Directed Security Policies: A Stateful Network Implementation

Cornelius Diekmann[†] Lars Hupel[‡] Georg Carle[†]

Technische Universität München

[†]Chair for Network Architectures and Services [‡]Chair for Logic and Verification

diekmann@net.in.tum.de lars.hupel@tum.de carle@in.tum.de

Large systems are commonly internetworked. A security policy describes the communication relationship between the networked entities. The security policy defines rules, for example that A can connect to B, which results in a directed graph. However, this policy is often implemented in the network, for example by firewalls, such that A can establish a connection to B and all packets belonging to established connections are allowed. This stateful implementation is usually required for the network's functionality, but it introduces the backflow from B to A, which might contradict the security policy. We derive compliance criteria for a policy and its stateful implementation. In particular, we provide a criterion to verify the lack of side effects in linear time. Algorithms to automatically construct a stateful implementation of security policy rules are presented, which narrows the gap between formalization and real-world implementation. The solution scales to large networks, which is confirmed by a large real-world case study. Its correctness is guaranteed by the Isabelle/HOL theorem prover.

1 Introduction

Large systems with high requirements for security and reliability, such as SCADA or enterprise landscapes, no longer exist in isolation but are internetworked [10]. Uncontrolled information leakage and access control violations may cause severe financial loss – as demonstrated by Stuxnet – and may even harm people if critical infrastructure is attacked. Hence, network security is crucial for system security.

A central task of a network security policy is to define the network's desired connectivity structure and hence decrease its attack surface against access control breaches and information leakage. A security policy defines, among others things, rules determining which host is allowed to communicate with which other hosts. One of the most prominent security mechanisms to enforce a policy are network firewalls. For adequate protection by a firewall, its rule set is critical [3, 2]. For example, let A and B be sets of networked hosts identified by their IP addresses. Let $A \rightarrow B$ denote a policy rule describing that A is allowed to communicate with B. Several solutions from the fields of formal testing [4] to formal verification [25] can guarantee that a firewall actually implements the policy $A \rightarrow B$. However, to the best of our knowledge, one subtlety between firewall rules and policy rules remains unsolved: For different scenarios, there are diverging means with different protection for translating $A \rightarrow B$ to firewall rules.

Scenario 1 Let A be a workstation in some local network and B represent the hosts in the Internet. The policy rule $A \rightarrow B$ can be justified as follows: The workstation can access the Internet, but the hosts in the Internet cannot access the workstation, i.e. the workstation is protected from attacks from the Internet. This policy can be translated to e.g. the Linux iptables firewall [18] as illustrated in Figure 1. The first rule allows A to establish a new connection to B. The second rule allows any communication

Submitted to:
ESSS'14

```
iptables -A INPUT -s A -d B -m conntrack --ctstate NEW -j ACCEPT
iptables -A INPUT -m conntrack --ctstate ESTABLISHED -j ACCEPT
iptables -A INPUT -j DROP
```

Figure 1: Stateful implementation of $A \rightarrow B$ in Scenario 1

```
iptables -A INPUT -s A -d B -j ACCEPT
iptables -A INPUT -j DROP
```

Figure 2: Stateless implementation of $A \rightarrow B$ in Scenario 2

over established connections in both directions, a very common practice. For example, A can request a website and the answer is transmitted back to A over the established connection. Finally, the last rule drops all other packets. In particular, no one can establish a connection to A; hence A is protected from malicious accesses from the Internet.

Scenario 2 In a different scenario, the same policy rule $A \rightarrow B$ has to be translated to a completely different set of firewall rules. Assume that A is a smart meter recording electrical energy consumption data, which is in turn sent to the provider's billing gateway B. There, smart meter records of many customers are collected. That data must not flow back to any customer, as this could be a violation of other customers' privacy. For example, under the assumption that B sends packets back to A, a malicious customer could try to infer the energy consumption records of their neighbors with a timing attack. In Germany, the requirement for unidirectional communication of smart meters is even standardized by a federal government agency [6]. The corresponding firewall rules for this scenario can be written down as shown in Figure 2. The first rule allows packets from A to B, whereas the second rule discards all other packets. No connection state is established; hence no packets can be sent from B to A.

These two firewall rule sets were created from the same security policy rule $A \rightarrow B$. The first implementation of "$\rightarrow$" is "can initiate connections to", whereas the second implementation is "can send packets to". The second implementation appears to be simpler and more secure, and the firewall rules are justifiable more easily by the policy. However, this firewall configuration is undesirable in many scenarios as it might affect the desired functionality of the network. For example, surfing the web is not possible as no responses (i.e. websites) can be transferred back to the requesting host.

A decision must be made whether to implement a policy rule $A \rightarrow B$ in the stateful (Figure 1) or in the stateless fashion (Figure 2). The stateful fashion bears the risk of undesired side effects by allowing packet flows that are opposite to the security policy rule. In particular, this could introduce information leakage. On the other hand, the stateless fashion might impair the network's functionality. Hence, stateful flows are preferable for network operation, but are undesirable with regard to security. In this paper, we tackle this problem by maximizing the number of policy rules that can be made stateful without introducing security issues.

We can see that even if a well-specified security policy exists, its implementation by a firewall configuration remains a manual and hence error-prone task. A 2012 survey [20] of 57 enterprise network administrators confirms that a "majority of administrators stated misconfiguration as the most common cause of failure" [20]. A study [22] conducted by Verizon from 2004 to 2009 and the United States Secret Service during 2008 and 2009 reveals that data leaks are often caused by configuration errors [12].

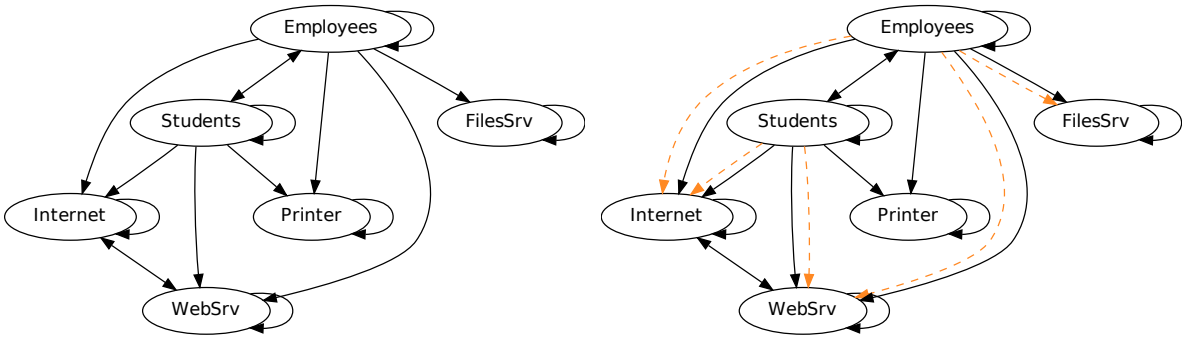
In this paper, we answer the following questions:

- What conditions can be checked to verify that a stateful policy implementation complies with the directed network security policy rules?
- When can a policy rule $A \rightarrow B$ be upgraded to allow a stateful connection between A and B?

