.

Algorithm 1 describes how we process the rule overlaps within an ACL to derive an intra-ACL interaction free version (ACL_V1) of each ACL. The algorithm has time complexity $O(n^2)$ where n is the number of ACL rules.

Secondarily, *FireP* also accounts for the inter-ACL interactions that alter an ACL rule's intended behaviour. Inter-ACL interactions can occur between ACLs that are assigned on different interfaces of the same firewall or between ACLs that are assigned on different firewalls. *FireP* analyses potential inter-ACL interactions

Algorithm 2 Process inter-ACL interactions.

input: An ACL that can interact with other ACLs
in the network, a list of all network ACLs and
the Zone-Firewall model of the network.
output: A rule-set containing inter-ACL interaction free rules.

Step
1. Find all ACLs that interact with provided ACL by:
   (i) considering each ACL rule's source, destination zones
   and identifying all elementary paths from source to
   destination in the Zone-Firewall model.
   These paths should exclude Firewall-Zones.
   (ii) locating the network ACLs that lie in each
   elementary path found.
2. Compute the net effect of provided ACL and each interacting
   ACL by deriving rule-wise net effect.
   So, rule r1 of input ACL and r2 of interacting ACL
   will render r1's net effect as:
   (i) r1; if r1 is shadowed by r2; or r1, r2 do not overlap; or
   r1 is a generalisation of r2 and both are deny rules; or
   r1 and r2 partially overlap and both are deny rules.
   (ii) r1-r2; if r1 is a generalisation of r2 and both
   are not deny rules.
   (iii) r1-(intersection of r1 and r2); if r1 and r2 partially
   overlap and both are not deny rules.
   (iv) NULL; if r1 and r2 conflict and both are deny rules
   and r1 is shadowed by r2.
   (v) any of (i)-(iii) above; if r1 and r2 conflict.
   So, update r2's action to r1's and use Step 2 recursively
   to determine net effect.
3. Build a new rule-set consisting of the net rules.

of each ACL_V1 using Algorithm 2, to derive a second version (ACL_V2) that now reflects the *net-effect* of all rule interactions possible for a given network. The time complexity of Algorithm 2 is $O(pnm + pn^2)$ where n is the average number of rules in an ACL, m is the average number of valid paths between a zone pair in the Zone-Firewall model and p is the number of ACLs in the network.

The Parser generates explicit service-flow views depicting the overall services enabled by each ACL using the net ACL effect (ACL_V2).

FireP also processes implicit rules based on interface security levels and builds an implicit service-flow view. This service-flow view describes IP-level traffic flow enabled implicitly between zones. Special Cisco configuration commands

that permit firewall management traffic above ACLs are also parsed by *FireP* and corresponding service-flow views are created.

The Parser combines the implicit and explicit service flow views above to derive the high-level security policy implemented in the network. This automated capability makes the task of redefining existing policy at a high-level easier. This task is often seen as requiring a larger effort than just analysing the deployed rulesets [3] because ruleset analysis tools are often standalone and cannot determine anomalies in distributed firewalls. *FireP* also generates results that we can use to test firewall configurations are correct and efficient.

3.1.7 VLAN Considerations

Commercial firewalls (*e.g.,* Cisco ASA 5500 series) can also support VLANs for managing broadcast traffic [30]. However, the type of segregation VLANs offer is only logical: there is no physical separation between each VLAN. A purely logical segregation between a SCADA and a enterprise network is inadequate and should be avoided [21]. For example, a DoS attack on a VLAN-separated enterprise network can render the SCADA network inoperable, as the (shared) physical network becomes saturated with malicious traffic, the SCADA networks is also effected.

VLANs also operate at the Ethernet layer and have no understanding of the traffic 'state'. VLAN tags can be spoofed with ease and there are many hacking tools [101, 106] designed to bypass their security. IT experts have warned on the dangers of using VLANs for security [69, 106]. *FireP* handles them but generates warnings to make the user aware of the fact.

In summary, *FireP* serves multiple purposes; (i) it functions as an operational tool, allowing users to check a network's segregation strategy and the high-level policy deployed; and (ii) it also serves as a research tool, allowing users to conduct firewall configuration case studies; and (iii) it can also be used as an automated high-level policy generator, useful for redefining existing policy deployed in a network. We use it in the case studies below to analyse seven real security architectures.

3.2 A series of case studies

Obtaining real firewall configurations from working SCADA networks is difficult due to the sensitive nature of the data. We were able to obtain such configurations from seven SCADA networks. A high-level summary of these Systems Under Consideration (SUCs) is provided in Table 3.1. We describe each SUC in detail here and order their discussion based on the number of security best-practice breaches found, from highest to lowest.

Table 3.1: High-level summary of the SUCs (* backup firewall, ** does not include system-generated zones such as Firewall-Zones and Abstract-Zones, *** conventional format, LoC - Lines of Code).

SUC	Configuration date	Firewall type	Firewalls	Gateways	Zones**	Average LoC per firewall	ACLs	Average rules per ACL***
1	Sep 2011	Cisco IOS	2	1	2	1360	10	237
2	Aug 2011	Cisco ASA	1(2)*	5	11	432	12	16
3	Oct 2011	Cisco PIX	2	2	5	125	8	6
4	Mar 2011	Cisco ASA	1	2	4	819	3	80
5	Apr 2015	Cisco ASA	1(2)*	2	12	853	12	677
6	Apr 2015	Cisco ASA	1(2)*	3	13	900	8	1034
7	Jul 2015	Cisco ASA, Cisco FWSM	2(4)*	4	11	860	13	724

Due to security concerns and non-disclosure agreements, a modified version of each real SCADA network is presented for discussion. Effort has been taken to ensure that the security strategies and underlying issues uncovered remain intact. However, details such as IP addresses are anonymised.

3.2.1 SUC 7

We start with System Under Consideration (SUC) 7 because it contained the highest number of security best-practice breaches.

The Network in Detail

The SUC used a Cisco Adaptive Security Appliances (ASA) 5500 series firewall and a Cisco Catalyst 6500 switch that included an integrated Firewall Services Module (FWSM). The FWSM treats VLANs on the switch as separate firewall ports [28]. The configurations of both devices were extracted in July 2015.

The Cisco FWSM (FW1) consisted of 917 LoC with eight ACLs that averaged 1075 conventional rules each. The Cisco ASA firewall (FW2) consisted of 804 Lines of Code (LoC) with nine ACLs averaging 412 conventional rules each. A conventional rule does not group protocols and ports. In contrast, Cisco object-groups classify devices, protocols and ports into groups. These groups can be very useful in practice given the potentially lengthy conventional ACLs. For example, the ACLs studied in the SUC contained on average 724 conventional rules each, as opposed to 18 object-group based rules.

The SUC is shown in Figure 3.8. Firewall FW1 uses VLANs to separate SCADA1 network traffic from enterprise-network traffic. Firewall FW2 physically separates SCADA2 network traffic from enterprise-network traffic.

FW2 has an active/standby fail-over configuration enabled via a dedicated Ethernet link. So, an identical standby firewall could take over the functionality of the primary unit on failure [30]. The primary unit automatically replicates its configuration to the standby unit once special configuration commands are issued [30]. Hence, the auxiliary unit mirrors the primary unit's configuration. The standby unit is also accessed only via the primary unit, so both are managed as one using a single Firewall-Zone. The Catalyst 6500 switch had built-in redundancy.

FW1 has four virtual interfaces operational, each pointing to the subnets below. All subnets except β-DMZ were configured to host up to 254 hosts. β-DMZ accommodated up to six hosts.

The α-Demilitarised-Zone (α-DMZ): Accommodates shared servers that receive updates from the SCADA1 network. The DMZ also facilitates file transfer and file sharing with enterprise-network hosts (reachable via β-DMZ gateway).

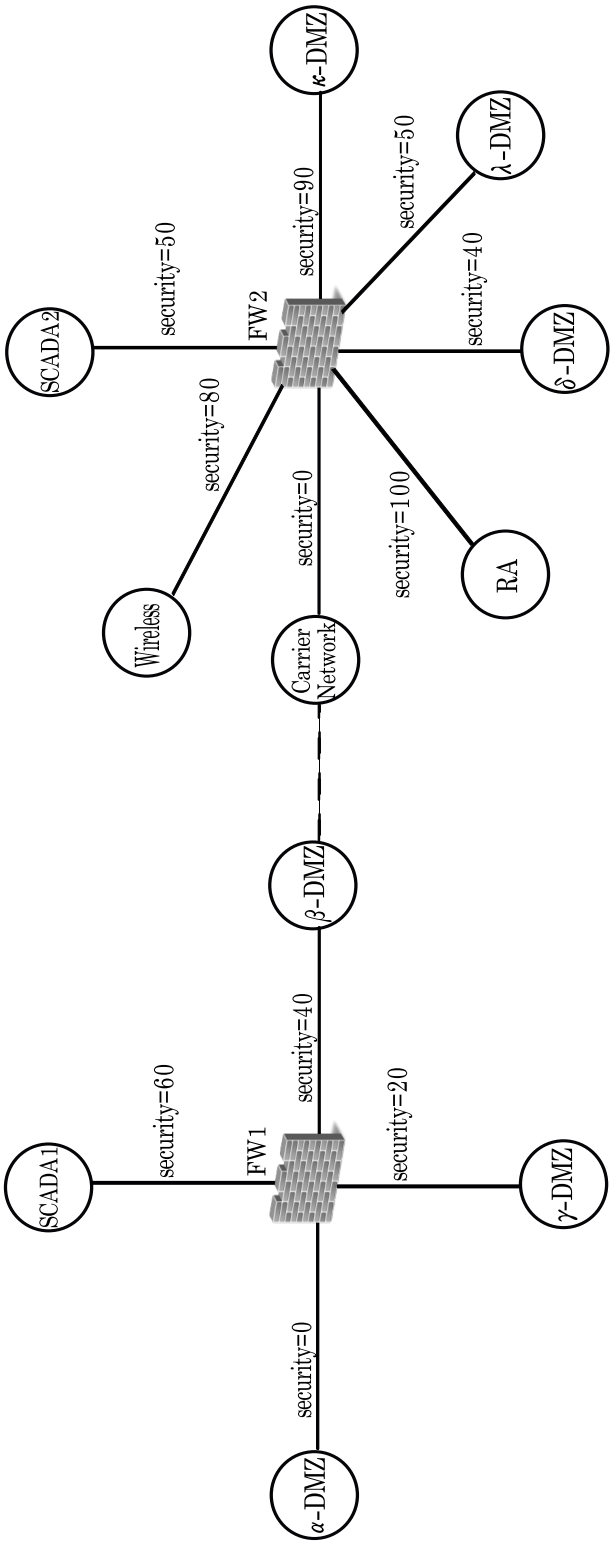


Figure 3.8: SUC 7.

The β-Demilitarised-Zone (β–DMZ): Hosts shared servers that can be accessed by γ–DMZ and enables access to the shared servers in α–DMZ.
The γ-Demilitarised-Zone (γ–DMZ): Accommodates Web servers that can be accessed by enterprise-network hosts.
The SCADA network1 (SCADA1): The utility distribution control centre that communicates with the SCADA2 network.

FW2 has seven physical interfaces, each pointing to a subnet described below. All subnets were configured to host up to 254 hosts.

The remote access network (RA): Allows enterprise-network users (reachable via Carrier network gateway) to remotely login to RA using SSH and two-factor authentication. Also facilitates remote access of SCADA by RA users.
The δ-Demilitarised-Zone (δ–DMZ): Accommodates shared servers that have file transfer and sharing enabled with enterprise-network hosts. The servers also have RDP-based remote access enabled to the SCADA2 network. The DMZ also allows its hosts to be accessed by SCADA1 and λ–DMZ.
The λ-Demilitarised-Zone (λ–DMZ): Facilitates file transfer, sharing and RDP-based remote access of its shared servers by enterprise-network users. Also enables SNMP alerts from the enterprise-network to the DMZ hosted SNMP server. The DMZ's shared servers can also be accessed from SCADA1 and δ–DMZ.
The κ-Demilitarised-Zone (κ–DMZ): Enables FTP-based file transfer and SSH-based remote login from enterprise-network users. The shared servers in this DMZ can also be accessed from SCADA1 and RA.
The SCADA network2 (SCADA2): Enables networked access to utility-performance infrastructure such as condition monitoring equipment.
The Carrier network (Carrier): The third-party operated wide-area network that links SCADA2 to SCADA1 and the Enterprise network.
The wireless network (Wireless): Facilitates FTP-based file transfer and SSH-based remote login from enterprise-network users. The wireless hosts can also be accessed from λ–DMZ, κ–DMZ and RA.

Varying security levels (from 0 to 100) are assigned to each firewall interface (Figure 3.8). With a security level of 100 assigned RA is classified as the highest-trust zone. Doing so, prevents the zone from being accessed by a lower trust zone by default. Carrier is classified as the lowest-trust zone in the group with a security level of 0 assigned to its firewall interface.

Traffic flow from a high-security interface to a low-security interface is permitted by default [30]. So, for instance, RA is allowed to access all other zones by default. In addition, same-security level traffic is permitted between firewall interfaces. So, for instance, SCADA2 and wireless can access each other by default.

The firewall uses Cisco extended ACLs [30] with object-group based rules. Of the nine ACLs defined in FW2's configuration, only seven were in use. Two ACLs

were defined as 'test' rulesets and not assigned to any of FW2's interfaces. These seven used ACLs were assigned inbound on FW2's interfaces. All four ACLs defined in FW1 configuration were used and each was assigned to an interface. The ACLs override, the default security-level based filtering behaviour [30].

In FW2's configuration, RA has an ACL, ra_access_in assigned inbound to its interface. The ACL plays an important role because it restricts traffic from RA to SCADA2. Without this ACL, all-IP traffic flow is enabled instead, due to the security levels assigned.

ACLs are also assigned on the λ-DMZ, κ-DMZ and δ-DMZ interfaces of FW2 to restrict access to and from their shared servers. For instance, δ_dmz_access_in is assigned inbound on δ-DMZ's firewall interface and enables RDP traffic from δ-DMZ to SCADA2.

In FW1's configuration, β-DMZ has an ACL, β_dmz_in assigned inbound on its firewall interface. The ACL restricts traffic from β-DMZ to SCADA1 and α-DMZ. Similar ACLs are assigned on the firewall interfaces of α-DMZ, β-DMZ and γ-DMZ to restrict access to and from their shared servers.

Static routing was used rather than dynamic routing protocols such as OSPF or EIGRP. This is common in such networks where network reliability is achieved via redundancy in layer-2 switching [109].

Results and Best Practice Implications

Figure 3.9(a) shows the Zone-Firewall model of the SUC generated by parsing the configuration data. Figure 3.9(b) shows the corresponding Zone-Conduit model. It includes a Carrier-Zone and multiple Firewall-Zones (FWZs).

In generating the models, we uncovered four gateways (GW1-GW4). Their configurations were unavailable, so we assume here the conservative security option: each gateway enforces a security policy and hence behaves as a firewall.

As a consequence, our assumption yields two additional zones attached to a gateway: one is the Firewall-Zone that represents the control plane of the gateway, the other zone encompasses all subnets reachable via the gateway (as per the static routes). We group these subnets to a single zone because without actual gateway configurations, precise identification of the disjoint zones the subnets may create, is impossible.

We can compare this network security model against industry recommended SCADA firewall architectures in [21], [117]. The model falls into the category of 'paired firewalls between the SCADA network and the Enterprise network' [21]. The model also complies with the critical requirement that the SCADA-Zone should not be directly connected to the Internet-Zone (reachable only via gateway G3). Direct communication between these two zones was also disabled as per the best practices.

(a) Zone-Firewall model including gateways.

(b) Zone-Conduit model of (a).

Figure 3.9: Security models of SUC 7.

HTTPS was enabled inbound to both SCADA1 and SCADA2 from Corp. Doing so, adhered to industry best practices in [21] as the protocol is deemed safe, and is recommended to carry Web traffic to and from a SCADA-Zone.

Security Violations: We identified multiple serious violations of the SCADA best practices. The most significant was the use of VLANs to separate SCADA traffic from enterprise-network traffic. Industry recommends physically separating the two types of traffic to prevent for instance, a DoS attack aimed at the Enterprise-Zone from rendering the SCADA-Zone non functional.

The firewall configurations also enabled FW1 and FW2 to be managed from Corp over Telnet, HTTP and SSH. The Telnet protocol lacks support for message encryption and strong authentication and hence industry recommends against using it for firewall management [21]. But moreover, industry recommends against managing a SCADA firewall from a less-trusted Enterprise-Zone because a compromise in the latter could compromise the management of this firewall.

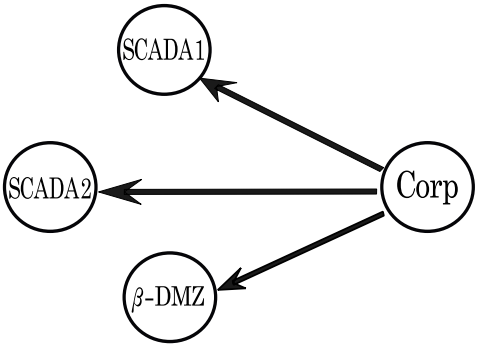
Service flow views in Figure 3.10 depict how direct traffic transition was also explicitly enabled between less-trusted Corp and the two SCADA-Zones. Doing so, increases the vulnerability of the SCADA-Zones to cyber attack [21, 117].

For instance, SMB based file sharing was enabled between Corp and the two SCADA-Zones (Figure 3.10(a)). Industry recommends against allowing the protocol across a SCADA firewall because it is often used to carry out attacks on networks [117].

In addition, unsafe HTTP, NTP and Telnet traffic was explicitly enabled inbound to both SCADA-Zones from Corp. These protocols are known to transport worms and attacks into networks, so enabling them also exposed the SCADA-Zones. The problem was exacerbated by all-IP traffic explicitly allowed inbound to SCADA2 from λ-DMZ. Enabling such a broad range of protocols meant a large number of unsafe protocols were admitted in to the protected SCADA-Zone. This illustrates the danger of *implicit allow* rules.

The service flows that stem from the security-levels assigned to the firewall interfaces are depicted in Figure 3.11. It shows that all-IP traffic flow from κ-DMZ, λ-DMZ, RA and Wireless to SCADA2 is allowed. The implicit rules contradict with industry guidelines in [21] which recommend terminating traffic at the DMZs and only enabling safe protocols inbound to a SCADA-Zone.

The implicit rules could also have made SCADA2 more vulnerable, by increasing the complexity of the configuration. The ACLs assigned on the firewall interfaces override the security-levels. However, incorrect security-levels lead to a confusing issue of precedence of rules. So, consistent security levels must be assigned to comply with industry best practices; the interface of κ-DMZ should have been assigned a lower trust level than the interface of SCADA2 interface and not vice versa. By assigning correct security levels, we also enforce defence-in-depth.



(a) SMB traffic flow.

(b) HTTP traffic flow.

Figure 3.10: Explicit service-flow views of SMB and HTTP traffic for SUC 7.

For instance, if the ACLs on the interfaces of SCADA2 and κ-DMZ were accidentally removed, correct security levels would continue to protect the SCADA-Zone by only allowing IP traffic flow to be initiated only from the more trusted SCADA2 interface to κ-DMZ interface.

Security-levels provide quick and easy access between firewall interfaces. But, as the case study shows, their default behaviour of enabling all-IP traffic flow from a high-security firewall interface to one with a lower security level, can easily expose a SCADA-Zone if assigned incorrectly. Hence, the need for ACLs to specify more granular traffic restrictions. However, as observed in the SUC, the defined security-levels and the ACLs can interact in complex ways and conflict one another, leaving a network vulnerable. We see that (i) allowing implicit rules that don't map to a clear high-level policy; and (ii) enabling security features that interact in complex ways, only promote network-security that is *broken-by-design*.

The firewall software version in FW1 (FWSM 2.3) was also over eight years old at the time its configuration was extracted. Many publicly listed security vulnerabilities were available for this software version since its release. For instance, FWSM 2.3 allows Access Control Entries (ACEs) in an ACL to be improperly evaluated, enabling remote authenticated users to bypass the protections intended by ACLs [35]. Some of these listed vulnerabilities were not applicable to the firewall's configuration context. But, enabling a restricted set of firewall functionality is a poor excuse for not upgrading/patching firewall software in a regular manner. Firewall configurations evolve with time, hence vulnerabilities that were inapplicable at first could eventually expose a network to cyber threats.

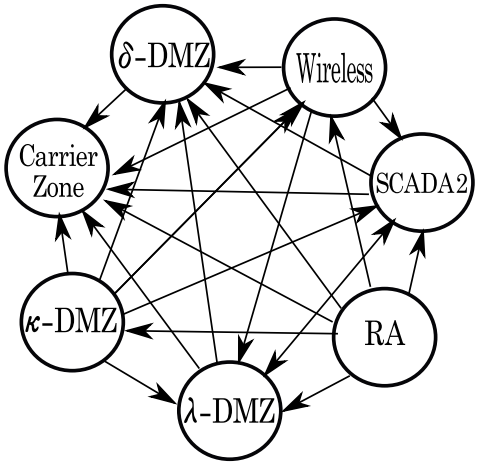


Figure 3.11: Implicit service-flow view of IP traffic for SUC 7.

Even worse, the decade-old Cisco FWSM was out of vendor support, but used in a production SCADA environment. Regular patching and upgrading of systems is less practical on the SCADA side of networks as discussed earlier, but that doubles the need to have up-to-date firewalls protecting the SCADA network.

ACL-rule comments were present in the SUC's firewall configurations, but, these comments lacked clarity on the purpose or requirement of the rules. This is not usually listed as a configuration error, but here we highlight the issue as it is common and leads to long-term maintenance problems.

FireP also identified ACL rules with invalid source and/or destination IP addresses. In some cases the order of the addresses was wrong while in others they were simply invalid. There were eight such occurrences.

In addition, the Parser also found 31 ACL rules that were either assigned to the incorrect firewall, firewall interface or direction. An *incorrect-firewall error* implies that the desired traffic filtering could not be achieved by placing the ACL rule in any of that firewall's interfaces. An *incorrect-interface error* implies that the desired traffic filtering could not be achieved by assigning the rule inbound or outbound of that firewall-interface. *Incorrect-direction* errors comprise of ACL rules that are assigned to the correct firewall and firewall-interface, but in the wrong direction (*e.g.,* outbound instead of inbound).

The ACL rules in both the above circumstances wasted configuration space. But more importantly, a security administrator reading the lengthy firewall configurations wouldn't easily understand that these rules were not enforcing their intended policy in any way. This *appearance of security* is particularly dangerous as it permits behaviour that would be more carefully scrutinised in an insecure domain.

Configuration inefficiencies: Several other occurrences of configuration space wastage were also detected by *FireP*. For instance, it found two unused ACLs in FW2 configuration that wasted 15 lines. There were also three redundant object-groups with identical semantics defined using different names, violating the single-source of truth doctrine.

The Parser also detected 34 intra-ACL interactions through rule processing. These interactions comprised of 12 shadowed rules and 22 generalisations. These shadowed rules further contributed to the appearance of firewall security, potentially misleading administrators on the enforcement of their intended policies.

There were also 89 inter-ACL interactions involving the ACLs. Most of these were shadowed-rules caused by identical rules in distinct ACLs, collectively enabling traffic flow between zones. These intra- and inter-ACL interactions hindered maintaining concise firewall configurations.

3.2.2 SUC 1

This SUC was the first dataset we examined. The data was extracted in September 2011.

The Network in Detail

SUC 1 used two Cisco IOS routers (running IOS ver 12.2 and 12.3) to physically separate the Enterprise and SCADA networks. Both routers had ACLs configured. One router (R1) used 1150 Lines of Code (LoC) with 5 ACLs averaging 184 conventional rules each. The other router (R2) consisted of 1571 LoC with 5 ACLs averaging 290 rules each. Once ACLs are enabled, the routers behave as firewalls.

The SUC is shown in Figure 3.12 that shows the firewalls/routers employed in a serial configuration. There were no network devices connected to the subnet between the firewalls. R1 connected to the Enterprise network while R2 connected to the SCADA network.

R1 has two physical interfaces operational, pointing to the `Corp` and `LAN` subnets. R2 also has 2 interfaces active, pointing to `SCADA` and `LAN` subnets. `Corp` could accommodate up to 2046 hosts and `SCADA` accommodated up to 65534 hosts. The subnets are summarised below.

The Enterprise network (`Corp`): Allows business application and Internet access.
Local Area network (`LAN`): Enables connectivity between R1 and R2.
The SCADA network (`SCADA`): Enables networked access to plant equipment.

Cisco standard and extended ACLs were used. All five ACLs defined in R1 were in use while only three of the five ACLs defined in R2 were in use.

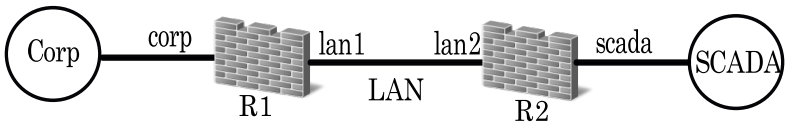


Figure 3.12: SUC 1.

An extended ACL, `corp_access_in`, is assigned inbound on `Corp`'s firewall interface. The ACL enables traffic flow from `Corp` to `SCADA`. For instance, it permits FTP-based file transfer from `Corp` users to `SCADA` hosted servers.

There are has two extended ACLs, `lan1_access_in` and `lan2_ access_in` assigned inbound on the `lan1` and `lan2` interfaces respectively. The first ACL permits traffic initiated from `SCADA` to `Corp`. For instance, it allows all ICMP messaging between the two subnets. The ACL also allows SMTP and DNS traffic initiated from selected `SCADA` hosts to the `Corp` hosted management and monitoring stations. In addition, the `SCADA` hosts can perform file transfer using FTP with `Corp` servers. `lan2_access_in` overlaps with `corp_access_in` to collectively enable traffic flow between `Corp` and `SCADA`.

R1 has three additional standard ACLs defined, `acl_time_sync`, `acl_snmp` and `acl_vty1`. The first ACL restricts NTP based time synchronisations to known NTP servers in `Corp`. SNMP-based alert exchanges are restricted by `acl_snmp` to a known SNMP server located in `Corp`. The ACL `acl_vty1` restricts SSH-based remote access of R1 to selected hosts in `Corp` and off gateway GW. This ACL also limits Telnet-based remote access of R1, to R2 only. Once remotely logged in, the ACL disallowed remoting out to other devices.

There is also an extended ACL, `scada_access_in` assigned inbound on the `SCADA`'s firewall interface. The ACL overlaps with `lan1_access_in` to collectively enable traffic flow between `SCADA` and `Corp`.

R2 uses a standard ACL, `acl_vty2` that restricts Telnet and SSH based remote access of R2, to selected hosts in `SCADA` and `Corp` as well as the Enterprise-network gateway and R1.

EIGRP routing was enabled on both firewalls. A default route was also present in each of the configurations, revealing a gateway located off `Corp`. It also informed us that R1 acted as the gateway for R2.

The configuration also revealed that a remote Syslog server located in `Corp` was used to store local firewall log messages. Both firewalls used the *ip inspect name* command for automated build of state-tables to allow return traffic to bypass the ACLs as necessary [33].

SNMP messages from the routers were sent to a Network Monitoring Server (NMS) in `Corp` on authentication, link-up and link-down events.

Results and Best Practice Implications

The Zone-Firewall model generated by parsing the configuration data is shown in Figure 3.13(a). In generating the model, we uncovered a gateway (GW), and again we assume here the conservative security option: each gateway enforces a security policy and hence behaves as a firewall (their configurations were unavailable).

The Zone-Firewall model falls into the category of 'paired-firewalls incorporating an empty demilitarised zone' (*i.e.,* our Abstract-Zone). As per industry recommendations, the Internet-Zone is not directly connected to the SCADA-Zone.

Figure 3.13(b) shows the corresponding Zone-Conduit model. It includes the Firewall-Zones (`FWZ1`, `FWZ2`) and an Abstract-Zone (`AZ`), to capture the individual policy requirements of the serial firewalls.

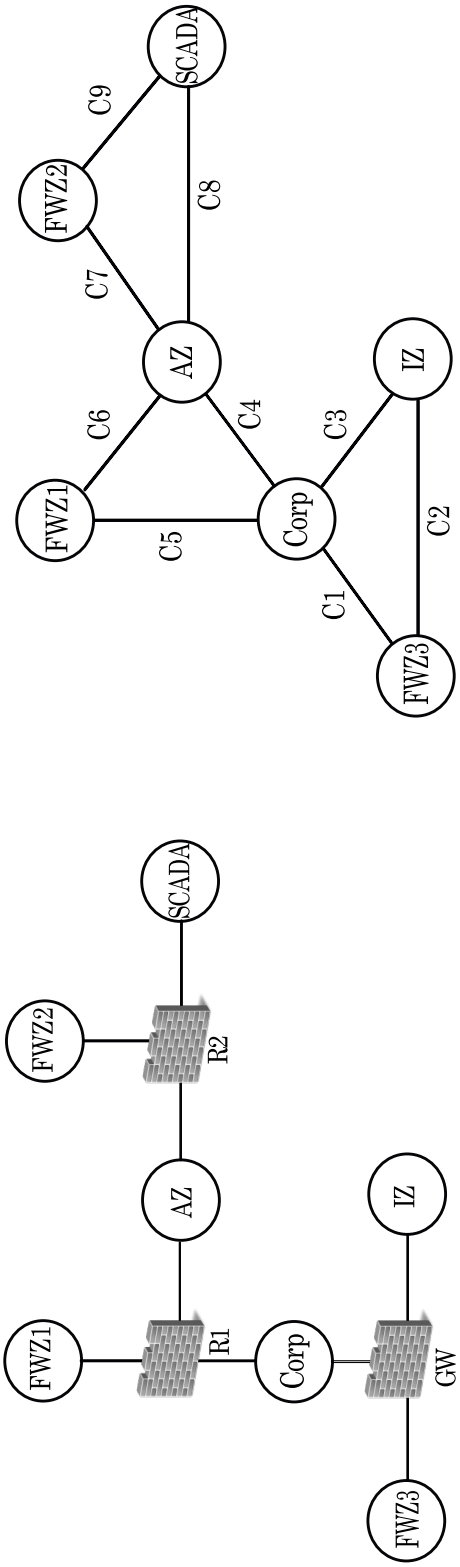
An example explicit service-flow view of SSH traffic is shown in Figure 3.14(a). We checked these permitted services against the industry best practices in [21]. SSH is considered a secure protocol, so, traffic can be allowed both inbound and/or outbound from the SCADA-Zone [21]. Figure 3.14(a) complies with this requirement. Figure 3.14(b) shows the explicit rules to allow devices in `AZ` to manage `FWZ2` but not `FWZ1`. This reasserts the distinct policy requirements of the serial firewalls, showcasing the need for an Abstract-Zone to separate these firewalls to accommodate such policy specification.

Figure 3.14(b) describes the service-flow view for implicitly enabled Telnet firewall management traffic. It shows that Firewall R2 can be managed using Telnet from both `Corp` and `SCADA`.

Security Violations: We found several significant violations of the industry standards. For one, direct communication was enabled between the SCADA-Zone and the Internet-Zone (Figures 3.15(a) and 3.15(b)). The best-practices recommend against doing so, for obvious reasons. For another, insecure DNS, HTTP, SMTP and FTP protocols were enabled inbound to `SCADA`. HTTP for example, can potentially transport many attacks and worms in to the SCADA-Zone [21]. These protocols were restricted to particular hosts but due to the lack of a real separation using firewalls between these hosts and the rest of the zone, these were only perceived exceptions; a compromise of one host could compromise the entire zone.

The explicit service flow views in Figure 3.15 show that direct traffic is enabled between `SCADA` and the less-trusted `Corp`. Doing so, exposes the SCADA-Zone and is hence discouraged [21, 117].

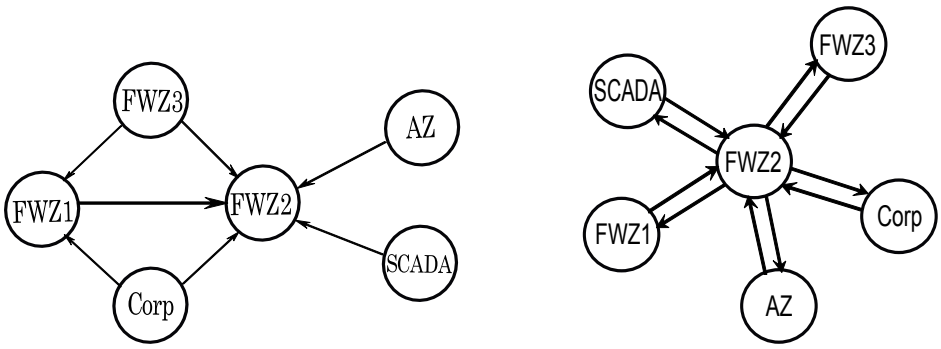
All-IP traffic was also explicitly enabled between `SCADA` and `Corp` through ACL rules. Such broad rules admit far too many services than necessary, and further increased the vulnerability of the SCADA-Zone [21].



(a) Zone-Firewall model including gateways.

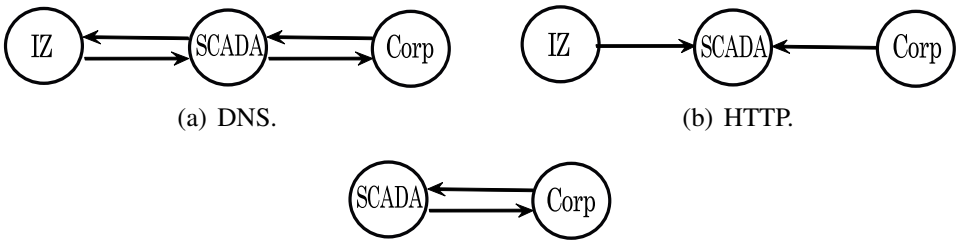
(b) Zone-Conduit model of (a).

Figure 3.13: Security models of SUC 1.



(a) SSH (explicit). (b) Telnet (implicit).

Figure 3.14: Service-flow views of SSH and Telnet traffic for SUC 1.



(a) DNS. (b) HTTP.

(c) FTP.

Figure 3.15: Explicit service-flow views of DNS, HTTP and FTP for SUC 1.

The firewall configurations also enabled firewall management over insecure Telnet which lacks support for message encryption and strong authentication and its use in firewall management is discouraged [21]. Moreover, R2 could be managed from `Corp`. Managing a SCADA firewall from a less-trusted Enterprise-Zone is not recommended because a compromise in the Enterprise-Zone could compromise the management of the SCADA firewall [21].

TFTP was also allowed from the Firewall-Zones to hosts off the `Corp` gateway. TFTP does not require user login prior to file transfer and industry recommends [21] avoiding its use altogether.

EIGRP configuration also deviated from the Cisco guidelines. Instead of enabling EIGRP *hello multicasts* between neighbours, broad IP-level multicasts were enabled between neighbours. To accommodate unicast EIGRP acknowledgments (responses), similar broad rules were used.

The SUC did not have fail-over firewalls configured. This lack of redundancy of critical infrastructure components in a real-world SCADA plant was alarming.

Our Parser identified ACL rules with invalid source and/or destination IP addresses. There were 70 occurrences of this type. *FireP* also found 47 ACL rules

Table 3.2: Inter-ACL interactions summary for SUC 1.

ACL1	ACL2	Interaction type	Interaction count
scada_access_in	lan1_access_in	shadow	229
corp_access_in	lan2_access_in	shadow	533
corp_access_in	lan2_access_in	generalisation	3

that were either assigned to the incorrect firewall, firewall interface or direction. All of these ACL rules wasted configuration space, but most importantly, none actually enforced their intended policies. The close coupling between the ACL rules and network-implementation, render the configurations human incomprehensible. Hence, an administrator cannot simply locate these errors by inspection.

Configuration Inefficiencies: We also found ACLs copied between the firewalls but left-in unused, wasting hundreds of lines in the configuration files. Additional copies were also made of these ACLs, renamed and modified to cater for the distinct policy requirements of the serial firewalls. This configuration approach is itself inefficient, but accentuates the overlapping nature of the security policies enforced by a serial firewall configuration.

There were 167 intra-ACL interactions involving the four ACLs assigned to the firewall interfaces. These consisted of seven generalisations, 146 shadowed-rules, 12 partial-overlaps and two conflicts. The intended policy of these shadowed-rules and conflicts were not enforced. But, this appearance of security is difficult for a human to scrutinise by simply reading the configurations.

FireP also identified 765 inter-ACL interactions involving these four ACLs, as summarised in Table 3.2. These interactions contained 762 shadowed-rules and three generalisations. The shadowed rules included different rules, which indicated the potentially distinct nature of serial-firewall policies. These inter- and intra-ACL interactions contributed to expansive firewall configurations.

FireP also identified ACL rules that attempted to explicitly block directed broadcasts from propagating between the firewalls. These rules were redundant as routers blocked these broadcasts by default.

3.2.3 SUC 2

This SUC was the first dataset that we analysed that used VLANs with 802.1Q trunking. The data was extracted in August 2011.

The Network in Detail

The SUC employs a single Cisco ASA 5510 firewall (Figure 3.16). The configuration consisted of 432 lines with 12 ACLs averaging 16 conventional rules each.

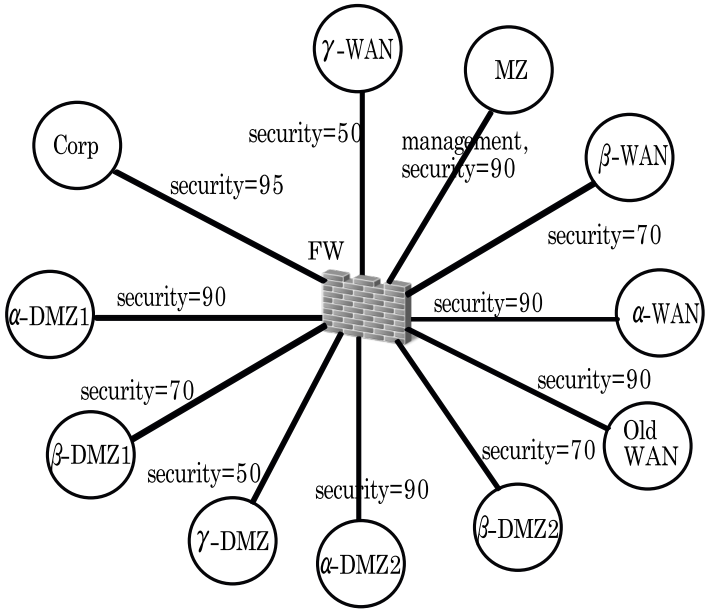


Figure 3.16: SUC 2.

The firewall physically separates the enterprise network from α-WAN; the entry point to the SCADA network. There is also an active/standby fail-over unit configured as a backup firewall.

The primary firewall has four physical interfaces operational; two of these were divided into sub-interfaces supporting five VLANs. These interfaces each pointed to the subnets described below. All subnets were configured to accomodate up to 254 hosts.

The enterprise network (Corp): Provides access to business applications and the Internet. Corp accommodates NTP servers, a TFTP server, an Email server and several RDP clients.
The α-WAN: High security WAN enabling access to the SCADA network.
The β-WAN: Medium security WAN enabling data updates to the β-DMZs.
The γ-WAN: Low security WAN enabling data updates to the γ-DMZ.
The Old-WAN: High security WAN enabling secure access to the α-DMZs.
The α-Demilitarised-Zone1 (α-DMZ1): High security DMZ that receives updates from the SCADA network and α-DMZ2. The DMZ facilitates inbound access from Corp and other DMZs. α-DMZ1 hosts Web servers, a Network Monitoring Station (NMS) and a timeserver.
The α-Demilitarised-Zone2 (α-DMZ2): High security DMZ responsible for receiving updates from the SCADA network, which sends syslog messages and alerts to the α-DMZ1. This DMZ hosts a data historian.

