Verifiable Policy-Defined Networking for Security Management

Dinesha Ranathunga[1], Matthew Roughan[1], Phil Kernick[2], Nick Falkner[3], Hung Nguyen[4], Michelle McClintock[3] and Marian Mihailescu[4]

[1]*ACEMS, University of Adelaide, Adelaide, Australia*

[2]*CQR Consulting, Unley, Australia*

[3]*School of Computer Science, University of Adelaide, Adelaide, Australia*

[4]*TRC, University of Adelaide, Adelaide, Australia*

{*dinesha.ranathunga, matthew.roughan, nick.falkner, hung.nguyen, michelle.mcclintock, marian.mihailescu*}*@adelaide.edu.au, phil.kernick@cqr.com*

Keywords: Security policy, Zone-Conduit model, SCADA security, Security management.

Abstract: A common goal in network-management is security. Reliable security requires confidence in the level of protection provided. But, many obstacles hinder reliable security management; most prominent is the lack of built-in verifiability in existing management paradigms. This shortfall makes it difficult to provide assurance that the expected security outcome is consistent pre- and post-deployment.

Our research tackles the problem from first principles: we identify the verifiability requirements of robust security management, evaluate the limitations of existing paradigms and propose a new paradigm with verifiability built in: Formally-Verifiable Policy-Defined Networking (FV-PDN). In particular, we pay attention to *firewalls* which protect network data and resources from unauthorised access. We show how FV-PDN can be used to configure firewalls reliably in mission critical networks to protect them from cyber attacks.

1 INTRODUCTION

"He is making a list, and checking it twice;"

(Coots and Gillespie, 1934)

It is common knowledge that St. Nicholas maintains a 'naughty and nice' list of everyone on the planet. Santa even checks this list *twice* to ensure it's comprehensive and accurate. The question is: "Why don't administrators check security policies even once?" We know they don't check, because every existing study of firewalls has found serious problems in the majority of the cases (Wool, 2004; Ranathunga et al., 2015b).

Deploying un-verified security policies often leaves networks vulnerable to cyber attack, but, administrators have little choice in the case of firewalls because they frequently have to configure complex policies on many heterogeneous devices manually, box-by-box. This approach to network management is tedious, error-prone and expensive.

The concept of programmable networking aims to overcome these challenges in network management by allowing creation, deployment and management of novel services rapidly in a network. The evolution of programmable networks has been slow (Feamster et al., 2013), an outcome largely dependent on timely technological advances and use-case demands. But, programmable networks continue to lower the barrier to deploying new services.

Examples of network-management paradigms include Bhattacharjee et al., 1997; Magedanz and Popescu-Zeletin, 1996; Feamster et al., 2013; Verma, 2002. These have focused on innovations in both the networking hardware and software. One paradigm that is the outcome of network-software innovation is Policy-Defined Networking (PDN), also known as Policy-Based Network Management (Verma, 2002). In PDN, administrators specify high-level declarative policies that are refined to device-level configurations automatically by the underlying framework.

Network-programming paradigms commonly aim to facilitate security. But, reliable security-management has challenges

- a security policy must be formally verified as error-free prior to deployment, for instance, we need to compare policies to best-practice standards, *e.g.,* ANSI/ISA-62443-1-1;
- a policy must be consistent with the underlying network and technology; and
- security administrators need assurance that the expected security outcome pre- and post-deployment are consistent.

In this paper, we propose *Formally-Verifiable Policy-Defined Networking* (FV-PDN) to overcome these challenges. In doing so, we compare existing network- and security-management paradigms to better understand their limitations for security. Our contributions in summary are

- clear expression of need for verification.
- framework describing types of verification.
- explanation of how they can be achieved.
- discussion of challenges faced.

In particular, we pay attention to *firewalls* which are a standard mechanism for enforcing network security. Firewalls may only protect systems behind it from external threats, but they are a necessary evil in protecting data and resources in a network from unauthorised access (Rubin and Geer, 1998).

For instance, firewalls protect the distributed assets of industrial control systems in Supervisory Control and Data Acquisition (SCADA) networks from cyber attacks in this domain (Byres et al., 2005). They help prevent network-security breaches that can render significant physical and environmental damage, financial-loss or worse, the loss of human lives! Much of our work is motivated by mission critical security applications such as these.