Our results apply not only to firewalls but to any network security mechanisms that shape network connectivity.

The outline of this paper is as follows. Section 2 presents a guiding example. Section 3 formalizes the key concepts of directed policies, security requirements, and stateful policies. Section 4 discusses the requirements for a stateful policy to comply with a directed policy. Section 5 presents an algorithm to automatically derive a stateful policy. Sections 6 and 7 evaluate our work: Section 6 discusses the computational complexity of the algorithm, and Section 7 presents a large real-world case study.

2 Example



(a) Network security policy (b) Stateful implementation

Figure 3: The network security policy and its stateful implementation

We introduce a network – for a hypothetical university department – to illustrate the problem with a complete example and outline the solution before we describe its formalization in the next section.

The network (depicted in Figure 3) consists of the following participants: the students, the employees, a printer, a file server, a web server, and the Internet. The network security policy rules are depicted in Figure 3a as a directed graph. A security policy rule $A \to B$ is denoted by an edge from A to B. The security policy is designed to fulfill the following security invariants.

Access Control Invariants The printer is only accessible by the employees and students; more formally, $\mathit{employees} \to \mathit{printer}$ and $\mathit{students} \to \mathit{printer}$. The file server is only accessible by employees, formally $\mathit{employees} \to \mathit{fileSrv}$. The students and the employees are in a joint subnet that allows collaboration between them but protects against accesses from e. g. the Internet or a compromised web or file server.

Information Flow Invariants The file server stores confidential data that must not leak to untrusted parties. Only the employees have the necessary security clearance to receive data from the file server. The employees are also trustworthy, i. e. they may declassify and reveal any data received by the file server. The printer is an information sink. Confidential data (such as an exam) might be printed by an employee. No other network participants, in particular no students, are allowed to retrieve any information from the printer that might allow them to draw conclusions about the printed documents. This can be formalized by "$* \to \mathit{printer}$" and "$\mathit{printer} \nrightarrow *$".

Stateful Policy Implementation

Considering Figure 3a, it is desirable to allow stateful connections from the employees and students to the Internet and the web server. Figure 3b depicts the stateful policy implementation, where the additional dashed edges represent flows that are allowed to be stateful, i.e. answers in the opposite direction are allowed. Only strict stateless unidirectional communication with the printer is necessary. The students and employees can, as already defined by the policy, freely interact with each other. Hence stateful semantics are not necessary for these flows.

In this paper, we specify conditions to verify that the stateful policy implementation (e.g. Figure 3b) complies with the directed security policy (e.g. Figure 3a). We present an efficiently computable condition and formally prove that it implies several complex compliance conditions. Finally, we present an algorithm that automatically computes a stateful policy from the directed policy and the security invariants. We formally prove the algorithm's correctness and that it can always compute a maximal possible set of stateful flows with regard to access control and information flow security strategies.

3 Formal Model

We implement our theory and formal proofs in the Isabelle/HOL theorem prover [15]. It is based on a small inference kernel. All proof steps, done by either the user or by the (embedded or external) automated proof tactics and solvers, must pass this kernel. The correctness of Isabelle/HOL proofs therefore only depends on the correctness of the kernel. This architecture makes the system highly trustworthy, because the proof kernel consists only of little code, is widely used (and has been for over a decade) and is rigorously manually checked. In this paper, all proofs are verified by Isabelle/HOL. The corresponding theory files are publicly available (c.f. Section Acknowledgements & Availability).

The following notations are used in this paper. A total function from $\mathscr{A}$ to $\mathscr{B}$ is denoted by $\mathscr{A} \Rightarrow \mathscr{B}$. A logical implication is written with a long arrow "$\Longrightarrow$". Function application is written without parentheses: $f\ x\ y$ means "f applied to x and y". The set of Boolean values is denoted by the symbol $\mathbb{B}$.

For readability, we only present the intuition behind proofs or even omit the proof completely. Whenever we omit a proof, we add an endnote that points to our formalization. We also add endnotes into the text which can be used to jump directly from a definition in this paper to the definition in the theory files. The endnotes are referenced by roman marks. For example, if the paper states "note[iv] that A is equal to B", then the corresponding formal, machine-verified proof can be found by following [iv].

Network Security Policy Rules We represent the network security policy's access rules as directed graph $G = (V, E)$. The type of all graphs is denoted by $\mathscr{G}$. For example, the policy that only consists of the rule that A can send to B, denoted by $A \to B$, is represented by the graph $G = (\{A, B\},\ \{(A, B)\})$. An edge in the graph corresponds to a permitted flow in the network. We call this policy a *directed policy*. In § 3.1, we will introduce the notion of a stateful policy.

We consider only syntactically valid graphs. A graph is *syntactically valid*[i] if all nodes in the edges are also listed in the set of vertices. In addition, since we represent finite networks, we require that V is a finite set. This does not prevent creating nodes that represent collections of arbitrary many hosts, e.g. the node *Internet* in Figure 3a represents arbitrarily many hosts.

Network Security Invariants A security invariant m specifies whether a given policy G fulfills its security requirements. As we focus on the network security policy's access rules which specify which

hosts are allowed to communicate with which other hosts, we do not take availability or resilience requirements into account. Instead, we deal with only the traditional security invariants that follow the principle "prohibiting more is more or equally secure". We call this principle *monotonicity*. To allow arbitrary network security invariants, almost any total function m of type $\mathscr{G} \Rightarrow \mathbb{B}$ can be used to specify a network security requirement.

This model landscape is based on the formal model by Diekmann [7]. We distinguish between the two security strategies that m is set to fulfill: *Information flow security strategies* (IFS) prevent data leakage; *Access control strategies* (ACS) are used to prevent illegal or unauthorized accesses.

