Packet flow analysis in IP networks using data-flow analysis

Raghavan Komondoor
Indian Institute of Science, Bangalore
raghavan@csa.iisc.ernet.in

K. Vasanta Lakshmi
Indian Institute of Science, Bangalore
kvasanta@csa.iisc.ernet.in

Deva P. Seetharam
IBM Research India
dseetharam@in.ibm.com

Sudha Balodia
Indian Institute of Science, Bangalore
sudha.balodia@gmail.com

ABSTRACT

Static analysis (aka offline analysis) of a model of an IP network is useful for understanding, debugging, and verifying packet flow properties of the network. Data-flow analysis is a method that has typically been applied to static analysis of programs. We propose a new, data-flow based approach for static analysis of packet flows in networks. We also investigate an application of our analysis to the problem of inferring a high-level policy from the network, which has been addressed in the past only for a single router.

1. INTRODUCTION

Analysis of the flow of packets across an IP network is an important problem. It has varied applications, such as identifying anomalies in configuration files in routers [14], testing of router implementations [3], checking whether a network configuration satisfies a high-level policy of a network administrator by querying properties of the configuration [8, 10], and inferring such a high-level policy automatically from the network configuration [11, 4]. However, such an analysis is challenging, because packet routing in an IP network is a complex activity. Routers intervene between subnets (i.e., fully connected collections of hosts), and perform operations on packets such as filtering, routing to adjacent routers or subnets, and transformation, e.g., for network address translation (NAT). Each operation performed by a router is predicated (i.e., guarded) by the current content of the header of the packet, which, due to transformations, changes as the packet flows through the network. Furthermore, a NATing rule may write any value from a specified range of values, and a firewall may have a choice in which interface to send a packet out of; this increases the number of alternative packet-flow scenarios that could occur during network operation. All of this means that it is quite difficult to precisely analyze the flow of packets across the network.

The state-of-practice for analyzing reachability is to send test packets in the actual network, using commercially available tools. However, testing does not give complete information about all possible packet flow outcomes, because it is infeasible to send all possible packets across a network. Several *static* (or offline) analysis approaches, e.g., [13, 14, 1], have been reported in the literature in order to overcome this disadvantage; these approaches analyze a specification of the network topology and router configurations (i.e., a *model* of the network), and emit information about possible packet flows in the network. Model checking is another technique that has been widely used in the literature [9, 5, 1] for static analysis of networks; while the former two approaches model the flow of a single packet through the network, Al Shaer's approach [1] models transitions of the set of all packets in a network.

1.1 Contributions

1) Our primary contribution is a novel approach for determining packet reachability *precisely* in an IP network, which we formulate as an instance of the *data-flow analysis* framework [6]. The previous static analysis techniques for IP networks that most closely resemble ours are the ones based on transitive closure analysis [13], and graph propagation with bounded unfolding of cycles [8, 10, 14]. All of these approaches are unsound, i.e., can miss certain packet flows, in the presence of certain kinds of cycles in the network. Data-flow analysis involves an iterative analysis until a fix-point is reached, and hence cleanly addresses this situation.

2) We also show how our analysis can be applied for inferring a high level-policy from a distributed network of firewalls. In previous work [11, 4] researchers have formulated the problem of inferring a high-level policy from a *single* router. We first generalize this problem to the setting of a network of multiple routers, and then show how to solve it using our reachability analysis.

Due to space constraints we omit from this writeup several aspects of our problem and approach, and refer the reader to an accompanying technical report [7] for more details.

2. MODEL AND TERMINOLOGY

A *concrete packet* is an IP packet in a network. We only model the headers of packets; let *pkSz* be the total number

ISEC '12, Feb. 22-25, 2012, Kanpur, UP, India

of bits in a packet header, partitioned into *nFlds* fields. We denote the fields of a packet p as $p.f_1, p.f_2, \ldots, p.f_p$. These fields include the source address and port, and destination address and port.

A network consists of a set of nodes $\mathbb{N}$, which are partitioned into two categories: a set of *zones* (i.e. subnets) $\mathbb{Z}$, which are terminal nodes, and a set of *firewalls* (i.e., routers) $\mathbb{F}$, which are intermediate nodes. We use zones to model organizational subnets as single units; i.e., we assume that each zone z has a set of publicly visible IP addresses $addr_z$ (with the sets of distinct zones being non-overlapping). We use n, n_i, etc., to represent individual nodes, z, z_i, etc., to denote individual zones, and f, f_i, etc., to denote individual firewalls. Each zone has a single interface connected to one or more firewalls, while each firewall has a set of one or more interfaces. When we say $(m, i_1) \rightarrow (n, i_2)$, we mean interface i_1 belongs to node m, interface i_2 belongs to node n, m and n are distinct nodes, and i_1 and i_2 are physically linked. Since physical links are undirected, the presence of the link $(m, i_1) \rightarrow (n, i_2)$ implies the presence of the link $(n, i_2) \rightarrow (m, i_1)$.