2 LOWERING THE BARRIER TO NETWORK INNOVATION - A BRIEF HISTORY

Deploying ideas for better network management historically required following a *path to production* through approvals from industry standardising bodies such as the ITU, followed by uptake by vendors.

This uptake of research by industry was slow in the 1970s and 1980s, with ideas pushed to implementation typically after ten years or more. The lag was primarily due to vendors initially finding the majority of the proposed innovations *too hard to implement* (Feamster et al., 2013), given the high cost of computing. Technological advances eventually reduced computing cost, but, vendors continued to overlook these innovations driven by their need to keep networks closed and proprietary (Feamster et al., 2013).

As a result, frustrated researchers and organisations looked to open up network control: an idea that stemmed from the ability to reprogram a stand-alone PC (Wetherall, 1999). This initiative brought about new network programming models that focused on innovations in both the network hardware and software, to facilitate better network management and enable improved network services.

For instance, *Intelligent Networks* (late 1980s to mid 1990s) were introduced to flexibly add new, sophisticated services to existing fixed-line and mobile telecommunication networks (Magedanz and Popescu-Zeletin, 1996). The need for service providers to incorporate these value-added services without requesting feature changes from switch vendors demanded innovations in the network software. These innovations enabled simple security features to be incorporated into the newly deployed services (*e.g.,* PIN based protection in account card calling). Intelligent Networks reduced the typical innovation time scale for simple services to about three years.

In the mid 1990s, *Active Networks* allowed custom functionality to selected traffic packets traversing a device-node (Bhattacharjee et al., 1997; Tennenhouse et al., 1997; Wetherall, 1999). This customisation was performed using special code that executed at the nodes. The approach demanded more processing in the network, code-execution safety and code cross-platform portability. Several technological advances helped meet these demands, particularly, reductions in costs of computing and enhancements in programming languages. The granular control provided by Active Networks helped enable flow-level security in enterprise networks (Casado et al., 2006).

With the rapid growth of the Internet in the early 2000s, ISPs faced significant challenges, particularly in Traffic Engineering (TE) (Lakshman et al., 2004). Operators needed to control the paths of heavy network-traffic more reliably and predictably. Conventional distributed routing protocols were replaced with centralised route-control (Caesar et al., 2005; Lakshman et al., 2004; Verkaik et al., 2007).

Centralising control required a network's control plane to be decoupled from the data plane (Caesar et al., 2005; Feamster et al., 2004). The decoupling and consolidation of the control plane were characterised as Software Defined Networking (SDN) (Feamster et al., 2013).

The control-plane and data-plane decoupling in SDN was achieved via an open API between the planes (McKeown et al., 2008). This open API could be extended to suit applications beyond route control, in particular to manage the security and QoS aspects of network traffic at large scale data centres (Prakash et al., 2015; Soulé et al., 2014).

OpenFlow is an example instantiation of this API (McKeown et al., 2008; Peterson et al., 2006). OpenFlow's flow-level control allows network applications to deploy granular (*i.e.,* per-host or per-user) access control policies. Real-world deployment of OpenFlow is also seamless because its flow-level control is readily supported by existing network hardware.

Having centralised control of the network is particularly useful for security applications as it allows, for instance, policy changes to be rapidly deployed to the network in response to cyber attacks. Having network-wide visibility allows accurate predictions of network behaviour under failure conditions (Vahdat et al., 2015); a useful feature to ensure the security of systems during a planned maintenance.

SDN has revolutionised network-management and offered network operators and organisations a real choice in deploying required features in a timely manner. Software programmers can now write control software to achieve end goals without needing to acquire closed, proprietary network hardware. SDN has proved very cost effective (Vahdat et al., 2015; Feamster et al., 2013) and has enabled ideas to be pushed to implementation within weeks.

In addition to network-programming models, *policy languages* have also helped improve network and security management (Han, 2012). Business and security goals are commonly described using natural language by non-technical corporate managers and alike. These goals are often ambiguous (Liu and Gouda, 2008) and require human-translation to a machine interpretable technical specification.