The β-Demilitarised-Zone1 (β-DMZ1): Medium security DMZ that periodically synchronises data with the NMS in α-DMZ1. It also shares updates with the enterprise network and β-DMZ2. This DMZ hosts Web, RDP and SNMP servers.
The β-Demilitarised Zone2 (β-DMZ2): Medium security DMZ that allows RDP clients in the enterprise network to access its servers.
The γ-Demilitarised-Zone (γ-DMZ): Low security DMZ that periodically synchronises data with the NMS in α-DMZ1. It also shares updates with the enterprise network. The DMZ hosts Web, RDP and SNMP servers.
IT Management Network (MZ): Hosts multiple management workstations that allow network-device management via Telnet.

Varying security levels (from 0 to 100) were assigned to each firewall interface (Figure 3.16). With a security-level of 95 assigned, Corp is classified as the highest-trust zone. This classification prevents the zone from being accessed by a lower-trust zone by default (*i.e.,* without the use of ACLs).

The five zones: Old-WAN, Management, α-DMZ1, α-DMZ2 and α-WAN are classified as second highest-trust zones with each assigned a security-level of 90. β-WAN, β-DMZ1 and β-DMZ2 are classified as third highest-trust zones with each assigned a security level of 70. With a security-level of 50 assigned, γ-DMZ and γ-WAN are classified as the lowest-trust zones in the group.

Traffic flow from a high-security interface to a low-security interface is permitted by default [30]. So, for instance, Corp is allowed to access all other zones by default. In addition, same-security level traffic is permitted between firewall interfaces, via a special Cisco CLI command. So, for instance, γ-DMZ and γ-WAN can access each other by default.

The firewall uses Cisco extended ACLs with conventional rules. Of the 12 ACLs defined in the configuration, only 10 were in use. Eight of these ACLs were assigned inbound to the interfaces of the following zones: Corp, α-DMZ1, β-DMZ1, γ-DMZ, α-DMZ2, β-DMZ2, α-WAN and MZ. Two ACLs were assigned outbound on the interfaces of α-DMZ1 and β-DMZ2. Once assigned to firewall interfaces, the default security-level based filtering behaviour is overridden by the ACLs [30].

There is an extended ACL, acl_corp_in assigned inbound on Corp's firewall interface. The ACL plays an important role as it restricts outbound traffic from Corp to SCADA and α-DMZ1. Without this ACL, *all-IP* traffic is enabled outbound from the Zone due to the security-levels assigned.

There is also an ACL, acl_α_wan_in assigned inbound on α-WAN's firewall interface. The ACL restricts traffic from α-WAN to α-DMZ1, α-DMZ2 and Old-WAN based VPN clients.

Similar ACLs are assigned on the firewall interfaces of the high, medium and low security DMZs to restrict access to and from their shared servers.

The firewall utilised NAT on all traffic traversing a high-security (*i.e.,* inside) interface to a low-security (*i.e.,* outside) interface. Thus, traffic that did not match

a specified NAT rule was blocked by the firewall [30]. NAT-exemptions were used to exclude certain traffic from having to undergo NAT. These exemptions were provided via ACLs.

Interfaces at the same security level and traffic traversing from outside to inside interfaces were not required to use NAT to communicate, since dynamic-NAT or Port Address Translation (PAT) was not configured on any of the interfaces.

The *stateful* nature of the Cisco ASA firewall permitted all legitimate return traffic.

The firewall also had an active/standby fail-over configuration enabled via a dedicated Ethernet link. The standby unit is also accessed only via the primary unit, so both are managed as one using a single Firewall-Zone.

Basic threat detection was also enabled on the firewall. This captured ACL statistics, packet denial rates and connection limit exceeds. This threat data was reported to a remote Syslog server located in `Corp` which had *info-level* traps enabled. SNMP messages from the firewall were also sent to a NMS on authentication, link-up and link-down events.

The configurations also enabled FW1 to be managed from `Corp` via Telnet. Also, only static routes were in use.

Results and Best Practice Implications

Figure 3.17 shows the Zone-Firewall model generated by parsing the configuration data. In generating the model, we uncovered five gateways (GW1-GW5). The Zone-Firewall model falls into the category of a 'single firewall employing demilitarised zones' as per [21]. The model complies with the critical requirement that the SCADA-Zone should not be directly connected to the Internet-Zone (reachable only via gateway GW2). Figure 3.18 shows the corresponding Zone-Conduit model including Firewall-Zones (FWZs).

Security Violations: The SUC employed less-secure VLANs to separate Enterprise-network traffic from SCADA-network traffic. Industry recommends against such virtual segregation of traffic for reasons already described.

Also, direct communication between the SCADA-Zone and the Internet-Zone was explicitly enabled. Industry strictly recommends against this. In addition, all-TCP traffic was explicitly enabled inbound to the SCADA-Zone from `UZ5` (Figure 3.19(c)). Such a broad rule allowed entry of unsafe FTP, HTTP and Telnet protocols to the SCADA-Zone. All-IP traffic was also (explicitly) enabled inbound to the SCADA-Zone from α-`DMZ1`, with similar adverse effects.

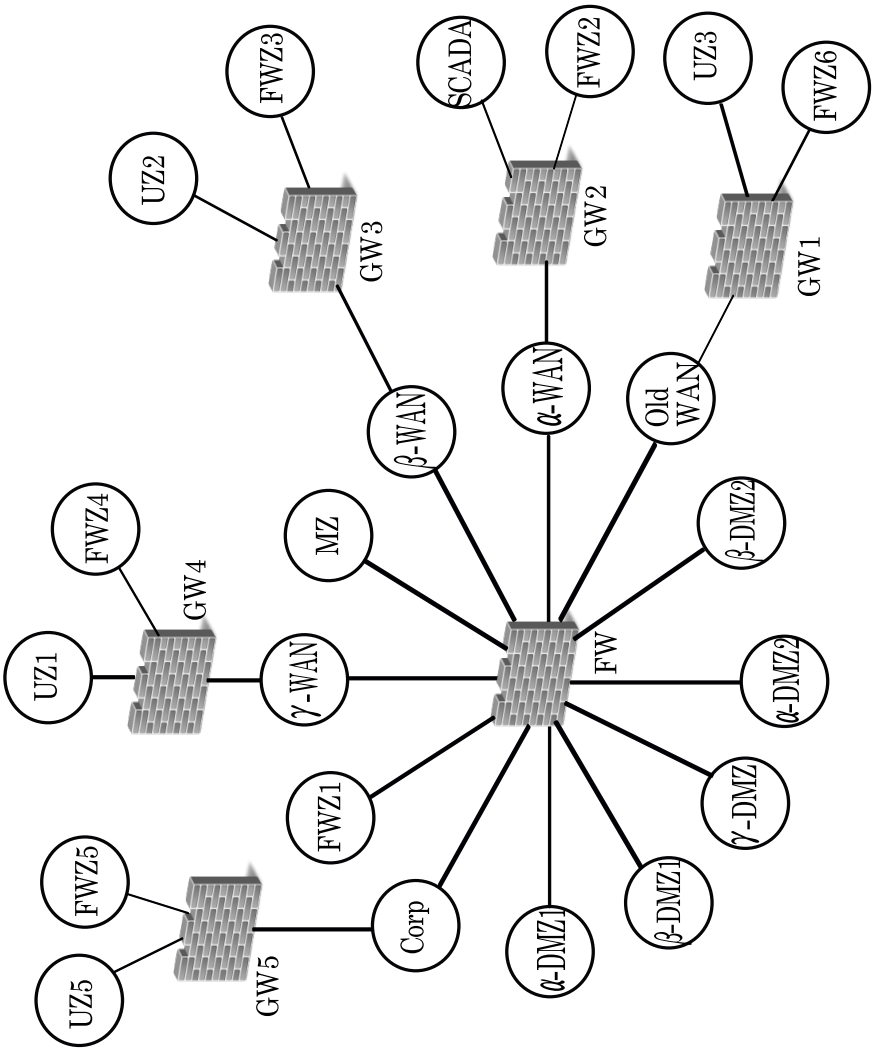


Figure 3.17: Zone-Firewall model including gateways of SUC 2.

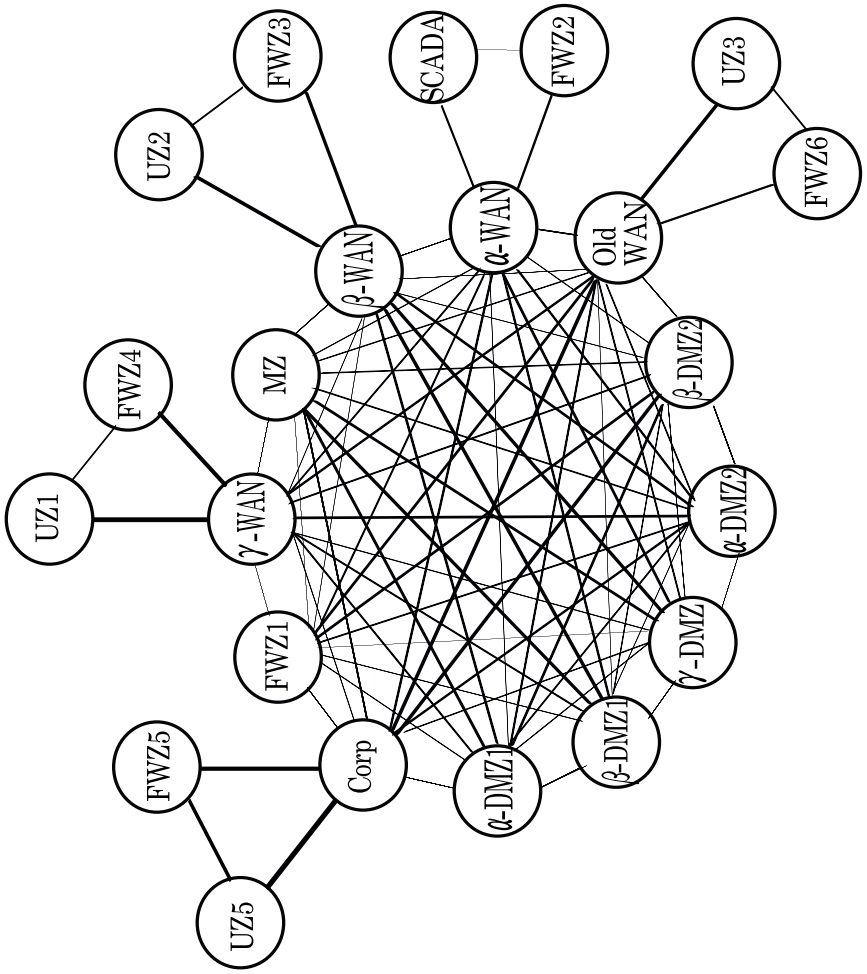
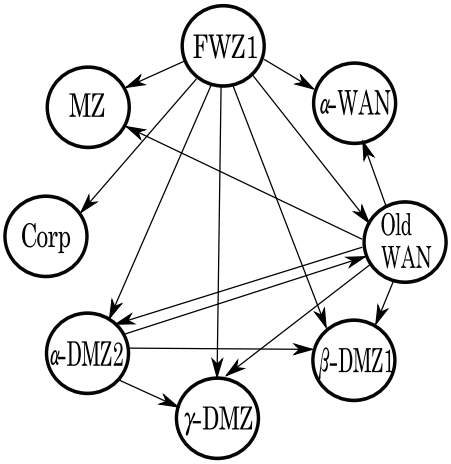
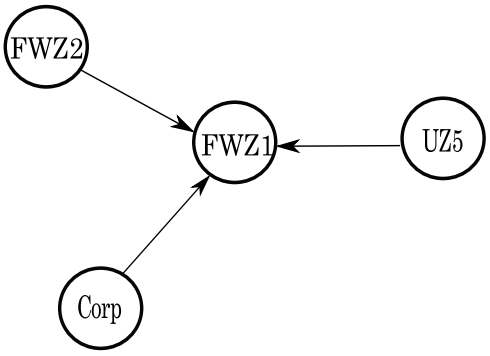


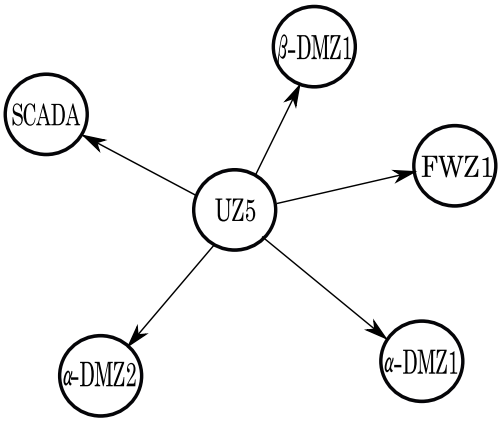
Figure 3.18: Zone-Conduit model of SUC 2.



(a) Generic IP (implicit).



(b) Telnet (implicit).



(c) Generic TCP (explicit).

Figure 3.19: Service-flow views of generic IP, Telnet and generic TCP traffic.

Security levels must be assigned to comply with best practices. So, `Corp`'s interface should have been assigned a lower trust level than α`-WAN`'s interface and not vice versa. Doing so, also enforces defence-in-depth. For instance, if the ACLs on the interfaces of α`-WAN` and `Corp` were removed by accident, correct security levels would have continued to protect the SCADA-Zone by only allowing IP traffic flow to be initiated from the more trusted α`-WAN` to `Corp`.

In fact, the service flows that stem from the security-levels assigned to the firewall interfaces are depicted in Figure 3.19(a). They show that all-IP traffic flow from `Corp` to α`-WAN` is allowed and α`-WAN` provides access to `SCADA`.

HTTPS was also enabled between `MZ`, `FWZ1` and α`-DMZ1`, but *excluded* from the SCADA-Zone. This safe protocol is encouraged by [21] to carry Web traffic to and from the SCADA-Zone, but was not used.

Table 3.3: Inter-ACL interactions summary for SUC 2

ACL1	ACL2	Interaction	Count
acl_β_dmz1_in	acl_α_dmz1_out	shadow	1
acl_α_wan_in	acl_α_dmz1_out	generalisation	12
acl_α_wan_in	acl_α_dmz1_out	shadow	4
acl_γ_dmz_in	acl_α_dmz1_out	shadow	1

The firewall configurations also enabled firewall management over insecure Telnet (Figure 3.19(b)) which lacks support for message encryption and strong authentication.

NAT-control was also incorrectly used. The feature ensures traffic flows have their private IP addresses translated to public ones when traversing from a high-security (*i.e.,* inside) firewall interface to a low-security (*i.e.,* outside) interface [30]. Given there were no explicit address translations (*i.e.,* through dynamic NAT or PAT) but only a few NAT-exemptions; the feature was misused to achieve a secondary effect: traffic filtering. Doing so, may be convenient, but these secondary effects do not construct robust security defences. Industry recommends using ACLs instead of NAT-control for reliable traffic filtering [30].

There were 14 ACL rules with invalid source and destination IP addresses, wasting configuration space. There were also 10 ACL rules that were either assigned to the incorrect firewall, firewall interface or direction, wasting further configuration space.

But most importantly, none of these ACL rules actually enforced their intended policies. So, they only contributed to the appearance of firewall security. Lack of human comprehensibility in the lengthy firewall configuration prevents an administrator from easily scrutinising this absence of security.

Configuration inefficiencies: The Parser also found two intra-ACL interactions within acl_α_ DMZ1_out. These were generalisations that consisted of a specific narrow rule in the ACL with an overlapping, broad rule further down the list.

There were also 18 inter-ACL interactions involving four ACLs, as summarised in Table 3.3. These consisted of 12 generalisations and six shadowed-rules. These inter- and intra-ACL interactions hindered maintaining a concise and hence clear firewall configuration.

3.2.4 SUC 3

The dataset we examined in this SUC was extracted in October 2011.

The Network in Detail

The SUC used two Cisco PIX version 7.2 firewalls. One firewall (FW1) consisted of 131 Lines of Code (LoC) with four ACLs averaging seven conventional rules each. The other firewall (FW2) consisted of 120 LoC with four ACLs that averaged four conventional rules each.

Figure 3.20 depicts how the Cisco PIX firewalls are deployed in the SUC. Firewall FW1 physically separates the SCADA network from the Enterprise network while FW2 facilitates a backup Disaster Recovery (DR) site.

FW1 has three physical interfaces operational, each pointing to the subnets below. All subnets except the DMZs were configured to host up to 254 hosts. Each DMZ accommodated up to 62 hosts.

The Enterprise network (`Corp`): Allows business application and Internet access.
The SCADA network (`SCADA`): Enables networked access to plant equipment.
Demilitarised Zone1 (`DMZ1`): Low-security DMZ that hosts a data historian and a Webserver for receiving updates from the SCADA network.

FW2 has two physical interfaces operational, each points to the subnets below

The Demilitarised Zone2 (`DMZ2`): Low-security DMZ that hosts a backup data historian and a Webserver.
The Disaster-Recovery SCADA network (`DR-SCADA`): Backup SCADA network that hosts auxiliary SCADA servers.

Varying security levels (from 0 to 100) are assigned to each firewall interface (Figure 3.20). With a security level of 100 assigned `SCADA` and `DR-SCADA` are classified as the highest-trust zones. These zones cannot be accessed by a lower-trust zone by default. `Corp`, `DMZ1` and `DMZ2` are classified as the lowest-trust zones in the group with a security level of 0 assigned to their firewall interfaces. All-IP traffic flow is enabled between these zones by default by a special Cisco command.

The firewalls use Cisco extended ACLs. There is an ACL, `corp_access_in` assigned inbound on `Corp`'s firewall interface. It enables enterprise-user access to the `DMZ1` hosted Web server.

Two ACLs: `dmz1_access_in` and `dmz1_access_out` are assigned inbound and outbound (respectively) on the firewall interface of `DMZ1`. The first ACL enables the `DMZ1` hosted data historian to access the `SCADA` hosted servers. The second allows all-TCP traffic flow from `SCADA` to `DMZ1`.

An ACL, `scada_access_in` is assigned inbound on `SCADA`'s firewall interface. The ACL enables all-TCP traffic flow from `SCADA` to the `DMZ2` hosted backup data historian and Webserver.

There are also two ACLs, `dmz2_access_in` and `dmz2_access _out` assigned inbound and outbound (respectively) on `DMZ2`'s firewall interface. `dmz2_access_in` allows the `DMZ2` hosted backup data historian and Webserver to access `DR-SCADA`

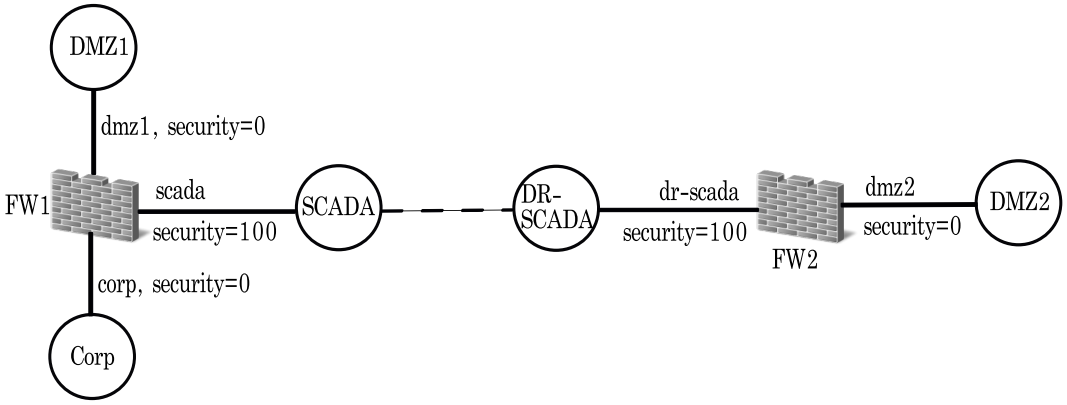


Figure 3.20: SUC 3.

and the live historian and Webserver located in DMZ1. dmz2_access_out allows DR-SCADA users to access the backup servers in DMZ2.

There is also an ACL, dr_scada_access_in assigned inbound on DR-SCADA's firewall interface. It enables all-TCP traffic flow from DR-SCADA to the DMZ2 hosted data historian and Webserver. The ACL also allows all-TCP traffic flow from these servers to a remote network residing off DR-SCADA's gateway.

OSPF routing was enabled on both firewalls; each firewall interface belonged to a distinct OSPF area. A default route was also present on each firewall configuration revealing two distinct gateways.

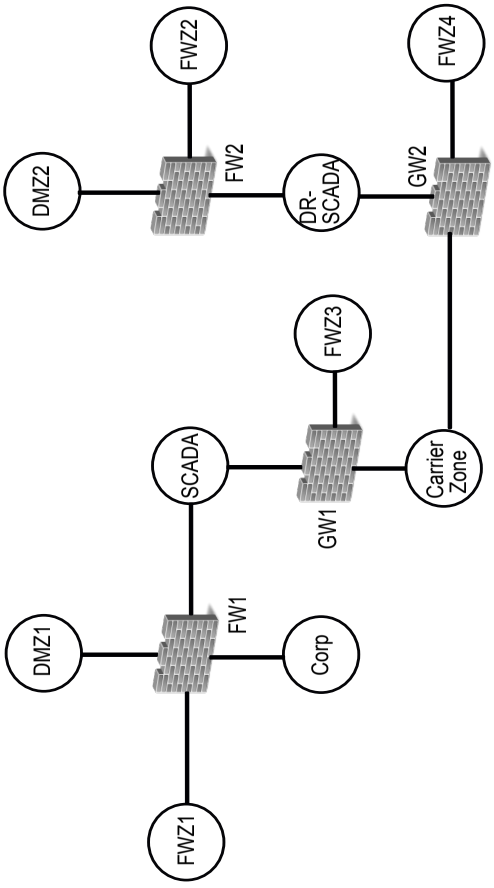
The configurations also allowed FW2 to be managed from DR-SCADA via SSH, HTTP and Telnet while FW1 could be managed from SCADA via SSH and HTTP only.

FW1 was also configured as a DHCP server for clients connecting via the interface of SCADA.

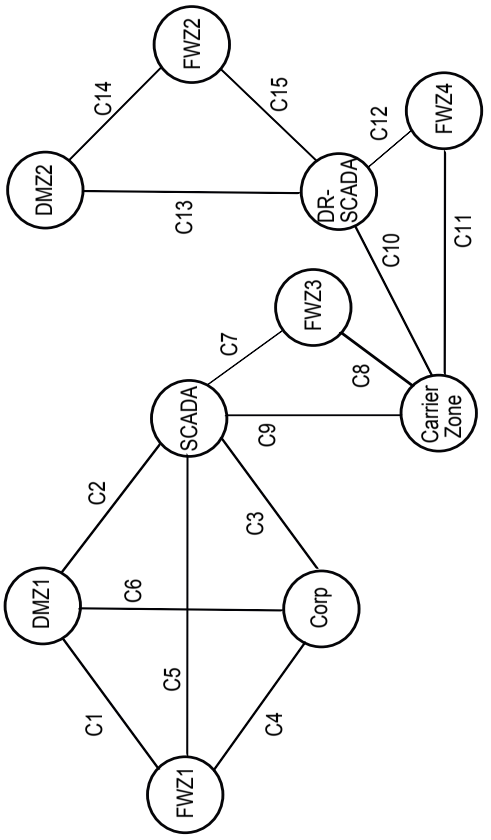
Firewall-generated syslog messages were logged internally in both the firewalls.

Results and Best Practice Implications

The zone-firewall model generated by parsing the configuration data for the SUC is shown in Figure 3.21(a). According to the industry recommend firewall architectures in [21], this model falls into the category of a 'single firewall employing a demilitarised zone'. Figure 3.21(b) shows the corresponding Zone-Conduit model generated. It includes the Firewall-Zones (FWZ1, FWZ2) and a Carrier-Zone.

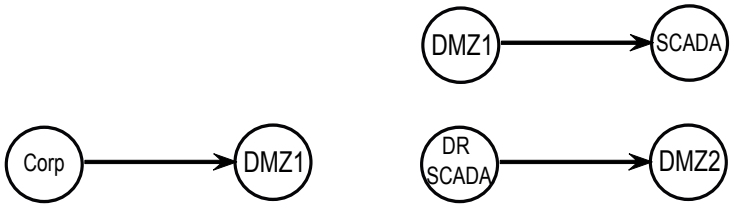


(a) Zone-Firewall model including gateways.



(b) Zone-Conduit model of (a).

Figure 3.21: Security models of SUC 3.



(a) HTTP traffic flow. (b) NTP traffic flow.

Figure 3.22: Explicit service-flow views of HTTP and NTP traffic for SUC 3.

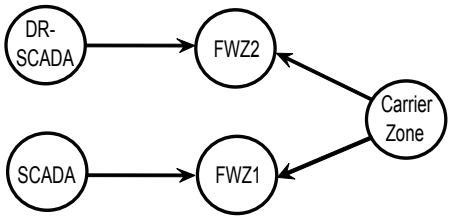


Figure 3.23: Implicit HTTP service-flow view for SUC 3.

An example explicit service-flow view of HTTP traffic is shown in Figure 3.22(a). It shows that the service is only enabled between `Corp` and `DMZ1` and does not involve the SCADA-Zone as recommended by best practices in [21].

The SUC also enabled redundancy of critical infrastructure components by the use of a backup SCADA network (*i.e.,* `DR-SCADA`).

Security Violations: All-TCP traffic was explicitly enabled inbound to `DR-SCADA`. This meant insecure protocols such as DNS, FTP and HTTP were enabled inbound to this protected SCADA-Zone, exposing it to cyber attacks. We also found that insecure NTP traffic was enabled inbound to `SCADA` from `DMZ1` (Figure 3.22(b)).

The service-flow depicted in Figure 3.23 describes implicitly enabled HTTP traffic for firewall management purposes. HTTP does not support encryption and is discouraged by industry for secure firewall management.

Direct transit of RDP traffic was also explicitly enabled between `SCADA` and less-trusted `Corp`, increasing the vulnerability of the former [21, 117].

The firewall software version (Cisco PIX 7.2) was over five years old and out of maintenance at the time the configurations were extracted. Many publicly listed security vulnerabilities were available for this software version since its release. For instance, this software allowed remote attackers to render a DoS attack using H.323 packets [35].

There were seven ACL rules that were either assigned to the incorrect firewall, firewall interface or direction, wasting configuration space. Moreover, the

Table 3.4: Inter-ACL interactions summary for SUC 3

ACL1	ACL2	Interaction	Count
acl_scada_access_in	acl_dmz1_access_out	generalisation	2
acl_dr_scada_access_in	acl_dmz2_access_out	generalisation	4

intended policy of these ACL rules were not enforced. So, they only contributed to the appearance of firewall security.

Configuration Inefficiencies: Configuration space was further wasted due to only seven of the eight ACLs defined across both firewall configurations, being used.

FireP also identified six inter-ACL interactions between the ACLs, as summarised in Table 3.4. These interactions were all generalisations, indicating the overlapping nature of the policies of firewalls configured in series (FW1, FW2). There were no intra-ACL interactions found across the seven ACLs used.

3.2.5 SUC 5

The dataset we examined for System Under Consideration (SUC) 5 was extracted in April 2015. SUC 5 is shown in Figure 3.24. It employs a single Cisco Adaptive Security Appliances (ASA) 5510 firewall. It contained 853 lines of code and 12 ACLs averaging 677 conventional rules each. There is also an active/standby failover unit configured as a backup firewall.

The Network in Detail

The firewall has seven physical interfaces operational, pointing to the subnets described below. All subnets except SCADA are configured to accommodate up to 254 hosts. SCADA is configured to accommodate up to 1022 hosts.

The Enterprise network (Corp): Allows business application and Internet access.
The production SCADA network (SCADA): Enables networked access to live plant equipment.
The test SCADA network (SCADA Test): Enables networked access to test plant equipment. All applicable software updates and patches are first applied to this test network, prior to rolling out to SCADA and SCADA Sim networks.
The SCADA network simulator (SCADA Sim): Hosts a full-replica emulator of the production SCADA network and is used for operator training.
The Management network (Mgt): Enables firewall management.
The remote access network (RA): Allows enterprise-network users and Internet users (reachable via Corp gateway) to remotely login to RA using a Cisco VPN and two-factor authentication. Also facilitates RA users with remote access of SCADA, Misc, SCADA Sim and SCADA Test.

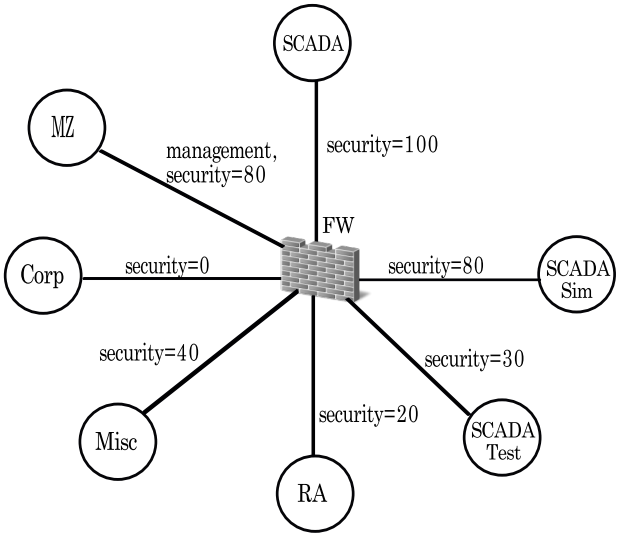


Figure 3.24: SUC 5.

The miscellaneous-controls network (`Misc`): Hosts Human-Machine Interface (HMI) stations for controlling auxiliary-plant equipment such as ash and dust collectors.

Varying security levels (from 0 to 100) are assigned to each firewall interface (Figure 3.24). With a security-level of 100 assigned, `SCADA` is classified as the highest-trust zones. Doing so prevents the zone from being accessed by a lower-trust zone in absence of overriding ACLs.

`MZ` and `SCADA Sim` are classified as the second highest-trust zones with security level of 80 assigned to their respective firewall interfaces. `Corp` is the lowest-trust zone in the group with a security level of 0 assigned.

The firewall uses Cisco extended ACLs with object-group based rules. Of the 12 ACLs defined in the configuration, only nine were in use. Seven of these ACLs were assigned inbound to the firewall interfaces of the following zones: `Corp`, `management`, `SCADA`, `SCADA Sim`, `SCADA Test`, `RA` and `Misc`. Two ACLs were used to divert traffic to the Intrusion Prevention System (IPS) module of the ASA firewall and to export flows that matched pre-defined traffic patterns to Netflow collectors.

An extended ACL, `scada_in` is assigned inbound on `SCADA`'s firewall interface. The ACL plays an important role as it restricts traffic from `SCADA` to `RA`, `SCADA Test` and `Corp`. For instance, The ACL enables DNS queries from `SCADA` to `Corp` hosted DNS servers. Without this ACL, *all-IP* traffic is enabled from `SCADA` due to the security-levels assigned.

Another ACL, `ra_in` is assigned inbound on `RA`'s firewall interface. The ACL

restricts traffic from RA to SCADA, Misc, Corp and SCADA Test. It enables remote access to SCADA using RDP and PCAnywhere protocols. The ACL also facilitates SMB-based file sharing with Corp users.

There is also an ACL, corp_in assigned inbound on Corp's firewall interface. The ACL restricts traffic from Corp to SCADA, SCADA Test, SCADA Sim and the firewall. For instance, it enables MS-SQL based access of the SCADA hosted data historian. The ACL also allows the firewall to be managed from Corp using HTTP and SSH. Similar ACLs are assigned on the firewall interfaces of SCADA Test, SCADA Sim, Misc and MZ interfaces to restrict access to and from their hosts.

The SUC employed static routing.

Results and Best Practice Implications

The Zone-Firewall model generated by parsing the configuration data for the System Under Consideration (SUC) is shown in Figure 3.25(a). In generating the models, we uncovered two gateways (GW1, GW2). The model falls in to the category of 'a single firewall employing demilitarised zones' as per [21]. The model also complies with the requirement that the Internet-Zone should not be directly connected to the SCADA-Zone (reachable only via gateway GW1). Figure 3.25(b) shows the corresponding Zone-Conduit model, including the Firewall-Zones.

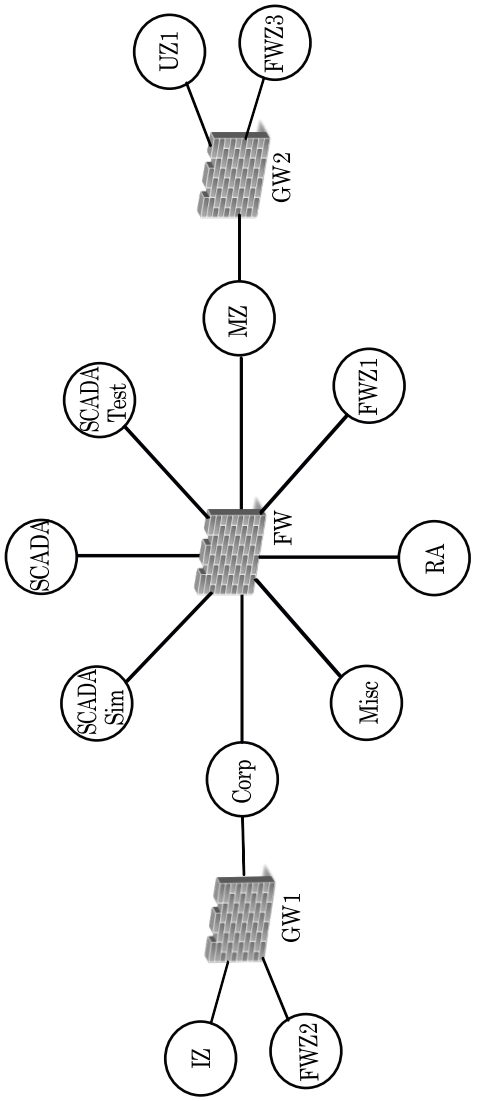
An example explicit service-flow view of SMTP traffic is shown in Figure Figure 3.26. SMTP is only enabled outbound from SCADA as recommended by industry guidelines in [21].

An example implicit service-flow view for the SUC is shown in Figure 3.27. The implicit flows stem from the security levels assigned on the firewall interfaces. By default, all-IP services can be initiated by the most trusted SCADA-Zone to all other zones. A highest security level of 100 assigned to the firewalls management interface facilitates this.

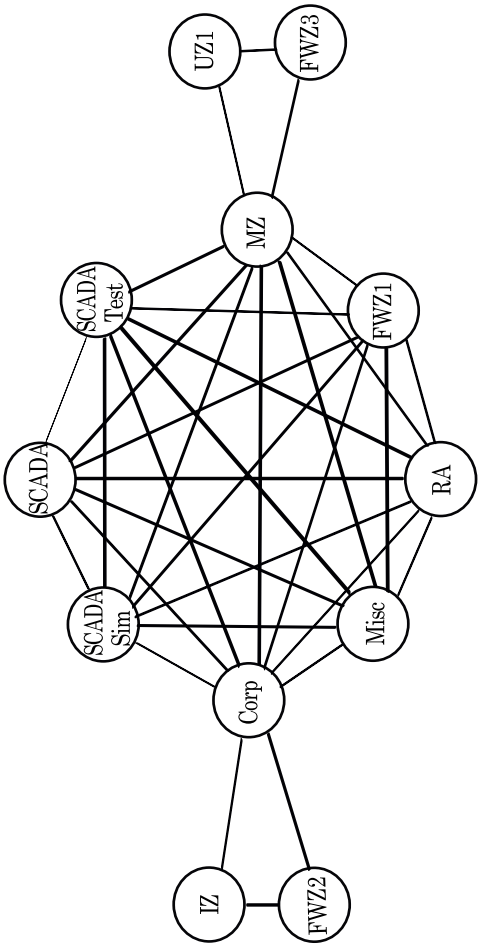
Security Violations: Remote access to the SCADA-Zone was not secured as per best practices. For instance, once authenticated on to RA, users were not required to use two- or multi-factor authentication to access SCADA. Instead, a simple shared password was used. Some enabled remote-access protocols (*e.g.,* RPC) were vulnerable to attack and their use was discouraged by the industry [117].

We also found insecure DNS traffic allowed inbound to SCADA. DNS is often used in carrying out DoS attacks and if successful could cause compromise or failure of SCADA equipment, leading to potentially catastrophic outcomes.

FW1 could be managed from Corp using HTTP protocol. HTTP lacks support for data encryption and is discouraged by industry for secure firewall management. Moreover, managing a SCADA firewall from a less-trusted Enterprise-Zone is a clear breach of industry best practices [21].



(a) Zone-Firewall model including gateways.



(b) Zone-Conduit model of (a).

Figure 3.25: Security models of SUC 5.

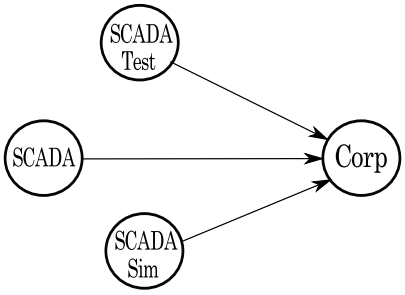


Figure 3.26: Explicit service-flow view of SMTP traffic for SUC 5.

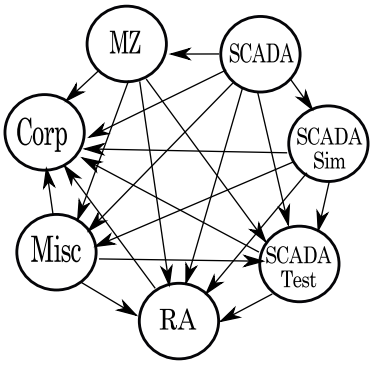


Figure 3.27: Implicit service-flow view of IP traffic for SUC 5.

Direct transition of DNS and LDAP traffic was explicitly enabled between `SCADA` and `Corp`. SMB based file sharing was also enabled between these two zones. Industry recommends against enabling both as they expose a SCADA-Zone [21, 117].

The firewall software version (ASA Version 8.2) was over three years old when the configuration was extracted. Many publicly listed security vulnerabilities were available since its release. For instance, the software had a memory-leak vulnerability that allowed remotely authenticated users to carry out DoS attacks [35]. It is imperative that the firewalls protecting SCADA networks are up-to-date, given that regular patching and upgrading of systems on these networks is less practical.

ACL-rule comments were present in the SUC's firewall configurations, but, some of these comments were incorrect or obsolete; their intended traffic flow mismatched that implemented by the rule. This is a common problem where comments are used to document policies.

There were also 17 ACL rules that were either assigned to the incorrect fire-

wall, firewall interface or direction, wasting configuration space. Moreover, the intended policy of these ACL rules were not enforced. So, they only contributed to the appearance of firewall security. The lack of human comprehensibility in the lengthy firewall configuration, prevents an administrator from detecting this absence of security by simple inspection.

Configuration Inefficiencies: Configuration space was further wasted due to only nine of the 12 ACLs defined in the firewall configuration, being used.

FireP also identified 128 intra-ACL interactions across the nine ACLs used. These consisted of 11 generalisations, 116 shadowed-rules and one conflicts. Most of these generalisations and shadows were caused by the use of 'any' as source and/or destination. These intra-ACL interactions led to expansive configurations.

3.2.6 SUC 6

The dataset we examined for SUC 6 was extracted in July 2015. The SUC is shown in Figure 3.28 and employs a single Cisco ASA 5510 firewall. The configuration consisted of 900 lines with eight ACLs averaging 1034 conventional rules each. The firewall physically separates the enterprise network from the SCADA network.

The Network in Detail

The primary firewall has four physical interfaces operational, three of these were divided into sub-interfaces supporting six VLANs with 802.1Q trunking enabled. Each of the interfaces pointed to the subnets below. All subnets except `SCADA Test` were configured to accommodate up to 254 hosts. `SCADA Test` was configured to host 65534 hosts.