Our model of a firewall is based on the widely used package `Iptables` [2]. Each firewall f has four tables: a Destination NATing table $f.dnat$, a filtering table $f.filt$, a Source NATing table $f.snat$, and a routing table $f.rt$. Each packet entering f through any of its interfaces goes through the above tables in the given order. We assume that firewalls are pure routers; i.e., they don't create or ultimately accept packets. A filtering table is a sequence of filtering rules, while each of the two NATing tables is a sequence of NATing rules. Each rule r (filtering or NATing) has two components: its "guard" $r.grd$, which is a propositional formula on the bits in a packet header, and "action" $r.act$. A packet c is said to *match* a rule r if c satisfies the formula $p.grd$. A packet entering a table is matched against each rule in the table sequentially until a matching rule is found; the matching rule's action is then taken on the packet, and the remaining rules in the table are ignored (for this packet). The final rule in any filtering table has the guard *true* (i.e., is a *default* rule). For filtering rules the action is either "drop" or "accept". For a NATing rule, the action part specifies the field f_i (source/destination address/port) to modify, as well as a range r of (new) values for the field. At run time the system chooses *one* of the values from this range and overwrites field f_i of the matched packet to this value. The default rule in a NATing table does no transformation of the packet. The routing table $f.rt$ of firewall f is a function from the interfaces in f to formulas, each of which is a constraint on destination addresses; i.e., if a packet c, after having gone through the DNAT, filtering, and SNAT tables in f, has destination address d, it is then sent out of one of the interfaces i of f such that d satisfies the formula $f.rt(i)$.

Note in the discussion above that choices may have to be made at run time by NATing rules as well as during the final routing step. We do not model how these choices are made during network operation, and instead, in our analysis, assume that *all* choices are possible. Also, we assume the following on the flow of concrete packets in the network: (a) There is no IP spoofing; i.e., every packet leaving a zone z has a source address that matches $addr_z$, and a source port that is within the valid port-range of z. (b) Every packet that enters the network either reaches a zone or is dropped by a firewall filtering rule.

1. *p.curr*: Formula representing the current contents of the set of concrete packets represented by p.
2. *p.orig*: Formula representing the original contents of the set of packets leaving a zone that, after flowing through the network, become the packets represented by $p.curr$.
3. *p.ifNated*: A vector of bits, one per field in a packet header. $p.ifNated.b_i$ is 1 if $p.curr.f_i$ contains a value overwritten by NATing (by some firewall).

(a)

1: *Inputs:* (1) A network configuration, (2) an originating zone z_0, and (3) an "initial" abstract packet p_0 at zone z_0.
2: *Outputs:* At each node n in the network a set of abstract packets $n.abs$, which represents all concrete packets that may reach n along some path during actual network operation.
3:
4: Initialize $z_0.abs$ to $\{p_0\}$. Mark z_0.
5: For all nodes n other than z_0 initialize $n.abs$ to the empty set.
6: **while** there exist marked nodes **do**
7: Choose a marked node m, and unmark it.
8: **for all** links $(m, i_1) \rightarrow (n, i_2)$ **do**
9: Replace $n.abs$ with $n.abs \sqcup ff_{(i_1,i_2)}(m.abs)$.
10: If node n was unmarked and if the new value of $n.abs$ is different from the old value, then mark n.
11: **end for**
12: **end while**

(b)

Figure 1: (a) Fields in an abstract packet $p \in AbsPk$ (b) Propagation of abstract packets.

3. THE BASE ALGORITHM

3.1 The data-flow lattice

Instantiating a data-flow analysis requires us to specify (a) a directed graph on which the analysis is to be performed, (b) a data lattice L, whose elements are called *abstract values*, which is closed wrt a *join* operation (written as "$\sqcup$"), (c) *transfer functions* for the edges in the graph, each of which is a function from L to L, (d) the *initial* abstract value at some designated *originating* node of the graph. In our setting the nodes in the network are the graph nodes, and each link $(m, i_1) \rightarrow (n, i_2)$ in the network results in a graph edge $m \rightarrow n$. The originating zone z_0, and an abstract value p_0 that leaves this zone, are assumed to be a given. In our setting each abstract value is a *set* of *abstract packets*, from the domain *AbsPk*, where each abstract packet in turn intuitively represents a set of concrete packets. The join operation on abstract values is set union. For us p_0 has to be a single abstract packet, which represents all concrete packets that may originate from z_0 during actual network operation, and that have destination addresses of other zones in the network.