Thus, policy languages were required for various disciplines; some to suit specific domains like network- or security-management and others for general purpose: *e.g.,* Ponder (Twidle et al., 2009). Network-management languages (*e.g.,* Merlin, NetKAT) help allocate resources such as bandwidth (Anderson et al., 2014; Soulé et al., 2014). Security-management languages (*e.g.,* XACML, Firmato) assist with access-control and help protect a user's privacy (OASIS, 2016; Bartal et al., 2004).

Many SDN languages also exist (*e.g.,* attire, flowlog, FML, Frenetic, HFT, Maple, Merlin, Netcore, NetKAT, Pyretic *etc.*), offering competing trends. For instance, NetKAT is a functional language that supports algebraic models (Anderson et al., 2014), Pyretic is a procedural language (Reich et al., 2013) and Merlin is logic based (Soulé et al., 2014).

Policy languages can also be device-specific, *i.e.,* low-level (Han, 2012). These languages are not user friendly: administrator's cannot specify high-level goals using them. High-level policy languages try to overcome the shortfall (Prakash et al., 2015; Bartal et al., 2004), by capturing policy intent, so, abstract goals can be specified. Typical goals include security and/or business objectives (*e.g.,* no single point of failure) (Kephart and Walsh, 2004).

Intent-based network modelling is also supported by some SDN programming languages (Cohen et al., 2013; Wijnen, 2015). Intent-based policies are high-level (*i.e.,* above network level) and are automatically refined to device-level by the underlying framework. Users can utilise these languages to program network behaviour in their service applications.

Policy-language advancements have seen another network-management paradigm develop: Policy-Defined Networking (PDN, also known as Policy Based Network Management) (Verma, 2002). PDN was developed independently and before the advent of SDN. An administrator's main task in PDN is to specify business-level goals via high-level policies (Verma, 2002). These policies are declarative and technology independent. So, a user who is non-conversant with the technology details required to deploy a policy, can still manage the network.

PDN has commercial use in many domains: *e.g.,* QoS management (Cisco Systems Inc., 1998), Internet services access-control (Tao, 2005), Web-user privacy management (Cranor et al., 2002), management of Cloud architectures (Goyal and Mikkilineni, 2009) and so on. Recent work (Cleder Machado et al., 2015), has also shown how PDN can be adapted for more flexible SDN management.

As PDN and SDN converge to provide an end-to-end solution for network security management, new challenges arise. Most notable is that despite the goals of transparency and predictability adapted in both communities, the addition of multiple layers makes policy verification more challenging; but also more vital. We discuss this below.

3 SECURITY-MANAGEMENT REQUIREMENTS

Security-management has stricter requirements than general network-management. We investigate these requirements in detail next.

3.1 Transparency

Network-programming models often include multiple layers of processing (*e.g.,* language compilation, policy refinement, etc.). These layers decrease transparency. 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, Microsoft Windows OS depicts storage of files in a hierarchical directory structure (Hall, 2003) 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 blocks the ability to locate and rectify an incorrect decision. In mission critical SCADA networks, this inability can lead to security malfunction or failure resulting in catastrophic outcomes including potential loss of human lives! Transparency can be provided for the processing layers through layer-wise verification of the processing. The more transparent, the more confidence security administrators will have of deploying and maintaining correct security policies in their networks.

3.2 Human-Comprehensible Policy

The ability to author a security policy correctly, largely depends on (i) how easy it is express the intentions of the authors in a policy language; and reciprocally (ii) the ability to clearly understand *what exactly* a specified policy aims to achieve. A security-policy language therefore, should be easily *human readable and writable* to say the least.

But surprisingly, even this fundamental requirement is lacking in some popular security-policy languages. 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 (OASIS, 2016). 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 (McKendrick, 2006). Thus, it is difficult for XACML policy authors to precisely know, for instance, if a security policy blocks potentially-unsafe HTTP traffic from entering a protected SCADA Zone or not. HTTP is known to carry attacks and worms, and if permitted inbound, could render fatal outcomes in SCADA networks (Byres et al., 2005).

Even simple XACML access-control policies can span hundreds of lines of code or more (Han, 2012). Keeping track of the myriad interactions within a policy manually is nearly impossible. A useful policy language should therefore, allow *concise* description of policies, in addition to being user friendly. In essence, human-comprehensibility in a policy is essential to easily verify that it captures the original intentions correctly.