The Demilitarised Zone (`DMZ`): Low security DMZ that enables PCAnywhere and Telnet access to its hosts from `RA`.

The following six subnets are analogous to those discussed earlier in SUC 5 in terms of their function: the Enterprise network (`Corp`), the Remote-Access network (`RA`), the Management network (`MZ`), the Production SCADA network (`SCADA`), the Test SCADA network (`SCADA Test`) and the Miscellaneous-controls network (`Misc`).

Security levels are also assigned on the firewall interfaces analogous to SUC 5: SCADA is classified as the highest-trust zone, `Corp` is classified as the lowest-trust zone and `MZ` is classified as a medium-trust zone.

The firewall uses Cisco extended ACLs with object-group based rules. Of the eight ACLs defined in the configuration, only seven were in use. These were assigned inbound on the interfaces of the following zones: `Corp`, `RA`, `SCADA`, α`-DMZ`, `SCADA Test`, `MZ`, `Misc`.

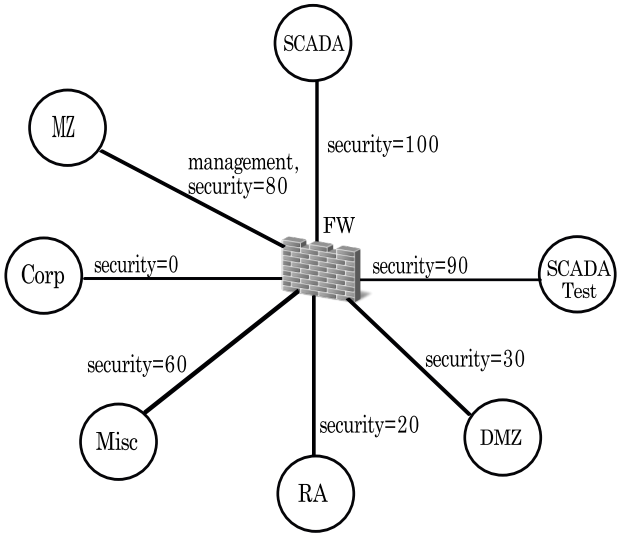


Figure 3.28: SUC 6.

There is an extended ACL, `corp_access_in` assigned inbound on `Corp`'s firewall interface. The ACL permits `Corp` users with FTP-based file transfer and RDP-based remote access to `RA` hosts. The ACL also permits NTP and DNS traffic flow to `SCADA` and `SCADA Test`.

Another extended ACL, `scada_access_in` assigned inbound on `SCADA`'s firewall interface. The ACL permits ICMP traffic flow from `SCADA` to all other zones. It also enables DNS, SMTP and NTP traffic to `Corp`. The ACL enables access of the SMB file server in `Corp`.

There is also an ACL, `ra_access_in`, assigned inbound on `RA`'s firewall interface. The ACL permits RDP-based remote access to `SCADA` and `SCADA Test`. It also allows HTTP access to the `SCADA` hosted historian. ICMP traffic flow and SMB based file sharing is also enabled by the ACL to `SCADA Test`.

Similar ACLs are assigned on the firewall interfaces of `MZ`, `SCADA Test`, `Misc` and `DMZ` to enable restricted access to and from their hosts.

The firewall also had an active/standby fail-over configuration enabled via a dedicated Ethernet link. The standby unit is also accessed only via the primary unit, so, both are managed as one using a single Firewall-Zone.

Static routing was used rather than dynamic routing protocols such as OSPF or EIGRP. The firewall could be managed from `Corp` using HTTP and SSH.

Basic threat detection was also enabled to capture ACL statistics and packet denial rates. This threat data was reported to a Syslog server located in `Corp` which had *info-level* traps enabled. SNMP messages were also sent to a NMS located in `Corp`, on authentication, link-up and link-down events.

Results and Best Practice Implications

Figure 3.29(a) depicts the Zone-Firewall model generated by parsing the configuration data. We uncovered three gateways (GW1-GW3) in generating this model. The model falls in to the category of a 'single firewall employing demilitarised zones' as per [21]. Figure 3.29(b) shows the corresponding Zone-Conduit model including Firewall-Zones (FWZs).

The model adheres to the critical requirements; (i) the SCADA-Zone must not be directly connected to the Internet; and (ii) SCADA traffic must be physically separated from the Enterprise traffic.

The service flows that stem from the security-levels assigned to the firewall interfaces are depicted in Figure 3.30. It shows that no IP traffic flow is enabled by default into `SCADA` from any other zone. Thus, the security-levels assigned, comply with best-practices. They reduce the vulnerability of the SCADA-Zone and enforce defence in depth.

For instance, even if the ACLs on the interfaces of `SCADA` and `Corp` were removed by accident, the correctly assigned security levels continue to protect the SCADA-Zone by only allowing IP traffic flow to be initiated from the more trusted `SCADA` to less-trusted `Corp` and not vice versa.

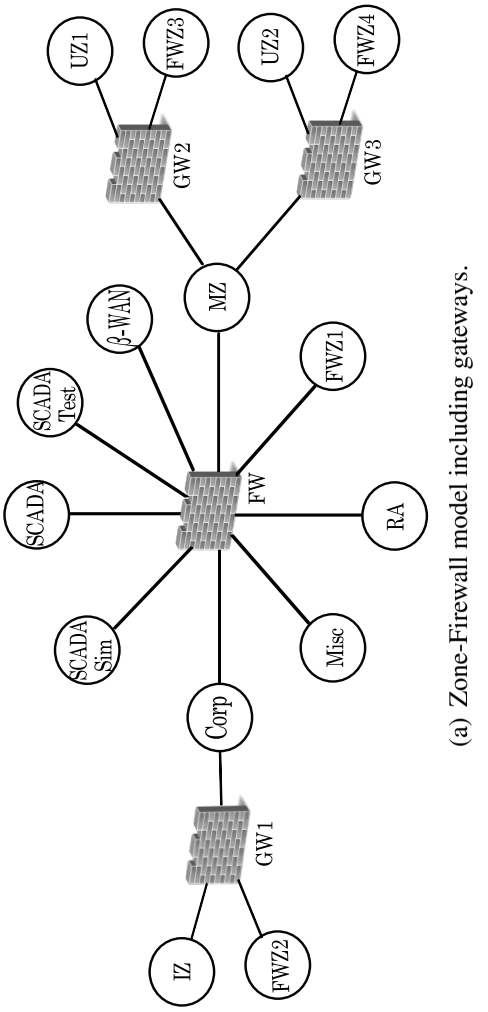
Security Violations: Unsafe protocols such as HTTP, DNS and NTP were enabled inbound to the SCADA-Zone, making the zone highly vulnerable.

The firewall configuration also allowed the firewall to be managed over insecure HTTP (Figure 3.31), which is known to carry attacks and worms. Moreover, the firewall could be managed from the least-trusted Enterprise-Zone; a clear violation of the best practices.

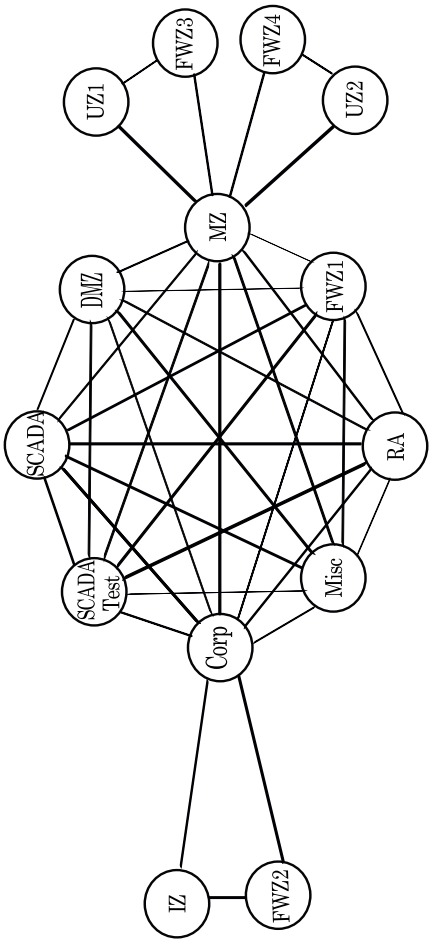
HTTPS was also enabled between `Corp`, `RA` and `SCADA Test`, but excluded the SCADA-Zone. This safe protocol is encouraged by best practices to carry Web traffic to and from the SCADA-Zone, but was not used.

There were eight ACL rules with invalid source and/or destination IP addresses. The Parser also detected 11 ACL rules that were assigned to the incorrect firewall, firewall interface or direction. These ACL rules wasted configuration space. But, most importantly, the intended policies of the ACL rules were not enforced.

Configuration Inefficiencies: There was one unused ACLs that further wasted configuration space. In addition, there were 33 intra-ACL interactions found across the ACLs used that lead to expansive configurations.



(a) Zone-Firewall model including gateways.



(b) Zone-Conduit model of (a).

Figure 3.29: Security models of SUC 6.

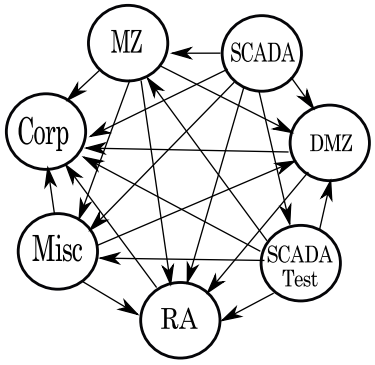


Figure 3.30: Implicit Service-flow View of IP traffic for SUC 6.

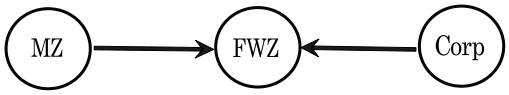


Figure 3.31: Implicit Service-flow View of HTTP traffic for SUC 6.

3.2.7 SUC 4

The SUC is shown in Figure 3.32. It employs a single Cisco ASA 5500 series firewall. The firewall configuration was extracted in March 2011. It contained 819 lines of code and three ACLs averaging 80 conventional rules each.

The Network in Detail

The firewall has four physical interfaces operational, pointing to the subnets described below. All subnets except `Corp` are configured to accommodate up to 254 hosts. `Corp` is configured to accommodate up to 1022 hosts.

The Enterprise network (`Corp`): Allows business application and Internet access.
The SCADA network (`SCADA`): Enables networked access to plant equipment.
The Demilitarised Zone (`DMZ`): Hosts business critical shared servers such as data historians that are accessed by both Enterprise- and SCADA-network users.
The Management network (`Mgt`): Enables firewall management.

Varying security levels (from 0 to 100) are assigned to each firewall interface (Figure 7.1). With a security-level of 100 assigned, `Mgt` and `SCADA` are classified as the highest-trust zones. Doing so, prevents these zones from being accessed by a lower-trust zone by default.

`Corp` is the second highest-trust zone with a security level of 60 assigned to its interface. `DMZ` is as the lowest-trust zone in the group with a level of 40 assigned.

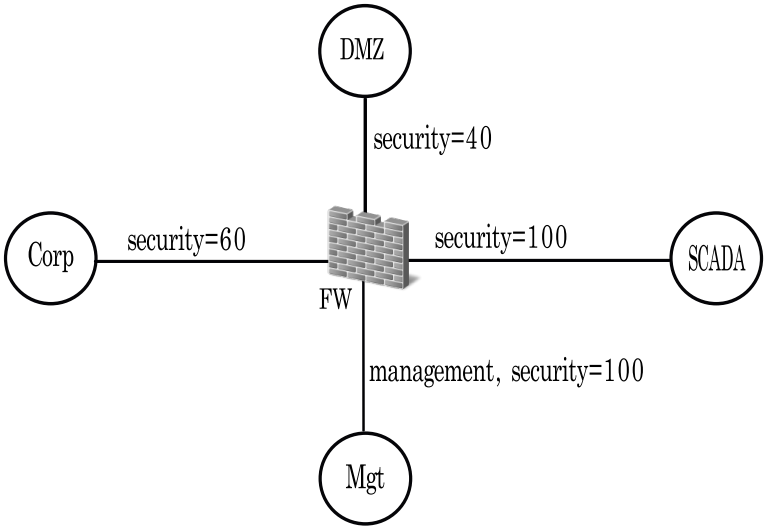


Figure 3.32: SUC 4.

The firewall uses Cisco extended ACLs with conventional rules. There is an ACL, `corp_in` assigned inbound on `Corp`'s firewall interface. The ACL enables all ICMP traffic from `Corp` to the other zones. It also enables DNS queries originating from `Corp` to the servers located in `SCADA`.

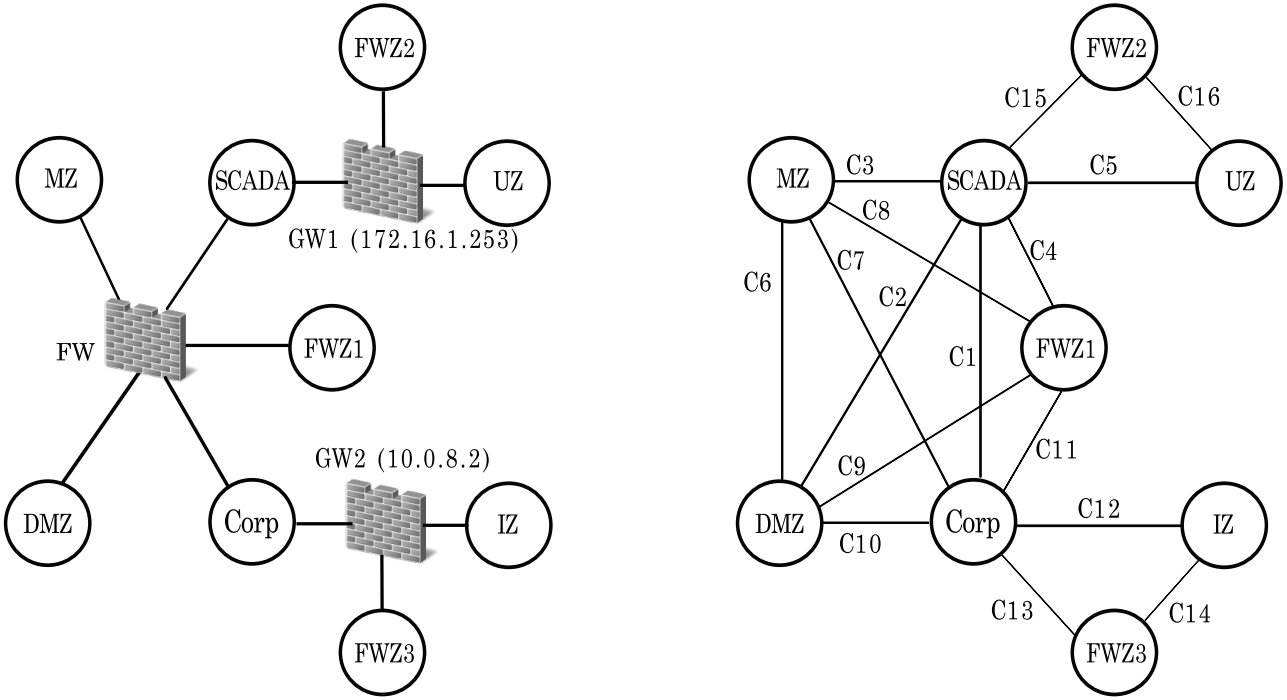
There is another extended ACL, `scada_in` assigned inbound on `SCADA`'s firewall interface. The ACL enables time synchronisation between `SCADA` hosted servers and the `Corp` hosted timeserver using NTP. It also enables email access between `SCADA` hosted servers and the `DMZ` hosted SMTP mail server. The ACL also permits all-ICMP traffic outbound from SCADA.

A third extended ACL: `dmz_in`, is applied inbound on the interface of `DMZ`. It permits several services for the `DMZ` hosted data historian: DNS queries to the `Corp` hosted DNS server, time synchronisation to the `Corp` hosted timeserver using NTP and Web access to the `SCADA` hosted servers. The ACL also enables email access between the `DMZ` and `Corp` hosted mail servers.

The SUC employed static routing.

Results and Best Practice Implications

The Zone-Firewall model generated by parsing the configuration data for the SUC is shown in Figure 3.33(a). The model falls in to the category of 'a single firewall employing a demilitarised zone' as per [21]. The model also complies with the requirement that the Internet-Zone should not be directly connected to the SCADA-Zone. Figure 3.33(b) shows the corresponding Zone-Conduit model which includes multiple Firewall-Zones.



(a) Zone-Firewall model including gateways.

(b) Zone-Conduit model of (a).

Figure 3.33: Security models of SUC 4.

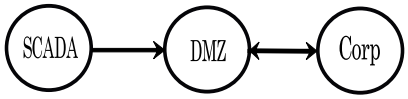
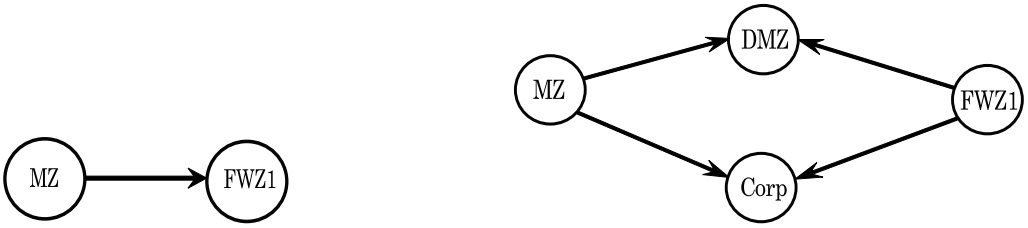


Figure 3.34: Explicit service-flow view of SMTP traffic for SUC 4.



(a) HTTP flow. (b) Generic IP flow.

Figure 3.35: Implicitly enabled HTTP and Generic IP traffic for SUC 4.

An example explicit service-flow view of SMTP traffic is shown in Figure 3.34. SMTP is only enabled outbound from `SCADA` as recommended by industry guidelines in [21]. Similar results were observed with Distributed Component Object Model (DCOM) protocol, where only outbound connections from `SCADA` were allowed.

Example implicit service-flow views generated for the SUC is shown in Figure 3.35. Figure 3.35(b) describes generic IP traffic enabled via security levels assigned to the firewall interfaces. By default, all-IP services are allowed to be initiated by the most trusted `MZ` to the less-trusted `DMZ` and the `Corp`. A highest security level of 100 assigned to the firewalls management interface facilitates this.

Traffic flow between `MZ` and `SCADA` was disabled by the *equal* security levels assigned to their firewall interfaces (an additional Cisco configuration command is required to enable this flow but was not supplied).

Security Violations: Figure 3.35(a) depicts implicitly enabled HTTP traffic used for firewall management purposes. The protocol does not support encryption and is discouraged by industry for secure firewall management [21].

Direct transition of DNS, NTP traffic was explicitly enabled between `SCADA` and `Corp`, increasing the vulnerability of the latter [21, 117]. We also found insecure DNS and HTTP traffic enabled inbound to `SCADA`.

Comprehensible ACL rule comments were present in the firewall configurations. But these comments included incorrect or obsolete rule descriptions: their intended traffic flow mismatched that implemented by the rule.

Alarmingly, the SUC also did not employ backup firewalls for redundancy.

We also observed that HTTPS was only used for communications between `Corp` and `DMZ` was excluded from the SCADA-Zone. This safe protocol is encouraged by industry best practices in [21] to carry Web traffic to and from the SCADA-Zone, but here it was under utilised.

FireP identified 12 ACL rules with invalid source and/or destination IP addresses. There were also 30 ACL rules that were either assigned to the incorrect firewall, firewall interface or direction. These ACL rules wasted configuration space. But, most importantly, the intended policies of these ACL rules were not enforced. It is nearly impossible to identify this appearance of security due to the lack of human comprehensibility in the lengthy firewall configuration.

Configuration Inefficiencies: There were also seven intra-ACL interactions found across the ACLs used. These were generalisations that lead to expansive configurations.

3.3 Discussion

Table 3.5 and Table 3.6 summarise the security best-practice violations and the configuration inefficiencies found across the SUCs.

Security violations of critical infrastructure such as power and water could have a disastrous impact; either financially, environmentally or potentially even life-end limits [62, 117]. Understandably, we expect a network controlling a power station to have rigorous protection orders of magnitude greater than an enterprise IT network. But, our study of firewall configurations from seven production SCADA networks revealed otherwise: critical infrastructures are insecure now.

There are many flaws in firewall configuration that need to be fixed and current mechanisms do not work. For instance, Table 3.5 shows how every firewall configuration we studied allowed insecure protocols such as DNS inbound to the SCADA-Zone. None of the SCADA firewalls were securely managed; a grave error given that if the management of the firewall is compromised the entire SCADA network could be compromised. Moreover, in every case, policy was incorrectly assigned to the firewalls, creating an appearance of security. This false sense of security permits behaviour that would be more carefully scrutinised in an insecure domain. For instance, we might leave systems behind a firewall unpatched, confident in the blanket of security provided.

Identifying flaws in firewall configuration is a non-trivial task; configurations are lengthy, involves a high-volume of low level detail and have many potentially interacting components. To date, firewall vendors have concentrated on new and impressive features to create systems with as much or more complexity as the base firewall configurations.

Table 3.5: Summary of SCADA security best-practice violations found in the SUCs (✗ indicates a violation).

Best practice	SUC1	SUC2	SUC3	SUC4	SUC5	SUC6	SUC7
Employ industry-recommended firewall architecture	✓	✗	✓	✓	✓	✓	✗
Physical separation of Enterprise and SCADA traffic	✓	✗	✓	✓	✓	✓	✗
Backup firewalls available	✗	✓	✗	✗	✓	✓	✓
Secure remote access of SCADA network	✗	✗	✗	✗	✗	✗	✗
Secure firewall management	✗	✗	✗	✗	✗	✗	✗
Prohibit Internet-SCADA direct traffic transit	✗	✗	✓	✓	✓	✓	✓
Prohibit Enterprise-SCADA direct traffic transit	✗	✓	✗	✗	✗	✗	✗
Prohibit inbound insecure traffic (*e.g.,* DNS, HTTP) to SCADA	✗	✗	✗	✗	✗	✗	✗
Prohibit use of generic rules (*e.g.,* allow all-TCP)	✗	✗	✗	✗	✗	✗	✗
Clear documentation of ACL rules	✓	✓	✗	✓	✗	✗	✗
Firewall firmware is current	✓	✓	✓	✗	✗	✗	✗
Firewalls have active vendor support	✓	✓	✓	✓	✓	✓	✗

Table 3.6: Summary of firewall-configuration inefficiencies found in the SUCs (* refers to Cisco object-groups).

Inefficiency	SUC1	SUC2	SUC3	SUC4	SUC5	SUC6	SUC7
Obsolete ACLs	2	2	1	0	3	1	2
Invalid or obsolete ACL rules	70	14	7	12	17	8	31
Rule inconsistencies (*i.e.,* redundancies and conflicts)	170	20	6	7	128	33	34
Redundant object-groups*	0	0	0	0	2	1	3

These firewall features are often independent of each other and uncoordinated (Cisco Security-levels and ACLs are good examples). Existing firewall configuration platforms fail to apply coherent concepts and relationships throughout- *i.e.,* they lack *conceptual integrity*. In his software engineering book- *The Mythical Man-Month* [18], Fred Brooks emphasises the importance of conceptual integrity in building easy-to-use systems:

> "For a given level of function, however, that system is best in which one can specify things with the most simplicity and straightforwardness. Simplicity and straightforwardness proceed from conceptual integrity. Every part must reflect the same philosophies and the same balancing of desiderata. Every part must even use the same techniques in syntax and analogous notions in semantics. Ease of use, then, dictates unity of design, conceptual integrity."
>
> *Fred Brooks [18]*

Brooks highlights that it is essential to separate architecture from implementation, to facilitate conceptual integrity. Otherwise, inter-twined implementation details prevent the ability to maintain a set of coherent design ideas. Our case studies also inform us of this fact: decoupling policy (*i.e.,* architecture) from network topology (*i.e.,* implementation) is necessary to enable high-level policy specification. A good set of security abstractions is key to achieving this separation.

Implicit rules are nascent attempts by firewall vendors to provide high-level abstractions, but are too restrictive in that you cannot write flexible rules. For example, Cisco security levels allow quick and easy access between internal and external firewall interfaces, but lack the flexibility to specify detailed traffic restrictions. Hence the ACLs we see overriding this mechanism in practice.

For another example, Cisco object-groups aim to provide convenience and modularity in a firewall configuration. But, these groups do not facilitate construction or composition of disjoint policies that can be maintained by users from distinct policy subdomains (*e.g.,* Enterprise admins and SCADA engineers). A pragmatic firewall policy specification needs to support such modular policies.

The ANSI/ISA Zone-Conduit abstraction is also too ambiguous and allows alternate ways of defining zones and conduits to cater for business models. The abstraction is good when used by humans, but for automation we need clarity and precision.

A good abstraction is, therefore, a tussle between these two approaches. It should provide clear mapping between policies and networks, with some restrictions, but also the required amount of flexibility.

For instance, the standards allow $1 : n$ or $n : 1$ mapping between conduits, firewalls and policy. We argue that maintaining a $1 : 1$ mapping between policies and

conduits leads to a simple and useful abstraction for high-level policy specification. Otherwise the ambiguity might lead to specification of policies that breach the restrictions implied by a zone, *i.e.,* a single policy within a zone.

For another instance, when firewalls are placed in series, the best practice is vague about how zones and conduits should be defined. We argue that there needs to be an Abstract-Zone to capture the distinct policies that could be reasonably applied to the two firewalls.

ANSI/ISA best practices also lacked specification on how to precisely capture firewall management traffic. Adding a Firewall-Zone addressed the problem.

The average firewall configuration length in our case studies was 764 lines. It is easy to accidentally leave-in lapsed ACL rules inside a lengthy configuration, when the composition of network devices changes with time. Our case studies confirm the fact; we found lapsed rules in each one of them. Apart from being wasteful and misleading, such lapsed rules can lead to latent errors. But, detecting these rules through simple inspection was impossible because the close coupling between policy and network implementation made the firewall configurations human incomprehensible. A pragmatic policy framework should automatically detect and remove obsolete rules.

Access Control List (ACL) rules can have complex interactions. For example, a rule within an ACL can overlap and conflict with other preceding rules in the same ACL, potentially altering or even reversing its intended effect. Smaller rule-bases have been observed to contain fewer errors [128, 129]. But, with lengthy ACLs, maintaining interaction free rule-sets by hand is nearly impossible without automation.

Implicit rules can override ACLs, rendering the effort tendered to the careful design and deployment of ACLs obsolete. For example, an ACL can be configured on a Cisco ASA firewall interface to block inbound HTTP traffic to the firewall. If an implicit management policy on the device (such as those defined using *http* command) allows the service, the management policy overrides the ACL [30].

Based on our studies, we assert that the use of implicit rules should be avoided where possible, and replaced with explicit ACL based access control instead. This will be the difference in being able to generate clear, simple and effective firewall configurations from confusing, complex and unsafe ones.

We also assert that NAT and VPN play a minimal role in configuring a SCADA firewall. NAT is not required because hosts within a SCADA network should not communicate directly through the firewalls to the Internet [21]. VPNs also lack use here because SCADA firewalls should not be remotely accessed from insecure enterprise networks and the Internet [21].

In every real-world SCADA case study we considered, the intended security policy was also incorrectly mapped to the network firewalls. The manual mapping process resulted in policy often being allocated either to the wrong firewall,

firewall interface or interface direction. The incorrect mapping meant that the intended security policy was not enforced in the network. But this false sense of security was concealed by the lengthy firewall configurations; administrators inspecting the configurations were unable to identify the fact, as evidenced by the basis these rules had remained in place.

This perception of firewall security could leave firewall configurations inefficient at best and produce potentially dangerous security holes at worst. A policy framework must map security policy to the network-firewalls *provably correctly* using formal semantics.

Every SCADA network we studied also breached industry security best practices, exposing the critical SCADA zone. But, detecting these breaches automatically was previously not possible due to the inability in existing firewall configuration tools to verify best-practice compliance. Hence, automating best-practice compliance tests is also necessary in a pragmatic policy framework.

It is likewise important to maintain a small set of coherent ideas (*e.g.,* single source of truth, implementation-free policy) in automating firewall configuration. Doing so, boosts conceptual integrity and keeps the system user friendly.

Our case studies did not comprise large, complex networks. This simplicity suggests that the task of configuring the network firewalls should be relatively easy. In addition, due to the critical nature of the industrial control equipment protected by these firewalls, one expects them to be configured with accuracy. As we found, this is far from reality. Even in the simplest of cases, SCADA firewalls are still badly configured. Needless to say, what chances do we have of correctly configuring firewalls in a large, complex network?

Firewall configuration is complex and difficult and contributes to network-security that is broken-by-design, as shown by our case studies. By refining the ANSI/ISA Zone-Conduit abstraction we make it precise and complete for the description of real-world SCADA firewall policies. This refined Zone-Conduit model is suitable for automation.

3.4 Conclusions

The components of a SCADA firewall configuration are diverse and have complex interactions, making it difficult to manage manually. The ANSI/ISA best practices provide a Zone-Conduit model for firewall policy specification. However, it lacks key aspects for automation of firewall configuration. We propose several extensions to address these missing pieces and evaluate the usefulness of these extensions in describing real-world SCADA firewall policies.

Several additional requirements for auto-configuration were also identified from the SCADA case studies. For instance, eliminating ACL interactions, au-

tomating best-practice compliance checks and mapping policy to network devices provably correctly. We will leverage these requirements to develop a firewall auto-configuration prototype system, later in Chapter 6.

Chapter 4

Mathematical Abstractions of Firewalls

In the previous chapter, we identified the missing pieces of the Zone-Conduit model required for precise high-level specification of firewalls. We then proposed solutions to bridge these gaps and evaluated their usefulness through real SCADA firewall case studies. We now mathematically model the Zone-Conduit policies in a network, and derive the formal properties and constraints that a firewall policy specification must satisfy. Incorporating formal foundations into the policy specification helps (i) make the specified policies precise and unambiguous; and (ii) eliminate wishful thinking pitfalls common in designing and checking firewall policies.

Building a high-level policy definition on formal mathematical constructions removes implicit properties and provides a truly sound foundation for everything that follows. The formalism allows construction of complex and flexible policies and support reasoning about the policies.

For instance, we could check policy consistency with precision. Policy inconsistencies such as redundancies and conflicts were a common occurrence in the real firewall case studies discussed in Chapter 3. Related works [46, 73, 128, 131] confirm this observation. Such policy inconsistencies often produce unintended consequences.

Several related works have used mathematical abstractions of IP packets and networks to construct formal policy rules. For instance, the firewall-debugging tool – FIREMAN [131] – employs symbolic model checking using Binary Decision Diagrams (BDDs) that cover all paths and IP packets to detect network-wide policy inconsistencies. They prove the soundness and completeness of the proposed algorithms, but their underlying assumption of a first-match strategy means their proposed semantics are firewall-implementation dependent.

Likewise, Liu *et al.* employ a packet model to describe firewall policies [73].

They do not mathematically model the underlying network because only a single pair of firewall configurations is compared at a time. In the approach, firewall rulesets are converted to Firewall Decision Diagrams (FDDs) and their functional discrepancies are derived. These discrepancies are then used to evaluate multiple policy designs and analyse change impact. It is more useful to incorporate a mathematical model of the underlying network and enable network-wide policy comparison because *per-firewall changes* can be made redundant through inter-firewall rule interactions.

Verifying a policy is correct with precision is a non-trivial task. But, progress can be made towards the goal using formal policy language semantics. SDN programming languages such as NetKAT, Pyretic [9, 98] and Policy Defined Networking (PDN) policy languages such as ASL and Rei [61, 65], often include semantics to check syntactic correctness.

Policy-consistency checking partly verifies syntactic correctness. A syntactically correct policy must also be semantically correct for it to be *well designed*. For instance, it is necessary to compare a specified policy with industry best practices in [21] for compliance. Doing so, is critical in SCADA networks where more restrictive practices are required to prevent serious injury of people, or even death.

Guttman *et al.* [49] have proposed an approach that models IP packets and the underlying network using formal semantics. Their semantics allow construction of network-wide access-control policies and security goals to verify the specified goals are met by the policy. Formal semantics are also defined for trajectories – (network-path, IP packet) tuples – to explicitly block unwanted traffic along paths. However, their approach is not aimed at comparing network-wide policies.

The algebras we propose allows to check policy semantic correctness via efficient comparison of high-level firewall policies. These formal foundations help us to determine the properties and constraints required in a firewall policy description to make the policy comparisons rule-order and firewall-implementation independent.

Likewise, it is also necessary to map security policy to network-firewalls using a formal approach. We can only be confident of the protection provided for our network if we can prove that an intended policy is implemented by the correct set of network-firewalls. Some top-down configuration languages (*e.g.,* [12, 92]) allow creation of network wide high-level policies, but lack means to allocate policy to network devices in provably correct way.

NetKAT [9] is a SDN programming language that allows specifying and reasoning about network behaviour. An implementation has been developed for NetKAT to handle high-level policies based on virtual network topologies [113]. The implementation uses an extension of Binary Decision Diagrams to generate OpenFlow entries from high-level policy. Another such is the SOL framework [53] that uses a path-based abstraction to capture optimisation requirements

of SDN applications. The abstraction allows definition of valid paths via predicates. However, as these are SDN frameworks, they cannot be used to map high-level policy to network devices in traditional networks. Our proposed algebras overcome this shortfall, which is vital in current SCADA networks using existing network equipment.

4.1 Policy on a single directed conduit

A firewall policy is constructed from an ordered series of rules $[p_1, p_2, \ldots, p_n]$ that act on packet sequences to *accept*, *deny*, or *modify* them. We formally define these rules starting with $S = \{all\ packet\ sequences\}$, which implicitly includes ϕ (no packets). In principle, a firewall can delay its decision on a packet sequence until it can decide if the whole sequence is valid, but practical limitations (on stored state, and packet delays) mean that decisions must be made on short sequences, and often on *single packets*. So, we define rules to operate on $S^* = \{complete\ packet\ sequences\}$, where *complete* is defined below.

We define the operation $s = s_1 + s_2$, to mean that s is the concatenation of the two subsequences. Both S and S^* are *closed* under + (an associative operator) and ϕ is the identity. Hence S and S^* are *monoids*.

A policy rule p is a function $p : S^* \to S^*$. Considering a family $\mathcal{F}$ of possible rules, *complete* means that $\forall s_1, s_2 \in S^*$ and policy rules $p \in \mathcal{F}$

$$p(s_1 + s_2) = p(s_1) + p(s_2). \tag{4.1}$$

In summary, a decision on two concatenated *complete* subsequences is the same as that on the joint sequence. Hence, valid policy rules $p(\cdot)$ are *monoid endomorphisms* on S^*; they are mappings from S^* to itself that preserve the semigroup structure of the operator + and the identity ϕ.

A *packet sequence* also includes packet timings. For instance, consider a packet u that is fragmented into two component packets u_a and u_b. A firewall might perform fragment reassembly in order to test the packet's validity. It might allow $u = u_a + u_b$, but not allow either fragment by itself. This appears to break (4.1). But, if the firewall sees u_a "by itself", it has in fact observed $u_a + timeout$, where the timeout helps the firewall to determine when to give up on the second fragment. And $u_a + timeout + u_b \neq u_a + u_b$, so timing information is a crucial component of the packet sequences.

We often define a policy rule to operate on $\mathcal{A} = \{atomic\ packet\ sequences\}$, where an atomic packet sequence is a *complete packet sequence* that cannot be decomposed into smaller subsequences (except themselves and ϕ). Typical policy rules accept or deny packets, so, for some $A \subset \mathcal{A}$ (which can be extended to S^*)

$$p_A(s) = \begin{cases} s, & \text{if } s \in A, \ // \ accept \\ \phi, & \text{if } s \in A^c, // \ deny. \end{cases} \tag{4.2}$$

This type of rule doesn't allow modification or creation of packets. Real firewall rules can modify or create packets by

- updating certain header fields related to QoS; or
- defragmenting, or fragmenting packets; or
- integrating with Network Address Translation (NAT) or Virtual Private Network (VPN) functionality.

The scope for packet modification in general, is quite large, but internal to a firewall, many modifications don't change fields that affect further rules in subsequent firewalls such as QoS or TTL changes. So, we restrict rules to such modifications, and thus consider them to be in the form given in (4.2).

Firewall rules can also generate new packets through event logging. We exclude logging from the scope of this chapter – see Chapter 8 for an extended discussion of this and related issues.

We also cannot construct a policy rule for any possible subset of $\mathcal{A}$, due to the limitations of technology used in a firewall. For instance, common IP packet header fields used by Cisco ASA firewalls in traffic filtering include source and destination IP addresses, protocol type and protocol-specific fields such as source/destination port numbers for TCP/UDP and message code for ICMP [30].

The subsets of $\mathcal{A}$ for which rules can be defined is a *sigma algebra* $\sigma(\mathcal{A})$ [17]. The particular sigma algebra is generated by the finest possible partition of $\mathcal{A}$ determined by the firewall technology used. So for a given firewall technology, $\mathcal{A}$ can be broken into sets $A_i \subset \mathcal{A}$ such that $A_i \cap A_j = \phi$ and $\cup_i A_i = \mathcal{A}$, and we can implement rules p_{A_i}, but cannot define any rule p_B where B is a strict subset of some A_i. Note that for real firewalls $\sigma(\mathcal{A})$ is finite.

4.1.1 Positive, explicit policies

A firewall policy in practice is made up of multiple predicate matching rules that can be combined with precedence determined by several strategies: *first match*, *last match* or *all match*. If we consider here the conservative security option: an implicit *deny-all* rule in place, then an accept rule q^a_m (where m is the predicate) defines an accept packet set $A = \{s \in \mathcal{A} \mid s \prec m\}$, where $s \prec m$ denotes 's matches predicate m'. A single deny rule q^d_m has no affect, so, $A = \phi$.

When we combine two (ordered) rules $(q^t_{m_1}, q^t_{m_2})$ where $t \in \{a, d\}$, we define operators based on the matching order

1. first match

$$q^t_{m_1} \oslash q^t_{m_2} = \begin{cases} q^t_{m_1}, & \text{if } s \prec m_1 \\ q^t_{m_2}, & \text{if } s \prec m_2 \text{ and } s \not\prec m_1 \\ deny, & \text{otherwise,} \end{cases}$$

2. last match

$$q^t_{m_1} \ominus q^t_{m_2} = \begin{cases} q^t_{m_2}, & \text{if } s \prec m_2 \\ q^t_{m_1}, & \text{if } s \prec m_1 \text{ and } s \not\prec m_2 \\ deny, & \text{otherwise.} \end{cases}$$

The operations $\oslash$ and $\ominus$ are associative, *e.g.,* $(q_1 \oslash q_2) \oslash q_3 = q_1 \oslash (q_2 \oslash q_3)$, but not commutative or equivalent. This is a problem. We need a policy to hold the same semantics regardless of the rule precedence, in order to simplify policy specification.

So, we restrict ourselves to *accept rules*, conditional on an implicit *deny-all* rule. This restriction

- is rich enough to represent all rules of the form (4.2); and
- the operators $\oslash$ and $\ominus$ are then commutative and equivalent.

Equivalent results also hold if we only allow deny rules, but this option is less secure as it is easier to accidentally leave something out of a deny list than to include it.

Theorem 2 (Commutativity)**.** *The order of matching is irrelevant iff the match rules are all* accept rules *overlaying an implicit* deny-all *rule (or visa versa).*

Proof. First note that if we allow only *accept rules*, with an implicit *deny-all*, then the operators above become

$$q^a_{m_1} \oslash q^a_{m_2} = q^a_{m_1} \ominus q^a_{m_2} = p_A,$$

where $A_1, A_2 \in \sigma(\mathcal{A})$ such that $A_i = \{s \in \mathcal{A} \mid s \prec m_i\}$, and $A = A_1 \cup A_2$. Hence when we limit ourselves to *accept rules* the operators are commutative (via commutativity of set intersection) and equivalent.