Definition 1 (Security Invariant) A network security invariant m is a total function $\mathscr{G} \Rightarrow \mathbb{B}$ with a security strategy (either IFS or ACS) satisfying the following conditions:

- If no communication exists in the network, the security invariant must be fulfilled: $m\ (V,\ \emptyset)$
- Monotonicity: $m\ (V,E)\ \wedge\ E' \subseteq E \Longrightarrow m\ (V,E')$

If there is a security violation for m in G, there must be at least one set $F \subseteq E$ such that the security violation can be remedied by removing F from E.[1] We call F offending flows. F is *minimal* if all flows $(s,r) \in F$ contribute to the security violation. For m, the set of all minimal offending flows can be defined. The definition *offending_flows* describes a set of sets, containing all minimal candidates for F.

$$\mathit{offending\_flows}\ m\ G = \left\{F \subseteq E \mid \neg m\ G\ \wedge\ m\ (V,\ E \setminus F)\ \ \wedge\ \forall (s,r) \in F.\ \neg m\ (V,\ (E \setminus F) \cup \{(s,r)\})\right\}$$

The offending flows inherit m's monotonicity property. The full proof can be found in our formalization.[ii]

Lemma 1 (Monotonicity of Offending Flows)

$$E' \subseteq E \Longrightarrow \bigcup \mathit{offending\_flows}\ m\ (V,E') \subseteq \bigcup \mathit{offending\_flows}\ m\ (V,E)$$

If there is an upper bound for the offending flows, it can be narrowed.[iii]

Lemma 2 (Narrowed Upper Bound of Offending Flows) *Let E' be a set of edges. If the offending flows are bounded, i.e. if $\bigcup$offending_flows $m\ (V,E) \subseteq X$ holds, then $\bigcup$offending_flows $m\ (V,E \setminus E') \subseteq X \setminus E'$.*

Proof. From Lemma 1, we have $\bigcup \mathit{offending\_flows}\ m\ (V,E \setminus E') \subseteq \bigcup \mathit{offending\_flows}\ m\ (V,E)$. This implies that $(\bigcup \mathit{offending\_flows}\ m\ (V,E \setminus E')) \setminus E' \subseteq (\bigcup \mathit{offending\_flows}\ m\ (V,E)) \setminus E'$. Since the set of offending flows only returns subsets of the graph's edges, the left hand side can be simplified: $\bigcup \mathit{offending\_flows}\ m\ (V,E \setminus E') \subseteq (\bigcup \mathit{offending\_flows}\ m\ (V,E)) \setminus E'$. From the assumption, it follows that $(\bigcup \mathit{offending\_flows}\ m\ (V,E)) \setminus E' \subseteq X \setminus E'$. We finally obtain

$$\bigcup \mathit{offending\_flows}\ m\ (V,E \setminus E') \subseteq \left(\bigcup \mathit{offending\_flows}\ m\ (V,E)\right) \setminus E' \subseteq X \setminus E'$$

by transitivity. □

Definition 2 (Security Invariants) We call a finite list of security invariants $M = [m_1, m_2, ..., m_k]$ a network's security invariants. The functions *getIFS M* (and *getACS M*) return all $m \in M$ with an IFS (and ACS, respectively) security strategy. Additionally, we abbreviate all sets of offending flows for all security invariants with $\mathit{get\_offending\_flows}\ M\ G = \bigcup_{m \in M} \mathit{offending\_flows}\ m\ G$. Similarly to *offending_flows*, it denotes a set of sets.

[1] Since $m\ (V,\ \emptyset)$, it is obvious that such a set always exists.

3.1 Stateful Policy Implementation

We define a stateful policy similarly to a directed policy.

Definition 3 (Stateful Policy) A stateful policy $T = (V, E_\tau, E_\sigma)$ is a triple consisting of the networked hosts V, the flows E_τ, and the stateful flows $E_\sigma \subseteq E_\tau$.

The meaning of E_σ is that these flows are allowed to be stateful. We consider the stateful flows E_σ as "upgraded" flows, hence $E_\sigma \subseteq E_\tau$. This means that if $(s,r) \in E_\sigma$, flows in the opposite direction, i.e. (r,s) may exist. For a set of edges X, we define the *backflows* of X as $\overleftarrow{X} = \{(r,s) \mid (s,r) \in X\}$. Hence, the semantics of E_σ can be described as that both the flows E_σ and $\overleftarrow{E_\sigma}$ may exist. We define a mapping that translates a stateful policy T to a directed policy G as $\alpha\ T = (V, E_\tau \cup E_\sigma \cup \overleftarrow{E_\sigma})$.

Example The ultimate goal is to translate a directed policy $G = (V, E)$ to a stateful implementation $T = (V, E_\tau, E_\sigma)$ that contains as many stateful flows E_σ as possible without introducing security flaws. The trivial choice is $T_{\text{triv}} = (V, E, \emptyset)$. It fulfills all security invariants because $\alpha\ T_{\text{triv}} = G$. Since $E_\sigma = \emptyset$, it does not maximize the stateful flows.

Before discussing requirements for the compliance of T and G, we first have to define the requirements for a *syntactically valid* stateful security policy.[iv] All nodes mentioned in E_τ and E_σ must be listed in V. The flows E_τ must be allowed by the directed policy, hence $E_\tau \subseteq E$, which also implies $E_\sigma \subseteq E$ by transitivity. The nodes in T are equal to the nodes in G. This implies that E_τ and E_σ are finite[v]. In the rest of this paper, we always assume that T is syntactically valid.

From these conditions, we conclude that T and G are similar and T syntactically introduces neither new hosts nor flows. Semantically, however, $\alpha\ T$ adds $\overleftarrow{E_\sigma}$, which might introduce new flows. Hence, the edges of $\alpha\ T$ need not be a subset of G's edges (nor vice versa).

4 Requirements for Stateful Policy Implementation

We assume that G is a *valid policy*. In addition to being syntactically valid, that means that all security invariants must be fulfilled, i.e. $\forall m \in M.\ m\ G$. We derive requirements to verify that a stateful policy T is a proper stateful implementation of G without introducing security flaws.

4.1 Requirements for Information Flow Security Compliance

Information leakages are critical and can occur in subtle ways. For example, the widely used transport protocol TCP detects data loss by sending acknowledgment packages. If A establishes a TCP connection to B, then even if B sends no payload, arbitrary information can be transmitted to A, e.g. via timing channels, TCP sequence numbers, or retransmits. Therefore, we treat information flow security requirements carefully: When considering backflows, all information flow security invariants must still be fulfilled.

$$\forall m \in \mathit{getIFS}\ M.\ \ m\ (\alpha\ T) \tag{1}$$

4.2 Requirements for Access Control Strategies

In contrast, the requirements for access control invariants can be slightly relaxed: If A accesses B, A might expect an answer from B for its request. If B's answer is transmitted via the connection that A established, B does not access A on its own initiative. Only the expected answer is transmitted back to A. If A's software contains no vulnerability that B could exploit with its answer, no access violation

occurs.[2] This behavior is widely deployed in many private and enterprise networks by the standard policy that internal hosts can access the Internet and receive replies, but the Internet cannot initiate connections to internal hosts.

Therefore, we can formulate the requirement for ACS compliance. Access control violations caused by stateful backflows can be tolerated. However, *negative side effects* must not be introduced by permitting these backflows. First, we present an example of a negative side effect. Second, we derive a requirement for verifying the lack of side effects.