3.3 Specialisation within Networking

Reliable security management also demands specialisation in networking. Division of labour is one of the fundamental hacks 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 IT specialists, but in reality, they are just better paid network engineers. For instance, consider how network infrastructure manufacturers (*e.g.,* Cisco) structure their certification programs. It is assumed, that a network architect will know as much, or more about devices than the person programming the devices. Decoupling policy from implementation also creates opportunities for specialisation.

Network-specialisation also means people can verify their work more easily. For one, network architects could easily check that their designed security policies meet high-level goals, in the absence of implementation-centric details. For another, network engineers only need to verify that their policy implementations match the network-architects' designs and not high-level business goals.

3.4 Verifiability

The above requirements indicate how *verifiability* is an important property in security-management. There is clear danger in assuming any software component functions correctly, from the network-devices up to and including the security-management platform. We need to provably check that the policies are specified and implemented correctly in the network at every level possible. We discuss these various dimensions of verifiability in detail next.

4 WHAT SHOULD WE VERIFY?

Policies play a key role in security management and it would be sensible therefore, to begin to understand what aspects of a policy needs to be verified for reliable security-management.

4.1 Verify Policy is Correct

Inconsistencies such as conflicts and redundancies can often occur in a policy (Prakash et al., 2015; Soulé et al., 2014). Conflicts indicate an *ill-defined* policy and redundancies imply an *inefficient* one. These inconsistencies can stem from ambiguities in the high-level policy goals and their error-prone (manual) translation to technical specification. For instance, it's easy for a policy author to accidentally leave in two conflicting access-control rules inside a complex security policy containing thousands of rules.

The problem is exacerbated by the need for policies to be administered by users from multiple subdomains (*e.g.,* SCADA engineers, Corporate admins), particularly in large distributed environments. Commercial network-management tools often lack the ability to compose distributed policies automatically (Han, 2012; Prakash et al., 2015). So, users must *manually* check with policy sub-domains for conflicts to maintain consistency.

Inconsistencies introduced by policy composition can produce unintended consequences (Wool, 2004). Correct policy deployment therefore, demands on the ability to check for policy inconsistencies accurately.

Policy inconsistency checking partly verifies it's 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. Doing so, is critical in SCADA networks where more restrictive practices are required to prevent serious injury of people, or even death!

Best-practice compliance checks are also only meaningful if we compare *policy intent* not implementation. How can one compare a SCADA security policy with a generic ISA-recommended policy if either is inter-twined with implementation details?

But, not all policies capture intent. Such policies need to be checked against organisational intent, supplementary to standards. Deriving organisational intent is usually a non-trivial manual task that requires conducting a risk assessment to identify the threats, vulnerabilities and impacts (Stoneburner et al., 2002) as well as appropriate risk mitigation strategies. Once the intended policy is derived, a deployed policy can be validated against it.

Existing security-management tools can comprehensively check policy syntactic correctness, but not semantic correctness. And there have been major incidents: *e.g.,* the hacking of a German steel mill (BBC, 2014), as a result.

Policy languages that do not capture high-level intent (*e.g.,* Cisco CLI) are often tightly coupled with the underlying network (Cisco Systems Inc., 2010). Using these low-level languages is similar to programming with Assembler in the 1970s, where users need to specify, for instance, hardware addresses in order to communicate with devices. Naturally, errors are common in Assembler programs (Endres, 1975).

High-level policy languages attempt to overcome these shortfalls and are analogous to present-day programming languages used by software developers. These programming languages support features such as modularisation and other constructs that automate common tasks. They 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.

To allow cross-policy comparisons, security policy languages should allow policies to be specified abstractly, flexibly enough and in detail. In every sense of the word these policies must be high-level: *i.e.,* they *must* be decoupled from the network and its implementation, separating the *what* from the *how*.

4.2 Verify Policy is Compatible with Network and Technology

A policy may not always be compatible with 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 security-policy creator for instance, may use the Zone-Conduit abstraction (ANSI/ISA-62443-1-1, 2007) 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 operators create out-of-policy connects, *e.g.,* to support wireless, but again, effectively fuses 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 (Byres et al., 2005).

Thus, a policy must be checkable against both the underlying network and its technology for compatibility, prior to deployment. A user could then swiftly rectify an incompatible policy or update the target network or technology, in preparation for deployment.