If we have one accept, and one deny rule, with an implicit *deny-all*, then for a non-trivial rule space we can construct an example that violates commutativity and equivalence. Take two sets $A_1, A_2 \in \sigma(\mathcal{A})$ with matching predicates m_1 and m_2 such that $A_1 \neq A_2 \neq \phi$. Then under

$$\begin{aligned} q^a_{m_1} \oslash q^d_{m_2} &= p_{A_1}, \\ q^d_{m_2} \oslash q^a_{m_1} &= p_{A_1 \setminus A_2}, \\ q^a_{m_1} \ominus q^d_{m_2} &= p_{A_1 \setminus A_2}, \\ q^d_{m_2} \ominus q^a_{m_1} &= p_{A_1}. \end{aligned}$$

Combinations yield different composite rules, implying a non-commutative and non-equivalent counter-example. □

Theorem 3 (Completeness)**.** *Any rule (4.2) for* $A \in \sigma(\mathcal{A})$ *can be represented by a series of accept rules* $(q^a_{m_1}, q^a_{m_2}, \ldots, q^a_{m_n})$.

Proof. When we constructed $\sigma(\mathcal{A})$ from the A_i we defined these as the finest partition of $\mathcal{A}$. This means that there is at least one predicate that matches A_i within the ruleset available.

By construction all elements $A \in \sigma(\mathcal{A})$ can be constructed by finite unions on the sets A_i, and any rule combination generates a union, *i.e.,* $(q^a_{m_i}, q^a_{m_j})$ defines set $A_i \cup A_j \subset \mathcal{A}$, so we can define any set A by listing rules for each member. □

The key advantage of adopting a security whitelisting model; *i.e.,* restricting to accept rules, is that *rule order doesn't matter*, and *neither does implementation strategy*. A policy holds the same semantics; irrespective of how its rules are organised, removing dependencies from the policy specification. So, we can add or remove a rule without considering the complete rule set and the potential interactions. By being explicit, we also guard against services being enabled implicitly by accident or by default.

Related works [12,92] have also adopted a security whitelisting model to simplify policy specification, but they do not use formal methods to show how the approach renders the policy-implementation strategy irrelevant or preserve completeness in policy specification.

We can now simplify the notation we use in this context so that

1. $(p_A \oplus p_B)(s) = p_{A \cup B}(s)$ denotes within-firewall composition, or construction through parallel firewalls ($\oplus$ replaces both $\oslash$ and $\ominus$);

2. $(p_A \otimes p_B)(s) = p_B(p_A(s)) = p_{A \cap B}(s)$ denotes the action of sequential firewalls.

4.2 Network policy

We considered policy rules on a *single directed conduit* in the previous section. Now we generalise policies to the simplified Zone-Conduit model of a network.

We explicitly label the address space of allowed sources and destinations when applying a policy to a conduit. Now we consider policy rules to act as part of a $(edge, rule)$ pair, where the $rule$ excludes details of sources and destinations, which are incorporated via the Zone-Conduit model.

The Zone-Conduit model is a directed graph $G = (Z, C)$, where Z is the set of zones, and $C \subseteq Z \times Z$ is the set of *directed* conduits. We include here indirect conduits that may be built up from multiple physical paths.

It is also necessary here to consider all topological paths that can be taken. So, we consider all feasible conduits in the Zone-Conduit model, and ignore the dynamics of the underlying routing plane. This is because firewalls must filter traffic in a consistent manner, regardless of the underlying routing decisions (which may result in packets traversing different conduit paths at different times). For instance, the reachability of desired IP packets can be guaranteed only if none of the firewalls on the feasible paths denies them. For another instance, prohibited IP packets can only be blocked with reliability, if they are disallowed by all feasible paths, even after network failures.

Definition 4 (Network Policy)**.** *A network policy* $\mathcal{P} = (G, P)$ *means a directed-graph* $G(Z, C)$ *with policy functions* $p_{ij} \in \Phi$ *for* $(i, j) \in C$.

The policy function p_{ij} here does not involve addresses, which are implicit in the directed-conduit label (i, j), and the mapping from zone to addresses.

The SCADA firewall configuration best practices discussed in Chapter 3 lacked guidelines on how to automate compliance checks. The advantage of the mathematical framework above is that we can address this shortfall. We do so, and propose semantic foundations for comparing network-wide firewall policies to check if they are equivalent; or one is contained in the other in meaningful ways, in the next section.

4.3 Policy comparison

Security administrators may naturally wish to compare the semantics of two firewall policies, for instance, to

- compare a policy to industry recommended practices;
- study change impact, for instance, how policies evolve over time;
- identify discrepancies between multiple policy designs;
- check an implemented policy is the intended policy; or
- change the underlying firewalls (*e.g.,* vendor), but ensure the policy remains the same.

The ability to automate the comparison of firewall policies is important in SCADA networks where more restrictive practices are required to minimise their vulnerability to cyber attacks. Hence, we must make accurate comparisons to industry guidelines [21, 117] available in this domain. But, checking firewall policies for best-practice compliance has challenges:

- there is a lack of common standards for firewall policy specification;
- equivalent policies can be specified in multiple ways, for instance, through accept or deny rules;
- network-centric details impinge on current means of specifying policies; and
- evaluating comparisons between two policies could be computationally expensive.

We tackle these issues from first principles by first comparing the policies of individual conduits and then extending the concept to network-wide policies.

4.3.1 Conduit policy comparison

A conduit policy can be a single rule or as a set of rules and operators. So, we can write a conduit policy as a single mapping function $p^X \in \Phi$ or as its components $p^X = (p_1^X \oplus \cdots \oplus p_m^X)$. Policy rules can be combined in multiple ways to achieve a given effect. Hence, we define two policies to be *equivalent* as follows

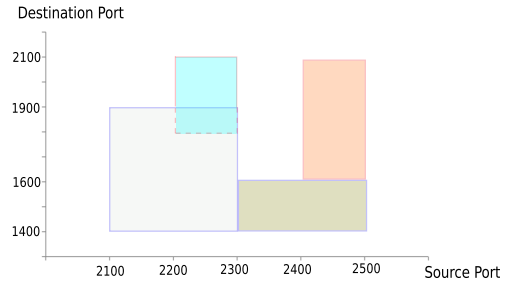
Definition 5 (Equivalence)**.** *Two policies p^X and p^Y are* equivalent *on $\mathcal{A}$ iff $p^X(s) = p^Y(s)$, $\forall s \in \mathcal{A}$. We denote this equivalence by $p^X \equiv p^Y$.*

When we compare policies on two networks, it is also useful to be able to partition the network policies in *equivalence classes* [10]. We call this a *semantic partition* (SP) and give it the following formal definition

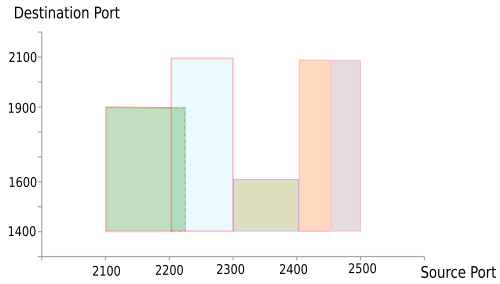
Definition 6 (Semantic partition)**.** *The* semantic partition *SP of a set of policies P is given by $SP = \{e_m\}$ where $P = \cup_m e_m$ and the $e_m \subset P$ are the minimal number of equivalence classes,* i.e.*, for all $p_i, p_j \in e_m$ we have $p_i \equiv p_j$.*

An *equivalence class* groups a set of conduits with identical semantics. Large equivalence class sizes in a network indicate many zones with identical reachability. This indicates an *inefficiently designed* Zone-Conduit model. The said zones may need to be amalgamated (at least at a logical level) to maintain a concise network and reduce policy specification complexity. Semantic partitions can help write better policies, by allowing to check if a collection of conduits have a given policy.

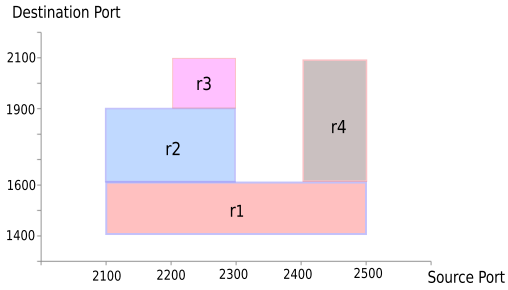
Two policies with different rule sets can have the same underlying semantics: *i.e.,* allow the same set of services between zones. For instance, Figures 4.1(a) and 4.1(b) illustrate the idea based on TCP port filtering of single packets. Each rectangle indicates the allowed packets of a single rule. Combined, the rules cover the same set of allowed packets.



(a) Four rules shown by (overlapping) rectangles.



(b) Five rules giving an equivalent policy to (a).



(c) Horizontal partitions of polygons in (a) or (b).

Figure 4.1: Canonicalisation of distinct rule sets of the same policy. Rectangles indicate the packets allowed by a particular rule. Horizontal partitions as shown in (c) is one strategy for canonicalisation.

We could compare policies by exhaustive comparison, but that is very inefficient. A more efficient approach is to derive a unique, *canonical*, representation of each policy. We can represent canonicalisation of policies through a mapping $c : \Phi \to \Theta$, where Θ is the canonical space of policies, in which all equivalent policies of Φ map to a singleton. Given a canonicalisation mapping, we note the following (the proof being the direct result of the definitions).

Lemma 7. *Policies* $p^X \equiv p^Y$ *iff* $c(p^X) = c(p^Y)$.

We need to evaluate our policies against industry best practices. When two flow-policies are *equivalent*, they accept and deny the same IP packets, but it will be common that some customisation of policies must be conducted locally. A policy *complies* with another if it is equally or more restrictive. In that context we define *inclusion* as follows

Definition 8 (Inclusion)**.** *A policy* p^X *is* included *in* p^Y *on* $\mathcal{A}$ *iff* $p^X(s) \in \{p^Y(s), \phi\}$, i.e., *X has the same effect as Y on s, or denies s, for all* $s \in \mathcal{A}$*. We denote inclusion by* $p^X \subset p^Y$.

In order to make practical use of this definition (via canonicalisation), we restrict our attention to the accept rules considered above, where the following lemma applies

Lemma 9. *A policy $p^X \subset p^Y$ iff $p^X \oplus p^Y \equiv p^Y$.*

Proof. We can describe all accept rules by functions such as (4.2) and $p_A \subset p_B$ means that $A \subset B$. Moreover $p_A \oplus p_B = p_{A \cup B}$, and $A \subset B$ iff $A \cup B = B$. Hence the result. □

We define the *semantic difference* of two policies as follows (again restricting to *accept rules* for practical use)

Definition 10 (Semantic Difference)**.** *The semantic difference between policies p^X and p^Y is given by $p^X - p^Y = (p^X \oplus p^Y) \otimes (p^X \otimes p^Y)^c$, where $(p_A)^c = p_{A^c}$ and A^c is A's complement.*

We present $c(p^X - p^Y)$ in our results. When two policies are *equivalent*, their semantic difference yields ϕ. If $p^X \subset p^Y$, the result will contain elements from p^Y that are not in p^X. This notation is useful in particular, in change impact analysis. It helps administrators understand in detail how a policy has really evolved with time, independent of minor implementation changes of no real impact. *Equivalence* and *inclusion* notations on the other hand, allow fast, high-level checks.

4.3.2 Comparison of network policies

We extend conduit policy *equivalence*, *inclusion* and *difference* definitions to compare policies of an entire network.

Definition 11 (Equivalence)**.** *A policy $\mathcal{P}_1 = (G, P_1)$ is* equivalent *to $\mathcal{P}_2 = (G, P_2)$ iff $\forall e \in C$, $p_1^e \equiv p_2^e$. We denote this by $\mathcal{P}_1 \equiv \mathcal{P}_2$.*

Definition 12 (Inclusion)**.** *A policy $\mathcal{P}_2 = (G, P_2)$* includes *$\mathcal{P}_1 = (G, P_1)$ iff $\forall e \in C$, $p_1^e \subset p_2^e$. We denote this by $\mathcal{P}_1 \subset \mathcal{P}_2$.*

Definition 13 (Semantic Difference)**.** *The semantic difference between $\mathcal{P}_1 = (G, P_1)$ and $\mathcal{P}_2 = (G, P_2)$ is given by $\mathcal{P}_1 - \mathcal{P}_2 = \bigoplus\{(p_1^e - p_2^e); \forall e \in C\}$.*

Similarly, we extend the idea of a Semantic Partition (SP) to a network. An SP groups a set of directed conduits with equivalent policies. These groups partition in the sense that they cover P with disjoint sets. We now extend the notion of comparison of two network policies to the comparison of their SPs.

Definition 14. *The semantic partitions SP_1 and SP_2 of policies P_1 and P_2, respectively, are* equivalent *iff $|SP_1| = |SP_2|$ and $\forall e_1 \in SP_1$, $\exists e_2 \in SP_2$ such that for any $p_1 \in e_1$ and $p_2 \in e_2$, we have $p_1 \equiv p_2$. We denote this by $SP_1 \equiv SP_2$.*

Semantic partition SP_1 includes *SP_2 iff $\forall e_2 \in SP_2$ $\exists e_1 \in SP_1$ s.t. $e_2 \subset e_1$. We denote this by $SP_2 \subset SP_1$.*

Definitions 11 to 14 assume that the underlying networks are the same. We may wish to make comparisons between different networks, or one network as it evolves over time. So we also extend the definitions to the case where the two graphs G_1 and G_2 are isomorphic, *i.e.,* $G_1 \simeq G_2$.

The essential property of our high-level language is that it has a 1:1 mapping between policies and conduits, so comparing policies of two networks is a matter of comparing policies of each conduit. The difficulty is that the zone labels are arbitrary. So, two networks might have entirely different labels and policies cannot be compared directly. In the end, we must evaluate if the two (unlabelled) digraphs have the same graph structure and identify a mapping between the zones of the two networks. This is the *graph isomorphism* problem.

The complexity of the graph isomorphism problem is unknown, but there are no known polynomial-time algorithms [63] (the best being a quasi-polynomial time algorithm released in 2015 [11]), so there is a practical difficulty to overcome.

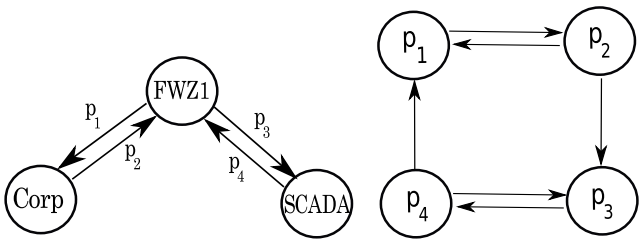
We also need to compare two different graphs (where $G_1 \not\simeq G_2$), to see the effect of a network change, or to check if firewall policies are deployed in a consistent manner across an organisation's multiple SCADA sites. The policies in this context cannot be strictly *equivalent* and we cannot apply *inclusion* given different target networks. So, we introduce a new concept: *incorporation*.

Definition 15 (Incorporation)**.** *Policy $\mathcal{P}_2$* strictly incorporates *$\mathcal{P}_1$ iff G_1 is isomorphic to a subgraph of G_2 and $\forall e \in C_1$, $p_1^e \equiv p_2^e$. We denote this by $P_1 \equiv P_2(G_1)$. Policy $\mathcal{P}_2$* partially incorporates *$\mathcal{P}_1$ iff G_1 is a subgraph of G_2 and $\forall e \in C_1$, $p_1^e \subset p_2^e$. We denote this by $\mathcal{P}_1 \subset \mathcal{P}_2(G_1)$.*

A network policy P is compliant with best practice RP if $P \subset RP$, through either *inclusion* or *incorporation*.

Thus, we also need to solve the *subgraph isomorphism* problem (which is NP complete) to check compliance. The two isomorphism problems seem intractable, but the Zone-Conduit policy model is not a completely unlabelled digraph. We have edge labels in the form of policies.

We exploit this information to avoid the graph isomorphism problem. The mathematical tool we work with is called the *line digraph* (directed graph), defined as follows [52]



(a) Zone-Flow model. (b) Line digraph of (a).

Figure 4.2: A policy digraph and its corresponding line digraph. Nodes in the line digraph are the edges of the original digraph.

Definition 16 (Line Digraph)**.** *Given a digraph $D = (N, E)$, its line digraph $\mathcal{L}(D) = (P, L)$ is defined by the non-empty set of points $P = E$, and edges $(uv, vw) \in L$ wherever $(u, v) \in E$ and $(v, w) \in E$.*

We derive the line-digraph by taking a node for each edge in D, and creating edges between these new nodes wherever the original edges connected through the same node (so they form part of a length-two path). Theorem 17 allows us to identify an isomorphism in the original digraphs from isomorphism in the line-digraphs as per below. The theorem is a generalisation of the Whitney graph isomorphism theorem to multi-digraphs [52].

Theorem 17. *Let D be a multi-digraph. If D' is a multi-digraph such that $\mathcal{L}(D) \simeq \mathcal{L}(D')$, then the digraphs formed by D and D' by deleting all* sources *and* sinks *are isomorphic.*

Proof. see [52]. □

A multi-digraph is a digraph that allows multiple directed edges between graph nodes [52]. So, Theorem 17 applies to our model. Conduits in the Zone-Conduit model are always formed from a directed pair (to allow for different policies), so, our digraphs are *source* and *sink* free. We need not delete any nodes to apply Theorem 17; if the line digraphs are isomorphic then the original policy digraphs are isomorphic.

The constructed line digraphs also have a (non-unique) node labelling defined by the policies that labelled the original conduits (Figure 4.2). We can use this labelling to avoid the full complexity of the graph isomorphism problem. We will describe the resulting algorithms later in Chapter 6.

In the introduction to this chapter we described how the ability to map security policies to the underlying network-firewalls is required in a policy specification

language. Doing so in a provably correct way, allows administrators to be confident of the level of protection provided by the security policies for their networks.

Previous related works [12, 53, 113, 115] have generated device-level configurations from high-level policy, but, they lack formal semantics to allocate policy to network devices provably correctly. We address this shortfall and propose an algebraic framework to map a given security policy to a set of network firewalls.

4.4 Mapping policies to network-firewalls

The Zone-Conduit abstraction allows us to describe firewall policies at a high-level, independent of network-topology intricacies. But, in order to implement such a policy on a network instance, the two must be re-coupled together. The coupling must be performed in a provably correct way to (i) provide administrators assurance that their security policies are enforced by the correct set of network-firewalls; and (ii) to avoid redundant policy updates to the firewalls.

4.4.1 The policy to network-firewall mapping problem

Consider the example network in Figure 4.3(a), our high-level policy might be "we want to enable only SSH flows from $Z1$ to $Z4$". This intent can be expressed through the Zone-Conduit model in Figure 4.3(b) as per below with an implicit *deny-all* in place ($p \in \Phi$ is an element of the possible space Φ of policies).

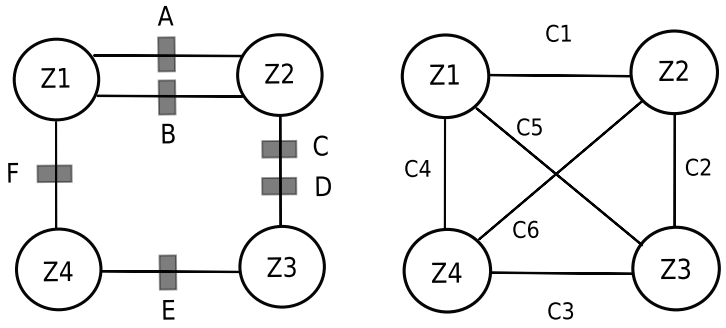
$$p\texttt{:=Z1} \rightarrow \texttt{Z4: ssh}$$

We now want to map the policy to the enforcing firewalls. It looks easy. Simply implement the policy on firewall F. However, the problem is in general, more complicated. For example, consider policy from $Z1$ to $Z2$. The parallel firewall architecture involved gives the resultant security policy (p_R) the meaning — *all packets that can possibly be allowed through* — and is the union of the packet sets allowed by the individual devices. We are conservative in taking union, because intrusions and attacks are often carried out through permitted traffic. So, if p_A and p_B allows packet sets Q and T respectively, then $p_R = p_A \oplus p_B = p_{Q \cup T}$.

For another example, consider policy from $Z2$ to $Z3$. When the devices are in series, the resultant policy permits the *intersection* of the packet sets V, W allowed by the policies of C, D respectively, *i.e.,* $p_R = p_C \otimes p_D = p_{V \cap W}$.

We have denoted the policy composition resulting from parallel routes as $p_A \oplus p_B$ and that resulting from serial routes as $p_A \otimes p_B$. In this example, $p_A \oplus p_B = p_{Q \cup T}$ and $p_C \otimes p_D = p_{V \cap W}$, but, the meaning of $\oplus$, $\otimes$ are policy-context dependent [95].

The problem of mapping our high-level policy p (from $Z1$ to $Z4$) to firewalls may be even more complicated than simply implementing the policy on F. What



(a) Network topology. (b) Zone-Conduit model of (a).

Figure 4.3: Example network consisting of four zones ($Z1-Z4$) and six firewalls $A,B,...,F$ (a). The corresponding Zone-Conduit model is shown in (b).

happens if F fails? We still want to enable SSH flows that traverse the redundant paths from $Z1$ and $Z4$ (for instance, path via $Z2$ and $Z3$). So, p must also be mapped to the firewalls along the path $Z1-Z2-Z3-Z4$, to cater for this redundancy. But then, should we map the policy to all of these firewalls or a subset of them?

Given the serial firewalls between $Z2$ and $Z3$ it is easy to see that we need to map p to both C and D to preserve the semantics of $\otimes$. But, with parallel routes such as that involving A and B, we have the choice of mapping policy to both or just one to preserve the semantics of $\oplus$. Mapping to both may be unnecessary and inefficient and firewalls may have limits on their capabilities.

Moreover, the firewalls may not always be in series or parallel. We have described earlier how mapping policy to topology becomes interesting when a star-topology is involved (Subsection 3.1.3). Loosely mapping policy to firewalls using a *hyper-edge* adds to the Zone-Conduit model complexity and decreases it's precision (it allows a $1:n$ relationship between hyper-edges and policies). We overcame these issues and increased the precision of the model by using a simple-edge mapping instead.

We may also need to consider paths that consist of one or more intermediate zones, when implementing policy between zones. For instance, consider implementing the security policy below in the network in Figure 4.3(a).

$$f := \texttt{Z1} \rightarrow \texttt{Z3: http}$$

Once implemented on the firewalls, the policy must allow HTTP traffic flow from `Z1` to `Z3`. But, the policy can only be deployed correctly if the intermediate zones `Z2,Z4` can route or forward HTTP traffic.

A zone's traffic routing or forwarding capability can be restricted. For instance, with regards to inbound traffic, only *secure* traffic that terminates at a SCADA-Zone must be permitted by a firewall policy [21]. Enabling the SCADA-

Zone to transit traffic can potentially expose the critical systems within to cyber attacks. So, in addition to link properties, node properties such as a zone's *traffic transitivity* capability must be considered with care, in constructing valid firewall-paths. This node property must also be captured *explicitly* in the mapping process.

4.4.2 Policy Mapping vs Routing

Our discussion above on mapping policies to network devices relates to network paths or routes and thus, has many parallels with routing. But, the aim in routing is to determine the path that optimises a given path-metric; for instance, shortest-path routing finds a minimum-distance path between nodes.

Our target problem is different: we need to determine the firewalls in a network a given policy needs to be implemented on. So, constructing all feasible paths is crucial because, for instance, a security policy between two endpoints can only be correctly implemented (*e.g.,* to block all Telnet traffic) if all redundant paths between them are taken into account.

In routing, the number of nodes (or gateways) can be high for a large network, so, distributed means to computing a solution is essential (requiring de-centralised protocols like OSPF and BGP). In contrast, we expect a relatively low number of zones when mapping security policies to firewalls (for instance, our SCADA case studies in Chapter 3 all comprised 15 zones or less). This low zone count makes our policy-mapping algorithm computationally tractable (as we will see later in Chapter 6). So, a logically centralised implementation can be considered.

Current meta-routing algebras [38], allow one to provably choose paths that optimise various path-metrics. These algebras support specification of link properties, but not the specification of node properties such as traffic transitivity. Our approach extends meta-routing to include node transitivity.

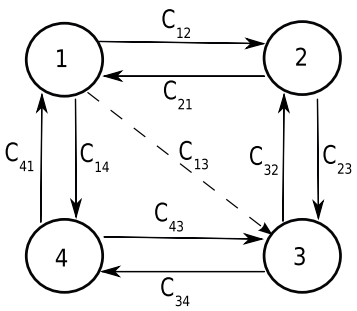
4.4.3 Mapping Algebra

We now outline our proposed algebras to map policy to network firewalls by first making a distinction between *primary* and *secondary* conduits.

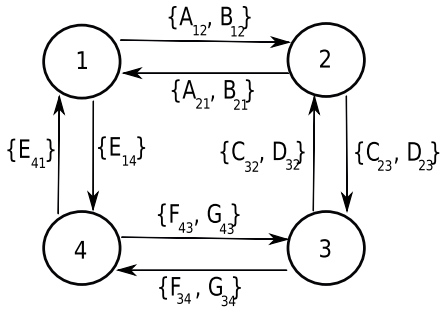
Firewall-path Construction

A conduit can be classified as primary or secondary based on how it enforces security policy. A primary-conduit enforces policy using firewalls only. A secondary-conduit enforces policy using firewalls and allows traffic to transit through one or more intermediate zones.

We demonstrate the idea using the simple Zone-Conduit graph $G = (Z, C)$ in Figure 4.4(a). The composition of each primary-conduit in terms of its component



(a) Zone-Conduit model with primary and secondary conduits.



(b) Firewalls comprising the primary-conduits.

Figure 4.4: Zone-Conduit model depicting primary-conduits C_{ij} and a secondary-conduit C_{13} enabled by the transit zones 2 and 4 are shown in (a). The firewall-paths of the primary-conduits are shown in (b).

firewalls the model is shown (Figure 4.4(b)). The example secondary-conduit C_{13} filters traffic flow from Zone 1 to 3, using several directed firewalls (defined below) and transit zones 2 and 4. The set of all directed-conduits is given by $DC = \{C_{ij} \mid (i,j) \in C\}$ and the set of network firewalls $W = \{firewalls\}$.

Definition 18. *A* directed firewall t_{ij} *is a firewall* $t \in W$ *that filters traffic on directed-conduit* $C_{ij} \in DC$.

The Zone-Conduit graph is an abstraction that moves traffic packets from one zone to another using conduits. So, regular expressions (the natural language of finite automata), can capture the packet-processing behaviour of this model; a single firewall-path encoding is a concatenation of directed-firewalls (or policy-arbitration steps *pq*) and a set of paths is encoded as a union of paths. Past work [9] has shown, all single-path encodings stem from the Kleene star operator (*) on the set of directed-devices. So, each firewall-path depicts a sequence of traffic filtering steps and is an element of F^* where $F = \{\text{directed firewalls}\}$.

But, the Zone-Conduit model is a logical representation, so, every firewall-path encoding in F^* may not be valid. We define the following to help filter-out invalid paths

Definition 19. *The mapping* $h : F \rightarrow W$ *retrieves the network firewall corresponding to a given directed firewall,* i.e., $h(t_{ij}) = t$ *where* $t_{ij} \in F$ *and* $t \in W, \forall i, j$.

Definition 20 (Single firewall-path concatenation)**.** *Take* $a \in F^*$ *s.t.* $a = t_{d_1d_2}t_{d_2d_3}t_{d_3d_4}...t_{d_nd_{n+1}}$ *where* $t_{d_id_j} \in F$. *Also take* $b \in F^*$ *s.t.* $b = s_{g_1g_2}s_{g_2g_3}s_{g_3g_4}...s_{g_mg_{m+1}}$ *where* $s_{g_ig_j} \in F$. *Then firewall-path concatenation from* $F^* \times F^* \rightarrow F^*$ *is*

$$ab = \begin{cases} t_{d_1 d_2} ... t_{d_n d_{n+1}} s_{g_1 g_2} ... s_{g_m g_{m+1}} \text{ if } d_{n+1} = g_1 \\ \quad \text{and } g_i \neq d_j;\ \forall i, j,\ i > 1 \\ \quad \text{and } h(t_{d_i d_j}) \neq h(s_{g_k g_l});\ \forall i, j, k, l \\ \phi; \text{empty set, otherwise.} \end{cases}$$

and $a\phi = \phi a = \phi;\ \forall a \in F^*$.

Concatenation defined above is a binary operation that constructs only elementary firewall-paths. We consider elementary paths here because they are most efficient and do not allow traffic to traverse a particular firewall or interface more than once. These paths also prohibit traffic flow through non-transit zones such as Firewall-Zones.

Consider two directed-firewalls X, Y with policies p_X and p_Y that accept packet sets R and T. The resultant policy of the concatenated firewall-path XY is that of sequential firewalls and can be denoted as $(p_X \otimes p_Y)(s) = p_{R \cap T}(s)$ where s is an atomic packet sequence.

We define a set of directed-firewall paths as a union of elements in F^*. Again, consider our directed-firewalls X, Y from before. If these described two distinct paths, then the resultant policy of the path union $X \cup Y$ is that of parallel firewalls, *i.e.,* $(p_X \oplus p_Y)(s) = p_{R \cup T}(s)$.

We also extend concatenation in Definition 2 to $S = \{\text{Power-set of } F^*\}$

Definition 21 (Multiple firewall-path concatenation)**.** *Take $a, b \in S$ s.t. $a = \{a_0, a_1 ..., a_x\}$, $b = \{b_0, b_1, ..., b_y\}$ where $a_i, b_j \in F^*$. Then firewall-path concatenation from $S \times S \to S$ is given by*

$$ab = \{a_i b_j \mid \text{for i=0,....,x, j=0,....,y}\}$$

Definition 3 allows to construct all possible firewall-path sets from the union of elements in S. Then $(S, \cup, \cdot, \hat{0}, \hat{1})$ is an idempotent semiring with

$\hat{0} = \phi$; the empty set; and

$\hat{1} = \{\varepsilon\}$; the empty-string set where ε is the identity element of the concatenation operation.

The properties of the operators $\cup$ and $\cdot$ dictate rules for firewall-path construction. So, these semantics must be preserved when composing firewall-paths. For instance, $\cup$ is commutative while $\cdot$ is not. So, the order of the directed-firewalls matter, when constructing a single firewall-path, but are irrelevant when constructing multiple paths.

Mapping Policy to Firewalls

We have described so far, how the semiring $(S, \cup, \cdot, \hat{0}, \hat{1})$ constructs sets of firewall-paths between zones in the Zone-Conduit model. The sequential: $\otimes$, and parallel:

$\oplus$, policy-composition operators in Subsection 4.4.1 can now construct the policies of these paths. Assume we have to implement a high-level policy p_{ij} on firewalls q_{kl} that lie in the paths from i to j. All applicable firewall-paths from i to j can be constructed as per Section 4.4.3 and given by

$$S_{ij} = \{q_{a_1a_2}q_{a_2a_3}...q_{a_{n-1}a_n}, ..., q_{b_1b_2}q_{b_2b_3}...q_{b_{m-1}a_m}\}.$$

The high-level policy p'_{ij} derived from the individual policies p'_{kl} then is

$$\begin{aligned} p'_{ij} \quad = \quad & (p'_{a_1a_2} \otimes p'_{a_2a_3} ... \otimes p'_{a_{n-1}a_n}) \\ & \oplus (.....) \\ & \oplus (p'_{b_1b_2} \otimes p'_{b_2b_3} ... \otimes p'_{b_{m-1}b_m}). \end{aligned} \tag{4.3}$$

Mapping policy p_{ij} to the firewalls is now a matter of finding p'_{ij} for all firewalls such that $p'_{ij} = p_{ij}$. For instance, a simple solution supporting defence in depth is to implement the security policy p_{ij} on every sequential firewall across all paths.

The underlying requirement when mapping security policy to network firewalls is to adhere to the semantics of (4.3).

Computation of All Firewall-paths

We reduced the policy-to-firewall mapping problem to the semantics of (4.3). We now develop an algorithm to compute the firewall-paths of (4.3) efficiently.

We can represent the primary-conduit firewall-paths using a generalised Adjacency matrix A. Here, $A(i, j)$ is the firewall-path of primary conduit $C_{ij} \in DC$.

For our example in Figure 4.4(b),

$$A = \begin{array}{c} \begin{matrix} Z_1 & Z_2 & Z_3 & Z_4 \end{matrix} \\ \begin{bmatrix} \{\varepsilon\} & \{A_{12}, B_{12}\} & \phi & \{E_{14}\} \\ \{A_{21}, B_{21}\} & \{\varepsilon\} & \{C_{23}, D_{23}\} & \phi \\ \phi & \{C_{32}, D_{32}\} & \{\varepsilon\} & \{F_{34}, G_{34}\} \\ \{E_{41}\} & \phi & \{F_{43}, G_{43}\} & \{\varepsilon\} \end{bmatrix} \end{array} \tag{4.4}$$

Then, the solution to the problem of finding all valid firewall-paths between zones is a matrix A^* s.t.

$$\begin{aligned} A^*(i, j) \quad = \quad & \{\text{valid primary- and secondary-conduit} \\ & \text{firewall-paths from zone i to j}\} \end{aligned} \tag{4.5}$$

We developed the following theorem to compute A^*, inspired by algorithms in meta-routing [38].

Theorem 22. *A^* can be calculated using the right iteration algorithm*
$$A^{<k+1>} = (A^{<k>}T \cup I)A \text{ where } A^{<0>} = I.$$

T and I are the Zone-transitivity matrix and the multiplicative-identity matrix (of semiring $(S, \cup, \cdot, \hat{0}, \hat{1})$) respectively, and are given by

$$T(i,j) = \begin{cases} \{\varepsilon\}, \text{ if } i = j \text{ and transitivity}(i) = 1 \\ \phi, \text{ otherwise.} \end{cases} \tag{4.6}$$

$$I(i,j) = \begin{cases} \hat{1}, \text{ if } i = j \\ \hat{0}, \text{ otherwise.} \end{cases} \tag{4.7}$$

Proof. Let's check that the result holds for $k = 0$ (*i.e.,* valid firewall paths between zones with up to $(0+1)$ hop).

$$\begin{aligned} LHS &= A^{<0+1>} = A^{<1>}. \\ RHS &= (A^{<0>}T \cup I)A \\ &= (IT \cup I)A\ ; \quad \because A^{<0>} = I \\ &= (T \cup I)A\ ; \quad \because IT = TI = T \ (I \text{ is multiplicative-identity}) \\ &= IA\ ; \quad \because T \cup I = I \\ &= A. \end{aligned}$$

$A^{<1>} = A$ *is correct*, since all valid single-hop firewall paths are represented by A. Let's assume the result holds when $k = n$, *i.e.,* $A^{<n+1>} = ((A^{<n>}T) \cup I)A$.

So, $A^{<n+1>}$ holds all valid firewall paths between the zones of up to $(n+1)$ hops. Then $A^{<n+1>}(i,j)$ represents all valid firewall paths of up to $(n+1)$ hops, from zone i to zone j (see Figure 4.5).

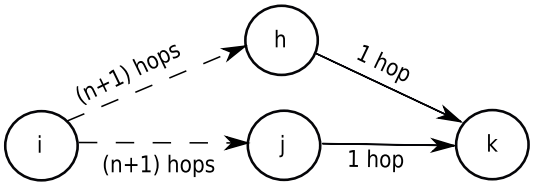


Figure 4.5: Construction of $(n+2)$ hop firewall paths from zone i to zone k via j.

So, valid firewall paths up to $(n+2)$ hops from i to k via j are given by

$$(t_{ik})_j = A^{<n+1>}(i,j)A(j,k).$$

But zone j may not be transitive, so we modify the above equation to account for this

$$(t_{ik})_j = A^{<n+1>}(i,j)T(j,j)A(j,k).$$

When $i = j$ the above equation yields ϕ, whereas we expect $\{\varepsilon\}$ to be the self-firewall path. Again, we update the equation

$$(t_{ik})_j = \Big[\, A^{<n+1>}(i,j)T(j,j) \,\cup\, I(i,j) \,\Big]A(j,k).$$

Then, all valid firewall paths up to (n+2) hops from i to k (via $j, h...etc.$) are given by

$$A^{<n+2>}(i,k) = \bigcup_{\forall j} \Big[\, A^{<n+1>}(i,j)T(j,j) \cup I(i,j) \,\Big]A(j,k);\ \forall i,k.$$

We simplify the equation further

$$\begin{aligned} RHS &= \bigcup_{\forall j} \Big[\, A^{<n+1>}(i,j)T(j,j)A(j,k) \Big] \,\cup \\ &\quad \bigcup_{\forall j} I(i,j)A(j,k). \end{aligned}$$

However,

$$A^{<n+1>}(i,j)T(j,j) = \bigcup_{\forall s} \Big[A^{<n+1>}(i,s)T(s,j) \Big]$$

So,

$$\begin{aligned} \bigcup_{\forall j} \Big[A^{<n+1>}(i,j)T(j,j)A(j,k) \Big] &= \bigcup_{\forall j} \Big[A^{<n+1>}T \Big](i,j)A(j,k) \\ &= \Big[A^{<n+1>}TA \Big](i,k). \end{aligned}$$

Also,

$$\bigcup_{\forall j} \Big[I(i,j)A(j,k) \Big] = \Big[IA \Big](i,k)$$

Hence,

$$\begin{aligned} A^{<n+2>}(i,k) &= \Big[A^{<n+1>}TA \Big](i,k) \,\cup \\ &\quad \Big[IA \Big](i,k);\ \forall i,k \end{aligned}$$

We can generalise the above to the matrices

$$\begin{aligned} A^{<n+2>} &= A^{<n+1>}TA \,\cup\, IA \\ &= (A^{<n+1>}T \,\cup\, I)A. \end{aligned}$$

□

For bounded semirings we only iterate $n-1$ times to converge to A^*, where n is the number of nodes in the Zone-Conduit model, *i.e.,* $A^* = A^{<n-1>}$.

We have defined T, A^* and the right-iteration algorithm for a security-policy context, but these can likewise be defined for other policy contexts such as traffic measurement, QoS and load balancing [95].

If we apply our algorithm in Theorem 22 to the example in Figure 4.4(b), $n = 4$, so, $A^* = A^{<3>}$ and for simplicity assume that all zones are transitive, then

$$T = \begin{bmatrix} \{\varepsilon\} & \phi & \phi & \phi \\ \phi & \{\varepsilon\} & \phi & \phi \\ \phi & \phi & \{\varepsilon\} & \phi \\ \phi & \phi & \phi & \{\varepsilon\} \end{bmatrix},$$

and we see that $I = T$ in this instance.

We calculate $A^* = A^{<3>} =$

$$\begin{bmatrix} \{\varepsilon\} & \eta & \kappa & \theta \\ \mu & \{\varepsilon\} & \nu & \beta \\ \gamma & \lambda & \{\varepsilon\} & \rho \\ \xi & \delta & \zeta & \{\varepsilon\} \end{bmatrix}.$$

All valid firewall-paths from Zone 1 to 3 are given by

$$\kappa \;=\; \{A_{12}C_{23}, A_{12}D_{23}, B_{12}C_{23}, B_{12}D_{23}, E_{14}F_{43}, E_{14}G_{43}\}. \tag{4.8}$$

Let's now assume that Zone 4 is non-transitive, so, $transitivity(4) = 0$, we re-calculate A^*,

$$A^* = \begin{bmatrix} \{\varepsilon\} & \{A_{12}, B_{12}\} & \eta & \theta \\ \{A_{21}, B_{21}\} & \{\varepsilon\} & \{C_{23}, D_{23}\} & \mu \\ \gamma & \{C_{32}, D_{32}\} & \{\varepsilon\} & \rho \\ \xi & \delta & \zeta & \{\varepsilon\} \end{bmatrix}.$$

The updated firewall-paths from Zones 1 to 3 are given by

$$\eta = \{A_{12}C_{23}, A_{12}D_{23}, B_{12}C_{23}, B_{12}D_{23}\}. \tag{4.9}$$

In comparison to (4.8), we see that the paths via Zone 4 ($E_{14}F_{43}, E_{14}G_{43}$) have now been removed, as the zone is no longer transitive. A^* can likewise be calculated for other policy contexts to determine all valid firewall-paths between zone pairs.