Example We examine a building automation network. Let B be the master controller, A a door locking mechanism, and C a log server that records who enters and who leaves the building. The controller B decides when the door should be opened and what to log. The directed policy is described by $G = (\{A,B,C\},\ \{(B,A),(B,C)\})$. The only security invariant m is that A is not allowed to transitively access C. Let $\rightarrow^*$ denote the transitive closure of $\rightarrow$. Then, m prohibits $A \rightarrow^* C$, but it does not prohibit $C \rightarrow^* A$. In this scenario, that means that the physically accessible locking mechanism must not tamper with the integrity of the log server.

Setting $E_\sigma = \{(B,A)\}$ gives $T = (\{A,B,C\},\ \{(B,A),(B,C)\},\ \{(B,A)\})$, and hence $\alpha\ T = (\{A,B,C\},\ \{(B,A),(B,C),(A,B)\})$. This attempt results in a negative side effect. We compute the offending flows for m of $\alpha\ T$ as $\{\{(B,C)\},\ \{(A,B)\}\} = \{\{(B,C)\},\ \overleftarrow{E_\sigma}\}$. Clearly, a violation occurs in $\overleftarrow{E_\sigma}$. Additionally, there is a side effect: the flow from B to C could now cause a violation. Applied to our scenario, this means that in case the locking mechanism sends forged data to the controller, that data could end up in the log. This is a negative side effect. Hence (B,A) cannot securely be made stateful. For completeness, note that because A is just a simple physical actor which only executes B's commands, there is no need for bidirectional communication. On the other hand, (B,C) can be made stateful without side effects.

We formalize the requirement of "no negative side effects" as follows: The violations caused by any subset of the backflows are at most these backflows themselves.

$$\forall X \subseteq \overleftarrow{E_\sigma}.\ \forall F \in \mathit{get\_offending\_flows}\ (\mathit{getACS}\ M)\ (V, E_\tau \cup E_\sigma \cup X).\ F \subseteq X \tag{2}$$

In particular, all offending access control violations are at most the stateful backflows. This is directly implied by the previous requirement by choosing X to be $\overleftarrow{E_\sigma}$ (recall the definition of α).

$$\bigcup \mathit{get\_offending\_flows}\ (\mathit{getACS}\ M)\ (\alpha\ T) \subseteq \overleftarrow{E_\sigma} \tag{3}$$

Also, considering all backflows individually, they cause no side effects, i.e. the only violation added is the backflow itself.

$$\forall (r,s) \in \overleftarrow{E_\sigma}.\ \bigcup \mathit{get\_offending\_flows}\ (\mathit{getACS}\ M)\ (V, E_\tau \cup E_\sigma \cup \{(r,s)\}) \subseteq \{(r,s)\} \tag{4}$$

It is obvious that (2) implies both (3) and (4).[vi] The condition of (2) is imposed on all subsets, thus ruling out all possible undesired side effects.

However, translating (2) to executable code results in exponential runtime complexity, because it requires iterating over all subsets of $\overleftarrow{E_\sigma}$. This is infeasible for any large set of stateful flows. In this paper, we contribute a new formula[vii], which implies (2) and hence (3) and (4). It has a comparably

[2] Note that we make an important assumption here. This assumption is justified as we only work on the network level and do not consider the application level, which is also the correct abstraction for network administrators when configuring network security mechanisms. It also implies that, as always, vulnerable applications with access to the Internet can cause severe damage.

low computational complexity and thus enables writing executable code for the automated verification of stateful and directed policies.

$$\bigcup get\_offending\_flows\ (getACS\ M)\ (\alpha\ T)\ \subseteq\ \overleftarrow{E_\sigma} \setminus E_\tau \tag{5}$$

Obviously, the runtime complexity of (5) is significantly lower than (2) (see §6). The formula also bears great resemblance to (3). We explain the intention of (5) and prove that it implies (2).

Note that $\overleftarrow{E_\sigma} \setminus E_\tau = \overleftarrow{\{(s,r) \in E_\sigma \mid (r,s) \notin E_\tau\}}$ [viii], which means that it represents the backflows of all flows that are not already in E_τ. In other words, it represents only the newly added backflows. For example, consider the flows between students and employees in Figure 3b: no stateful flows are necessary as bidirectional flows are already allowed by the policy, and the newly added backflows are represented by the dashed edges. Therefore, (5) requires that all introduced violations are only due to the newly added backflows. This requirement is sufficient to imply (2).[ix]

Theorem 1 (Efficient ACS Compliance Criterion) *For ACS, verifying that all introduced violations are only due to the newly added backflows is sufficient to verify the lack of side effects. Formally,* (5) $\implies$ (2).

Proof. We assume (5) and show (2) for an arbitrary but fixed $X \subseteq \overleftarrow{E_\sigma}$. We need to show that $\forall F \in get\_offending\_flows\ (getACS\ M)\ (V, E_\tau \cup E_\sigma \cup X).\ F \subseteq X$. We split $\overleftarrow{E_\sigma}$ into $\overleftarrow{E_\sigma} \setminus E_\tau$ and $\overleftarrow{E_\sigma} \setminus (\overleftarrow{E_\sigma} \setminus E_\tau)$. Likewise, we can split X into $X_1 \subseteq \overleftarrow{E_\sigma} \setminus E_\tau$ and $X_2 \subseteq \overleftarrow{E_\sigma} \setminus (\overleftarrow{E_\sigma} \setminus E_\tau)$. Hence, $X_2 \subseteq E_\tau$ and immediately $E_\tau \cup X_2 = E_\tau$. This simplifies the goal as X_2 disappears from the edges:

$$\forall F \in get\_offending\_flows\ (getACS\ M)\ (V, E_\tau \cup E_\sigma \cup X_1).\ F \subseteq X$$

We show an even stricter version of the goal since $X = X_1 \cup X_2$.

$$\forall F \in get\_offending\_flows\ (getACS\ M)\ (V, E_\tau \cup E_\sigma \cup X_1).\ F \subseteq X_1$$

This directly follows[x] by using Lemma 2 and subtracting $(\overleftarrow{E_\sigma} \setminus E_\tau) \setminus X_1$ from (5). □

5 Automated Stateful Policy Construction

In this section, we present algorithms to calculate a stateful implementation of a directed policy for a given set of security invariants using (1) and (5).

Instead of a set, the algorithms' last parameter is a list because the order of the elements matters. We denote the list cons operator by "::". For example, "$e :: es$" is the list with the first element e and a remainder list es. Since lists can be easily converted to finite sets, we make this conversion implicit for brevity. For example, for a list a, we will write the stateful policy as (V, E, a), where a is implicitly converted to a finite set.