4.3 Verify Expected Security Outcome Pre-Deployment

Policy-creator oversights can also cause a policy to be flawed (Wool, 2004). 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 would prefer to avoid network disruptions in the interim. Checking a policy for oversights prior to deployment also needs to be made *compulsory* and *automated*. So, a user would be unable to bypass the verification step.

4.4 Verify Expected Security Outcome Post-Deployment

Pre-deployment verification doesn't necessarily guarantee expected policy outcome in the real network. For instance, unintended security policy behaviour can still occur due to firewall software bugs (Stouffer et al., 2008). So, in security management it is equally necessary to be able to verify that the real network operates as intended, post-deployment.

A policy may also function correctly in the real network at first, but produce unexpected behaviour later. For instance, following the upgrade and/or patching of network firewalls (Stouffer et al., 2008). Thus, pragmatic security-management also demands on the ability to continuously monitor the security mechanisms of a network post-deployment.

We have described so far, verifiability requirements of security management. Next, we show how these requirements can be implemented concretely in our proposed network-programming paradigm.

5 FORMALLY-VERIFIABLE PDN

SDN and PDN together support high-level policies decoupled from network implementation. They both build on centralised network control: a feature that helps deliver new functionality more rapidly and drive capital and operational expenditure savings (Feamster et al., 2013). A key difference between the two is that SDN requires the use of a standard network API (*e.g.,* OpenFlow) to manage heterogenous network devices, while PDN does not.

This requirement in SDN can be too restrictive for traditional networks: *e.g.,* deploying OpenFlow capable devices is a distant reality in mission critical SCADA networks where TCP is a recent innovation! High availability requirements in these control networks (Byres et al., 2005) mean the legacy network equipment employed within them is unlikely to be upgraded except during a major overhaul of a plant, which might happen once in a decade (if that often).

PDN doesn't require an open network API (*e.g.,* OpenFlow), thus, heterogeneous devices in traditional networks can be managed through support for proprietary configuration languages (*e.g.,* by use of templates) (Verma, 2002). Administrators don't require learning these arcane languages; the PDN engine automatically invokes them when refining policies.

With this added flexibility, PDN can deliver the benefits of SDN, while overcoming SDN shortfalls (*e.g.,* lack of scalability). But, like SDN, traditional PDN lacks built-in verifiability capabilities in §4.

Precisely checking a policy is correct is a non-trivial task. But, progress can be made towards the goal using formal policy language semantics. SDN programming languages: *e.g.,* NetKAT, Pyretic (Anderson et al., 2014; Reich et al., 2013) and PDN policy languages: *e.g.,* Rei, ASL (Kagal, 2002; Jajodia and Samarati, 1997), often include semantics to facilitate this checking. Additionally in FV-PDN, formalisms would also be included to allow automated-reasoning about policies: *e.g.,* to precisely compare a security policy with a best-practice policy for compliance. Such a reasoning framework would involve deriving canonical representations of the security policies (Ranathunga et al., 2016) and developing algebras for accurate policy comparisons.

A correct high-level policy should in theory refine down to correct network- and device-level policy. But, we simply cannot 'trust' FV-PDN to always correctly refine high-level policy. So, we must additionally validate that the refined policies are likewise correct. Alloy (Jackson, 2011) is an example formal verification tool that can be employed to perform this validation. But as a pre-requisite, the refined policies need to be converted to an Alloy compatible format.

A high-level policy can be checked against the underlying network for compatibility by validating the abstract network model perceived by the policy creator with that of the physical network. For instance, with Zone-Conduit policies, FV-PDN could derive and compare the Zone-Conduit model of the actual network against that the policy is built on, for a match.

It is also possible to check that a policy can be supported by a network's technology capabilities, during policy refinement (Verma, 2002). So, FV-PDN could for instance, validate an application-layer filtering policy against the DPI firewalls available in the network, when refining the policy to network-level. If the available DPI firewalls cannot implement the policy correctly, an error is raised.

FV-PDN would also have pre-deployment verification built-in. A simple, cost effective way to do this is by using an emulated network together with automated pathological traffic tests. Emulation tools have been used in SDN to test prototype applications (Gupta et al., 2013) and in PDN to test policies (Vacante and Houck, 2003). But, these tools are standalone, so, prototypes and policies need to be hand deployed and tested using pathological-traffic. The manual tedium leads users to bypass this step.