4.5 Conclusions

Building a high-level firewall policy description on formal foundations allows construction of complex and flexible policies and supports reasoning about these policies. Two important aspects of policy reasoning include (i) checking if policies are industry best practice compliant; and (ii) verifying whether the policies are mapped to the underlying network-firewalls correctly.

However, there are many obstacles to achieving both these policy-reasoning aspects. For one, the lack of standards for firewall policy specification makes policy semantics rule-order and firewall-implementation dependent, hindering their meaningful comparison. For another, lack of decoupling between policy and network makes policy sensitive to network-intricacies and vendor changes.

We addressed these challenges, by proposing algebraic frameworks that (i) enable direct comparisons of firewall policies; and (ii) allow policy to be mapped to network-firewalls provably correctly. We will use these proposed algebras in Chapter 6, to automate best-practice compliance tests and deploy firewall policies with precision to create networks that are secure-by-design.

In the next chapter, we will describe our design of the high-level firewall policy specification that is built upon these proposed mathematical foundations.

Chapter 5

ForestFirewalls Policy Specification Language Design

> "Everyone knows that debugging is twice as hard as writing a program in the first place. So if you're as clever as you can be when you write it, how will you ever debug it?"
>
> *Brian Kernigan [68]*

All studies of firewall configurations have found signifiant errors [97, 128, 129]. The introductory quote is part of the reason. Current firewall and security specifications have evolved by adding features and complexity. Many modern PDN solutions (*e.g.,* see Chapter 2) have provided new tools, but in making configuration more clever, they have actually made tasks such as debugging more difficult.

In this chapter we describe the design of our high-level firewall policy specification language: *ForestFirewalls*[1]. Our design takes the above tenet seriously; it enforces good security in a network at the outset to minimise a myriad of security problems later (*i.e.,* enables secure-by-design networks), but the design also facilitates fast, accurate policy debugging for when things do go wrong.

Our discussion of related works in Chapter 2 illustrated how existing firewall policy languages lack high-level configuration capability. Some nascent attempts [12] to decouple policy from the underlying network implementation still require minutiae such as IP addresses and hostnames to be provided in-policy when re-coupling policy to a network instance. The consequent increase in policy complexity for instance, contributed to the obsolete rules found in each of our firewall case studies in Chapter 3. These obsolete rules were not enforced by the firewalls, but the high policy complexity hindered debugging of this appearance of security.

[1]The name derives from the analogy of not being able to see the forest for the trees.

Many of the security violations observed in our firewall case studies were also due to the lack of verifiability in existing firewall policy languages. How can security administrators confidently deploy their policies to a network if they can't even verify that the specified policies are industry best-practice compliant, prior to deployment? It's no surprise that the security of the networks in our case studies and others found in practice [128, 129] is broken-by-design. Verifiability is a central tenet of our design of *ForestFirewalls* as we will elaborate later in the Chapter.

A pre-requisite to our design of *ForestFirewalls* is a rigorous survey of the requirements of a firewall policy language and we conduct this below.

5.1 Requirements of policy framework

In *ForestFirewalls*, we enforce good designs on its users through the principles and requirements derived from our study of real SCADA firewall configurations in Chapter 3 and industry best practices [19, 21, 59, 117]. Namely

- *Single source of truth:* more specifically, "Security managers need a single place to look for the corporate policies on who gets in and who doesn't." [56]. Single source of truth is a general principle in computer science [15] and has a higher importance here, where a broken policy can have serious consequences.
- *Decouple structure from function* [89]*:* separating network topology (*i.e.,* structure) from policy specification (*i.e.,* function) allows administrators to maintain consistent policy in the presence of changes to network details. The separation also promotes policy reuse across multiple sites and keeps policy design unpolluted by vendor/device specifics. Thus, heterogenous network devices can be managed in a uniform manner and transition between vendors can become seamless. We will discuss this decoupling in more detail in Subsection 5.1.2.
- *Verify rigorously:* there is a clear danger in assuming any one piece of software functions correctly, from the firewall up to and including our own system. We must check the configuration works both pre- and post-deployment. Such rigorous verification requires strong mathematical underpinnings. The formal foundations we developed in Chapter 4 provide precise verification techniques for administrators to reason about their network security; for instance to (i) show that certain types of traffic flows are impossible; (ii) compare their security to industry best practices to check it complies; and so on. Verifiability is discussed in detail in Subsection 5.1.1.
- *Secure firewall management:* firewall management must be restricted to secure protocols such as SSH and HTTPS and a small set of hosts in the network [21, 117]. The firewalls must be made 'invisible' to all other hosts in the

network; *i.e.,* made stealthy. In particular, management of the firewalls that protect the perimeter of a SCADA-Zone must not be conducted from a non-SCADA domain. Existing policy languages do not enforce these best practices and as we observed in our SCADA case studies in Chapter 3, this has lead to insecure firewall management practices that can compromise the entire network.

- *No implicit rules:* our real SCADA case studies also revealed how implicit rules allow unexpected interactions, and undesirable consequences. Desired flows must be explicitly allowed.
- *Proactive security:* top-down provisioning of a network's security controls from high-level policy is a proactive way of protecting the network from unauthorised access. Pre-verifying and enforcing these security controls, prior to a cyber attack, can prevent significant, expensive damage to system infrastructure. This idea is analogous to banks using thick steel and concrete vaults with sophisticated electronic systems to detect and prevent break-ins.
- *Modularity:* the policy framework must cater for both non-technical users (*e.g.,* business managers) and technical users (*e.g.,* network programmers). Non-technical users need to define high-level policies to meet their business goals. Incorporating simple, modular and reusable libraries within the policy language (such as well known IANA services) enables easy construction of such high-level policies. On the other hand, technical users may want to extend the policy libraries for instance, to add support for new protocols. A modular policy framework design promotes such extensibility, but is lacking in existing firewall policy languages.
- *Firewall reporting:* our real SCADA case studies in Chapter 3 demonstrated how easy it is to either misconfigure firewalls or, for some part of the security setup to otherwise malfunction, either breaking the network or reducing its security. One important step to guard against such failure, is for firewall to report their status of security.
  This can be performed externally, and by using the diverse logs, alerts and alarms from the firewalls involved. We refer to these together as *firewall reports*. We will discuss firewall reporting in detail in Chapter 8.
- *Flexibility and extensibility:* our real SCADA case studies in Chapter 3 showed that different SCADA networks have different size, equipment and operational requirements. The firewall architectures employed within these networks can also vary. A policy specification framework must therefore be flexible to cater for these varying requirements. In addition, the framework must be extensible enough to allow new semantics to be added in future versions.
- *Rule order should not matter:* it must be possible to add, or subtract a policy rule without considering its effect on every other rule. None of the existing

firewall configuration platforms [24, 29, 34, 64] achieve this. Administrators using these tools must provide correct rules *and* maintain correct rule order to avoid adverse interactions. Our case studies in Chapter 3 showed that this added configuration burden leads to the many policy inconsistencies found in practice, making it impossible to know if an intended policy is actually enforced by a firewall.

- *Distributed management:* multiple staff, potentially at different locations should be able to collaborate on policy specifications.
- *Simplicity:* it is unnecessary to attempt to provide every possible security feature. At best, advanced features create confusion, and at worst, bad implementations can create security flaws.
- *Convenience:* security and convenience are often at odds, but wherever possible convenience needs to be provided. This is not a luxury – lack of convenience is one of the main reasons administrators circumvent *their own* security.
- *Scale at lower cost:* current approach to firewall configuration has a linear cost with the size of the network, at best. With increasing network size, the number of firewalls often increases and each firewall is often managed individually. A pragmatic policy specification framework must allow configuring groups of firewalls consistently and simultaneously to reduce management costs. Describing concise policies that contain only a small number of elements to express policy for a large number of network-devices, promotes usability and drives economies of scale.
- *Support layer-3 or higher network firewalls:* firewalls that operate at the Ethernet layer exist (*e.g.,* a Cisco ASA 5505 in 'transparent' mode [30], Lucent firewall brick [5]). Such a firewall forwards Ethernet frames based on their MAC addresses, between its interfaces. Traffic filtering is achieved by checking the IP packets inside the Ethernet frames against an explicit firewall ACL, prior to being forwarded.
  However, Ethernet layer firewalls impose serious design and usage limitations. For one, they only support two interfaces, so, a DMZ cannot be enabled. For another, troubleshooting is made more difficult by their lack of visibility (they are not router hops connected to devices). Thus, industry best practices recommend against deploying these firewalls in SCADA networks [117]. So, the policy framework should focus on network firewalls that are layer 3 and higher.

The policy framework we propose comprise a suite of tools to write, validate and debug policy, create real configurations, and test configurations, while meeting this list of requirements. We describe some of these requirements in more detail below, and then explain how our approach fits the requisites.

5.1.1 Verifiability

One requirement for our system underpins all of the others. In the context of SCADA networks, administrators require a high degree of trust that the network operates as intended. We simply cannot assume any software component functions correctly, from the network-devices up to and including the policy specification framework. It is necessary to verify that the firewall policies are specified and implemented correctly at every level possible. We discuss below, the dimensions of *verifiability* that need to be considered to achieve this objective.

Verify Policy is Correct

Our firewall configuration case studies in Chapter 3 and others [46, 73, 128, 131] have demonstrated that policy inconsistencies such as redundancies and conflicts are a common occurrence. These inconsistencies often lead to unintended consequences that result in, for instance, an *appearance of security*. An input high-level policy must therefore, be first checked for consistency to prevent such outcomes.

Checking policy consistency partially verifies its syntactic correctness; in addition, we must verify its semantic correctness. For instance, we need to check a specified policy against SCADA best practices to identify security violations. An example violation is when direct communication is enabled between the SCADA-Zone and the Internet [21]. In Chapter 4, we determined the formal foundations required in a firewall policy specification to verify semantic correctness.

Several properties make verifying policy correctness in a specification easier. We describe these properties below.

(1) Transparency: Policy frameworks often include multiple processing layers such as language compilation and policy refinement that reduce transparency; *i.e.,* they blur the ability to view the relationship between a change made and its result.

This black-box like behaviour is analogous to that of a modern Operating System (OS). For instance, the Microsoft Windows OS depicts storage of files in a hierarchical directory structure [50] for easy understanding. But the layers underneath this representation are non-transparent; you need special skills to learn how files are stored physically on disk.

Lack of transparency in security-management hinders locating and rectifying an incorrect decision. For instance, notifying a user that their input policy is not standards compliant, is of little use unless the incorrect semantics (the cause(s) of non-compliance), can be identified with clarity.

Particularly, in mission critical SCADA networks, the inability to quickly debug firewall configuration problems, can lead to security malfunctions or failures that result in catastrophic outcomes. The more transparent the security-management platform is, the more confidence security administrators will have of deploying and maintaining correct security policies in their networks.

(2) Human-comprehensible policy: The ability to author a security policy without errors, largely depends on (i) how easy it is to express the intentions of the authors in a policy language; and reciprocally (ii) the ability to understand *what exactly* a specified policy aims to achieve. A security-policy language therefore, needs to be easily *human readable and writable* to say the least.

A natural language based policy representation should make the policy human comprehensible. But due to the ambiguity inherent to natural languages, current state its processing requires significant improvements before such policies can be expressed [22]. It is more practical to specify policy in a simple language interpretable by current computers.

However, even popular security-policy languages are often human incomprehensible. For instance, XACML is an XML based security-policy language that is accepted as a *de facto* standard for access-control management in distributed systems [81]. The attribute-based language model offers flexibility, but, the syntax and semantics of its underlying XML structure makes it intrinsically difficult for humans to interpret [78]. Thus, a XACML policy author cannot know with precision, for instance, if a security policy blocks potentially-unsafe HTTP traffic from entering a protected SCADA-Zone or not.

Even simple XACML access-control policies can span hundreds of lines of code or more [51]. Keeping track of the myriad interactions within a policy by hand is almost impossible. A useful policy language must therefore, allow *concise* description of policies, in addition to being user friendly.

(3) Specialisation within networking: Division of labour is one of the fundamental hallmarks of modern culture. Specialisation allows people to work more efficiently, and effectively. For example, in building construction, architects do not learn how to lay bricks. They might learn some structural engineering, but it isn't necessary. Other specialists can perform that task.

Why then is network engineering still a monolithic area of expertise? We have network architects and/or security specialists, but in reality, they are highly-qualified network engineers. For instance, consider how network infrastructure manufacturers such as Cisco structure their certification programs. It is assumed, that a network architect will know as much, or more about devices than a person programming the devices. Decoupling policy from implementation removes this monolithicity by creating opportunities for specialisation.

Network specialisation also simplifies policy verification. For instance, it allows network architects to check with ease that their designed security policies meet high-level goals, in the absence of implementation-centric details. For another instance, network-engineers only need to verify that their policy implementations match the architects' designs and not high-level business goals.

Verify policy is feasible for target network and technology

A policy may also not be feasible for the target deployment network. For instance, the target topology may be different from that perceived by the policy creator; a common occurrence when high-level policies are built on abstract views of the underlying network. A policy author for instance, may use the Zone-Conduit abstraction [59] and consider two zones to be distinct in the policy specification, but in the absence of one or more firewalls enforcing a real separation between the zones, only a single zone may actually exist. Another example is when network administrators create out-of-policy connects, for instance, to support wireless, but again, effectively fuse two zones.

A policy may also not be supported by the underlying network's technology. For instance, a security policy may intend to filter application-layer traffic. But, if Deep Packet Inspection (DPI) capable firewalls are unavailable in the target network, the policy cannot be correctly implemented [21].

A proactive policy framework must therefore guard against deploying policies that are infeasible for the target network and its technology.

Verify expected security outcome pre-deployment

Policy-creator oversights can also cause a policy to be flawed [128]. Flawed policies can result in security holes or non-functional networks. A common instance is failure to enable required routing protocols. Debugging such problems can be confusing and slow and we prefer to avoid network disruptions in the interim. Checking a policy for oversights prior to deployment must be made *compulsory* and *automated*, to prevent users from bypassing the check altogether. Through this verification, administrators can confidently deploy correct policies to their production environments.

Verify expected security outcome post-deployment

Pre-deployment verification doesn't always guarantee expected security outcome in the real network. For instance, unintended security policy behaviour can still occur due to firewall software bugs [117]. So, it is also necessary to verify that the real network operates as intended, post-deployment.

Moreover, a policy may function correctly in the real network at first, but produce unexpected behaviour later, following the upgrade and/or patching of network firewalls [117]. A pragmatic policy framework must also allow continuous monitoring of the security mechanisms of a network post-deployment.

5.1.2 Decouple policy from network

Another requirement that is fundamental for our system is removing network intricacies (*i.e.,* decoupling the network) from policy. In practice, network architects and business managers decide what types of services are allowed through firewalls. Network engineers then implement these policies. Intuitively, separation of the network intricacies from policy specification better suits these distinct phases. Conceptually this is analogous to the separation of architects and building contractors in construction. Contractors don't, in general, decide what roof shape a building should have, for instance.

Network topologies also often change in response to new business needs, upgrades and service demands. This may alter the devices and administratively assigned parameters such as IP addresses and hostnames in the network.

In comparison, security policies are relatively static. These policies often only involve dozens of distinct services [97], so policy complexity is low compared to that of the network. This relative simplicity and invariant nature leads to a natural decoupling of policy from the network. The adage *"Structure and function should be independent"* [89] truly applies here.

Decoupling *structure* from *function*, has the following advantages: it

- *Provides high-level network configuration capability via vendor neutral policy:* so, management-level policy makers can configure their firewalls against simple business and engineering requirements without needing to translate them to complex machine-dependent scripts. As a result, the syntax and semantics of the high-level policy language can also be kept intuitively simple.
- *Centralises policy management and promotes reuse:* a single topology-independent organisation policy (*i.e.,* source of truth) can be maintained across sites.
- *Streamlines network changes and upgrades:* the policy can be quickly re-mapped to a new topology, retaining previous levels of protection. The device-independent configuration process also reduces potential disruptions in network upgrades.
- *Simplifies best-practice enforcement:* best practice standards can be specified with precision in the absence of proprietary details of specific networks.
- *Avoids vendor lockdown:* administrators can replace existing network-devices with another vendor's equipment that proves most cost effective, without needing to be fluent in the proprietary and possibly arcane configuration platforms that ship with these devices.
- *Helps enforce conceptual integrity:* maintaining a set of coherent design ideas in the presence of inter-twined implementation details is almost impossible. Separating architecture from implementation helps enforce conceptual integrity in the design [18].

Separating policy from the underlying network also helps to implement a centralised control plane for physically distributed networks. This is the foundation on which SDN is designed and implemented. However, approaches with this respect that are applicable to traditional networks are far and few. Policy Defined Networking (PDN) for instance, is one such solution, but it lacks several key features such as built-in verifiability. We will discuss PDN in detail in Subsection 5.2.1.

In the next section, we will describe our design of *ForestFirewalls* that meets the policy framework requirements discussed above.

5.2 Policy specification framework design

There are two main network-programming paradigms available for consideration when designing a policy specification framework: SDN and Policy Defined Networking (PDN) (or Policy Based Network Management) [42, 126]. We compare these paradigms below to identify the model that best fits our purpose.

5.2.1 Choice of network-programming model

Both SDN and PDN support high-level policies decoupled from network implementation. They both build on decentralised network control: a feature that helps deliver new functionality rapidly and drive operational expenditure savings [42]. The main difference is that SDN requires the use of a standard network API such as OpenFlow to manage heterogeneous networks, while PDN does not.

This requirement in SDN is too restrictive for SCADA networks: deploying OpenFlow capable devices is unrealistic in these mission critical networks where TCP is a recent innovation. High availability requirements in these control networks [21], mean the legacy network equipment deployed within them is unlikely to be upgraded except during a major overhaul of a plant (which might happen once in a decade).

PDN does not require an open network API such as OpenFlow; so, heterogeneous devices in traditional SCADA networks can be managed via support for proprietary configuration languages (*e.g.,* by use of templates) [126]. Users do not require learning these arcane languages; the PDN engine automatically invokes them when refining policies. So, users who are non-conversant with technology details required to deploy a policy can still manage the network with PDN using high-level declarative policies.

There is a clear need to make new and emerging technologies easier to manage by promoting multi-vendor interoperability in a PDN framework. So, the Internet Engineering Task Force (IETF) has introduced the Policy Core Information Model

(PCIM) model [80] (later extended to PCIMe [79]) in order to standardise this policy framework and achieve ease of management. As an added benefit, the network-management process is also simplified and automated.

The PCIM model is comprised of four core components: a Policy Administration Point (PAP), Policy Decision Point (PDP), a Policy Repository (PR) and a Policy Enforcement Point (PEP) [80]. The PAP module is used to define the high-level declarative policies that need to be enforced in the network. The module often includes a user interface to input the high-level policies, a resource discovery component to identify the network topology, users and applications in the network. PAP also includes policy transformation logic to ensure specified policies are consistent and feasible for the underlying network and refine the policy to device-level configurations for deployment [80]. Authored policies are stored in the central PR (often a LDAP directory or database), using a rule-based representation. The PDP retrieves the stored policies from the PR and communicates them through to the PEPs as necessary.

The PCIM model is a generic policy framework that aims to standardise how policy information is modeled. Implementations that suit specific policy disciplines such as QoS require extension of this generic framework to incorporate policy-discipline specific details, *e.g.,* QPIM [114]. This effort is best justified when the target network devices are IETF policy architecture compliant (so for instance, the devices can directly retrieve their policies from the PR). But, our target SCADA networks often include legacy devices that may not adhere to the IETF policy architecture. In such circumstances, additional functionality is required from the core PCIM components to distribute policies to network devices (*e.g.,* to configure the device-level policies using a command line interface) [126].

So, we employ the general concept of PDN for security management; *i.e.,* we leverage policies to alter network-security functionality without explicitly changing implementation details within network devices, but we build our own implementation of these concepts: *ForestFirewalls*. There are similarities in the functional components of *ForestFirewalls* to those in the PCIM model (for instance, we employ a simple Console-based user interface for policy administration and include policy transformation logic within it). But, we do not strictly adhere to the PCIM model in designing our policy specification framework and concentrate here instead on the more important end goal: reliable configuration of SCADA firewalls.

5.2.2 A modular approach

This section describes our system in more detail. We described in Section 5.1 how a useful network policy specification framework must cater for management-level policy makers as well as competent programmers. Policy makers need to define

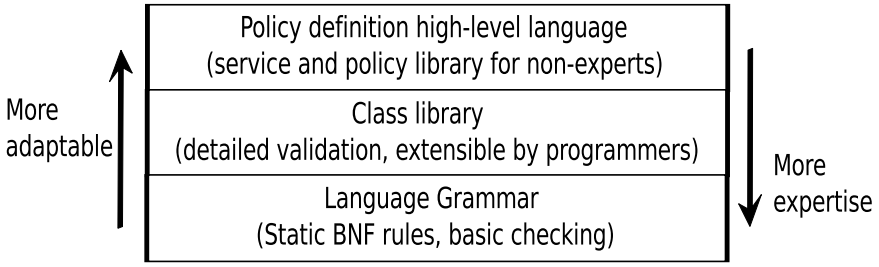


Figure 5.1: Modular policy specification.

high-level policies to meet business goals. Programmers may wish to extend the framework to add more features. A modular approach (Figure 5.1) supports both cases.

Policy definition high-level language: This component is designed primarily for non-expert users to define services and security policies. It uses a modular library of services and security policies in conjunction with a simple language. The language is based on the application whitelisting model discussed in Chapter 4. The service library consists of Internet Assigned Numbers Authority (IANA) well-known services and the policy library contains common SCADA best-practice security policies, all easily extensible by a non-expert user.

We primarily support network-layer firewall policies in the initial version of *ForestFirewalls*. These policies enable control of legitimate access to network applications and establish a strong security baseline. This baseline could be extended later to support policies for filtering traffic at higher layers (*e.g.,* at the application layer) to for instance, distinguish between allowed and prohibited use of these applications.

In addition to security policies, the *ForestFirewalls* language also supports specification of high-level reporting policies.

Class library module: This is an Intermediate Language (IL) dedicated to expert Programmers. The Class library features an Object Oriented Programming (OOP) based and well-defined object hierarchy that consists of rules for constructing TCP and UDP based services. The respective classes handle the detailed checking of object specific attributes (*e.g.,* TCP/UDP port numbers are between 0-65535).

A direct mapping between the grammar rules and the Classes makes the library easily extensible, but it is only intended that expert protocol engineers will extend this. Most operators will use the higher layer.

Language grammar: This component is dedicated for the designers of *ForestFirewalls*. It consists of Backus-Naur Form (BNF) rules that control our language semantics. The grammar includes basic checking such as argument length and null checks, but delegates detailed checking to the class library module. The language

is static and can only be altered by us; the language designers. This preserves the original objectives of a high-level abstraction that is intended to change slowly.

The overall architecture leads to a vendor and device neutral policy-specification framework. The system suits naive users, but the framework is easily extensible to cater for new network applications and protocols.

5.2.3 Verification tiers

The verifiability requirements discussed in Subsection 5.1.1 must be met by the design in a modular, systematic manner. *ForestFirewalls* uses multiple verification stages to achieve this goal (Figure 5.2).

Upper verification tier: the framework initially checks a specified firewall policy for correctness. So, it automates policy consistency checks (*i.e.,* we consider only redundancies here because conflicts are removed by design). Formal model checking tools such as Alloy can increase the accuracy of detecting these inconsistencies. *ForestFirewalls* automatically constructs tests in Alloy [60].

This verification stage also involves checking a policy for compliance with industry best practices [21, 117]. These automated compliance tests are critical in SCADA networks where more restrictive practices are required to minimise their vulnerability to cyber threats. The verification stage also checks the policy is feasible for the target network and technology.

Middle verification tier: the second verification stage primarily involves debugging configuration problems prior to deployment. Network emulation offers a cost effective way to test configurations [70]. For instance, the *Netkit* open source software package [91] provides an emulation platform with virtual devices and interconnections via User Mode Linux (UML). Pathological traffic tests, automated together with Netkit emulations, can help verify that the generated configurations produce the expected outcome prior to deployment. *ForestFirewalls* creates such emulations and allow them to be deployed through its command line.

Lower verification tier: the final stage guarantees that the real firewalls operate as intended, post deployment. The automated tests can be extended from emulations to the real network, to generate live-traffic and reveal unexpected configuration behaviour in the real firewalls.

5.3 Design outcomes

Details of the language syntax and semantics will be discussed in Chapter 6, but we want to briefly summarise how our design decisions meet our requirements.

Several outcomes stem from our design. For instance, building our high-level policy specification on the refined Zone-Conduit model derived in Chapter 3

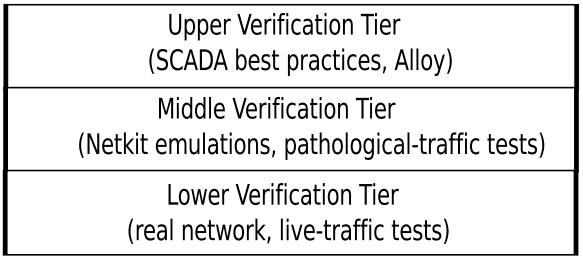


Figure 5.2: Policy verification tiers.

means the specifiable policies are decoupled from the network-implementation, allowing us to maintain a single network-wide policy (*i.e.,* source of truth) to easily assess who gets in and who doesn't. Decoupling also helps to keep the syntax and semantics of the high-level language intuitively simple for non-expert users.

For another instance, the explicit, application-whitelisting model adopted by *ForestFirewalls* renders the rule order irrelevant, so, policy makers need not consider the order when adding or removing policy rules. By being explicit, we also prevent services being accidentally enabled implicitly, enforcing the secure-by-design doctrine.

ForestFirewalls policies are also mapped to the conduits in the network using the formal semantics derived in Chapter 4. Since a conduit can be implemented by one or more network-firewalls; the high-level language allows to define,validate and debug policy for a group of firewalls at once, enabling it to scale at lower cost.

The modular design of *ForestFirewalls* also helps to dedicate the use of the high-level language, the Class library and the language grammar to users with varying levels of technical expertise. The design minimises the duplication of functionality across these layers, enabling easy maintenance and checking.

In addition, our use of multiple verification tiers allows us to proactively verify correct security outcome from the firewall(s) up to and including our system. For instance, the best-practice policies built into our high-level language enable policy compliance checks and the Class library supports checking for policy inconsistencies. Such proactive measures however, may sometimes be inadequate against some cyber attacks (*e.g.,* zero-day vulnerability attacks), prompting for more reactive measures such as a disaster recovery plan.

Many of the design principles we adopt in *ForestFirewalls* favour simplification, but it primarily stems from centralisation and the use of security abstractions. Centralisation allows network-wide firewall configurations to be specified at a single point (*i.e.,* a single source of truth) and security abstractions allow policy to be defined using a language closer to an organisation's business needs rather than a specific technology needed to deploy it.

5.4 Conclusions

Developing a reliable and efficient firewall policy specification isn't easy; only adhering to coherent design principles and requirements will make such a policy specification pragmatic. Existing firewall policy languages fail to apply such principles and requirements consistently and produce complex, uncoordinated firewall features that lead to the broken-by-design network-security found in practice.

We addressed these shortfalls here in our design of *ForestFirewalls:* a high-level firewall policy specification language that is built on mathematical foundations. The formal semantics enable verifiability: a property that is fundamental to making the system transparent, easily debuggable and providing administrators assurance of expected security outcome. Our application of coherent concepts and relationships throughout the design of *ForestFirewalls* enforce conceptual integrity. As we will see in the next chapter, this leads to a set of simple, coordinated policy specification features that is easy for users to administer correctly. We will also realise our design into a software prototype in the next chapter and apply it to the real SCADA case studies from Chapter 3 to demonstrate its use.

Chapter 6

ForestFirewalls Implementation

In the previous chapter, we described the design of our high-level firewall policy specification language: *ForestFirewalls*. We applied coherent design principles to the problem and considered the key underlying requirements of a security policy specification. We now describe the realisation of this design in a software prototype that supports simple, coordinated policy specification features that are easy to administer correctly.

6.1 System Overview

We start by describing the implementation of our policy framework as depicted in Figure 6.1. A summary of each system component is outlined below. We will describe the details of these components later in the Chapter.

High-level security policy: The topology-independent policy input file created using *ForestFirewalls*. See Section 6.2 for details.

Compile to intermediate-level (IL) policy: Parses the high-level policy into an intermediate format for checking.

Network topology: The input network topology described in the XML-based graph file format *GraphML* [48]. The file contains information of all relevant devices of the underlying network and their interconnections.

Generate network-level, vendor-neutral policy: Couples the security policy with network details to create a network specific, but still device-independent specification. See Section 6.3.

Verify IL policy against best-practices, via Alloy: Formally checks an Intermediate Level (IL) policy for SCADA best-practice violations and for correctness. Best-practice checks employ the mathematical semantics developed in Chapter 4 while a model checker, Alloy [60], finds anomalies within the policy. See Section 6.4.

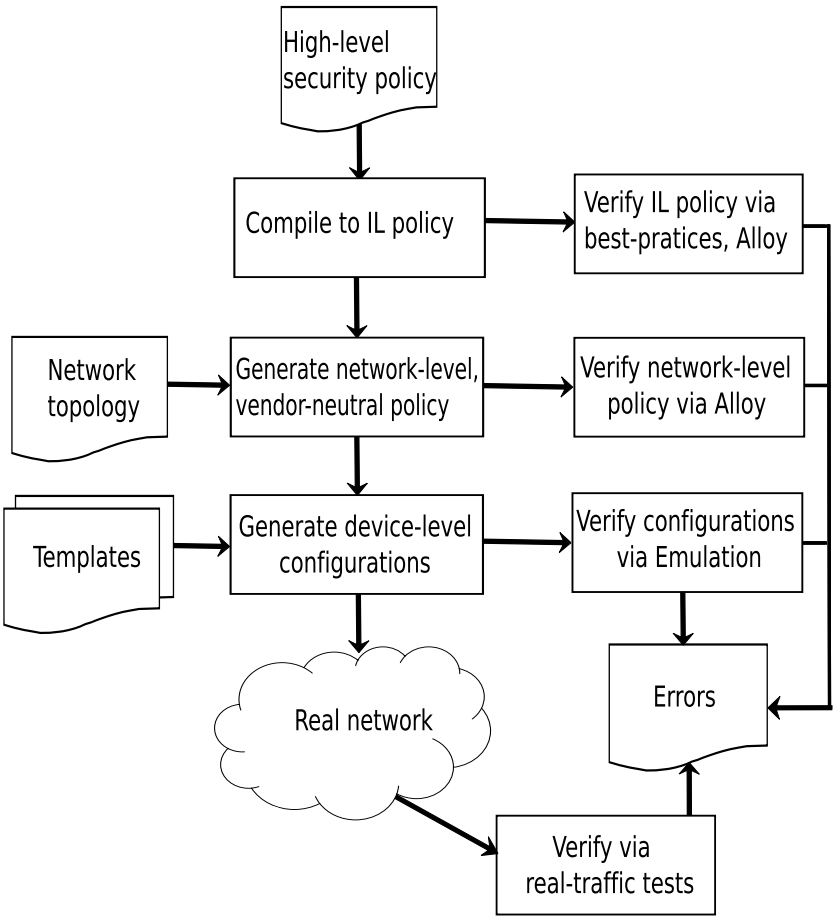


Figure 6.1: Firewall auto-configuration process.

Verify network-level policy via Alloy: Formally checks network-level policy for correctness via Alloy [60]. See Section 6.4.

Device templates: A vendor and device-specific meta-configuration repository that supports layer 3 and higher firewall models (currently it supports User Mode Linux (UML) IP-Tables, Cisco ASA5505 models, and can be extended with ease).

Generate device-level configurations: The rendering of device-specific configurations for firewalls using the network-level policy and the device templates.

Verify via emulations: Device configurations are pushed to an emulated network for pre-deployment testing. Test scripts are auto-executed in this network, to generate pathological traffic and validate configurations. See Subsection 6.4.4.

Real network: Device-specific configurations are pushed to hardware in a real network. At present this is conducted manually, but we intend to automate it.

Verify via real traffic tests: Automated tests are created for the real-network, generating real-traffic, to verify post-deployment behaviour of firewall configurations. See Subsection 6.4.4.

ForestFirewalls is a Policy Defined Networking (PDN) solution. So, the primary task of its users is to author declarative and technology independent firewall policies using its policy language. Therefore, we begin our detailed description of the system implementation by outlining the policy language first.

6.2 ForestFirewalls high-level language

The *ForestFirewalls* specification language allows a user to instantiate a high-level firewall policy. Below is the definition (a complete example can be found in Chapter 7).

ForestFirewalls' parser is currently implemented in Python and Ply (a `lex` and `yacc` implementation for Python). It translates a *ForestFirewalls* specification (*i.e.,* a `.policyml` file) into its Intermediate Level (IL) representation using object definitions from the underlying class library, also implemented in Python.

6.2.1 Service and Service-group description

A service is defined using the syntax

```
service <service-name> { protocol=<protocol-base>;
                         <protocol-attributes-list>; }
```

For example, a custom implementation of HTTP, based on the above service description format is given by

```
service custom_http {  protocol=tcp;
                       tcp.dest_port=8080;
                       comment=``Internal Web''; }
```

All unspecified attribute values have defaults assigned, for instance, here `tcp.source_port=1024-65535` is implicit. The language supports a modular library of services that comprise of IANA well known ports (*i.e.,* `iana_services`) and is easily extensible.

Service specific comments are enabled via the `comment` field. This type of self-documenting code allows commentary in the lower tiers to be auto-generated. The aim is to help document network and device level firewall rules to avoid the common problems, *e.g.,* rules that cannot be deleted because no one remembers why they exist.

ForestFirewalls prohibits the description of *generic services* such as *all-TCP* or *all-IP* for several reasons. For one, our SCADA case studies in Chapter 3 reveal that users exploit generic rules where possible for convenience, such as allowing *all-IP* traffic just to enable EIGRP traffic. Doing so, leads to invisible breaks in firewall security because far more services than necessary are thus admitted.

For another reason, such broad services don't contribute towards forming well-defined security policies. They decrease transparency; *i.e.,* they cloud the ability to precisely understand the type of cyber threats a network is being protected from.

We do allow a `service-group` to bundle services and other service-groups (*i.e.,* nesting is allowed), to enable concise and clear specification of rules, *e.g.,*

```
service_group <group-name>{<service-or-group-list>}
```

A `service-group` is essentially a set of `service` objects. *ForestFirewalls* supports the standard set operations: union (`,`), intersection (`^`) and difference (`\`), so, new service groups can be constructed by applying these operators on existing service groups.

Service-groups provide a level of indirection, so application protocols used to achieve network functionality (*e.g.,* Web services) can conveniently change without needing policy alterations.

The following snippet defines a `service-group` containing various example Web services

```
service_group Web { http, https, dns }
```

We will note that some checks are applied to the construction of groups, *e.g.,* to avoid redundancy.

Multi-channel service description

Some services use a fixed control channel and a dynamically agreed data channel to transfer bulk data. An example is FTP which uses TCP port 21 for its control channel and data channels that use dynamic TCP ports (*i.e.,* port numbers 1024-65535). Firewalls enabling these services need to parse the underlying application layer protocol to open the ports that were negotiated on the fly. But, allowing dynamically negotiated inbound connections can increase the vulnerability of a network behind a firewall. So, in the case of FTP, industry recommends enabling passive mode FTP instead [21, 117], which uses TCP ports within a pre-defined range for its data channel (an example is shown below). Inbound connections to this port range can then be enabled by a firewall policy in advance.

Vendors also often pre-define these services with special syntactic constructs to control them (*e.g., ftp* in Cisco PIX and ASA firewalls [30]). But, in *ForestFirewalls* we ignore these special vendor constructs and simply define multi-channel

services using regular port numbers. We do so, because vendor implementations of these special constructs do not always comply with industry best practices and are not portable. For example, we specify passive-mode FTP as follows:

```
// passive-mode FTP data channel
service ftp_data {  protocol=tcp;
                    tcp.dest_port=24500-24600; }

// FTP using standard control channel and
// passive-mode data channel
service_group ftp { iana_services.ftp_control, ftp_data }
```

Connection-like UDP or ICMP services

Connection-less protocols such as UDP and ICMP can also be used in applications with connection-like semantics. For instance, consider the *Ping* program. Sending a `ping` from host X to host Y causes *icmp-echo-request* packets to be sent from X to Y. In response, *icmp-echo-reply* packets are returned from Y to X. *ForestFirewalls* automatically generates and incorporates supplementary rules for these known special cases as necessary, to simplify configuration of such services.

6.2.2 Zone and Zone-group description

The following syntax allows a *ForestFirewalls* user to specify the Zone-Conduit graph on which the high-level policy is based upon. This graph is the network-security model perceived by the policy author.

```
load_zone_conduit_model ``<full-path-of-zone-conduit-graph-file>"
```

A zone in *ForestFirewalls* is then a label that matches a node label in this input Zone-Conduit graph.

A `zone-group` bundles a set of zones or other zone-groups and is defined using the syntax

```
zone_group <group-name> {<zone-or-group-list>}
```

A `zone-group` (*ZG*) is a set of zones. So, if $\Omega = \{\textit{all zones in the network}\}$, then $ZG \subset \Omega$. When a service-flow is enabled to/from a `zone-group`, each zone in the group *inherits* the same ability; *i.e.,* the same service flow is conveniently enabled for each zone.

Multiple zone-group declarations are automatically checked for duplicates to minimise code redundancy. The snippet below describes an example zone-group; `three_zones`, which is made up of three zones.

```
zone_group three_zones {corp_zone, scada_zone, dmz}
```

We can see that our high-level policy is easily adapted to incorporate new zone additions to a network. The updated policy can be quickly re-implemented on the network to protect the new zones. We also allow similar syntax and set operators as for defining groups of services.

6.2.3 Policy-rule description

A high-level security policy rule is defined using an `operator` to direct the *inter-zone flow* explicitly, as per below. The application whitelisting model we adopt makes the rule-order irrelevant and the required operators simple; the operators must enable unidirectional and bidirectional flows. The end-zones defined by `zone-or-group-name` have traffic flow of type `service-or-group-name` enabled. Permitting service flows at a zone-level (rather than a host-level) keeps the policy rules user friendly and concise.

```
policy_rule <rule-name> { <first-zone-or-group-name> operator
                          <second-zone-or-group-name> :
                          <service-or-group-name>  }
```

where `operator == '->' or '<->'`.

For example, the following policy allows `Web` traffic (see service group definition) from the enterprise network to the DMZ.

```
policy_rule corp_web_rule {Corp_zone -> DMZ : Web}
```

Formally, the above description can be represented as a policy rule q_m^a with accept packet set $A = \{s \in \mathcal{A} \mid s \prec m\}$ where predicate m is given by

$$m = (s.hdr.source\_address\ in\ Corp\_zone \wedge \\ s.hdr.dest\_address\ in\ DMZ \wedge \\ s.hdr.service == Web).$$

The policy specification in *ForestFirewalls* is network implementation independent because the policies are specified against an abstract-network model (*i.e.,* Zone-Conduit model) and not a concrete network instance. The decoupling allows network architects and business managers to decide on what flows are permitted and not be concerned with how to implement the policy on a target network. Users have a much reduced configuration burden, as they can be oblivious to the different firewall architectures employed within networks and the diverse network sizes.