5.1 Information Flow Security Strategies

We start by presenting an algorithm which selects stateful edges in accordance to the IFS security invariants. The algorithm filters a given list of edges for edges which fulfill (1). It also takes as input the directed policy G, the security invariants M, and a list of edges as accumulator a.

$$\begin{aligned}
&\texttt{filterIFS}\ G\ M\ a\ \texttt{[]} &&= a\\
&\texttt{filterIFS}\ G\ M\ a\ (e :: es) &&= \texttt{if}\ \forall m \in getIFS\ M.\ m\ (\alpha\ (V,E,e :: a))\ \texttt{then}\\
&&&\quad \texttt{filterIFS}\ G\ M\ (e :: a)\ es\\
&&&\texttt{else}\\
&&&\quad \texttt{filterIFS}\ G\ M\ a\ es
\end{aligned}$$

The accumulator, initially empty, returns the result in the end. It is the current set of selected stateful flows. The algorithm is designed such that (1) always holds for $T = (V, E, a)$. It simply iterates over all elements e of the input list and checks whether the formula also holds if e is added to a. If so, e is added to the accumulator; otherwise, a is left unchanged.

Depending on the security invariants, multiple results are possible with this filtering criterion. The algorithm deterministically returns one solution. Users can influence the choice of edges that they want to be stateful by arranging the input list such that the preferred edges are listed first. If only one arbitrary solution is desired, lists and finite sets are interchangeable.

The algorithm is sound[xi] and complete.[xii]

Lemma 3 (`filterIFS` Soundness) *If the directed policy $G = (V, E)$ is valid, then for any list $X \subseteq E$, the stateful policy $T = (V, E,$ `filterIFS` $G\ M$ `[]` $X)$ fulfills (1).*

Lemma 4 (`filterIFS` Completeness) *For $G = (V, E)$, let $E_\sigma =$ `filterIFS` $G\ M\ E$. Then, no non-empty subset can be added to E_σ without violating (1).*

$$\forall X \subseteq E \setminus E_\sigma,\ X \neq \emptyset.\ \neg \forall m \in \mathit{getIFS}\ M\ (\alpha\ (V, E, E_\sigma \cup X))$$

5.2 Access Control Strategies

The algorithm `filterACS` follows the same principles as `filterIFS`.

$$\begin{aligned}
&\texttt{filterACS}\ G\ M\ a\ \texttt{[]} \quad = a \\
&\texttt{filterACS}\ G\ M\ a\ (e :: es) = \\
&\qquad \texttt{if}\ e \notin \overleftarrow{E}\ \wedge\ \left(\forall F \in \mathit{get_offending_flows}\ (\mathit{getACS}\ M)\ (\alpha(V, E, e :: a)).\ F \subseteq \overleftarrow{e :: a}\right)\ \texttt{then} \\
&\qquad\quad \texttt{filterACS}\ G\ M\ (e :: a)\ es \\
&\qquad \texttt{else} \\
&\qquad\quad \texttt{filterACS}\ G\ M\ a\ es
\end{aligned}$$

As previously, the order of the elements in the list influences the choice of calculated stateful edges. Edges listed first are preferred. The algorithm is sound[xiii] and complete.[xiv]

Lemma 5 (`filterACS` Soundness) *If the directed policy $G = (V, E)$ is valid, then for any list $X \subseteq E$, the stateful policy $T = (V, E,$ `filterACS` $G\ M$ `[]` $X)$ fulfills (5).*

To show that `filterACS` computes a maximal solution, we must first identify the candidates that `filterACS` might overlook. Flows that are already bidirectional need not be stateful. As illustrated in the example of Figure 3b, no added value is created if stateful connections between students and employees were allowed as no communication restrictions exist between these groups in the first place. Hence only $E \setminus \overleftarrow{E}$ is considered.

Lemma 6 (`filterACS` Completeness) *For $G = (V, E)$, let $E_\sigma =$ `filterACS` $G\ M\ E$. Then, no non-empty subsets $X \subseteq E \setminus (E_\sigma \cup \overleftarrow{E})$ can be added to E_σ without violating (5).*

$$\forall X \subseteq E \setminus (E_\sigma \cup \overleftarrow{E}),\ X \neq \emptyset.\ \neg \left(\bigcup \mathit{get\_offending\_flows}\ (\mathit{getACS}\ M)\ (\alpha\ (V, E, E_\sigma \cup X)) \subseteq \overleftarrow{E_\sigma \cup X} \setminus E \right)$$

5.3 IFS and ACS Combined

Finally, we combine the previous section's algorithms to derive algorithms which compute a solution that satisfies all requirements of a stateful policy.

The first algorithm[xv] simply chains `filterIFS` and `filterACS`.

$$\texttt{generate1}\ G\ M\ e = (V, E, \texttt{filterACS}\ G\ M\ (\texttt{filterIFS}\ G\ M\ e))$$

The second algorithm[xvi] takes the intersection of `filterIFS` and `filterACS`.

$$\texttt{generate2}\ G\ M\ e = (V, E, (\texttt{filterACS}\ G\ M\ e) \cap (\texttt{filterIFS}\ G\ M\ e))$$

Both algorithms are sound.[xvii] It remains unclear whether both are equal in the general case. Furthermore, it is difficult to prove (or disprove) their completness, because both algorithms work on almost arbitrary functions M. However, we have formal proofs for the completeness of `filterACS` and `filterIFS` and the structure of `generate1` and `generate2` suggest completeness. In our experiments, `generate1` and `generate2` always calculated the same maximal solution.

Theorem 2 (`generate{1,2}` Soundness) *The algorithms* `generate1` *and* `generate2` *calculate a stateful policy that fulfills both IFS and ACS requirements.*

Example Recall our running example. We illustrate how Figure 3b can be calculated from Figure 3a and the security invariants. All ACS invariants impose only local—in contrast to transitive—access restrictions. Therefore, the ACS invariants lack side effects and `filterACS` selects all flows (excluding already bidirectional ones). The invariant that the file server stores confidential data also introduces no restrictions: Both *filesSrv* $\rightarrow$ *employees* and *employees* $\rightarrow$ *filesSrv* are allowed and since the employees are trusted, they can further distribute the data. Therefore, `filterIFS` applied on only this invariant correctly selects all flows. Up to this point, the network's functionality is maximized. However, since the printer is classified as information sink, it must not leak any data. Therefore, `filterIFS` applied to this invariant selects all but the flows to the printer. Ultimately, both `generate` algorithms compute[xviii] the same maximal stateful policy, illustrated in Figure 3b. The soundness and completeness of the running example is hence formally proven. The case study in Section 7 will focus on performance and feasibility in a large real-world example.