In FV-PDN, we would make this step *compulsory* and *automated*. For instance, FV-PDN could use the Netkit open source software package (Knight et al., 2013) that provides an emulation platform for traditional networks with virtual devices and interconnections via User Mode Linux. Pathological traffic tests can be automatically conducted, for instance, using test scripts generated in Expect– a UNIX scripting and testing utility (Libes, 1995). These automated tests can check for firewall-policy oversights and verify that a policy only allows the expected traffic services through a firewall (*i.e.,* positive vetting) and blocks all other services (*i.e.,* negative vetting).

A simple way to automate post-deployment verification is by extending the test scripts from the emulated network to the real network. Post-deployment network behaviour can also be monitored using the various network-device reports available (*i.e.,* logs, alerts, traps etc.) (Ranathunga et al., 2015a). For instance, a network's security mechanisms can be monitored using the diverse firewall reports generated. A firewall-reporting framework proposed (Ranathunga et al., 2015a) shows that reporting policy must be *coupled* with security policy, to be useful. Otherwise, one could deploy policies that are not verified. FV-PDN would enforce this coupling automatically.

Next, we consider the challenges that need to be overcome to achieve the verifiability requirements in §4.

6 FV-PDN CHALLENGES

Achieving verifiability in FV-PDN has challenges

What policies do we need to specify, at what level of detail? We need a clear understanding of the types of security policies that must be supported and the level of policy-detail needed, to formulate methods to accurately check policy correctness. A comprehensive study of the various security use cases (*e.g.,* traffic filtering, access-control *etc*.) would help identify the different policy types. Analysis of policy granularity (*i.e.,* level of policy detail) with respect to dimensions such as network, service, time *etc*. (Ranathunga et al., 2015a), is needed to understand the level of detail required in policy specification.

What policy abstractions should we choose to assist verifiability? The choice of security abstraction in a security policy language can also assist or hinder verifiability. For example, firewall access-control policies can be described using the Zone-Conduit abstraction (ANSI/ISA-62443-1-1, 2007) or the RBAC (Role Based Access Control) abstraction (Bartal et al., 2004). The first abstraction helps to check these policies are best-practice compliant (Ranathunga et al., 2015b), whereas the latter can make the task difficult.

How should we manage conflicts? The ability to verify a policy is correct and consistent also depends on managing conflicts reliably and efficiently. The type of conflicts that arise in a policy are context dependent. For instance, a policy language that employs an ABAC (Attribute Based Access-Control) abstraction, will produce two primary conflicts: (1) conflict between permission and prohibition; and (2) conflict between obligation and dispensation (OASIS, 2016).

The nature of the conflicts determine the management scheme(s) required (Di Vimercati et al., 2007b; Di Vimercati et al., 2007a; Strassner and Schleimer, 1998). For instance, the conflicts above can be managed by assigning explicit priorities to policies or by overriding positive policies with negative ones. But, assigning priorities is *infeasible* for large distributed environments where distinct policy-domain users could specify policies. Thus, conflict-management is a balancing act: abstractions must be chosen to reduce the possible conflict types to a manageable few; and conflict-management schemes must be developed being mindful of their limitations.

7 CONCLUSIONS

The inability to verify security policies comprehensively makes it difficult to provide assurance that the expected security outcome is consistent pre- and post-deployment. We propose Formally-Verifiable Policy-Defined Networking (FV-PDN) as a new management paradigm to allows users to comprehensively check a policy is correct, consistent and compatible with the underlying network and its technology. We illustrate FV-PDNs use in reliable security management by considering the administration of firewall policies in mission critical SCADA networks.

ACKNOWLEDGEMENTS

This project was supported by an Australian Postgraduate Award, Australian Research Council Linkage Grant LP100200493, and CQR Consulting.

REFERENCES

Anderson, C. J., Foster, N., Guha, A., Jeannin, J.-B., Kozen, D., Schlesinger, C., and Walker, D. (2014). NetKAT: Semantic foundations for networks. *ACM SIGPLAN Notices*, 49(1):113–126.