6.2.4 Policy description

We define a `rule_group` to hold one or more security-policy rules as per below. The semantics of the policy rules are order-independent; so, a `rule_group` is essentially a set of `policy_rule` objects. New rule groups can be constructed by applying set operators on existing rule groups. All services that are not explicitly enabled by the set of policy rules are denied by the implicit *deny-all* rule associated with a `rule_group`.

```
rule_group security_policy {<rule1>, <rule2>, ... }
```

Likewise, a `reporting_rule` defines a firewall reporting policy (we will discuss reporting policies in detail in Chapter 8):

```
reporting_rule reporting_policy {<attributes-list>}
```

A global `policy` object (*i.e.,* single source of truth) then encapsulates a security policy with a reporting policy to ensure firewall security is monitored as soon as traffic flow is enabled. The modular approach allows reuse of both types of policies (which may come from a library), but couples them together in the final specification.

```
policy <policy-name> { security_policy,
                       reporting_policy }
```

We will illustrate via a concrete example, later in Chapter 7 how this global `policy` is easily human comprehensible. Its security policy component is a composition of conflict-free high-level policy rules. The reporting policy component consists of high-level attributes (*e.g.,* reporting use case) and attribute values. Both components describe exactly what they aim to achieve allowing them to be easily understood. The increased human-comprehensibility makes troubleshooting and maintaining policies relatively straightforward. Of course there is a requirement for users to use standard coding practices, *e.g.,* good names for groups, but ease of use facilitates this behaviour.

6.2.5 Topology description and mapping policy to topology

ForestFirewalls does not require users to describe the details of the target network topology or map policy onto the target network explicitly through the policy specification. As we described earlier, the topology is provided via a separate GraphML file. The Zone-Conduit abstraction and the algebras developed in Section 4.4 perform the mapping of policy to topology transparently underneath.

6.2.6 ForestFirewalls library file imports

In large industrial control plants, multiple policy sub-domains exist that set their own policies to be applied to the network components they own or manage. Namespace importation facilitates such distributed policy management and simplifies reuse of policy. For instance, we can imagine the ISA creating a best practice ruleset for SCADA as a baseline for new installations.

Non-expert programmers may, however, be unfamiliar with the use of a complex namespace library with many features. We developed a namespace hierarchy that is simple, yet provides rich features for managing and reusing namespaces. *ForestFirewalls'* namespace hierarchy consists of generic library definitions for all users with additional support for custom namespace creation. This allows one to compose distributed policies into a coherent policy set, free of inconsistencies.

It's worth mentioning here that in practice, administrators sometimes disable filtering rules on the trusted 'inside' firewall interfaces to save CPU resources. But doing so enables *all-IP* traffic to go through the firewall by default, which is not intended by the high-level policy. This leads to invisible breaks, which makes it impossible to know why traffic filtering does not function as expected in a network. Moreover, if a policy specification allows implemented policy to mismatch that intended, how can it provide assurance of the protection provided by the security policies for a network? A specification that permits disabling of firewall rule generation for selected firewalls or firewall interfaces is hence *broken-by-design*. *ForestFirewalls* prohibits doing so, and is rigorous in ensuring that every flow enabled in the network matches explicit high-level policy intent.

Once a high-level policy has been authored in *ForestFirewalls* and provided as input (as a `.policyml` file), the policy framework refines it first to network-level and then to device-level (*i.e.,* firewall-level). We outline next, the details of this policy refinement process.

6.3 Policy refinement

A *ForestFirewalls* policy is first compiled to an Intermediate Level (IL) language and checked for correctness: *i.e.,* consistency and best-practice compliance. If correct, the policy can then be implemented on a network instance, by re-coupling the two. Re-coupling essentially involves mapping the zones in the high-level policy to hosts and/or subnets in the network implementation. The resultant network-level ACL rules are vendor and device neutral. This generic format enables easy checking of ACL rules for inconsistencies such as redundancies.

Implementing policy on a network instance is conducted via a Console-based User Interface (UI), which applies the policy transformation logic outlined below. This UI is analogous to the PAP of the PCIM model discussed in Subsection 5.2.1.

(i) Zone-Conduit model construction*: ForestFirewalls* first builds a *Zone-Firewall model*, containing the disjoint zones and their firewall interconnections from the input network topology. Additional Firewall-Zones, Abstract-Zones and Carrier-Zones are added to the model as required.

Next, the conduits are automatically defined to create the Zone-Conduit model of the input network. In doing so, *ForestFirewalls* generates a mapping between each zone and its IP address range. Each IP range is described by the smallest number of CIDR blocks that exactly cover it, to keep the rule base compact.

This real Zone-Conduit model may not *always* match that perceived by the high-level policy creator. So, the real model is checked against the perceived Zone-Conduit model provided through the high-level specification. If *mismatched*, an error is reported indicating incompatibility.

(ii) High-level policy rule translation: We created an implicit mapping between each zone and its host/subnet composition in step (i) above. The mapping readily translates the high-level policy to network-level. The source and destination zone of each high-level rule can be substituted with the corresponding IP address range(s) using this mapping. The network-policy equivalent is the cross product of the IP address range(s), the original rule operator and service description.

Multicast rules may also be required for the correct operation of certain protocols. For instance, when the user specifies OSPF as a dynamic routing protocol, multicast rules are required to enable neighbour relationships to correctly form within a single OSPF area [82]. Likewise, stateful protocols such as TCP require return path rules in addition to the forward path rules for correct operation. *ForestFirewalls* handles these requirements automatically and generates and incorporates any supplementary rules as necessary. This avoids the problem that low-level policy implementations have to know detailed policy syntax.

(iii) Mapping high-level policy to network firewalls: The system next employs the policy-to-firewall mapping algebras developed in Section 4.4 to map the high-level Zone-Conduit policies to the network firewalls. These algebras automatically eliminate firewall paths that are deemed invalid. For instance, a valid path does not allow traffic to transit a Firewall-Zone. A Firewall-Zone only enables traffic flow to and from a firewall but cannot forward traffic.

Our high-level policy can incorporate new zone additions to a network with ease. The policy-mapping algebras then allow the updated policy to be swiftly re-implemented on the network to protect the new zones.

(iv) Firewall-configuration generation: *ForestFirewalls* automatically renders the device-specific configurations from the network-level policy, using vendor and device-specific *Mako* templates. Mako is a template library written in Python [75] that enables fast and easy integration into *ForestFirewalls*. Mako templates have been used to generate virtual-machine configurations for emulated networks [70].

In our initial version of ForestFirewalls, the vendor and device-specific meta-configurations repository containing these templates supports UML IP-Tables and Cisco ASA5505 firewall models. But, this repository is easily extensible.

We describe next our implementation of the policy verification steps in Figure 6.1. Collectively, these steps constitute our tiered verification design in Chapter 5 and increase the overall transparency of the policy specification (so users can know if their policies will work or not).

6.4 Verification

ForestFirewalls starts by checking the IL policy for correctness. Incorrect policies can stem from redundancies and conflicts. But, our system only supports *positive permissions*. So, conflicts are removed by design. Redundancies are still possible, so we check for these as per below.

6.4.1 Redundant-rule detection

We use a model checker: Alloy [60], to find redundancies in the IL policy. Formal model checking is generally complex, so Alloy aims to find counter-examples to illustrate problems. In essence Alloy is a refuter [60] not a prover. But, its ability to analyse a model, even within finite bounds, makes it very useful [60].

A partial snippet of the Alloy specification (*i.e.,* `.als`) file auto-generated by *ForestFirewalls* for an IL policy is shown in Listing 6.1. It depicts a formal model with three signatures: `Service`, `PolicyRule` and `SecurityPolicy`. In our current model, a `Service` has the basic members: `ip_protocol`, `source_port`, `dest_port` and `icmp_type`. Of these members, only `ip_protocol` is mandatory.

A `PolicyRule` has four members: `zone1` and `zone2` to capture the zone names, an `operator` and a single `service` element. The global constraints are partially shown (lines 17–26), requiring the universal set (*Univ*) of `PolicyRule` to comprise exclusively of rules in the policy. The universal set of `Service` must also comprise of `PolicyRule` services.

Predicates can determine if two given rules or services overlap (not shown). `Service` overlaps are found by computing their intersection and testing if the result has members. *String* type members (*e.g.,* `zone1`, `zone2`) can be directly compared. `PolicyRule` overlaps are checked in a similar fashion.

A 'no_rule_overlaps' assertion (also not shown) is used to locate distinct rules with overlapping criteria. If found, a counter-example is returned, indicating potential redundancies in the IL policy (and the high-level policy). Counter-examples can be inspected through Alloy's GUI to find the underlying cause(s).

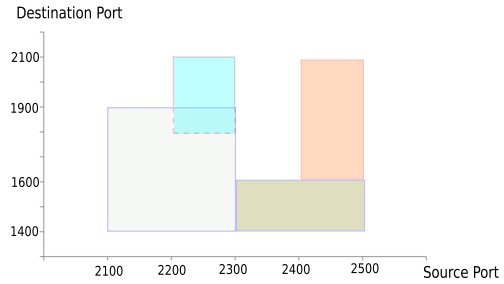
Listing 6.1: High-level policy verification framework using Alloy (partially shown).

```
1  abstract sig Service {
2     ip_protocol: some Int,
3     source_port: set String,
4     dest_port: set String,
5     icmp_type: set Int }
6
7  abstract sig PolicyRule {
8     zone1: one String,
9     zone2: one String,
10    operator: some Int,
11    service: one Service }
12
13 // Policy definition
14 one sig SecurityPolicy { rules: some PolicyRule }
15
16 // List of global constraints
17 fact {
18
19   // All defined rules are in the policy to check
20   all r: PolicyRule | r in SecurityPolicy.rules
21
22   // Policy rules make up universe of PolicyRule
23   SecurityPolicy.rules = PolicyRule
24
25   // A service belongs to at least one PolicyRule
26   all s: Service | some r: PolicyRule | s in r.service }
```

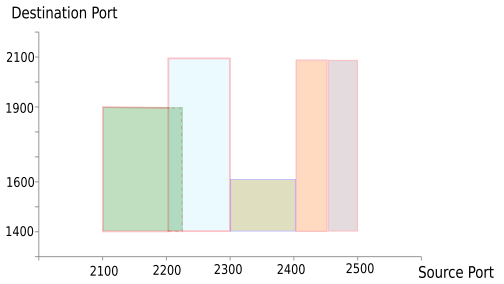
At the moment, Alloy is itself run manually and output counter-examples help debug data. We do not auto-correct rules as this requires human discretion.

When overlaps are absent in the IL policy, there must also be none in network-level policy. But we cannot simply 'trust' our system to always consistently generate network-level policy. So, we re-check the generated policy for overlaps using Alloy and verify the fact.

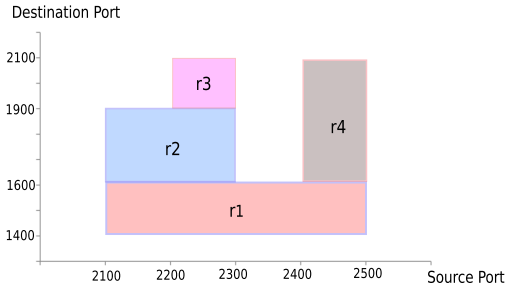
The network-policy equivalent Alloy export is similar to the IL export, but has the source and destination zone names replaced with IP address ranges in an `ACLRule`. The `Service` signature also has members depicting protocol state. Again, we use an assertion to check for `ACLRule` overlaps. We will demonstrate the use of this redundancy checking framework via an example later in Chapter 7.



(a) Four rules shown by (overlapping) rectangles.



(b) Five rules giving an equivalent policy to (a).



(c) Horizontal partitions of polygon in (a) or (b).

Figure 6.2: Canonicalisation of distinct rule sets of the same policy. Rectangles indicate the packets allowed by a particular rule.

In addition to redundant-rule checks, the IL policy verification step in Figure 6.1 also involves testing the policy for best-practice compliance to ensure semantic correctness. We present next, our implementation of the policy-comparison algebras developed in Chapter 4 to test compliance.

6.4.2 Best-practice compliance

A given policy semantic can be constructed from multiple distinct rule sets. We defined in Chapter 4, a canonicalisation mapping $c : \Phi \rightarrow \Theta$ to compare policies efficiently. Here, Θ is the canonical space of policies and all equivalent policies of Φ map to a singleton.

Identical policy semantics imply that the rule sets allow the same set of services between them. Figures 6.2(a) and 6.2(b) illustrate the ideas based on TCP port filtering of single packets. Each rectangle indicates the allowed packets of a single rule. Combined, the rules cover the same set of allowed packets.

We can dissect the polygon formed in our example policy in Figure 6.2 (a) or (b) into horizontal partitions (Figure 6.2(c)), using a Rectilinear-Polygon to Rectangle conversion algorithm [47]. Each partition is chosen to guarantee its

uniqueness in a provable manner. Canonical policy elements are derived by translating each partition back to a rule and ordering the resulting rule-set uniquely in increasing IP protocol number and source and destination port numbers.

Our algorithm to perform this partitioning is designed to be fast, rather than providing a guaranteed minimal partition. The result is a deterministic, ordered set of non-overlapping rules.

We test the best-practice compliance of a *ForestFirewalls* specified policy using their canonicalised policy forms and the *inclusion* and *incorporation* semantics defined in Subsection 4.3.2. The algorithms that implement these semantics are described below.

We denote the recommended policy by $\mathcal{RP} = (G_2, P_2)$ and the policy we check by $\mathcal{P}_1 = (G_1, P_1)$.

Check whether $\mathcal{P}_1 \subset \mathcal{RP}$*:* For *inclusion*, we first derive the semantic partitions $SP_1 = \{e_1, e_2, ..., e_m\}$ and $SP_2 = \{e'_1, e'_2, ..., e'_t\}$ with equivalence class sizes given by $\{c_1, c_2, ..., c_m\}$ and $\{c'_1, c'_2, ..., c'_t\}$ respectively. Our algorithm then checks $SP_1 \subset SP_2$ as per Subsection 4.3.2.

If $\mathcal{P}_1$ is included, we then determine the feasible mappings between the two semantic partitions. So, our algorithm finds *inclusive* policy mappings and *equivalences*. Table 6.1 captures this setting and illustrates the process by considering example mappings between SP_1 and SP_2 as shown. The set of all feasible mappings between these partitions are given by $\{M_1, M_2, ..., M_m\}$.

We determine the cardinality of each M_i's in the table, by taking equal number of elements from the corresponding classes and then considering all permutations. For instance,

$$|M_1| = (c_1)! \times \binom{c'_1 + c'_2}{c_1}, \tag{6.1}$$

where we know that $c'_1 + c'_2 \geq c_1$ in order for the cardinality constraints to be satisfied. More generally

$$|M_i| = (c_i)! \times \binom{c''_i}{c_i}, \tag{6.2}$$

where $c''_i \geq c_i$ is the *total size* of all matching classes.

All Feasible Mappings (FM) between $\mathcal{P}_1$ and $\mathcal{RP}$ stem from the cross product of the class-level mappings in Table 6.1; *i.e.,*

$$FM = M_1 \times M_2 \times \ldots \times M_m. \tag{6.3}$$

We then derive the line digraphs: $\mathcal{L}(G_1)$ and $\mathcal{L}(G_2)$, of the Zone-Conduit models of P_1 and RP. We construct the adjacency matrices A_1, A_2 of these line digraphs for each mapping in FM in (6.3). If there exists a mapping for which $A_1 = A_2$, then $\mathcal{P}_1 \subset \mathcal{RP}$.

Table 6.1: Illustration of the policy inclusion process using example inclusive equivalence-classes of two semantic partitions SP_1 and SP_2. The first line of the table for instance, implies that $e_1 \subset e'_1$ and $e_1 \subset e'_2$.

SP_1 class	class size	included in SP_2 class(es)	total size of SP_2 class(es)	mappings-set
e_1	c_1	e'_1, e'_2	$c'_1 + c'_2$	M_1
e_2	c_2	e'_2	c'_2	M_2
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
e_m	c_m	e'_1, e'_3, e'_t	$c'_1 + c'_3 + c'_t$	M_m

The time complexities of the algorithm components are given in Table 6.2. The dominant component of the algorithm in the worst case is the evaluation of FM, with time complexity $O(\prod_{i=1}^{m} c''_i!)$. In the worst case this is no better than that of an exhaustive algorithm ($O(n!)$ [94]), but in the best case where the equivalence classes are singular, then the algorithm performance is $O(n^2)$. We will describe in Chapter 7 why, in real networks, this is the typical real performance.

Check whether $\mathcal{P}_1 \equiv \mathcal{RP}(G_1)$*:* For *strict incorporation*, the node-count (NC) and edge-count (EC) must be such that ($NC_{G_2} > NC_{G_1}$ and $EC_{G_2} \geq EC_{G_1}$) OR ($NC_{G_2} = NC_{G_1}$ and $EC_{G_2} > EC_{G_1}$). We first check these conditions.

We derive SP_1 and SP_2 and check if every SP_1 class has an equivalent SP_2 class (with equal or higher class size). $\mathcal{RP}$ can *strictly incorporate* $\mathcal{P}_1$, only if this requirement is met. We then generate the FM by examining all of the permutations of policies within equivalent partitions. For each feasible mapping in the set FM, we construct and check their adjacency matrices for a match.

Check whether $\mathcal{P}_1 \subset \mathcal{RP}(\mathbf{G_1})$*:* For *partial incorporation*, we follow the approach of strict incorporation, but test if $SP_1 \subset SP_2$ instead of testing for equivalence. We perform this test by determining the feasible mappings between the two semantic partitions as per the approach in Table 6.1 and construct FM. Finally; we determine the adjacency matrices for each mapping in FM and check for a match. If a match is found, then $\mathcal{P}_1 \subset \mathcal{RP}(G_1)$.

The *strict-* and *partial-incorporation* algorithms have the same time complexity of that in *inclusion* above, with similar best and worst-case values.

We will use these inclusion and incorporation algorithms later, to automate the best-practice compliance checks of our case studies discussed in Chapter 3.

As a consequence of these compliance tests, firewall-management policies in *ForestFirewalls* are restricted to the industry recommended (secure) protocols. Our system also warns users when a SCADA firewall is attempted to be man-

Table 6.2: Time complexity of the semantic-partitioning algorithm components used to check policy inclusion.

algorithm component	time complexity	comments
cannonicalise policy	$O(n)$	$n =$ # of flow-policies
derive semantic partitions	$O(n^2)$	$n =$ # of flow-policies
construct line digraph	$O(n^2)$	$n =$ # of flow-policies
check partitions are inclusive	$O(mt)$	$m,t =$ semantic-partition sizes
evaluate mappings in FM	$O(\prod_{i=1}^{m} c_i''!)$	$c_i'' =$ total size of matching classes

aged from a non-SCADA domain. By placing the firewall interfaces in a separate Firewall-Zone on their own, our system automatically conceals the firewall from all hosts except those with explicit firewall-management privileges. Collectively, these steps enforce the secure firewall management best-practices of [21, 117].

Supplementary to best-practice compliance tests, it is also necessary to verify that the policy refinement steps in Figure 6.1 are provably-correct. Our policy-to-firewall mapping algebras developed in Chapter 4 help achieve this objective. In the next section, we describe the algorithm that implements these algebras.

6.4.3 Mapping policy to network-firewalls correctly

ForestFirewalls uses the right iteration algorithm described in Chapter 4 to calculate a matrix A^* that contains the valid policy-to-firewall mappings. We describe below how (i) the various input matrices of this algorithm are constructed; and (ii) how the right-iteration algorithm itself is implemented.

(i) Zone-transitivity matrix and adjacency matrix construction: The zone transitivity capabilities can be provided through the high-level specification. When the high-level policy is refined to an IL policy, these capabilities are collated to construct the $n \times n$ zone-transitivity matrix T (n is the number of zones in the network). The implicit mappings created between the primary conduits and their firewall compositions (Section 6.3) are used to construct the adjacency matrix A.

(ii) A^* ***calculation:*** Algorithm 3 shows our right-iteration algorithm implementation. A centralised implementation is considered (given typically low n), but, it can also be distributed across multiple nodes performing parallel computations.

We can see that Algorithm 3 has time complexity $O(n^4)$ (the operations in lines 8, 10, 12-15 and 20 introduce a fixed computational cost). The space complexity of this algorithm is also $O(n^4)$. These complexity values suggest that the algorithm performance decreases as the number of zones in the network (n) in-

Algorithm 3 Right-iteration algorithm to compute A^*. The additive and multiplicative operators of Semiring S are $\cup, \cdot$ (Section 4.4.3). The additive and multiplicative identities of S are $\hat{0}, \hat{1}$. The multiplicative-identity matrix and the zone-transitivity matrix are I and T. The set of zones in the network is Z.

```
1:  procedure RIGHTITERATION((∪, ·, 0̂, 1̂, A))
2:      R_0 ← I     // a n × n identity matrix
3:      k ← 1
4:      while k ≤ |Z| − 1 do
5:          for each i ∈ Z do
6:              for each j ∈ Z do
7:                  if i=j then
8:                      R_{k+1}(i, j) ← I(i, j)
9:                  else
10:                     R_{k+1}(i, j) ← 0̂
11:                     for each q ∈ Z do
12:                         s ← R_k(i, q) · T(q, j)
13:                         R_{k+1}(i, j) ← R_{k+1}(i, j) ∪ s
14:                         R_{k+1}(i, j) ← R_{k+1}(i, j) ∪ I(i, q)
15:                         R_{k+1}(i, j) ← R_{k+1}(i, j) · A(q, j)
16:                     end for
17:                 end if
18:             end for
19:         end for
20:         k ← k+1
21:     end while
```

creases. Specifying policy per individual host (*i.e.,* allowing a large n) in general, makes policy mapping *very inefficient*, but it does so particularly, in the context of this algorithm. It proves more efficient to create network-groups, for instance, by subnet and specify policy between them; so, a reasonably low value can be maintained for n.

In calculating A^*, we considered all valid paths between each pair of zones (hence time complexity of $O(n^4)$). The decision allows us to permit or block traffic along all possible communication paths between two zones, providing redundancy and *defence-in-depth* in the network. We also map policy uniformly to every firewall along a single-path, to further boost defence-in-depth. These decisions collectively create a robust defence against cyber attacks.

Later in this Chapter, we will employ Algorithm 3, to precisely deploy the high-level policies of our case studies in Chapter 3 to their underlying networks.

The device-configuration verification steps in Figure 6.1, help us check that the generated firewall configurations deliver the expected security outcome pre- and post-deployment. The implementation of these steps involve network emulations and traffic-based tests, which we will describe next.

6.4.4 Automated testing

We use a network emulator – Netkit – for our pre-deployment testing of the generated device configurations. The emulator is open source and enables virtual devices and interconnections using User Mode Linux (UML) [91]. AutoNetkit is a tool designed to automate emulated network experimentation via Netkit [70]. We have extended AutoNetkit, to include firewalls, to generate our emulations.

Our Autonetkit extensions allow users to specify the location of the firewalls in the input network graph of their target network. These extensions currently support Cisco ASA 5505 and UML IP-Tables firewalls to cater for the initial *ForestFirewalls* prototype. Methods have also been added to the Autonetkit API to allow setup of ACL rulesets within these firewalls. The API extensions also facilitate setup of test sets; *i.e.,* test sources and sinks in the network to match the input policy.

Our extensions particularly make it easier to use Autonetkit with *ForestFirewalls*. These extensions can be used stand-alone to generate firewall emulations, but doing so, would require more effort than to have it seamlessly driven by *ForestFirewalls*. For instance, it is necessary to verify that firewall ACL rulesets are consistent and best-practice compliant prior to deploying them to an emulated network. *ForestFirewalls* fulfils this pre-requisite automatically.

When the emulated network is run, tests specific to the input policy create pathological traffic to verify expected firewall configuration behaviour. *ForestFirewalls* uses *Expect* – a UNIX scripting and testing utility – to generate these test-scripts. Expect enables automated interactions with programs that expose a text terminal interface [72]. The scripts run sequentially, with independent outcomes.

The test scripts verify that the permit rules in a policy work correctly (referred to as positive vetting). A permit rule fails if its observed behaviour is different from that anticipated. We can use a *result function* R, to track what rules fail. For a permit rule q_m^a with accept packet set $A = \{s \in \mathcal{A} \mid s \prec m\}$ and a test-packet sequence $s_1 \in A$ the result function is

$$R(q_m^a, s_1) = \begin{cases} 0, & \text{if } s_1 \text{ fails for permit rule } q_m^a \\ 1, & \text{if } s_1 \text{ succeeds for permit rule } q_m^a. \end{cases} \tag{6.4}$$

A failed permit rule means its corresponding test packets are not delivered to the intended destination.

In addition to positive vetting, we need to check that all other services not explicitly enabled are blocked. This negative vetting is conducted using exhaustive port-scans employing *nmap* and *tshark*.

The same test-suite can be deployed to the real network, to generate live-traffic and verify expected real-firewall behaviour. Re-use of the emulation test-suite is possible because our test tools are industry standard and the test sets are already (automatically) matched to the input policy prior to emulation.

6.5 Conclusions

Existing firewall policy languages do not help industrial engineers operating a critical infrastructure plant to reliably manage their firewalls from high-level requirements. We address this shortfall with *ForestFirewalls:* a high-level firewall policy specification that is built on formal foundations, as described in this Chapter. Our implementation of *ForestFirewalls* significantly reduces the firewall configuration effort required and consistently generates correct firewall configurations.

In the next chapter, we apply *ForestFirewalls* to the real case studies in Chapter 3, to demonstrate its ability to significantly reduce the firewall configuration effort required in a SCADA network.

Chapter 7

Application to real case studies

We realised our design of *ForestFirewalls* in a software prototype in the previous chapter. We now apply our implementation to the real SCADA case studies in Chapter 3 to demonstrate that (i) our system reduces the firewall configuration effort required by an order of magnitude; and (ii) that it *always* generates consistent and industry best-practice compliant firewall configurations; and (iii) that our implementation algorithms in *ForestFirewalls* are computationally feasible.

We begin with a concrete example, illustrating our methodology and the prototype system. The example is based on SUC 1 discussed in Chapter 3- a real SCADA case study employing the multi-firewall architecture shown in Figure 7.1. We utilise here, a Cisco ASA 5505 firewall (R1) and two Linux IP-Tables firewalls (R2, GW) instead of IOS routers as per the original case study. IOS images are Cisco proprietary and obtaining them without a commercial contract is difficult.

A summary of the SUC 1 subnets is recaptured below

The Enterprise network (Corp): Allows business application and Internet access.
Demilitarised Zone (DMZ): Enables connectivity between R1 and R2. The distinct vendor firewalls provide *defence in depth* [21] by having different modes of failure and firewall-software redundancy.
The SCADA network (SCADA): Enables networked access to plant equipment.
`Corp` and `SCADA` accommodated 2,046 and 65,534 hosts respectively. `Corp` hosts management workstations, two Web servers, a FTP server, an Email server, a syslog server, a SNMP server, a NTP server and a DNS server. `SCADA` has `Oracle` database servers, management workstations and a Web server.

7.1 Policy goals

The high-level policy we consider is based on the SUC 1 policy described in Chapter 3. We also incorporate several elements that were observed in the other SUC

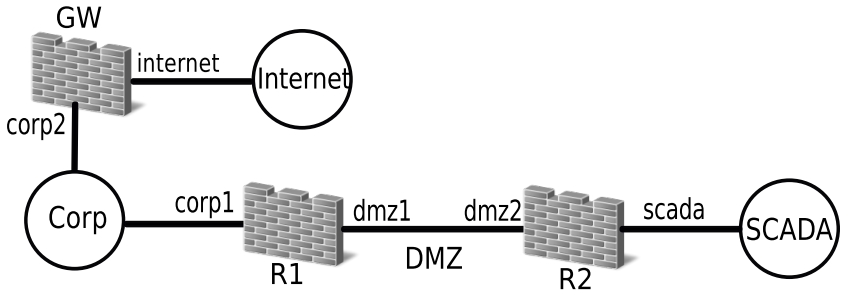


Figure 7.1: The SCADA network under study. `Corp` and `SCADA` are the enterprise and SCADA subnets while the firewalls are R1, R2 and GW. Each firewall has two interfaces, labelled with lower case names.

policies, to show how *ForestFirewalls* handles such specifications. In summary, our policy has the following goals:

- `Corp` hosts can access the `Oracle` servers and the Web server in `SCADA`. These `Corp` hosts can also *Ping* hosts in `SCADA`.
- `SCADA` hosts can access the Web, Email and DNS servers in `Corp`. These `SCADA` hosts can also *Ping* hosts in `Corp` and perform file transfer with the `Corp` servers using FTP. The `SCADA` hosted Web server can be accessed from the Internet.
- R1, R2 can be managed from `Corp` using HTTP, Telnet and SSH. R2 can be managed from `SCADA` using SSH and Telnet. R1 can also be managed from R2 using SSH and Telnet.
- The Syslog server and the SNMP server located in `Corp` store firewall logs and SNMP based firewall messages. Both R1 and R2 synchronise their system times with the NTP server located in `Corp`.
- OSPF is enabled across the entire site.
- Firewall reporting is enabled for policy verification.

7.2 Network implementation

A partial snippet of the *ForestFirewalls* declarative policy implementing the above security goals is depicted in Listing 7.1. This is the policy version prior to our system validating its correctness. We will see later in Section 7.3 that *ForestFirewalls* finds errors in this policy version prompting us to correct them.

At the beginning of the policy, the required library files containing the predefined lists of IANA well known services are imported. Next, the Zone-Conduit security model is supplied as a GraphML file (details of this model are shown

in Figure 7.2). The zones in this model can be grouped as necessary within the specification (line 9-14), to simplify policy authoring and increase readability. We also define custom port-groups, services and service-groups as needed.

A passive mode FTP data service is enabled through the firewalls (lines 17–18) as it's the best-practice approach [21, 117]. Ping (ICMP) is also defined to allow connectivity tests. High-level rules are defined to match the policy goals listed earlier (lines 27-32 depict these partially). We can see that these rules are easy to map to the human written security policy in Section 7.1, *i.e.,* the policy rules are transparent.

A `rule-group` object is used to hold all the security-policy rules (lines 33-34). We also define a `reporting_rule` to specify our reporting policy for verification (lines 37-43) (reporting policy will be discussed in detail in Chapter 8). Finally, both security policy and reporting policy are encapsulated within a global `policy` object (line 46).

7.3 Procedure and Results

Upon checking the IL policy for best-practice compliance, *ForestFirewalls* compiler located several significant violations. For one, direct communication was enabled between `SCADA` and the `Internet`. For another, potentially unsafe HTTP, DNS and FTP traffic was enabled inbound to `SCADA`. The compiler also warned of the direct traffic transition enabled between `SCADA` and `Corp`.

ForestFirewalls also flagged the use of insecure Telnet and HTTP protocols for firewall management. Moreover, our system warned that the SCADA firewalls: `R1` and `R2`, were both managed from `Corp`; *i.e.,* a non-SCADA domain.

We next proceeded to rectify these shortfalls flagged in the high-level policy. For instance, we removed line 32 in Listing 7.1, to prohibit direct communication between `SCADA` and the Internet. Similar actions were taken until all issues pertaining to non-compliance were resolved.

Once the policy is best-practice compliant, *ForestFirewalls* generates an Alloy export corresponding to the IL policy for verification (see Listing 6.1 earlier). Our checking of the 'no_rule_overlaps' assertion (not shown) returns a counter-example, indicating potential redundancies in the specification. Upon inspecting the counter-example in detail through Alloy (Figure 7.3), we see that rules enabling HTTP services (`ip_protocol=6`, `dest_port=80`) between `SCADA` and `Corp`, trigger the redundancy. The root cause is the `web` and `file_transfer` service-groups in the policy (lines 21 and 24 in Listing 7.1), both contain HTTP. We rectified this (by removing HTTP from `file_transfer`) for Alloy to return no further counter examples.

Listing 7.1: ForestFirewalls policy description (partially shown).

```
1  // library files
2  import system.services.iana_services;
3  import system.services.iana_icmp;
4
5  // zone-conduit security topology
6  load_zone_conduit_model ''zone_conduit.graphml''
7
8  // define zone groups
9  zone_group all_zones {z1,z2,z3,az1,fwz1,fwz2,fwz3}
10 zone_group scada_zone {z3}
11 zone_group corp_zone {z1}
12 zone_group internet_zone {z2}
13 zone_group all_firewall_zones {fwz1, fwz2, fwz3}
14
15 // passive mode FTP using custom port numbers
16 port_group ftp_data_ports {24500-24600}
17 service ftp_data {protocol=tcp; tcp.dest_port=ftp_data_ports;}
18
19 // service groups using standard port numbers
20 service_group ftp {iana_services.ftp_control, ftp_data}
21 service_group web {iana_services.http, iana_services.https}
22 service_group ping {iana_icmp.icmp_echo}
23 service_group dns {iana_services.dns_tcp, iana_services.dns_udp}
24 service_group file_transfer {iana_services.http, ftp}
25
26 // define security policy
27 policy_rule file_transfer_rule {
28               scada_zone -> corp_zone : file_transfer}
29 policy_rule ping_rule {corp_zone <-> scada_zone : ping}
30 policy_rule dns_rule {scada_zone -> corp_zone : dns}
31 policy_rule web_rule1 {scada_zone -> corp_zone : web}
32 policy_rule web_rule2 {internet_zone -> scada_zone : web}
33 rule_group security_policy {file_transfer_rule, ping_rule,
34               dns_rule, web_rule1, web_rule2}
35
36 // enable policy verification reporting in firewalls
37 reporting_rule verify_rules {
38  use_case=verification;
39  granularity.network={zone_or_group={all_zones}};
40  granularity.policy={rule_or_group={security_policy}};
41  granularity.traffic={measurement={counter}; type={connection};};
42  granularity.temporal={per_hour};
43  granularity.performance={process};}
44
45 // define global policy
46 policy company_policy {security_policy; verify_rules}
```

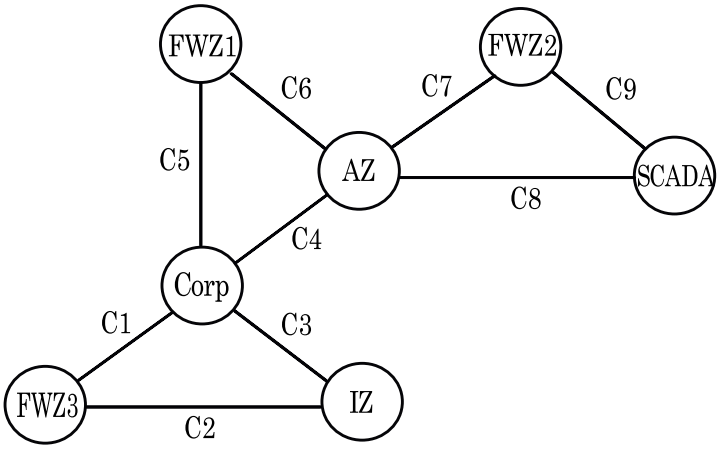


Figure 7.2: The Zone-Conduit model described by the *zone_conduit.graphml* file associated with the high-level policy in Listing 7.1. This is the network-security model perceived by the policy author.

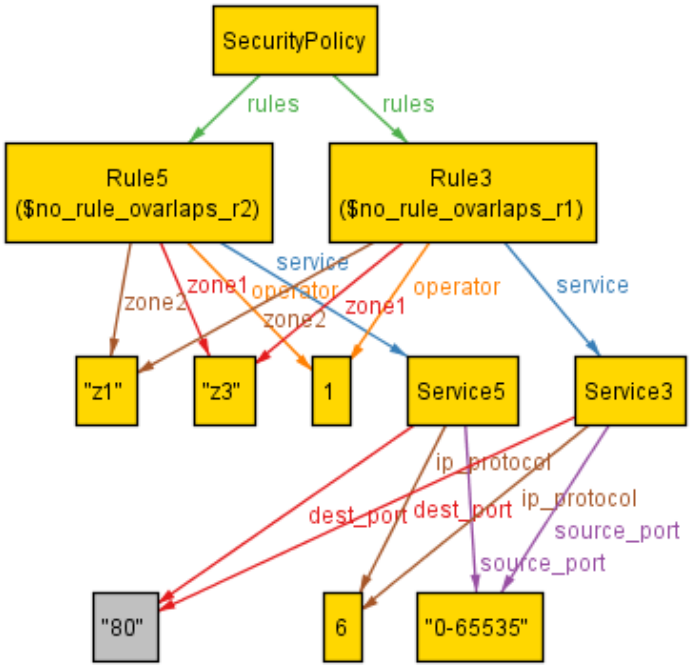


Figure 7.3: Counter-example thrown by Alloy, as a result of checking the IL security policy for redundancies via a 'no_rule_overlaps' assertion. The error indicates a flaw in the high-level policy. In particular, there are two rules that enable HTTP (IP protocol=6, destination port=80) from scada_zone to corp_zone.

ForestFirewalls then couples the IL policy to the input network topology (provided as a *.graphml* file). Listing 7.2 shows the resulting ACL-allocation map for `R1`, indicating how ACLs are assigned to the firewall's interfaces. Listing 7.3 partially shows the generated vendor-neutral ACL rules. Note the explicit *deny all* rule supplementing the explicit permit rules at the end. The corresponding network-level Alloy exports generated consisted of 828 `Service` and `ACLRule` objects, in total. Checking the 'no_rule_overlaps' assertion on these exports yielded no counter-examples, indicating that inconsistencies have not been introduced by our system during the refining of the high-level policy to the network-level.

The generated device-level configurations were first auto-deployed to a Netkit-based emulated network. Once the Netkit Virtual Machine (VM)s booted up, *ForestFirewalls*-generated test scripts were run. Traffic tests confirmed that the firewalls correctly admitted the services enabled in the high-level policy. Moreover, exhaustive port-scans using *nmap* and *tshark* showed that no additional services were permitted through the firewalls.

Post emulation testing, the device configurations were deployed to a real-network. This network consisted of a Cisco ASA 5505 firewall (running ASA version 8.4) and two Dell laptop computers running Linux IP-Tables firewalls (one laptop was an Intel Core CPU 2.7-GHz computer with 8GB of RAM and the other an Intel Core CPU 2.2-GHz computer with 6GB of RAM, both were running Ubuntu Linux). The firewalls were networked as per Figure 7.1.

We currently deploy these automatically generated configurations to the network-devices by hand. The intent is that in a real-world system, deployment would be automated. However, the building of an automated configuration deployment system is beyond the scope of this thesis because several existing mechanisms (*e.g.,* Yet Another Markup Language (YAML) enabled SaltStack network configuration-management software [105], SNMP and NETCONF based network-configuration management systems [112]) achieve this goal. Hence, it is more of a development challenge rather than novel research.

Once deployed, we re-executed the emulation test scripts on hosts in the various zones of the network. These real-traffic tests confirmed that the services enabled by the input policy were passing across the network firewalls as expected, and port scans confirmed no additional services were allowed through. This multi-level testing provides administrators the assurance that their security policies deliver the expected outcome pre- and post-deployment.