6 Computational Complexity

The computational complexity of all presented formulae depends on the computational complexity of the security invariants $m \in M$. As we allow almost any function m as security invariant, the computational complexity can be arbitrarily large. However, most of the security invariants we use in our daily business check a property over all flows in the network. Thus, the computational complexity of m is linear in the number of edges, i.e. $\mathrm{O}(|E|)$. The trivial computational complexity of *offending_flows* is in $\mathrm{O}(2^{|E|} \cdot |E|^2)$, since it iterates over all subsets of E. However, given the structure of the security invariants we use, we provide proof[xix] that the offending flows for our security invariants are uniquely defined [7]. They can be computed in $\mathrm{O}(|E|)$. The result is a singleton set whose inner set size is also in $\mathrm{O}(|E|)$. We present the computational complexity of our formulae and algorithms in this section for security invariants and offending flows with the mentioned complexity.[3] Our solution is not limited to these security invariants, but the computational complexity increases for more expensive security invariants.

We assume that set inclusion can be computed with the hedge union algorithm in $\mathrm{O}(k_i + k_j)$ for sets of size $k_{\{i,j\}}$. Since E_τ and E_σ are bounded by E, set inclusion is in $\mathrm{O}(|E|)$.

[3] The computational complexity results are not formalized in Isabelle/HOL, because in its present state, there is no support for reasoning about asymptotic runtime behavior.

Verifying information flow compliance, i.e. (1), can be computed in $O(|E| \cdot |M|)$. Hence, for a constant number of security invariants, the computational complexity is linear in the number of policy rules.

To verify access control compliance, we first note that (2) is in $O(2^{|E|} \cdot |E| \cdot |M|)$ which is infeasible for a large policy. However, we provide (5), which implies (2), and can be computed in $O(|E| \cdot |M|)$. Hence, for a constant number of security invariants, the computational complexity is linear in the number of policy rules.

The `filter` and `generate{1,2}` algorithms only add $O(|E|)$ to the complexity. Hence, for a constant number of security invariants, computing a stateful policy implementation from a directed policy is quadratic in the number of policy rules, which is feasible even for large policies with thousands of rules.

7 Case Study

In a study, Wool [23] analyzed 37 firewall rule sets from telecommunications, financial, energy, media, automotive, and many other kinds of organization, collected in 2000 and 2001. The maximum observed rule set size was 2671, and the average rule set size was 144. Wool's study "indicates that there are no good high-complexity rule sets" [23]. If in a scenario complicated rule sets are unavoidable, formal verification to assert their correctness is advisable.

In this section, we analyze the firewall rule set of TUM's Chair for Network Architectures and Services. With a rule set size of approximately 2983 as of November 2013, this firewall configuration can be considered representatively large. Almost all rules are stateful, hence the firewall generally allows all established connections and only controls who is allowed to initiate a connection. We publish our complete data set, allowing others to reproduce our results and reuse the raw data for their research.

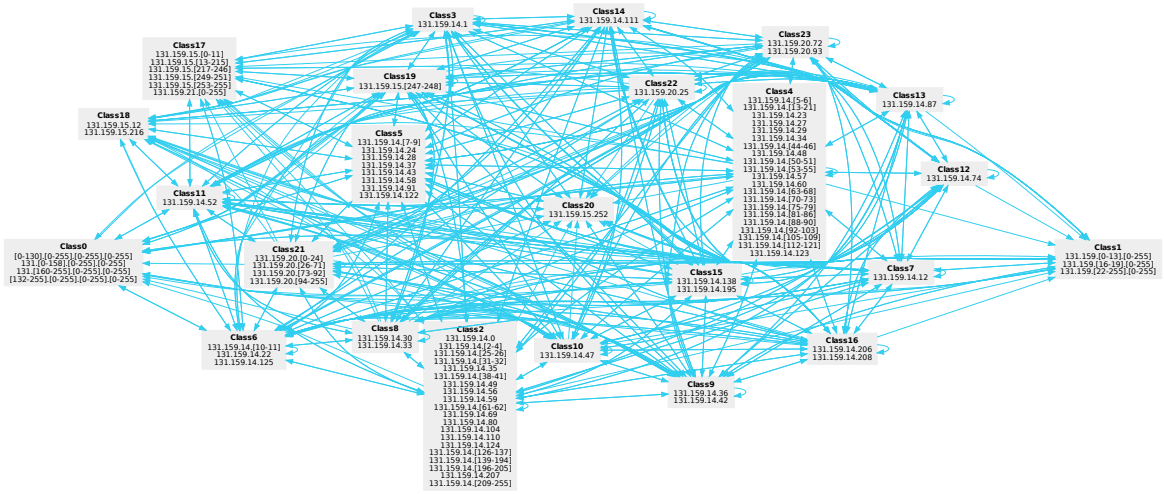


Figure 4: SSH landscape of the TUM Chair for Network Architectures and Services

As there is no written formal security policy for our network, we reverse-engineered the security policy and invariants with the help of our system administrator. The firewall contains rules per IP range that permit the services which are accessible from some IP range. Most rules are similar to rule one in Figure 1. We regard the firewall rules about which hosts can initiate a connection as security policy. It

is not unusual that the implementation is also the documentation [17, §1]. We verify that the so derived security policy, i.e. which hosts can initiate connections, corresponds to the stateful implementation, i.e. all connections are stateful.

In order to prepare the firewall rules as graph, we used *ITval* [13] to first partition the IP space into classes with equivalent access rights [14] which form the nodes of our policy. For each of these classes, we selected representatives and queried ITval for "which hosts can this representative connect to" and "which hosts can connect to this representative". This method is also suggested by Marmorstein [14]. The resulting IP ranges were mapped back to the classes. This generates the edges of the security policy graph. We asserted that these two queries result in the same graph. For brevity, we restrict our attention to the SSH landscape, i.e. TCP port 22. The full data set is publicly available. The SSH landscape results in a security policy with 24 nodes (sets of IP ranges with equal access rights) and 496 edges (permissions to establish SSH connections). The resulting graph is shown in Figure 4.

A detailed discussion with our system administrator indicated that the graphical representation of the computed graph contains helpful information. It reveals that the computed policy does not exactly correspond to the firewall's configuration. We could not clearly identify the cause for this discrepancy. However, the graph provides a sufficient approximation of our security policy. In the future, we will try to generate the graph using the approach by Tongaonkar, Niranjan, and Sekar [21], of which unfortunately no code is publicly available. In the long term, we see the need for formally verified means of translating network device configurations, such as firewall rule sets, SDN flow tables, routing tables, and vendor specific access control lists to formally accessible objects, such as graphs.