ANSI/ISA-62443-1-1 (2007). Security for industrial automation and control systems part 1-1: Terminology, concepts, and models.

Bartal, Y., Mayer, A., Nissim, K., and Wool, A. (2004). Firmato: A novel firewall management toolkit. *ACM TOCS*, 22(4):381–420.

BBC (2014). Hack attack causes 'massive damage' at steel works, `http://www.bbc.com/news/technology-30575104`.

Bhattacharjee, S., Calvert, K. L., and Zegura, E. W. (1997). An architecture for Active Networking. In *High Performance Networking VII*, pages 265–279. Springer.

Byres, E., Karsch, J., and Carter, J. (2005). Good practice guide on firewall deployment for SCADA and process control networks. *NISCC*.

Caesar, M., Caldwell, D., Feamster, N., Rexford, J., Shaikh, A., and van der Merwe, J. (2005). Design and implementation of a routing control platform. In *NSDI*, pages 15–28. USENIX Association.

Casado, M., Garfinkel, T., Akella, A., Freedman, M. J., Boneh, D., McKeown, N., and Shenker, S. (2006). SANE: A protection architecture for enterprise networks. In *Usenix Security*.

Cisco Systems Inc. (1998). Delivering end-to-end security in policy based systems. Tech report.

Cisco Systems Inc. (2010). *Cisco ASA 5500 Series Configuration Guide using the CLI*.

Cleder Machado, C., Araujo Wickboldt, J., Zambenedetti Granville, L., and Schaeffer-Filho, A. (2015). Policy authoring for software-defined networking management. In *IEEE IM*, pages 216–224.

Cohen, R., Barabash, K., Rochwerger, B., Schour, L., Crisan, D., Birke, R., Minkenberg, C., Gusat, M., Recio, R., and Jain, V. (2013). An intent-based approach for network virtualization. In *IEEE IM*, pages 42–50.

Coots, J. F. and Gillespie, H. (1934). *Santa Claus is Comin' to Town*. Leo Feist Pub 6752-4, NY.

Cranor, L., Langheinrich, M., Marchiori, M., Presler-Marshall, M., and Reagle, J. (2002). The platform for privacy preferences (P3P1. 0) specification. *W3C*.

Di Vimercati, S., Foresti, S., Jajodia, S., and Samarati, P. (2007a). Access control policies and languages in open environments. In *Secure Data Management in Decentralized Systems*, pages 21–58. Springer.

Di Vimercati, S., Foresti, S., Samarati, P., and Jajodia, S. (2007b). Access control policies and languages. *IJCSE*, 3(2):94–102.

Endres, A. (1975). An analysis of errors and their causes in system programs. In *ACM SIGPLAN Notices*, volume 10, pages 327–336.

Feamster, N., Balakrishnan, H., Rexford, J., Shaikh, A., and Van Der Merwe, J. (2004). The case for separating routing from routers. In *FDNA*, pages 5–12. ACM.

Feamster, N., Rexford, J., and Zegura, E. (2013). The road to SDN. *Queue*.

Goyal, P. and Mikkilineni, R. (2009). Policy-based event-driven services-oriented architecture for cloud services operation & management. In *IEEE CLOUD*, pages 135–138.

Gupta, M., Sommers, J., and Barford, P. (2013). Fast, accurate simulation for SDN prototyping. In *HotSDN*, pages 31–36. ACM.

Hall, M. (2003). Understanding the file system architecture in Windows CE .NET, `https://msdn.microsoft.com/en-au/library/aa459155.aspx`.

Han, W. (2012). A survey on policy languages in network and security management. pages 477–489. Elsevier.

Jackson, D. (2011). *Software Abstractions: Logic, Language, and Analysis*. The MIT Press.

Jajodia, S. and Samarati, P. (1997). A logical language for expressing authorizations. In *IEEE S&P, pages 31-42*.

Kagal, L. (2002). Rei: a policy language for the Me-Centric project. *HP Labs*.

Kephart, J. O. and Walsh, W. E. (2004). An artificial intelligence perspective on autonomic computing policies. In *PDSN*, pages 3–12. IEEE.

Knight, S., Nguyen, H., Maennel, O., Phillips, I., Falkner, N., Bush, R., and Roughan, M. (2013). An automated system for emulated network experimentation. In *ACM CoNEXT*, pages 235–246.