Listing 7.2: System generated ACL-allocation map (partially shown, t1 is a router in Corp).

```
1 INFO Firewall-ACL map for firewall: R1
2   Interface: dmz1(R1 to R2) direction: in ACL: acl_4
3   Interface: dmz1(R1 to R2) direction: out ACL: acl_3
4   Interface: corp1(R1 to t1) direction: in ACL: acl_1
5   Interface: corp1(R1 to t1) direction: out ACL: acl_2
```

Listing 7.3: System generated network-level policy (partially shown, comments are denoted by remark).

```
1 INFO Vendor neutral network-level ruleset for ACL: acl_2
2   remark~enable corp_zone to scada_zone HTTPS traffic (return path)
3   permit~tcp~from~10.0.0.16/29~to~10.0.0.0/29~sport~[443]~dport~['0-65535']~state~ESTABLISHED~log
4   permit~tcp~from~10.0.0.16/29~to~10.0.128.4/30~sport~[443]~dport~['0-65535']~state~ESTABLISHED~log
5   remark~enable scada_zone to corp_zone WEB traffic (forward path)
6   permit~tcp~from~10.0.0.16/29~to~10.0.0.0/29~sport~['0-65535']~dport~[443]~state~NEW,ESTABLISHED~log
7   permit~tcp~from~10.0.0.16/29~to~10.0.128.4/30~sport~['0-65535']~dport~[80]~state~NEW,ESTABLISHED~log
8   deny~ip~from~any~to~any~sport~~dport~~state~
```

7.4 Cyber attack mitigation

The following serious cyber-security vulnerabilities were present in the original SUC 1 policy and the six other real SCADA firewall policies studied in Chapter 3

- *Insecure protocols enabled through explicit generic rules:* all-TCP, all-UDP and all-IP traffic flow were explicitly enabled inbound to SCADA, permitting far more services than necessary. Unsafe protocols such as FTP and HTTP were thus allowed into SCADA exposing the zone to cyber attacks.
- *Insecure protocols enabled through implicit rules:* incorrect use of implicit rules such as Cisco security levels [29] enabled *all-IP* traffic to flow between Enterprise and SCADA zones, making the latter more prone to cyber attack.
- *Direct communication enabled between the SCADA-Zone and the Internet:* allowing so, violated industry best-practices and elevated the risk of a cyber attack on the SCADA-Zone.
- *Insecure protocols enabled through explicit and implicit-rule interactions:* in some cases, insecure protocols such as HTTP were explicitly disallowed inbound to the SCADA-Zone, but were then implicitly enabled into the same zone, exposing the latter to cyber attack.

ForestFirewalls removed these vulnerabilities, either by a design language that did not allow the error, or through the rigorous verification built-in to it. For instance, Table 7.1 shows a comparison of the firewall configurations observed in SUC 1 against those generated by *ForestFirewalls*. We can observe that there are *no explicit generic permit rules* generated by our system (*i.e.,* all-TCP, all-UDP or all-IP based rules). Recall that in the original case study, all-IP traffic was enabled through the firewall, just to get dynamic routing working! Removing the ability to specify generic services by design, prohibits unwanted services from being enabled implicitly between zones altogether.

In addition, all traffic flows must be explicitly enabled in *ForestFirewalls*. So, the use of implicit rules is also removed. This step prevents interactions between explicit and implicit rules, so, one cannot override the other to enable services by accident.

We also recall that there were two redundant ACLs and 117 redundant ACL rules in the original case study. These redundancies lead to inefficient and expansive configurations and ultimately latent errors. As Table 7.1 shows, there are *no redundant ACLs* and *no redundant ACL rules* generated by *ForestFirewalls*. Each ACL serves a purpose and is assigned to an active firewall interface. There are also *no intra-ACL rule interactions* in the rule sets generated. Collectively these outcomes make *ForestFirewalls* generated firewall configurations more efficient and concise.

Table 7.1: Comparison of the original configuration of SUC1 against the configuration generated by *ForestFirewalls* (LoC - Lines of Code).

Attribute	Original case study	ForestFirewalls
System-level LoC	N/A	80
Device-level LoC	2720	714
Obsolete-ACL count	2	0
Invalid ACL rules	70	0
Rule inconsistencies (*e.g.,* conflicts)	170	0
Generic permit-rule count	324	0
Secure firewall management	✗	✓
No Internet-SCADA direct traffic transit	✗	✓
No Enterprise-SCADA direct traffic transit	✗	✓
No insecure traffic (*e.g.,* DNS) inbound to SCADA	✗	✓

These are direct consequences of our design approach and might seem obvious, but the real firewalls [96] had all of these defects.

Our system only requires 80 high-level LoC (only 41 LoC are policy specific) to generate 714 device-level LoC to configure all three firewalls in the case study discussed. A high-level policy with only 80 LoC has replaced the 2720 attack and error prone, inefficient, device-level LoC in the original case study. This ability to manage network security functionality using concise, high-level policies characterises *ForestFirewalls* as a PDN based network-security management solution.

We next extend the application of our policy best-practice compliance checks to all the SUCs in Chapter 3. These SUCs involve various firewall architectures and allow us to demonstrate (i) *ForestFirewalls*' automated capability to detect non-compliance; and (ii) that the computational complexities of our policy-comparison algorithms are tractable in practice.

7.5 Best-practice compliance

A high-level policy comparison summary of the SUCs is shown in Table 7.2. The table shows details of the equivalence classes of the semantic partitions, for each network's firewall policy. We described in Chapter 4 how an *equivalence class* groups a set of conduits that have identical policies.

Table 7.2: High-level policy comparison summary of the SCADA case studies discussed in Chapter 3 (* includes gateways, ** includes system-generated zones such as Firewall-Zones and Abstract-Zones).

SUC	Firewall type	Firewalls*	Zones**	Conduits	Flow-policies	Equivalence classes	Maximum class size	Policy
1	Cisco IOS	3	7	11	22	12	7	$\mathcal{P}_1 = (G_1, P_1)$
2	Cisco ASA	6	21	81	162	87	8	$\mathcal{P}_2 = (G_2, P_2)$
3	Cisco PIX	4	10	17	34	15	8	$\mathcal{P}_3 = (G_3, P_3)$
4	Cisco ASA	3	9	16	32	16	5	$\mathcal{P}_4 = (G_4, P_4)$
5	Cisco ASA	3	12	34	68	24	6	$\mathcal{P}_5 = (G_5, P_5)$
6	Cisco ASA	4	14	37	74	28	8	$\mathcal{P}_6 = (G_6, P_6)$
7	Cisco ASA, Cisco FWSM	6	21	53	106	67	7	$\mathcal{P}_7 = (G_7, P_7)$

As the table shows, the maximum size of the equivalence classes (*i.e.,* the number of conduits with identical policies in an SUC), is *small* relative to the flow-policy count. This is to be expected, we argue that if many conduits have identical policies then the zone-conduit diagram of a network is *inefficiently designed*. Many equivalent policies lead to zones with identical reachability, which might be amalgamated. The subsequent reduction in complexity makes policy specification easier and less error prone.

More importantly, these zones need to be human understandable and in real networks these are implemented (at the moment) by hand. This leads to a small number of zones with equivalent policies.

As a result, the maximum equivalence class size across our case studies is eight (Table 7.2). Moreover, most of the class sizes are small (the majority have only *one member*). That makes the computational complexity of the policy-comparison algorithms presented in Subsection 6.4.2 closer to $O(n^2)$ rather than $O(n!)$. This allows us to describe these networks as close to being *semantically disjoint* and the complexity is certainly manageable.

We demonstrate this computational feasibility in detail by comparing the policies of SUC 1 and SUC 4 (*i.e.,* $\mathcal{P}_1$ and $\mathcal{P}_4$ respectively), below. We have performed similar tests on the other pairs. The zone-flow models of the SUCs are shown in Figure 7.4.

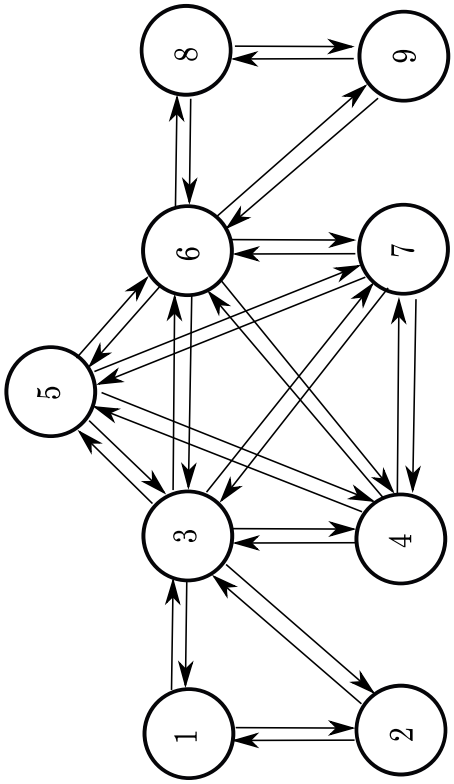
Given $\mathcal{P}_1 = (G_1, P_1)$ where $G_1 = (Z_1, C_1)$ and likewise $\mathcal{P}_2$, we denote the set of conduit policies $P_1 = \{p_{i,j} \mid i, j \in Z_1 \text{ and } ij \in C_1\}$. Similarly, we denote the set of conduit policies $P_2 = \{p'_{m,n} \mid m, n \in Z_2 \text{ and } mn \in C_2\}$.

$\mathcal{P}_1$ and $\mathcal{P}_4$ cannot be *equivalent* or *inclusive* as the node and edge counts don't match, so we immediately proceed to test incorporation, *i.e.,* $\mathcal{P}_1 \equiv \mathcal{P}_4(G_1)$ and $\mathcal{P}_1 \subset \mathcal{P}_4(G_1)$.

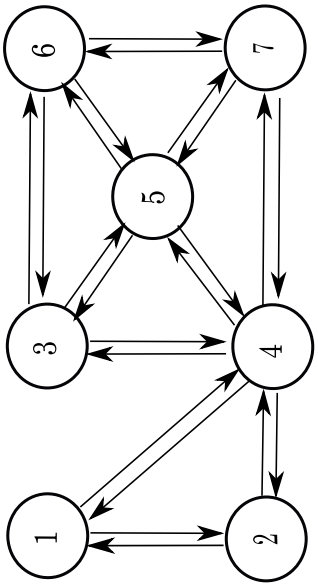
The node and edge counts of $\mathcal{L}(G_1)$ and $\mathcal{L}(G_2)$ satisfy the *incorporation* criteria in Chapter 4. So, we proceed to derive and compare the equivalence classes of SP_1 with those of SP_4 and find that not every SP_1 class has an *equivalent* SP_4 class. So, $\mathcal{P}_4$ *does not strictly incorporate* $\mathcal{P}_1$. But as Table 7.3 illustrates, $SP_1 \subset SP_4$.

The table also shows the equivalence class sizes; *e.g.,* $|e_{12}| = 7$ and $|e'_{17}| = 5$. Since $|e_{12}| > |e'_{17}|$, every policy in e_{12} cannot be *included* in a policy in e'_{17}. So, P_1 policy tuple $(p_{1,2}, p_{2,1}, \ldots, p_{6,7}, p_{7,6})$ with 22 elements cannot be *included* in any P_4 tuple. Thus, $\mathcal{P}_4$ *cannot partially incorporate* $\mathcal{P}_1$.

Our System automates all of these comparisons, and in this case takes 24 seconds to run on a standard desktop computer (*e.g.,* Intel Core CPU 2.7-GHz computer with 8GB of RAM running Mac OS X). We sought to compare our system's performance to previous work [73] that enabled less comprehensive firewall policy comparisons, but have been unable to obtain their software, to date.



(b) Zone-Flow model of SUC4.



(a) Zone-Flow model of SUC1.

Figure 7.4: Zone-Flow models of two real SCADA case studies.

Table 7.3: Mappings between inclusive equivalence-classes of semantic partitions SP_1 and SP_4.

SP_1 class	$\subset SP_4$ class(es)	total size	mappings
$e_1 = \{p_{7,4}\}$	e'_1	1	M_1
$e_2 = \{p_{4,7}\}$	e'_2	1	M_2
$e_3 = \{p_{2,4}\}$	e'_3	1	M_3
$e_4 = \{p_{7,6}, p_{3,6}\}$	e'_4	4	M_4
$e_5 = \{p_{6,3}\}$	$e'_4 - e'_7$,	15	M_5
$e_6 = \{p_{4,3}, p_{4,1}\}$	$e'_7 - e'_9$	7	M_6
$e_7 = \{p_{5,4}, p_{3,4}\}$	e'_1, e'_{10}	2	M_7
$e_8 = \{p_{1,4}\}$	$e'_2, e'_3, e'_5, e'_{11} - e'_{14}$	9	M_8
$e_9 = \{p_{7,5}\}$	e'_1, e'_{15}	2	M_9
$e_{10} = \{p_{6,7}\}$	e'_1, e'_{12}, e'_{14}	3	M_{10}
$e_{11} = \{p_{4,5}, p_{2,1}\}$	e'_1, e'_{16}	2	M_{11}
$e_{12} = \{p_{4,2}, p_{5,3}, p_{3,5}, p_{5,6}, p_{6,5}, p_{5,7}, p_{1,2}\}$	e'_{17}	5	M_{12}

Moreover, we checked each policy in Table 7.2 for compliance with the best-practice policy ($\mathcal{RP}$) [59] through *inclusion* and *incorporation*. Our system generates the semantic-partitions SP_i and SP_{RP} and checks whether $SP_i \subset SP_{RP}$. Alarmingly *none* of the policies of any SUC complied with best practices, confirming our conclusion in Chapter 3 that the SUCs were significantly vulnerable to cyber attack.

We also used our policy-comparison algorithms to compute the semantic difference between each policy and $\mathcal{RP}$. Listing 7.4 shows a snippet of SUC 1's policy deviation warnings. *ForestFirewalls* provides these warnings to allow a user to rectify the problem prior to deployment.

The system can also compute the functional discrepancies between multiple policy designs. By doing so, users from multiple policy subdomains can identify design differences and negotiate what variations are acceptable.

However, *ForestFirewalls* also has several limitations, for one, checking a policy against a best-practice policy relies on the best-practices being correct. Also, when a policy is a subset of a best-practice policy that may not always mean the policy is valid. For instance, ISA best-practices recommend enabling TCP port 443 (*i.e.,* HTTPS) inbound to the SCADA-Zone because it's deemed safe. But compliance doesn't guarantee safety. A particular HTTPS implementation could still have flaws.

Listing 7.4: ForestFirewalls output warnings of the recommended practice violations found in SUC1 policy (partially shown).

```
WARNING suc1_policy.policyml violates ISA practices
WARNING policy violations::
 cz->sz: { protocol=17; udp.dest_port=135; udp.source_port=1024-65535;}
 cz->sz: { protocol=17; udp.dest_port=53; udp.source_port=1024-65535;}
 cz->sz: { protocol=6; tcp.dest_port=53; tcp.source_port=1024-65535;}
 cz->sz: { protocol=6; tcp.dest_port=21; tcp.source_port=1024-65535;}
 cz->sz: { protocol=6; tcp.dest_port=20; tcp.source_port=1024-65535;}
 cz->sz: { protocol=6; tcp.dest_port=80; tcp.source_port=1024-65535;}
 sz->az1: { protocol=17; udp.dest_port=53; udp.source_port=1024-65535;}
 sz->az1: { protocol=6; tcp.dest_port=53; tcp.source_port=1024-65535;}
 sz->cz: { protocol=17; udp.dest_port=53; udp.source_port=1024-65535;}
 sz->cz: { protocol=6; tcp.dest_port=53; tcp.source_port=1024-65535;}
```

We now employ *ForestFirewalls* to map policies to the underlying network firewalls of all SUCs in Chapter 3. By doing so, we demonstrate (i) the accurate policy deployment capabilities of our system; and (ii) that the computational complexity of our policy-to-firewall mapping algorithm is tractable.

7.6 Policy mapping examples

A high-level summary of how policy was incorrectly mapped to the firewalls in the original SUCs is shown in Table 7.4. An important feature depicted in Table 7.4 is the number of security zones in each network. This number is small (*i.e.,* ≤ 21).

This is to be expected; a zone groups a set of hosts or subnets with identical policies. If every host had a distinct policy then a large number of firewalls are needed to enforce a real separation between the hosts, making it impractical. By grouping hosts into zones, we reduce policy complexity, so their specification becomes easier and less error-prone. The small number of zones contributes to the computational feasibility of our approach.

Each of our SUCs in Chapter 3 contained ACL rules that were incorrectly assigned to the underlying network firewalls. As Table 7.4 suggests, on average there were 10 ACL rules allocated to the wrong firewall, 8 rules allocated to the wrong firewall-interface and 15 rules allocated in the wrong interface-direction, per case study.

There were *zero* incorrectly allocated policy rules, when the policy was mapped to the firewalls using *ForestFirewalls*. Through precise policy deployment, we reduce the vulnerability of these SCADA networks to cyber attack and prevent potentially-catastrophic outcomes.

Table 7.4: High-level policy-mapping summary of Chapter 3 case studies (* includes gateways, ** includes system-generated zones such as Firewall-Zones and Abstract-Zones, # ACL rule allocation error).

SUC	Firewalls*	Zones**	Conduits	Max. hosts	ACLs	Average rules per ACL	Incorrect firewall#	Incorrect interface#	Incorrect direction#	Runtime (s)
1	3	7	11	67580	8	237	15	13	19	40
2	6	21	81	2794	12	16	3	2	5	70
3	4	10	17	886	8	6	2	1	4	43
4	3	9	16	2038	3	80	5	12	13	61
5	3	12	34	2664	12	677	15	8	26	47
6	4	14	37	3562	8	1034	21	15	19	63
7	6	21	53	3810	17	724	9	5	17	49

7.7 Conclusions

In this chapter, we apply *ForestFirewalls* to the real case studies in Chapter 3, to demonstrate its use in the high-level configuration of firewalls in a SCADA network. By doing so, we show the significant reduction in the firewall-configuration effort required and the increase in configuration accuracy achieved by our system by only generating consistent, best-practice compliant firewall configurations. These configurations also deliver the expected security outcome pre- and post-deployment. Such outcomes collectively create SCADA networks that are secure-by-design.

In addition to security policies, *ForestFirewalls* also supports reporting policies to monitor firewall security, post deployment. In the next chapter, we will describe the architecture of our firewall-reporting framework that enables reporting policy specification in a modular and systematic manner.

Chapter 8

Firewall Reporting

We demonstrated the use of our high-level firewall policy specification: *Forest-Firewalls*, in the previous chapter. In addition to security policies, the specification also allows to define the level of monitoring and reporting of firewall security, post-deployment. We describe here in detail the development of a framework for firewall reporting that helps understand (i) what reporting functions should be conducted by a firewall; and (ii) the level of detail required in these reports.

Our real SCADA case studies in Chapter 3 and related works [128, 129] have shown that firewall misconfigurations are a common occurrence in production networks. Other parts of the network-security setup can also malfunction (*e.g.,* due to network upgrades and patches). Either outcome can break the network or reduce its security.

There are many steps to guard against such failure, but one of the most important is to constantly examine the security mechanisms of the network. This can be performed externally, and by using the logs, traps, alerts and other information provided for instance via SNMP (Simple Network Management Protocol) polling of the firewalls involved. We term all these informational contents together as *reporting*.

Firewall reporting is also important to get right. Firewall vendors have introduced many products and security management tools with varying levels of sophistication [24, 30, 32, 64] to this end. However, what these tools have typically done is to increase the range of options. They provide more power, but there is a lack of guidelines on how to best utilise this power.

Even in domains such as Supervisory Control and Data Acquisition (SCADA) networks, which control the distributed assets of many critical systems, the standards for reporting and analysing firewall data are scant and vague. In particular, we lack clear direction in what firewalls *should* report. Given the primary function of a firewall is traffic filtering, we need to carefully scrutinise what secondary reporting functions should be conducted by a firewall. Only then can we begin to specify reporting policies for firewalls in a pragmatic and useful manner.

Existing SCADA best-practices provide direction on several aspects of firewall deployment; from suitable firewall architectures to service-specific firewall policies. These guidelines also nascently propose report-retention best practices.

For instance, North Atlantic Electric Reliability Corporation (NERC) provides a framework for the identification and protection of critical cyber assets. The framework consists of a group of standards collectively known as the Critical Infrastructure Protection (CIP) standard [86], which accommodates Incident Reporting and Response planning (IRRP). An incident response plan ensures the identification, location, response and reporting of cyber-security incidents related to critical cyber assets [84]. The IRRP standard requires cyber-security incidents to be classified as *reportable* or otherwise, but does not provide a clear definition of what is reportable.

The CIP standards also address Systems Security Management (SSM). The SSM guidelines require the definition of methods and procedures for securing critical cyber assets (*e.g.,* firewalls) [85] by critical infrastructure operators. In particular, the standard mandates the monitoring of security status of cyber assets by responsible entities. Both IRRP and SSM provide report-retention durations, but the rationale for these lengths are unclear.

Given these scant and vague industry standard surrounding firewall reporting, we sensibly begin our discussion on how reporting policy should be specified by

1. Looking at the ways firewall reports can be used, and how that impacts reporting requirements.

2. From this, formalising notions of the *scope* and *granularity* of reports.

3. Considering which use cases are actually reasonable uses of firewall resources.

We find that reporting at the right granularity is key to saving valuable firewall and network resources while achieving the required use case outcomes. In some cases a low (*i.e.,* coarse) granularity such as the reporting of configuration changes at a firewall level is sufficient. In other cases a higher (*i.e.,* refined) reporting granularity such as recording individual IP packets is required. The volume of firewall reports generated usually increases with granularity and can have a detrimental effect on a SCADA firewall's primary function- traffic filtering. Identifying reporting granularity requirements can help justify whether a potential use case is best served if it is conducted by a SCADA firewall or not.

We also find that reporting needs to be coupled with security policy specification to be of use. This important principle has been overlooked in the best practices [21, 117]. Coupling reporting and policy also provides security managers with a *single source of truth* for easy reference.

8.1 Firewall reporting use cases

Firewalls can generate logs, alerts, traps, and provide other information for instance via SNMP (the Simple Network Management Protocol). We term all these informational contents together as *reporting*.

Most firewalls allow a panoply of highly flexible and configurable reports. It is hard to even categorise all of the possibilities without some framework. Here we use the classical hierarchy of knowledge:

$$data \rightarrow information \rightarrow knowledge \rightarrow decisions.$$

If the data does not inform decisions its value is zero, so simply collecting data is insufficient motivation for the cost of collection. Hence we base our framework on *use cases*. That is, we frame our discussion of what to report based on how the data will be used.

We reviewed the literature on firewall reporting in both SCADA and Corporate domains [67, 107, 108] and classified the various uses of reports. Out of necessity, we grouped certain activities together, and simplified the nomenclature. Our resulting classification is as follows:[1]

Accounting measures network usage, for instance, to monitor network bandwidth usage for network planning.

Network-based Intrusion Detection (ID) is the near real-time monitoring of network traffic to identify traffic from unauthorised sources [108]. Near real-time implies without a significant delay (transmission and automated processing delays are allowed), usually up to a few minutes [108]. Network-based ID particularly helps administrators to secure a SCADA network by blocking an attack before it causes too much damage to critical systems.

For example, some firewalls can monitor traffic that passes key network locations, and generate alarms when known attack signatures are present. Other firewalls might send traffic data to an ID system for analysis.

Post-mortem analysis is the analysis of an *incident* after the fact. Analysis aims to identify root causes in order to prevent future occurrences. In a cyber-physical domain, analysis may be mandatory to meet regulatory compliance.

Security policy verification checks a firewall rule base for invalid rules. Our real SCADA case studies in Chapter 3 and related works [128, 129] show that configuration inefficiencies in firewalls are largely due to obsolete rules (*e.g.,* rules pertaining to a decommissioned server) and incorrect rules (*e.g.,* rules with source and destination IP address in wrong order). Efficient firewall configurations can be maintained by identifying such invalid rules through firewall reports.

[1] At this point we classify existing activities without stating which of these cases is a sensible use of firewall resources.

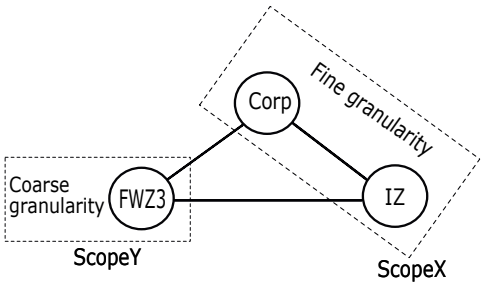


Figure 8.1: Illustration of scope vs granularity.

Troubleshooting operational issues is the debugging of network errors associated with firewalls, for instance, to identify the root cause(s) of connection problems through a firewall.

Given the primary function of a firewall is traffic filtering, we need to carefully consider what secondary reporting functions should be conducted by a firewall. To do so, we need to identify the reporting granularities associated with each use case above. We discuss these requirements in detail next.

8.2 Report Granularity and Scope

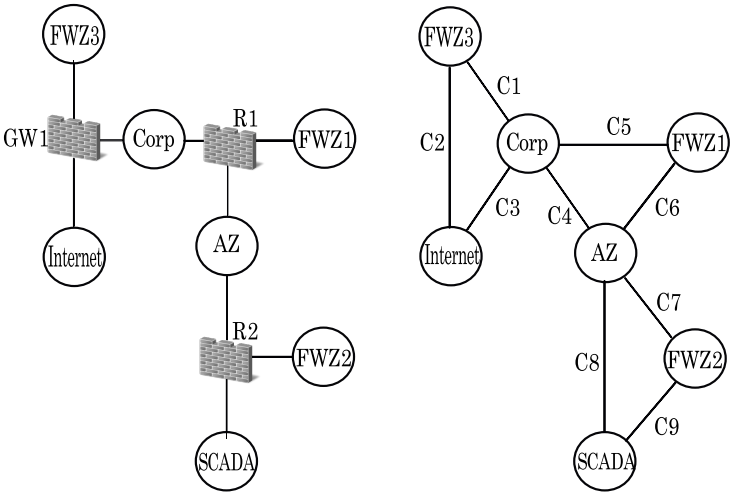
Firewall reports can be thought of as a form of *lossy data compression*. A firewall will internally register myriad events and data. A report, whether it be an alarm or log-message or polled counter, is a compressed form of this information.

Ideally, we would lose as little information as possible in compressing the data. Even the most effective compression mechanisms (*e.g.,* JPEG images) allow some loss, but try to ensure that it is not important information. In the context of firewalls, we discuss here what is important by considering the *resolution* or *granularity* of reports[2]. Granularity refers to the finest level of the discrimination we can make. For instance, in an image, resolution or granularity is the pixel size – we can't separate objects in an image that are smaller. An initial concern when compressing an image would be how many pixels do we need, and likewise this is a first step in considering firewall reports.

Related to granularity is *scope*. In image terms, this is analogous to field-of-view, *i.e.,* how widely is the data collected. We can refer to both scope and granularity with similar terminology, though the details can vary across the network as shown in Figure 8.1.

We refer to SUC 1 in our case studies discussed in Chapter 3, to refine our discussion here (the corresponding security models are depicted in Figure 8.2).

[2]Although we view the two terms as close to synonymous in this context, resolution is overloaded with meaning and so we prefer the term granularity.



(a) Zone-Firewall model. (b) Zone-Conduit model.

Figure 8.2: Security models of SUC 1 from the case studies in Chapter 3.

Most vendor firewalls support different reporting *verbosity* levels, *e.g.,* debug, info, warning [24, 29, 64, 93]. However, the terminology (for instance *warning* level) is not universal. We seek *vendor-independent* notions of reporting, and so will avoid these terms.

These terms also obscure the multiple dimensions of firewall reporting: for instance, the level could refer to nature of events with respect to the network, policy definitions, or time. Hence, in the next section, we will tease these aspects of reporting detail by considering granularity and scope with respect to the multiple aspects of a network that firewalls observe.

8.2.1 Granularity dimensions

Network granularity

This is the level of detail resolved in a network. Firewall reports may need to resolve network-specific detail, often at a network-wide, Zone-Conduit, interface, prefix or IP address level. We describe these levels of detail in order of increasing granularity next.

(i) ***Network-wide-level*** refers to an entire network. At this level we only distinguish between events internal, and external to the network. This is important for intrusion detection or post-mortem analysis, and for general accounting of network usage.

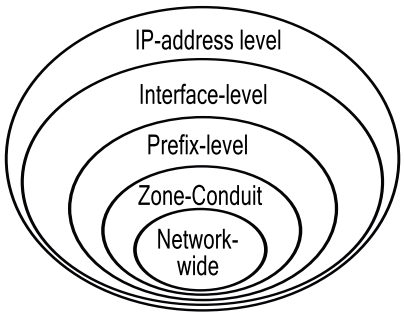


Figure 8.3: Network granularity hierarchy: coarse granularities are subsets of the finer.

(ii) ***Zone-Conduit-level*** resolves information per zone. For instance, in intrusion detection, we may want to understand which components of the network (*e.g.,* SCADA or Corporate) were potentially compromised, and the zone concept requires that all network elements in a zone are treated equally (*i.e.,* even if only one IP address is *observed* to be attacked, we assume all might have been, at least through proxy attacks).

(iii) ***Prefix-level*** resolves information per block of IP addresses. Where zones are not used, we can naturally group data for analysis by subnet.

(iv) ***Interface-level*** resolves information per firewall interface. Large firewall installations can have multiple interfaces serving a zone, and/or there is the question of where traffic triggers an event: when it enters a firewall, or leaves it. This level of resolution is needed when such fine discrimination is important, for instance, in troubleshooting firewall problems.

(v) ***IP-address-level*** resolves information per network host (*e.g.,* a server). This is the finest network-granularity usually possible, though sometimes virtual machines could be hosted by separate ports at a single IP address.

We seek to define granularity in terms of a hierarchy in which the information present in a coarser granularity can be derived from the finer. This hierarchy is shown in Figure 8.3, and note that it should recur in other granularity dimensions.

Policy granularity

This is the level of detail resolved in a security policy. Firewall reports may need to resolve policy detail, often at a global, zone, service, rule or sub-rule level. We describe these below.

(i) ***Global-level*** resolves rule-set wide information for a policy. *i.e.,* it separates events or traffic into *in-policy* and *out-of-policy* traffic. Out-of-policy traffic includes 'defective' traffic (*e.g.,* packets with ill-formed headers). Such data can help track the amount of attack traffic aimed at a network.

(ii) ***Zone-policy-level*** refines the global-level by resolving traffic that is in- and out-of-policy at the zone level. Doing so, allows to understand for instance, the amount of attack traffic breaching the outer level of a defence-in-depth.

(iii) ***Service-policy-level*** of detail resolves information per service either allowed or explicitly denied in a firewall's security policy.

(iv) ***Rule-level*** refines the Service-policy-level by noting that a "service" may be defined by several rules. An example is a service such as WWW, which could include HTTP and HTTPS. Another is a service that requires requests to be allowed in one direction, and responses in the other (such as Connection-like ICMP and UDP services discussed in Section 6.2.1)

Traffic measurement granularity

One of the common reports firewalls produce concerns the traffic they observe, either generally, or because it matches some particular pattern (*e.g.,* it is invalid, or matches a particular type of attack) or translation (*e.g.,* IP header changes by NAT, or IP payload encryption by IPSec in a VPN tunnel). We describe here the granularity with which traffic can be reported.

(i) ***Counter-level*** resolves traffic measurement details into certain bins, and then counts the amount of traffic into these bins. The granularity of the bins is defined by the previous two dimensions of network- and policy-level.

For instance, in policy verification, the key interest is in whether a policy rule has hits (*i.e.,* if IP packet matches have been recorded for the corresponding ACL rules).

The nature of the information recorded in each bin can vary: commonly we measure packets, bytes, or connections (or all three). However, the meaning within the packets are lost.

(ii) ***Connection-level*** records traffic per connection. It is very similar to flow-level, which we discuss in detail below.

(iii) ***Flow-level*** resolves traffic measurements per IP flow: *i.e.,* by grouping a series of connected packets, typically those with the same IP protocol, IP source and destination addresses, and TCP source and destination ports.

For instance, in accounting for network usage, it may be necessary to resolve the source IP from the destination IP address of a flow. This can resolve the upload and download traffic of a host or subnet. Such flows are almost analogous to connections, but are easier to collect.

(iv) ***Packet-header-level*** records each packets' headers. It includes information such as the *flags* and *fragment-offset* fields in the IP header, which indicate whether IP fragments can produce a complete datagram.

(v) ***Packet-level*** requires us to store whole packets, or at least a substantial part of each packet. This allows us to reconstruct, for instance, the details of a particular attack.

Our traffic granularity discussion above illustrates an important issue. Traffic measurements are collected using many different *mechanisms*, and the mechanism is often related to the type of measurement. For instance, SNMP is often used to collect counter-level data, and NetFlow to collect flow-level data.

However, this is just how it is done now. Our aim is to define universal concepts, and leave their implementation mechanisms to the engineers building devices. The goal of device- and vendor-independence requires this approach of decoupling what we want to measure, from how it is measured.

Performance measurement granularity

Another common type of report firewalls may produce concerns performance metrics. Of the many measurements attributable to a firewall or its interface performance, the memory and CPU utilisation are most significant. Both metrics can help identify ongoing attacks (*e.g.,* DoS) or other problems. Similarly, packet queue-length and packet loss measurements can indicate attacks through overloaded queues resulting from high traffic, *e.g.,* that resulting from a DoS attack.

We describe here the granularity with which performance can be reported.

(i) ***Firewall-level*** records performance measurements per individual firewall. For instance, post-mortem analysis might use firewall CPU and memory utilisation to help resolve the root cause of a network problem.

(ii) ***Process-level*** records performance measurements per firewall software process. For instance, when a new policy is pushed to a firewall, the policy processing module may fail to load the policy, if the memory required for the policy exceeds the module's allowance [31]. Module's CPU and memory utilisation reports can help troubleshoot firewall problems in this scenario.

(iii) ***Interface-level*** records performance metrics specific to a firewall interface. For instance, in troubleshooting firewall errors, it may be necessary to resolve reverse DNS lookup errors per firewall interface.

Temporal granularity

This is the level of detail (*e.g.,* measurements or counts) resolved per set of time instances. Granularity is not exactly the right concept here, but we will explain below.

(i) ***Per-T*** records detail per time interval T. The measurements can be counter measurements of traffic, configurations changes and so on. Common intervals T vary from

- daily;
- hourly;
- minutes;

and potentially finer intervals. However, the mechanisms used to support granularities down to minutes are often not suitable (*e.g.,* SNMP polling) for measurements at per-second granularity and finer, so we qualitatively separate very fine intervals into the following category.

(ii) ***Near-real-time*** means reporting data as soon as possible considering only the limitations of processing and network speed. Delays of up to seconds are reasonable, but not minutes. While the previous granularity can be reported through *pull* mechanisms, nearly all near-real-time support is provided by *push* mechanisms, *e.g.,* traps or notifications or alarms.

Note that it is non-trivial to set up accurate distributed clocks, but doing so is vital for any level of temporal granularity for the data collected to be meaningful.

Operational measurements

Another common type of report firewalls can produce concerns the events they observe. Granularity does not seem to be a useful concept in this domain because there is no reason we would ever store data at a coarser granularity than its origin. The standard means to describe the level of detail in event logs are vague terms such as *debug*, *error*, *warning*, *informational*. They specify the "level" of events to be reported, not the level of detail of the actual reports. Also, this notion of level is too context dependent to be universally agreed; *e.g.,* a warning in one domain is an error in another.

Errors can also be ambiguous to interpret. Some report abnormal behaviour that require no response action (*e.g.,* traffic with a broadcast destination address dropped). Others relate to a significant breakdown in operation that needs urgent attention. We increase clarity by classifying the latter type of errors as *failures*.

We studied corporate firewall logs and identified some common events that occur in real firewall deployments. Logs from five such firewalls were analysed. These aggressively reported on traffic denials at the external firewall interfaces.

Of over 4.5 million log messages in a month, 97.58% were traffic denial events. These were both in- and out-of-policy denials (in-policy denials were caused by explicit 'deny' rules in the policy). In addition. 1.18% log messages were power-state changes; 0.78% were VPN-state changes; 0.42% were *failures*; and 0.03% related to firewall-user activity.

Motivated by our findings, and avoiding bland ambiguous terms in favour of precision, we classify events by their nature [3]:

(i) ***State transitions:*** reports of state changes of the firewall and its components (*e.g.,* interface power-up, software process startup- such as for a HTTP or VPN server).

(ii) ***User activity:*** reports of user login activity, commands executed by a user logged into firewall including actions to change its configuration, and their consequences, including any errors or warnings.

(iii) ***Failures:*** reports of breakdown of normal operation (*e.g.,* VPN tunnel failure), that require immediate action. These are not regular state changes and hence are excluded from *state transitions*.

(iv) ***Diagnostics:*** reports of self-tests deployed on the firewall and their outcomes (*e.g.,* test failover interface).

(v) ***Table dumps:*** tables of comprehensible information (*e.g.,* active NAT translations) or potential events, internal to a firewall.

We can depict the granularity requirements of the reporting use-cases via a summary plot such as Figure 8.4. This example plot shows that a *Zone-Conduit level* network-granularity is required for all reporting use cases.

[3]Note that some types of events are already implicitly included in traffic or performance measurements, for instance, denied packet counts.

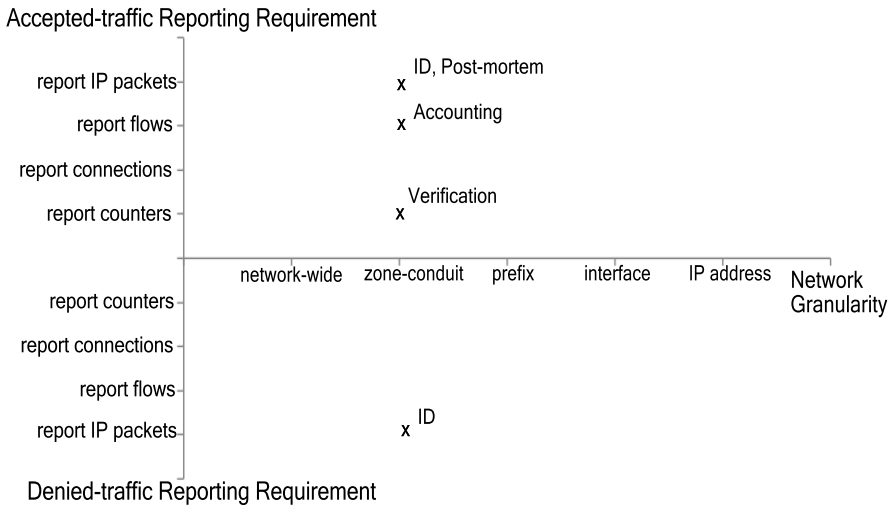


Figure 8.4: Traffic reporting requirement vs Network granularity: depicts the *zone-conduit level* network-granularity required for each reporting use case.

8.2.2 Relation between Reporting Granularity Dimensions

Reporting granularity dimensions may not be entirely orthogonal. We describe the relation that exist between some dimensions below.

Traffic measurement and Time: Packets, flows and connections can arrive at variable rates in time to a firewall. So, the traffic granularity required for each use case can restrict the temporal granularity achievable (and vice versa). For instance, if a use case requires reporting every IP packet, that typically implies reporting at a per millisecond temporal granularity. Conversely, if a firewall reports per minute, a packet level of granularity cannot be achieved.

Traffic measurement and Policy: Traffic granularity requirements can also restrict the policy granularity achievable (and vice versa). For instance, reporting every IP packet implies a rule-level policy granularity. Conversely, if reporting at a rule-level, a packet or flow or connection-level traffic granularity is achievable.

Traffic management and Network: Traffic granularity requirements can also restrict the network granularity achievable (and vice versa). For instance, reporting every connection implies an address-level network granularity. Conversely, if reporting at an address-level, a connection or flow or packet-level traffic granularity is achievable.

Each reporting use case requires a *Zone-Conduit level* network granularity. But the relation between traffic and network granularity implies that the actual network granularity achieved depends on the required traffic granularity of a use case. For instance, intrusion detection requires reporting every IP packet and yields a finer *address-level* network-granularity.

8.3 Reporting Cost

Report generation costs CPU, memory and network resources. The cost can potentially impact the performance of a firewall, and compromise its primary function: traffic filtering, so it is an important factor to consider.