After having constructed the security policy, we implemented our security invariants. They state that our IP ranges form a big set of mostly collaborating hosts. As a general rule, internal hosts are protected from accesses from the outside world, but there are many exceptions.

No IFS invariants exist and our ACS invariants cause no side effects. Note that we are evaluating neither the quality of our security policy nor the quality of our security invariants, but the quality of the stateful implementation in this large real-world scenario. As expected, our `generate{1,2}` algorithms identify all unidirectional flows as upgradable to stateful. This shows that the standard practice to declare (almost) all rules as stateful, combined with common simple invariants does not introduce security issues. For our invariants, our algorithms always generate a graph T such that $\alpha\ T = (V,\ E \cup \overleftarrow{E})$. This means that in this scenario, we have a formal justification that all directed policy rules correspond to their stateful implementation, without any security concern. This maximizes the network's functionality without introducing security risks and is thus the optimal solution.

This statement can be generalized to all networks without IFS invariants and without side effects in the ACS invariants. We provide formal proofs for both `generate{1,2}` algorithms.[xx] Due to its simplicity, universality, and convenient implications for everyday use, we state this result explicitly.

Corollary 1 *If there are no information flow security invariants and all access control invariants of a directed policy lack side effects, a security policy can be smoothly implemented as stateful policy, without any security issues concerning state.*

Our algorithms return this result, i.e. α (`generate G M E`) $= (V,\ E \cup \overleftarrow{E})$. If there are information flow security invariants or access control invariants with side effects, our algorithms also handle these problems.

All results can be computed interactively on today's standard hardware. The graph preparation, which needs to be done only once, takes several seconds. Our `generate` algorithms take a few seconds. This shows the practical low computational complexity for a large real-world study.

8 Related Work

In the research field of firewalls, several successful approaches to ease management [2] and uncovering errors [25] exist. In [17], the authors propose that a network security policy should exist in an informal language. A translation from the informal language to a formalized policy with an information content comparable to the directed policy in this work must be present. The same model for firewall rules and security policy is used. The authors model services, i.e. ports, explicitly but ignore the direction of packets in their firewall model and are hence vulnerable to several attacks, such as spoofing [24]. Constraint Satisfaction Problem (CSP) solving techniques are used to test compliance of the security policy and the firewall rule set. Using Logic Programming with Priorities (LPP), Bandara et al. [1] build a framework to detect firewall anomalies and generate anomaly-free firewall configurations from a security policy. The authors explicitly point out the need for solving the stateful firewall problem.

Brucker et al. [4] provide a formalization of simple firewall policies in Isabelle/HOL and simplification rules for them. With this, they introduce HOL-TestGen/FW, a tool to generate test cases for conformance testing of a firewall rule set, i.e. that the firewall under test implements its rule set correctly. In [5], the authors augment their work with user-friendly high-level policies. This also allows the verification of a network specification with regard to these high-level policies.

Guttman et al. [9, 8] focus on distributed network security mechanisms, such as firewalls, filtering routers, and IPsec gateways. Security goals centered on the path of a packet through the network can be verified against the distributed network security mechanisms configuration.

Using formal methods, network vulnerability analysis reasons about complete networks, including the services and client software running in the network. Using model checking [19] or logic programming [16], network vulnerabilities can be discovered or the absence of vulnerabilities can be shown. One potential drawback of these methods is that the set of vulnerabilities must be known for the analysis, which can be an advantage for postmortem network intrusion analysis, but is also a downside when trying to estimate a network's future vulnerability.

Kazemian et al. [11] present a method for the packet forwarding plane to identify problems such as reachability issues, forwarding loops, and traffic leakage. Considering the individual packet bits, the header space is represented by a $\langle$*maximum packet header size in bits*$\rangle$-dimensional space. An efficient algebra on the header space is provided which enables checking of the named use cases.

9 Conclusion

Stateful firewall rules are commonly used to enforce network security policies. Due to these state-based rules, flows opposite to the security policy rules might be allowed. On the one hand, we argued that under presence of side effects or information flow invariants, a naive stateful implementation might break security invariants. On the other hand, declaring certain firewall rules to be stateless might impair the functionality of the network. This problem domain has often been overlooked in previous work.

Verifying that a stateful firewall rule set is compliant with the security policy and its invariants is computationally expensive. In this work, we discovered a linear-time method and contribute algorithms for verifying and also for computing stateful rule sets. We demonstrated that these algorithms are fast enough for reasonably large networks, while provably maintaining soundness and completeness.

Since the complete formalization, including algorithms and proofs, has been carried out in Isabelle/HOL, there is high confidence in their correctness. For the future, we see the need for verified translation methods from network device configurations to formally accessible objects, such as graphs.

Acknowledgements & Availability

We thank our network administrator Andreas Korsten for his valuable input, his time and commitment. We appreciate Heiko Niedermayer's and Jasmin Blanchette's feedback.

This work has been supported by the German Federal Ministry of Education and Research (BMBF), EUREKA project SASER, grant 16BP12304, and by the European Commission, FP7 project EINS, grant 288021.

The Isabelle/HOL theory files can be obtained at `https://github.com/diekmann/topoS`. The complete raw data set of the firewall rules and a dump of our LDAP database, used to automatically construct some firewall rules, can be obtained at `https://github.com/diekmann/net-network`.

References

[1] Arosha K. Bandara, Antonis C. Kakas, Emil C. Lupu & Alessandra Russo (2009): *Using argumentation logic for firewall configuration management*. In: *IFIP/IEEE International Symposium on Integrated Network Management*, IEEE, pp. 180–187, doi:10.1109/INM.2009.5188808.

[2] Yair Bartal, Alain Mayer, Kobbi Nissim & Avishai Wool (1999): *Firmato: A novel firewall management toolkit*. In: *IEEE Symposium on Security and Privacy*, IEEE, pp. 17–31, doi:10.1109/SECPRI.1999.766714.

[3] M. Bishop (2003): *Computer Security: Art and Science*. Addison-Wesley.

[4] Achim D. Brucker, Lukas Brügger & Burkhart Wolff (2008): *Model-based Firewall Conformance Testing*. In: *Testing of Software and Communicating Systems*, Springer, pp. 103–118, doi:10.1007/978-3-540-68524-1_9.

[5] Achim D. Brucker, Lukas Brügger & Burkhart Wolff (2013): *HOL-TestGen/FW: An Environment for Specification-based Firewall Conformance Testing*. In: *International Colloquium on Theoretical Aspects of Computing – ICTAC 2013*, *Lecture Notes in Computer Science* 8049, Springer Berlin Heidelberg, pp. 112–121, doi:10.1007/978-3-642-00593-0_28.