Lakshman, T., Nandagopal, T., Ramjee, R., and Woo, T. (2004). The softrouter architecture. *ACM HotNets*.

Libes, D. (1995). *Exploring Expect: A Tcl-based toolkit for automating interactive programs*. O'Reilly.

Liu, A. X. and Gouda, M. G. (2008). Diverse firewall design. *IEEE TPDS*, pages 1237–1251.

Magedanz, T. and Popescu-Zeletin, R. (1996). *Intelligent Networks: Basic Technology, Standards and Evolution*. Thompson Computer Press.

McKendrick, J. (2006). Another view:XML not meant to be human readable, `http://tinyurl.com/hytddnt`.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM CCR*, 38(2):69–74.

OASIS (2016). OASIS Extensible Access Control Markup Language (XACML) version 3.0.

Peterson, L., Anderson, T., Blumenthal, D., et al. (2006). Geni: global environment for network innovations.

Prakash, C., Lee, J., Turner, Y., Kang, J.-M., Akella, A., Clark, C., Ma, Y., and Sharma, P. (2015). PGA: Using graphs to express and automatically reconcile network policies. In *ACM SIGCOMM, pages 29-42*.

Ranathunga, D., Roughan, M., Kernick, P., and Falkner, N. (2015a). Towards standardising firewall reporting. In *WOS-CPS*. Springer LNCS.

Ranathunga, D., Roughan, M., Kernick, P., and Falkner, N. (2016). Malachite: Firewall policy comparison. In *ISCC*. IEEE.

Ranathunga, D., Roughan, M., Kernick, P., Falkner, N., and Nguyen, H. (2015b). Identifying the missing aspects of the ANSI/ISA best practices for security policy. In *CPSS*, pages 37–48. ACM.

Reich, J., Monsanto, C., Foster, N., Rexford, J., and Walker, D. (2013). Modular SDN programming with Pyretic. *Technical Report of USENIX*.

Rubin, A. D. and Geer, D. E. (1998). A survey of Web security. *IEEE Computer*, pages 34–41.

Soulé, R., Basu, S., Marandi, P. J., Pedone, F., Kleinberg, R., Sirer, E. G., and Foster, N. (2014). Merlin: A language for provisioning network resources. In *ACM CoNEXT*, pages 213–226.

Stoneburner, G., Goguen, A. Y., and Feringa, A. (2002). Guide to Industrial Control Systems (ICS) security. *NIST Special Publication*, 800(30):16–16.

Stouffer, K., Falco, J., and Scarfone, K. (2008). Guide to Industrial Control Systems (ICS) security. *NIST Special Publication*, 800(82):16–16.

Strassner, J. and Schleimer, S. (1998). Policy framework definition language. *Internet Draft, IETF*.

Tao, H. (2005). A XACML-based access control model for Web service. In *WiCOM*, pages 1140–1144. IEEE.

Tennenhouse, D. L., Smith, J. M., Wetherall, D. J., and Minden, G. J. (1997). A survey of active network research. *Communications Magazine, IEEE*, (1):80–86.

Twidle, K., Dulay, N., Lupu, E., and Sloman, M. (2009). Ponder2: A policy system for autonomous pervasive environments. In *ICAS'09*, pages 330–335.

Vacante, R. C. and Houck, P. T. (2003). Testing of policy prior to deployment in a policy-based network management system. US Patent 6,651,191.

Vahdat, A., Clark, D., and Rexford, J. (2015). A purpose-built global network: Google's move to SDN. *Queue*.

Verkaik, P., Pei, D., Scholl, T., Shaikh, A., Snoeren, A. C., and Van Der Merwe, J. E. (2007). Wresting control from BGP: Scalable fine-grained route control. In *USENIX ATC*, pages 295–308.

Verma, D. C. (2002). Simplifying network administration using policy-based management. *IEEE Network*, 16(2):20–26.

Wetherall, D. (1999). ANTS: network services without the red tape. *IEEE Computer, pages 42-48*.

Wijnen, B. (2015). Intent Based Network Modeling (IBNEMO), `http://tinyurl.com/h95ecfl`.

Wool, A. (2004). A quantitative study of firewall configuration errors. *IEEE Computer*, 37(6):62–67.