The cost depends on several factors: the granularity and scope; whether the reports are distributed or centralised; and the retention period. We have already discussed granularity and scope in detail: finer granularity measurements cost more. We discuss the other two issues below.

8.3.1 Distributed vs Centralised Analysis and Retention

Firewalls can be configured to report to several destinations including internal storage, NAS (Network-Attached Storage), a report server or historian, a Telnet or SSH session, or an email account [24, 29, 64].

Distributed collection and retention: *i.e.,* collating and storing reports at the firewall, has a low network communications cost, but the trade-off is that it requires local storage. Firewalls rarely have large internal storage, so it must be frequently overwritten or cleared [29].

Moreover, the data lacks utility in this form. For example, accurate ID and troubleshooting require correlating reports from multiple sources (firewalls, routers, servers, *etc.*) and to manually extract these from each device is cumbersome.

At the other end of the spectrum lies centralised collection. In this strategy, network devices perform minimal analysis of data: they just collect it and pass it to a single repository. Reports can then be analysed together. However, centralised collection creates potentially large volumes of network traffic.

Centralised collection can also introduce security vulnerabilities in a network. The centralised server, by its nature, must collect data across zones with different levels of security. However, many reporting mechanism (for instance syslog) are based on UDP, so a central syslog server would inherit any UDP vulnerabilities. Therefore, a central syslog server should not be located in a high-security zone (*e.g.,* in a SCADA-Zone) [21]. Another vulnerability stems from the unsafe storage of sensitive information (*e.g.,* firewall-admin passwords) in the reports.

However, these vulnerabilities can be minimised with a carefully constructed collection strategy. For example, a syslog server can be placed in a DMZ and polled from the internal networks. Reports can also be sanitised to ensure that an attacker cannot obtain sensitive information. Doing so would allow each use case to reap the benefits of centralised collection, and reduce associated risks. However, this does require a careful and considered report architecture.

In reality, the two polarised extremes of decentralised and centralised are rare. There is actually a spectrum (Figure 8.5) where some processing and analysis is

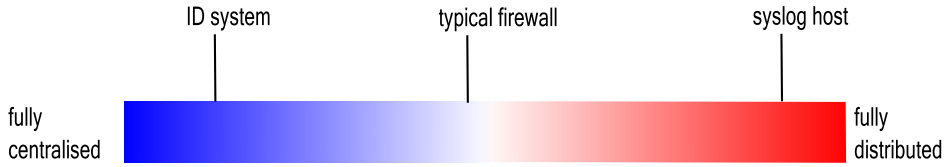


Figure 8.5: Spectrum of centralised vs distributed report collection.

distributed but eventually summary data is brought to one or small number of data historians. For example, firewalls typically perform significant processing of network traffic prior to sending back some notifications to a central historian.

8.3.2 Report Retention

Reported information needs to be retained for some time determined by the requirements of the use case. There are cyber-security industry standards [84, 85] for data retention. The rationale for the lengths suggested are, however, not clear, and the standards also make a distinction between *reportable events* and others, without clearly specifying what is reportable.

Given the progressively decreasing electronic storage costs, and the original data collecting cost, we argue that, from a cyber-security industry point of view, it is more cost-effective today to retain firewall reports (almost) perpetually.

If storage really becomes a problem, then progressive reduction in granularity can be a useful tool. For instance, see RRDtool [88].

However, we do consider minimum retention periods for each use case here.

Accounting for network usage reports are useful until the end of an usage evaluation period. Once processed and usage information is extracted, these reports can be purged. NERC recommends retaining these for 90 days [85]. So, we suggest the minimum of the two periods.

Intrusion detection: Some attacks (*e.g.,* DoS, Port scans) can last longer than others. So, at a minimum, reports must be retained until the attack has passed. Post-attack retention also helps identify the types of attacks common in a network, so better defence mechanisms can be formulated for the future. NERC recommends retaining these reports for a minimum of 90 days [85], and this seems a reasonable lower bound.

Policy verification reports are useful until a verification period ends. Post report-processing, invalid rules are located and rectified, so reports can be purged. NERC recommends retaining these reports for a minimum of 90 days [85], and this seems a reasonable lower bound.

Post-mortem analysis reports should allow tracing back of, for instance, the origins of a SCADA network attack that circumvented intrusion detection systems.

Doing so, may require processing of historical records that date back several years. NERC suggests the reports should be retained for a minimum of 3 years [84], and this seems a reasonable lower bound.

Troubleshooting reports are useful to monitor firewall configuration errors in near-real time. Once the error is rectified, they may be archived for future reference. NERC recommends retaining these reports for a minimum of 90 days [85], and again we suggest this is a reasonable lower bound.

Note again, that while standards propose reasonable lower bounds, we suggest that if data is worth collecting, it is worth keeping. Retention policies should therefore aim to store data indefinitely and if this is not possible, it is necessary to re-consider why the data is being collected. This leads to our next point.

8.4 What *Should* Firewalls Report?

The previous sections discussed what a firewall *could* report, and the use of that data. However, the ability to generate data does not mean we should. There is a cost to data collection, and some functions are better supported elsewhere.

For instance, firewalls have a limited perspective of a network so effective Intrusion Detection (ID) requires additional sensors (for instance at hosts and on wireless networks [108]). However, given such sensors why use the firewall at all? Collecting data at the firewall could compromise its main function through the cost of data collection, which may be amplified by a DoS attack. Moreover, a firewall, by its nature, is a visible target. It is far better to use passive (invisible) sensors for ID.

The conclusion is that a firewall is a poor source of data for ID.

Post-mortem analysis has fine granularity requirements, similar to ID, but lacks the near-real-time requirement. Hence, this use case is a marginally more acceptable use of firewall resources, but should not be implemented lightly.

Network usage accounting also requires traffic reporting though only at *flow-level* traffic-granularity. During a DoS attack, many flows can amplify the effect of the attack. But, if accounting focuses on accepted traffic, this effect is mitigated.

Policy verification is key to maintaining robust and efficient firewall configurations. The granularity requirements for this use case are quite coarse, and other devices cannot provide accurate information on packets that are matched by a firewall's policy, so this type of use case should definitely be supported.

Similarly, troubleshooting requires coarse information that can only be collected by the firewall, *e.g.,* an accurate change history can only be obtained from the firewall itself. Hence, firewall reporting should be enabled for this use case.

8.5 Implementation

We make the ideas discussed so far concrete, by presenting our implementation next. One of our goals is to extend *ForestFirewalls*: our high-level firewall specification developed in the earlier chapters. We describe here the mechanism for the two use cases that we identified above as requiring firewall reporting: (i) *policy verification* and (ii) *troubleshooting*.

Current firewall configuration platforms present "policy" and reporting as separate functions. Decoupling can be useful in some contexts to separate *structure* from *function*, *e.g.,* to separate security policy from the underlying network.

However, reporting and security policy are *inter-dependent* network functions. Scope and granularity are intimately related to the use of data, which, in turn, is related to the corresponding policies. Decoupling the two allows bad decisions to be made: for instance, addition of policies that aren't verified.

The Object Oriented Programming [36] paradigm makes it clear that *encapsulation* of related concepts and code together is vital for reliable and maintainable systems. A useful policy specification platform should encapsulate related specifications together. This implies that security policy and reporting specifications should be encapsulated together. Doing so, gives SCADA security managers a *single source of truth* to see who gets in and who doesn't, along with the audit trails required to check the configuration. We show how this works in the following.

8.5.1 Reporting Policy for Verification

The granularity and scope requirements for a typical policy verification scenario are given in Table 8.1. We then list, in Table 8.2, the resulting reporting *attributes* grouped by granularity dimension, as we define them in *ForestFirewalls*. We specify these attribute values through *ForestFirewalls* as shown in Listing 8.1.

A `rule_group` contains a set of security policy rules, and a `reporting_rule` object defines a set of reporting requirements. The `policy` object encapsulates the security policy rules and the reporting rule together. This facilitates reuse both of policies and reporting rules (which could come from a library), but encapsulates them together in the final specification.

Table 8.1: Summary of reporting granularity requirements for policy verification (terminology defined in Section 8.2).

Granularity dimension	Required granularity	Required scope
Network	zone-conduit	network-wide
Policy	rule	global
Traffic-measurement	connection-count	counter
Temporal	per-hour	per-day
Performance	process	firewall

Table 8.2: Reporting policy attributes resulting from the granularity requirements described in Table 8.1.

Reporting attribute	Example value
use_case	verification;
granularity.network	{zone_or_group={SCADA}; traffic_direction={inbound};}
granularity.policy	{rule_or_group={rule1}; policy_action={permit};}
granularity.traffic	{measurement={counter}; counter_type={connection};}
granularity.temporal	{per_hour}
granularity.performance	{process}

Listing 8.1: ForestFirewalls reporting-policy description for verification (from SUC 1 policy listed earlier in Section 7.2).

```
1 rule_group security_policy_rules {<rule1>, <rule2>,..., <ruleN>}
2
3 reporting_rule verify_a_rule {
4  use_case=verification;
5  granularity.policy={rule_or_group={<rule1>}};
6  granularity.traffic={measurement={counter}; counter_type={connection};};
7  granularity.temporal={per_hour};}
8
9 policy <policy-name> { security_policy_rules, verify_a_rule }
```

8.5.2 Reporting Policy for Troubleshooting

A summary of the granularity requirements for this use case is given in Table 8.3. The list of specification attributes is given in Table 8.4. We use these attributes to specify troubleshoot reporting for one or more firewalls as shown Listing 8.2.

Table 8.3: Summary of reporting granularity requirements for troubleshooting firewall errors (terminology defined in Section 8.2).

Granularity Dimension	Required granularity	Scope
Network	Zone-Conduit	network-wide
Policy	rule	global
Temporal	near real-time	per-day
Performance	interface	firewall

Table 8.4: Reporting policy attributes resulting from the granularity requirements described in Table 8.3.

Reporting attribute	Example value
use_case	troubleshoot;
granularity.network	{zone_or_group={SCADA}; traffic_direction={inbound};}
granularity.policy	{rule_or_group={rule1}; policy_action={permit};}
granularity.temporal	{near_realtime}
granularity.performance	{measurement={interface};}

Listing 8.2: ForestFirewalls reporting-policy description for troubleshooting.

```
1  rule_group security_policy_rules {<rule1>, <rule2>,..., <ruleN>}
2
3  zone_group FIREWALL_ZONES {<zone1>, <zone2>,..., <zoneM>}
4
5  reporting_rule debug_firewalls {
6    use_case=troubleshoot;
7    granularity.network={zone_or_group={FIREWALL_ZONES}; traffic_direction=
8       {inbound, outbound}};
9    granularity.policy={rule_or_group={security_policy_rules}};
10   granularity.performance={measurement={interface}; performance_type=
11      {memory, CPU, packet_loss, queue_length}};
12   granularity.temporal={near_realtime};}
13
14 policy <policy-name> { security_policy_rules, debug_firewalls }
```

As before, the `reporting_rule` object for the use case includes the attributes, *e.g., interface-level* performance statistics and *near real-time* temporal-granularity. The reporting rule is encapsulated in the `policy` statement.

8.6 Lessons Learned

There are several takeaways from our study

1. A SCADA firewall should not cater for every use case. For some use cases, it is better to employ additional dedicated infrastructure to meet requirements, and let firewalls conduct their primary function – *traffic filtering* – unimpeded by other complexities.

2. Firewall reporting must be configured at the right granularity for its use. Data that is collected but not used is just wasting resources.

3. Reporting and security policy must be coupled. Both are inter-dependent network functions and there is little sense in deploying one without the other.

4. Firewall vendors need to support standard firewall features to consistently map high-level reporting policy to firewall capabilities. These include: performance, operational and traffic measurements, and policy actions.

8.7 Conclusions

The standards and best practices for reporting and analysis of firewall data lack clarity in what firewalls *should* report.

Our research utilises the use cases of firewall reports and specifies reporting in terms of scope and granularity. From this we identify reporting requirements with respect to several dimensions: time, network elements, policies, *etc*, and evaluate costs. We provide clarity on what a SCADA firewall *should* report, and demonstrate our high-level reporting implementation.

Chapter 9

Conclusions

This thesis addresses the problem of error-prone firewall configuration in Supervisory Control And Data Acquisition (SCADA) networks. These networks control the distributed assets of many critical infrastructure including electricity generation, water distribution and sewage management. Firewalls are a primary defence mechanism in safeguarding these critical assets from cyber threats and hence, their correct configuration is paramount to the upkeep of national security.

However, firewall configuration in practice is tedious and error-prone due to the complex and proprietary configuration platforms offered by vendors. These platforms require policies to be managed manually, box-by-box, using large volumes of network-intricacies such as IP addresses. Moreover, intra- and inter-firewall policy interactions can produce adverse outcomes. Lack of built-in verifiability in existing configuration platforms means SCADA owners must rely on external cyber-security audits (which are perforce infrequent) to accurately detect these firewall-policy flaws.

The problem is also exacerbated by the lack of precise standards for firewall policy specification. The shortfall makes policy semantics rule-order and firewall-implementation dependent, hindering meaningful comparison, for instance, against best practices. Moreover, the lack of decoupling between policy and network makes policy sensitive to network-intricacies and vendor changes.

Existing firewall configuration platforms also do not allow designing a firewall policy first, and then determining how to implement it, as recommended by best-practices [21]. Instead network installations are built from bottom-up, *i.e.,* a firewall policy is the outcome of its configuration, which in turn is a consequence of a purchasing decision. Thus, a firewall policy is *implicitly* defined as a consequence of many small decisions that interact in complex ways.

The solution we propose: *ForestFirewalls*, addresses these challenges and offers high-level firewall policy specification based on formal foundations. The formal semantics enable construction of complex and flexible firewall policies and

support reasoning about the policies. For instance, it allows one to (i) check policies are consistent and best-practice compliant; and (ii) verify whether the policies are mapped to the underlying network-firewall correctly.

The coherent concepts and relationships we enforce throughout *ForestFirewalls* produce a high-level firewall specification that (i) is simple, intuitive and reduces the configuration burden on its users by enabling abstract policies that are decoupled from network-implementation details; and (ii) allows to construct a single network-wide firewall policy (*i.e.,* source of truth) to determine who gets in and who doesn't; and (iii) has automated, rigorous verification built-in to improve the baseline security policy deployed in a SCADA network. Such verification ensures that only consistent, feasible policy that delivers the expected security outcome is deployable ; and (iv) supports automated policy refinement to ensure that only efficient and concise firewall configurations are deployable.

ForestFirewalls promotes secure firewall-management practices by restricting firewall management to secure protocols such as SSH and HTTPS and ensuring that the firewalls protecting the perimeter of a SCADA-Zone are managed from the SCADA domain. Our system also relies on the principle that desired flows must be explicitly allowed; implicit rules contributes to unexpected interactions, and undesirable consequences and are hence not supported.

Our policy specification framework supports both security policies and reporting policies. Reporting policies are important to continuously monitor firewall security, post deployment. But, the standards and best practices for reporting and analysis of firewall data lack clarity in what firewalls *should* report. *ForestFirewalls* bridges the gap by utilising the use cases of firewall reports and enabling reporting specification in terms of scope and granularity.

We demonstrate the ease with which our specification features can be administered by applying it to real SCADA firewall configuration case studies. There is a significant increase in configuration accuracy achieved by our system by only generating consistent, best-practice compliant firewall configurations that deliver the expected security outcome pre- and post-deployment.

Future work could involve extending *ForestFirewalls* to also cater for the Enterprise domain. However, there are additional aspects of policy specification that need to be taken into account in enabling this capability. For instance, NAT and VPN were not dominant firewall features used in SCADA networks. But, they play a significant role in Enterprise networks which commonly have direct connectivity to the Internet enabled. Hence, careful consideration must be given in enabling these features through the policy specification.

ForestFirewalls could also be extended to support application-layer filtering (*i.e.,* Deep Packet Inspection (DPI)) policies. The network-layer policies currently supported by our system create a robust security baseline by controlling the legitimate-access of network applications. But, these policies do not distinguish

between allowed and prohibited use of these (legitimately-accessed) applications. Thus, security administrators require additional support in the specification for DPI policies to refine traffic filtering at the application and user layers.

Glossary

ABAC	Attribute Based Access Control.
ACE	Access Control Entry.
ACL	Access Control List.
ANSI	American National Standards Institute.
ASA	Adaptive Security Appliance.
BDD	Binary Decision Diagram.
BNF	Backus-Naur Form.
CIP	Critical Infrastructure Protection.
CLI	Command Line Interface.
CN	Carrier Network.
COTS	Commodity Off-The-Shelf.
CZ	Corporate-Zone.
DCOM	Distributed Component Object Model.
DMZ	Demilitarised-Zone.
DNS	Domain Name System.
DoS	Denial of Service.
FDD	Firewall Decision Diagram.
FM	Feasible Mapping.
FTP	File Transfer Protocol.
FWSM	Firewall Services Module.
FWZ	Firewall-Zone.
HTTP	Hypertext Transfer Protocol.
HTTPS	Hypertext Transfer Protocol Secure.
IANA	Internet Assigned Numbers Authority.
ICMP	Internet Control Message Protocol.

IDS	Intrusion Detection Systems.
IETF	Internet Engineering Task Force.
IL	Intermediate Level.
IOS	Internetwork Operating System.
IP	Internet Protocol.
ISA	International Society for Automation.
IZ	Internet-Zone.
MS-SQL	Microsoft SQL Server.
MZ	Management-Zone.
NAT	Network Address Translation.
NERC	North Atlantic Electric Reliability Corporation.
NIDB	Network Inventory Database.
NMS	Network Monitoring Station.
NTP	Network Time Protocol.
OOP	Object Oriented Programming.
OSI	Open System Interconnection.
PAP	Policy Administration Point.
PAT	Port Address Translation.
PCIM	Policy Core Information Model.
PDN	Policy Defined Networking.
PDP	Policy Decision Point.
PEP	Policy Enforcement Point.
PIX	Private Internet eXchange.
PLC	Programmable Logic Controller.
PR	Policy Repository.
QoS	Quality of Service.
RBAC	Role Based Access Control.
RISI	Repository of Industrial Security Incidents.
SCADA	Supervisory Control and Data Acquisition.
SMTP	Simple Mail Transfer Protocol.
SNMP	Simple Network Management Protocol.
SSH	Secure Shell Protocol.
SUC	System Under Consideration.
SZ	SCADA-Zone.

TCP	Transmission Control Protocol.
UDP	User Datagram Protocol.
UML	User Mode Linux.
UTM	Unified Thread Management.
VM	Virtual Machine.
VPN	Virtual Private Network.
WAF	Web Application Firewall.
WAN	Wide Area Network.

Bibliography

[1] D. Acosta. PCI DSS engineering controls part II: Firewall. [Online]. Available: www.pcihispano.com/controles-tecnicos-de-ci-dss-parte-ii-cortafuegos-firewall/, 2014.

[2] E. Al-Shaer and H. Hamed. Design and implementation of firewall policy advisor tools. *DePaul University, CTI, Technical Report*, 2002.

[3] E. Al-Shaer, H. Hamed, R. Boutaba, and M. Hasan. Conflict classification and analysis of distributed firewall policies. *IEEE Journal on Selected Areas in Communications*, 23(10):2069–2084, 2005.

[4] E. S. Al-Shaer and H. H. Hamed. Discovery of policy anomalies in distributed firewalls. In *IEEE INFOCOM*, volume 4, pages 2605–2616, 2004.

[5] Alcatel-Lucent Inc. *Alcatel-Lucent VPN Firewall Brick 20 Security Appliance*, 2007.

[6] Algorithmic Security Inc. AlgoSec Firewall Analyser. [Online]. Available: www.algosec.com/firewall-analyzer, 2006.

[7] Algorithmic Security Inc. The practitioner's guide to deploying, optimizing and managing next generation firewalls. [Online]. Available: www.pages.algosec.com/PractitionersGuidetoNGFWs.html, 2012.

[8] M. Allman and S. Ostermann. FTP security considerations. *RFC 2577, Internet Engineering Task Force*, 1999.

[9] C. J. Anderson, N. Foster, A. Guha, J.-B. Jeannin, D. Kozen, C. Schlesinger, and D. Walker. NetKAT: Semantic foundations for networks. *ACM SIGPLAN Notices*, 49(1):113–126, 2014.

[10] C. Avelsgaard. *Foundations for Advanced Mathematics*. Scott, Foresman, Brown Higher Education, 1990.

[11] L. Babai. Graph isomorphism in quasipolynomial time. *arXiv preprint arXiv:1512.03547*, 2015.

[12] Y. Bartal, A. Mayer, K. Nissim, and A. Wool. Firmato: A novel firewall management toolkit. *ACM Transactions on Computer Systems*, 22(4):381–420, 2004.

[13] BBC. No internet for Singapore public servants. [Online]. Available: www.bbc.com/news/world-asia-36476422, 2016.

[14] BBC. Hack attack causes 'massive damage' at steel works. [Online]. Available: www.bbc.com/news/technology-30575104, 2014.

[15] S. Bellovin and R. Bush. Configuration management and security. *IEEE Journal on Selected Areas in Communication*, 27(3):268–274, 2009.

[16] I. C. Bertolotti, L. Durante, L. Seno, and A. Valenzano. A twofold model for the analysis of access control policies in industrial networked systems. *Computer Standards & Interfaces*, 42:171–181, 2015.

[17] P. Billingsley. Probability and measure. *A Wiley-Interscience Publication,Wiley & Sons, New York*, 1995.

[18] F. P. Brooks Jr. *The mythical man-month (anniversary ed.)*. Addison-Wesley Longman Publishing Co., Inc., 1995.

[19] E. Byres. Using ANSI/ISA-99 standards to improve control system security. White paper, Tofino Security, May 2012.

[20] E. Byres. The air gap: SCADA's enduring security myth. *Communications of the ACM*, 56(8):29–31, 2013.

[21] E. Byres, J. Karsch, and J. Carter. NISCC good practice guide on firewall deployment for SCADA and process control networks. *NISCC*, 2005.

[22] E. Cambria and B. White. Jumping NLP curves: a review of natural language processing research. *IEEE Computational Intelligence Magazine*, 9(2):48–57, 2014.

[23] CBS. "Godzilla Attack!" San Francisco traffic sign hacked to warn drivers. [Online]. Available: www.cbsnews.com/news/godzilla-attack-s-f-traffic-sign-hacked-to-warn-drivers/, 2014.

[24] Check Point. *NGX R65 CC Evaluated Configuration User Guide*. Check Point, software technologies Ltd., USA, 2008.

[25] F. Chen, A. X. Liu, J. Hwang, and T. Xie. First step towards automatic correction of firewall policy faults. *ACM Transactions on Autonomous and Adaptive Systems*, 7(2):27, 2012.

[26] X. Chen, Z. M. Mao, and J. Van der Merwe. Towards automated network management: network operations using dynamic views. In *SIGCOMM workshop on Internet network management*, pages 242–247. ACM, 2007.

[27] W. R. Cheswick, S. M. Bellovin, and A. D. Rubin. *Firewalls and Internet security: Repelling the wily hacker*. Addison-Wesley, 2003.

[28] Cisco Systems Inc. *Catalyst 6500 Series Switch and Cisco 7600 Series Router Firewall Services Module Configuration Guide Using the CLI*, 2010.

[29] Cisco Systems Inc. *Cisco ASA 5500 Series Configuration Guide using the CLI*, 2010.

[30] Cisco Systems Inc. *Cisco ASA Series CLI Configuration Guide, 9.0*, 2013.

[31] Cisco Systems Inc. ASA 8.3 and later: Monitor and troubleshoot performance issues. White paper, March 2014.

[32] Cisco Systems Inc. Cisco ASA 5585-X adaptive security appliance architecture. White paper, May 2014.

[33] Cisco Systems Inc. *Cisco IOS Security Configuration Guide, Release 12.4*, 2014.

[34] Cisco Systems Inc. *Cisco Virtual Security Gateway for Nexus 1000V Series Switch Configuration Guide*. 170 West Tasman Drive, San Jose, CA 95134-1706, USA, 2014.

[35] Cisco Systems Inc. CVE Details, The ultimate security vulnerability datasource. [Online]. Available: www.cvedetails.com/vulnerability-list/, January 2016.

[36] D. De Champeaux, D. Lea, and P. Faure. *Object-oriented system development*. Addison Wesley, Reading, MA, 1993.

[37] R. Droms. Dynamic Host Configuration Protocol (DHCP). *RFC 2131*, 1997.

[38] S. Dynerowicz and T. G. Griffin. On the forwarding paths produced by Internet routing algorithms. In *ICNP*, pages 1–10, 2013.

[39] W. M. Eddy. Defenses against TCP SYN flooding attacks. *The Internet Protocol Journal*, 9(4):2–16, 2006.

[40] A. Endres. An analysis of errors and their causes in system programs. In *ACM SIGPLAN Notices*, volume 10, pages 327–336, 1975.

[41] N. Falliere, L. O. Murchu, and E. Chien. W32. Stuxnet dossier. *White paper, Symantec Corp., Security Response*, 5, 2011.

[42] N. Feamster, J. Rexford, and E. Zegura. The road to SDN. *Queue*, 11(12):20, 2013.

[43] D. Ferraiolo, J. Cugini, and D. R. Kuhn. Role-Based Access Control (RBAC): Features and motivations. In *Proceedings of the 11th annual computer security application conference*, pages 241–48, 1995.

[44] FireMon Homepage. [Online]. Available: www.firemon.com, July 2016.

[45] J. Garcia-Alfaro, F. Cuppens, N. Cuppens-Boulahia, S. Martinez, and J. Cabot. Management of stateful firewall misconfiguration. *Computers & Security*, 39:64–85, 2013.

[46] M. G. Gouda and A. X. Liu. Structured firewall design. *Computer Networks*, 51(4):1106–1120, 2007.

[47] K. D. Gourley and D. M. Green. Polygon-to-rectangle conversion algorithm. *IEEE CGA*, pages 31–32, 1983.

[48] Graph Drawing Steering Committee. GraphML. [Online]. Available: www.graphml.graphdrawing.org, June 2002.

[49] J. D. Guttman and A. L. Herzog. Rigorous automated network security management. *International Journal of Information Security*, 4:29–48, 2005.

[50] M. Hall. Understanding the file system architecture in Windows CE .NET. [Online]. Available: www.msdn.microsoft.com/en-au/library/aa459155.aspx, 2003.

[51] W. Han and C. Lei. A survey on policy languages in network and security management. *Computer Networks*, pages 477–489, 2012.

[52] F. Harary and R. Z. Norman. Some properties of line digraphs. *Rendiconti del Circolo Matematico di Palermo*, 9(2):161–168, 1960.

[53] V. Heorhiadi, M. K. Reiter, and V. Sekar. Simplifying software-defined network optimization using SOL. In *USENIX NSDI*, pages 223–237, 2016.

[54] S. Hinrichs. Policy-based management: Bridging the gap. In *Proceedings of 15th Annual Computer Security Applications Conference*, pages 209–218. IEEE, 1999.

[55] Homeland Security. Recommended practice: Improving industrial control systems cybersecurity with defense-in-depth strategies, 2009.

[56] C. Howe, B. Erwin, C. Barth, and S. Elliot. *What's beyond firewalls?* The Forrester Report, Forrester Research Inc., 1996.

[57] V. C. Hu, D. R. Kuhn, and D. F. Ferraiolo. Attribute-based access control. *Computer*, 48(2):85–88, 2015.

[58] ICS-CERT. ICS-CERT Monitor, September 2014-February 2015. [Online]. Available: www.ics-cert.us-cert.gov/sites/default/files/Monitors/ICS-CERT_Monitor_Sep2014-Feb2015.pdf, 2015.

[59] International Society of Automation. Security for industrial automation and control systems part 1-1: Terminology, concepts, and models, ANSI/ISA-62443-1-1 (99.01.01), 2007.

[60] D. Jackson. *Software Abstractions: Logic, Language, and Analysis*. The MIT Press, 2011.

[61] S. Jajodia and P. Samarati. A logical language for expressing authorizations. In *IEEE Security and Privacy, pages 31-42*, 1997.

[62] R. Jamieson, L. Land, S. Smith, G. Stephens, and D. Winchester. Critical infrastructure information security: Impacts of identity and related crimes. In *PACIS*, page 78, 2009.

[63] D. S. Johnson. The NP-completeness column. *ACM Transactions on Algorithms*, 1(1):160–176, July 2005.

[64] Juniper Networks, Inc. *Firewall Filter and Policer Configuration Guide*. 1194 North Mathilda Avenue, Sunnyvale, California 94089, USA, 2011.

[65] L. Kagal. Rei: a policy language for the Me-Centric project. *HP Labs*, 2002.

[66] G. Kambourakis, T. Moschos, D. Geneiatakis, and S. Gritzalis. A fair solution to DNS amplification attacks. In *Second International Workshop on Digital Forensics and Incident Analysis*, pages 38–47. IEEE, 2007.

[67] K. Kent and M. Souppaya. Guide to computer security log management. *NIST Special Publication*, 800(92):16–16, 2006.

[68] B. W. Kernighan and P. J. Plauger. *The elements of programming style*. McGraw-Hill, PJ New York, 1978.

[69] Kindervag, John. Virtual Network Segmentation for PCI? [Online]. Available: www.blogs.forrester.com/security_and_risk/2010/01/virtual-network-segmentation-for-pci.html, January 2010.

[70] S. Knight, H. Nguyen, O. Maennel, I. Phillips, N. Falkner, R. Bush, and M. Roughan. An automated system for emulated network experimentation. In *ACM CoNEXT*, pages 235–246, 2013.

[71] D. Levin, M. Canini, S. Schmid, and A. Feldmann. Panopticon: Reaping the benefits of partial SDN deployment in enterprise networks. *TU Berlin/T-Labs, Technical Report*, 2013.

[72] D. Libes. *Exploring Expect: A Tcl-based toolkit for automating interactive programs*. O'Reilly, 1995.

[73] A. X. Liu and M. G. Gouda. Diverse firewall design. *IEEE International Conference on Dependable Systems and Networks*, pages 1237–1251, 2008.

[74] W.-Z. Lu and S.-Z. Yu. An HTTP flooding detection method based on browser behavior. In *International Conference on Computational Intelligence and Security*, volume 2, pages 1151–1154. IEEE, 2006.

[75] Mako. Mako templates for Python. [Online]. Available: www.makotemplates.org, 2016.

[76] R. M. Marmorstein and P. Kearns. Firewall analysis with policy-based host classification. *Large Installation System Administration Conference*, 6:4–4, 2006.

[77] A. Mayer, A. Wool, and E. Ziskind. Fang: A firewall analysis engine. In *IEEE Security and Privacy*, pages 177–187, 2000.

[78] J. McKendrick. Another view: XML not meant to be human readable. [Online]. Available: www.tinyurl.com/hytddnt, 2006.

[79] B. Moore. Policy core information model (PCIM) extensions. 2003.

[80] B. Moore, E. Ellesson, J. Strassner, and A. Westerinen. Policy core information model (PCIM)–version 1 specification. Technical report, RFC 3060, February, 2001.

[81] T. Moses. OASIS eXtensible Access Control Markup Language 2.0, core specification. *OASIS XACML Technical Committee Standard*, 2005.

[82] J. Moy. Open Shortest Path First (OSPF) version 2. *IETF RFC 2328*, 1998.

[83] J. Myers. CGI Security: Escape Newlines. [Online]. Available: www.seclists.org/bugtraq/1996/Feb/16, 1996.

[84] NERC. Cyber security: Incident reporting and response planning. *Critical Infrastructure Protection Standards*, 008(3), 2009.

[85] NERC. Cyber security: Systems security management. *Critical Infrastructure Protection Standards*, 007(3a), 2013.

[86] NERC. CIP Compliance. [Online]. Available: www.nerc.com/pa/CI/Comp/Pages/default.aspx, 2016.

[87] H. Nguyen, M. Roughan, S. Knight, N. Falkner, O. Maennel, and R. Bush. How to build complex, large-scale emulated networks. In *TridentCom*, pages 3–18. Springer, 2010.

[88] T. Oetiker. RRDtool. [Online]. Available: www.oss.oetiker.ch/rrdtool/, 2014.

[89] J. Pearce. *Programming and Meta-Programming in Scheme*. Springer, 1998.

[90] J. Pescatore and G. Young. Defining the next-generation firewall, www.gartner.com/doc/1204914/defining-nextgeneration-firewall. *Gartner RAS Core Research Note*, 2009.

[91] M. Pizzonia and M. Rimondini. Netkit: Easy emulation of complex networks on inexpensive hardware. In *TRIDENTCOM*, page 7. ICST, 2008.

[92] C. Prakash, J. Lee, Y. Turner, J.-M. Kang, A. Akella, C. Clark, Y. Ma, P. Sharma, and Y. Zhang. PGA: Using graphs to express and automatically reconcile network policies. In *ACM SIGCOMM, pages 29-42*, 2015.

[93] G. N. Purdy. *Linux iptables pocket reference*. O'Reilly Media Inc., 2004.

[94] D. Ranathunga, M. Roughan, P. Kernick, and N. Falkner. Malachite: Firewall policy comparison. In *IEEE Symposium on Computers and Communication (ISCC)*, pages 310–317, June 2016.

[95] D. Ranathunga, M. Roughan, P. Kernick, and N. Falkner. The mathematical foundations for mapping policies to network devices. In *13th International Joint Conference on e-Business and Telecommunications*, 2016.

[96] D. Ranathunga, M. Roughan, P. Kernick, N. Falkner, and H. Nguyen. Identifying the missing aspects of the ANSI/ISA best practices for security policy. In *Workshop on Cyber-Physical System Security*, pages 37–48. ACM, 2015.

[97] D. Ranathunga, M. Roughan, P. Kernick, N. Falkner, and H. Nguyen. Case Studies of SCADA Firewall Configurations and the Implications for Best Practices. In *IEEE Transactions on Network and Service Management (TNSM), early access link* http://ieeexplore.ieee.org/document/7529047/, 2016.

[98] J. Reich, C. Monsanto, N. Foster, J. Rexford, and D. Walker. Modular SDN programming with Pyretic. *Technical Report of USENIX*, 2013.

[99] Y. Rekhter and T. Li. A Border Gateway Protocol 4 (BGP-4). *RFC 4271*, 1995.

[100] Reuters. Austria's FACC, hit by cyber fraud, fires CEO. [Online]. Available: www.reuters.com/article/us-facc-ceo-idUSKCN0YG0ZF, 2016.

[101] S. A. Rouiller. Virtual LAN security: Weaknesses and countermeasures. *SANS Institute InfoSec Reading Room*, 2003.

[102] D. Rovniagin and A. Wool. The geometric efficient matching algorithm for firewalls. In *23rd IEEE Convention of Electrical and Electronics Engineers in Israel*, pages 153–156, 2004.

[103] A. D. Rubin and D. E. Geer. A survey of Web security. *IEEE Computer*, pages 34–41, 1998.

[104] O. Rysavy, J. Rab, and M. Sveda. Improving security in SCADA systems through firewall policy analysis. In *Federated Conference on Computer Science and Information Systems*, pages 1435–1440. IEEE, 2013.

[105] SaltStack. Intelligent orchestration for the software-defined data center. [Online]. Available: www.saltstack.com/, February 2017.

[106] SANS Institute. Intrusion detection FAQ: Are there vulnerabilites in VLAN implementations? VLAN security test report. [Online]. Available: www.sans.org/security-resources/idfaq/vlan.php, 2016.

[107] K. Scarfone and P. Hoffman. Guidelines on firewalls and firewall policy. *NIST Special Publication*, 800(41), 2009.

[108] K. Scarfone and P. Mell. Guide to Intrusion Detection and Prevention Systems (IDPS). *NIST Special Publication*, 800(94):16–16, 2007.

[109] G. W. Scheer and D. J. Dolezilek. Comparing the reliability of Ethernet network topologies in substation control and monitoring networks. *Western Power Delivery Automation Conference, Spokane, Washington*, 2000.

[110] Security Incidents Organization. RISI database. [Online]. Available: www.risidata.com/Database/event_date/desc, 2016.

[111] Skybox Security. Skybox Firewall Assurance. [Online]. Available: www.skyboxsecurity.com/products/skybox-firewall-assurance, 2016.

[112] M. Słabicki and K. Grochla. Performance evaluation of snmp, netconf and cwmp management protocols in wireless network. In *Proceedings of the 4th International Conference on Electronics, Communications and Networks (CECNET IV)*, page 377. CRC Press, 2015.

[113] S. Smolka, S. Eliopoulos, N. Foster, and A. Guha. A fast compiler for NetKAT. In *ACM SIGPLAN*, pages 328–341, 2015.

[114] Y. Snir, Y. Ramberg, J. Strassner, R. Cohen, and B. Moore. Policy quality of service (QoS) information model. *IETF RFC 3644*, 2003.

[115] R. Soulé, S. Basu, P. J. Marandi, F. Pedone, R. Kleinberg, E. G. Sirer, and N. Foster. Merlin: A language for provisioning network resources. In *CoNEXT*, pages 213–226. ACM, 2014.

[116] Statista. Global market share held by security appliance vendors from 1st quarter 2012 to 4th quarter 2015. [Online]. Available: www.tinyurl.com/jgssmpp.

[117] K. Stouffer, J. Falco, and K. Scarfone. Guide to Industrial Control Systems (ICS) security. *NIST Special Publication*, 800(82):16–16, 2008.

[118] Y.-W. E. Sung, S. G. Rao, G. G. Xie, and D. A. Maltz. Towards systematic design of enterprise networks. In *ACM CoNEXT*, page 22, 2008.

[119] Symantec. Adware.Zango. [Online]. Available: www.symantec.com/security_response/writeup.jsp?docid=2005-050416-3519-99, 2007.

[120] Symantec. DRIDEX and how to overcome it. [Online]. Available: www.symantec.com/connect/blogs/dridex-and-how-overcome-it, 2016.

[121] Symantec. Duqu 2.0: Reemergence of an aggressive cyberespionage threat. [Online]. Available: www.symantec.com/connect/blogs/duqu-20-reemergence-aggressive-cyberespionage-threat, 2016.

[122] Symantec. Trojan.Zlob. [Online]. Available: www.symantec.com/security_response/writeup.jsp?docid=2005-042316-2917-99, 2006.

[123] Symantec. Cyberespionage Attacks Against Energy Suppliers, version 1.21. White paper, Mountain View, California, 2014.

[124] T. Tuglular, F. Cetin, O. Yarimtepe, and G. Gercek. Firewall configuration management using XACML policies. *In 13th International Telecommunications Network Strategy and Planning Symposium, Sep*, 2008.

[125] K. Twidle, N. Dulay, E. Lupu, and M. Sloman. Ponder2: A policy system for autonomous pervasive environments. In *International Conference On Autonomic and Autonomous Systems*, pages 330–335, 2009.

[126] D. C. Verma. Simplifying network administration using policy-based management. *IEEE Network*, 16(2):20–26, 2002.

[127] A. Wool. Architecting the Lumeta firewall analyzer. In *USENIX Security Symposium*, pages 85–97, 2001.

[128] A. Wool. A quantitative study of firewall configuration errors. *IEEE Computer*, 37(6):62–67, 2004.

[129] A. Wool. Trends in firewall configuration errors: Measuring the holes in Swiss cheese. *IEEE Internet Computing*, 14(4):58–65, 2010.

[130] yEd. yEd graph editor manual. [Online]. Available: www.yed.yworks.com/support/manual/index.html, 2016.

[131] L. Yuan, H. Chen, J. Mai, C.-N. Chuah, Z. Su, and P. Mohapatra. FIREMAN: A toolkit for firewall modeling and analysis. In *IEEE Security and Privacy*, pages 15–213, 2006.

[132] 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 *Symposium on Access control models and technologies*, pages 185–194. ACM, 2007.

[133] E. D. Zwicky, S. Cooper, and D. B. Chapman. *Building Internet firewalls*. O'Reilly, 2009.