[6] Bundesamt für Sicherheit in der Informationstechnik (2013): *Technische Richtlinie BSI TR-03109-1 – Anforderungen an die Interoperabilität der Kommunikationseinheit eines intelligenten Messsystems*, 1.0 edition. `https://www.bsi.bund.de`.

[7] Cornelius Diekmann, Stephan-A. Posselt, Heiko Niedermayer, Holger Kinkelin, Oliver Hanka & Georg Carle (2014): *Verifying Security Policies using Host Attributes*. In: *Proc. FORTE*, Springer, Berlin, Germany. Available at `http://www.net.in.tum.de/pub/diekmann/forte14.pdf`. To appear.

[8] J. D. Guttman (1997): *Filtering postures: local enforcement for global policies*. In: *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, Washington, DC, USA, doi:10.1109/SECPRI.1997.601327.

[9] Joshua D. Guttman & Amy L. Herzog (2005): *Rigorous automated network security management*. *International Journal of Information Security* 4, pp. 29–48, doi:10.1007/s10207-004-0052-x.

[10] Andrew H. R. Hansen (2012): *Protecting Critical Infrastructure*. *ASA Institute for Risk & Innovation*, pp. 1–12. `http://anniesearle.com/web-services/Documents/ResearchNotes/ASA_ResearchNote_ProtectingCriticalInfrastructure_June2012.pdf`.

[11] Peyman Kazemian, George Varghese & Nick McKeown (2012): *Header space analysis: static checking for networks*. In: *Networked Systems Design and Implementation*, NSDI'12, USENIX, pp. 113–126. Available at `https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/kazemian`.

[12] Jeremy Kirk (2010): *Verizon: Data breaches often caused by configuration errors*. networkworld. `http://www.networkworld.com/news/2010/072910-verizon-data-breaches-often-caused.html`.

[13] Robert M. Marmorstein & Phil Kearns (2005): *A Tool for Automated iptables Firewall Analysis.* In: *USENIX Annual Technical Conference, FREENIX Track*, pp. 71–81. Available at `https://www.usenix.org/legacy/publications/library/proceedings/usenix05/tech/freenix/full_papers/marmorstein/marmorstein.pdf`.

[14] Robert M. Marmorstein & Phil Kearns (2006): *Firewall Analysis with Policy-based Host Classification.* In: *LISA*, 6, pp. 4–4. Available at `http://usenix.org/event/lisa06/tech/full_papers/marmorstein/marmorstein.pdf`.

[15] Tobias Nipkow, Lawrence C. Paulson & Markus Wenzel (2002, last updated 2013): *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*. *LNCS* 2283, Springer. Available at `http://isabelle.in.tum.de/doc/tutorial.pdf`.

[16] Xinming Ou, Sudhakar Govindavajhala & Andrew W Appel (2005): *MulVAL: A logic-based network security analyzer*. In: *14th USENIX Security Symposium*, pp. 113–128. Available at `https://www.usenix.org/legacy/publications/library/proceedings/sec05/tech/full_papers/ou/ou.pdf`.

[17] S. Pozo, R. Ceballos & R. M. Gasca (2007): *CSP-Based Firewall Rule Set Diagnosis using Security Policies*. *International Conference on Availability, Reliability and Security*, pp. 723–729, doi:10.1109/ARES.2007.63.

[18] The netfilter.org project: *netfilter/iptables project*. Available at `http://www.netfilter.org/`.

[19] R.W. Ritchey & P. Ammann (2000): *Using model checking to analyze network vulnerabilities*. In: *IEEE Symposium on Security and Privacy*, pp. 156–165, doi:10.1109/SECPRI.2000.848453.

[20] Justine Sherry, Shaddi Hasan, Colin Scott, Arvind Krishnamurthy, Sylvia Ratnasamy & Vyas Sekar (2012): *Making middleboxes someone else's problem: Network processing as a cloud service*. *ACM SIGCOMM Computer Communication Review* 42(4), pp. 13–24, doi:10.1145/2377677.2377680.

[21] Alok Tongaonkar, Niranjan Inamdar & R Sekar (2007): *Inferring Higher Level Policies from Firewall Rules.* In: *LISA*, 7, pp. 1–10. Available at `https://www.usenix.org/legacy/event/lisa07/tech/full_papers/tongaonkar/tongaonkar.pdf`.

[22] Verizon Business RISK team & United States Secret Service (2010): *2010 Data Breach Investigations Report*. `http://www.verizonenterprise.com/resources/reports/rp_2010-DBIR-combined-reports_en_xg.pdf`.

[23] Avishai Wool (2004): *A quantitative study of firewall configuration errors*. *Computer, IEEE* 37(6), pp. 62 – 67, doi:10.1109/MC.2004.2.

[24] Avishai Wool (2004): *The use and usability of direction-based filtering in firewalls*. *Computers & Security* 23(6), pp. 459–468, doi:j.cose.2004.02.003.

[25] Lihua Yuan, Hao Chen, Jianning Mai, Chen-Nee Chuah, Zhendong Su & P. Mohapatra (2006): *FIREMAN: a toolkit for firewall modeling and analysis*. In: *IEEE Symposium on Security and Privacy*, pp. 199–213, doi:10.1109/SP.2006.16.

Definitions, Lemmas and Theorems

[i] FiniteGraph.valid-graph [ii] offending-flows-union-mono [iii] Un-set-offending-flows-bound-minus-subseteq [iv] valid-stateful-policy, stateful-policy-compliance [v] valid-stateful-policy.finite-$*$ [vi] stateful-policy-compliance.compliant-stateful-ACS-only-state-violations-union, stateful-policy-compliance.compliant-stateful-ACS-no-state-singleflow-side-effect [vii] stateful-policy-compliance.compliant-stateful-ACS [viii] backflows-filternew-flows-state [ix] stateful-policy-compliance.compliant-stateful-ACS-no-side-effects [x] stateful-policy-compliance.compliant-stateful-ACS-no-side-effects-filternew-helper [xi] filter-IFS-no-violations-correct [xii] filter-IFS-no-violations-maximal-allsubsets [xiii] filter-compliant-stateful-ACS-correct [xiv] filter-compliant-stateful-ACS-maximal-allsubsets [xv] generate-valid-stateful-policy-IFSACS [xvi] generate-valid-stateful-policy-IFSACS-2 [xvii] generate-valid-stateful-policy-IFSACS-stateful-policy-compliance, generate-valid-stateful-policy-IFSACS-2-stateful-policy-compliance [xviii] Impl_List_Playground_ChairNetwork_statefulpolicy_example.thy [xix] BLP-offending-set, CommunicationPartners-offending-set, ... [xx] generate-valid-stateful-policy-IFSACS-noIFS-noACSsideeffects-imp-fullgraph, generate-valid-stateful-policy-IFSACS-2-noIFS-noACSsideeffects-imp-fullgraph