We show the data-flow analysis algorithm in Figure 1(b); this is basically Kildall's algorithm [6], instantiated to our setting. When the algorithm terminates, the set of abstract packets $n.abs$ computed at each node n represents all concrete packets that may potentially enter n (through any interface of n) through some path beginning at z_0.

Each abstract packet $p \in AbsPk$ is a structure with a three fields *curr*, *orig* and *ifNated*; see Fig. 1(a). $p.curr$ is a propositional formula on the bits $b_0, b_1, \ldots, b_{pkSz}$ in a packet header. Due to NATing, the current form of the packets (as

represented by $p.curr$) could be different from their *original* form when they originally left the designated originating zone z_0. Therefore, $p.orig$ is a formula that represents the original forms of the packets represented by p when they left z_0. The field *ifNated* is a bit vector with one bit for each field in the packet header. If a bit i is set to 1 it means that the i^{th} field of concrete packets represented by p have been overwritten by some NATing rule along some path.

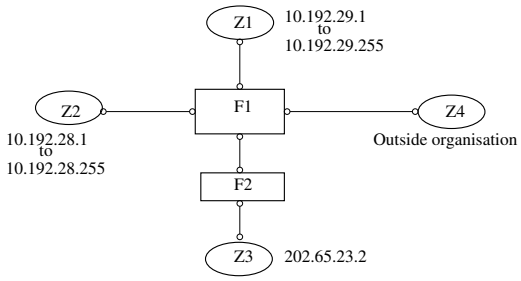
The formal correctness guarantee of the algorithm is that if an abstract packet p is in the set $n.abs$ at some node n, then for every concrete packet c_1 that satisfies the formula $p.orig$ and for every concrete packet c_2 that satisfies the formula $p.curr$ there is a path in the network from z_0 to some of interface i of n such that c_1 satisfies $p_0.curr$ and c_1 becomes transformed to c_2 by the time it reaches i along the path. We assume that $p_0.curr = p_0.orig$.

3.2 Transfer functions of links

The transfer function $ff_{(i_1,i_2)}$ for a link $(m, i_1) \rightarrow (n, i_2)$ is the composition of transfer functions for the DNAT table of m, filtering table of m, SNAT table of m, followed by a filtering rule which only accepts concrete packets whose destination address satisfies $m.rt(i_i)$. The transfer function of a table operates as follows. An abstract packet that *matches* a rule in the table is transformed by the rule's transfer function and then sent directly to the end of the table. On the other hand, an abstract packet that represents concrete packets that *don't* match a rule is sent to the next rule in the table. Note that a single abstract packet's formula may *partially* overlap a rule's guard; in this case the packet is specialized into two packets, one that matches the rule and one that doesn't; then, both the packets are processed as described above. We omit formal specifications of the transfer functions due to lack of space, and instead illustrate them below using the example network in Figure 2(a).

In the example we have annotated each of the zones Z1, Z2, and Z3 with the address(es) of the hosts in the zone. Part (b) of the figure shows the filtering and NATing rules in the two firewalls. The guard s=*pat* (d=*pat*) matches packets whose source (destination) address matches *pat*. In the *Action* part of the NATing rules, "SNAT r" means that the rule overwrites the source address of the matching packet with an address from the range r.

We use the notation p =<[s_c:d_c],[s_o:d_o]> to represent an abstract packet p, where the s's (d's) are formulas (i.e., ranges) for the source (destination) address field. We ignore port numbers for now. The first pair of square brackets represents $p.curr$, whereas the second pair represents $p.orig$. Let zone Z1 be the originating zone, and consider the abstract packet p_0 = <[10.192.29.1-255:*true*], [10.192.29.1-255:*true*]> leaving this zone. This abstract packet is shown in the first row in Part (c) of the figure. When this abstract packet enters firewall F1, it gets refined into the packet p_1 = <[10.192.29.1-255:¬{209.85.153.85}], [10.192.29.1-255: ¬{209.85.153.85}]> by filtering Rule 1, and flows out of the filtering table. Subsequently, this abstract packet matches SNATing Rule 3, and leaves F1 as p_2 = < [202.67.34.6-10:¬{209.85.153.85}], [10.192.29.1-255: ¬{209.85.153.85}]>. (Note that if Rule 3's guard had matched a subset of packets represented by p_1 then we would have had an additional abstract packet leaving F1, representing un-NATed packets that did not match Rules 3 or 4.) Subsequently, F1 forwards p_2 to Z2, F2, and Z4. Consider



(a)

#	*Guard*	*Action*
F1 filtering table:		
1.	s=10.192.29.[1-255], d=209.85.153.85	DROP
2.	s=10.192.28.[1-255], d=209.85.153.85	DROP
F1 SNAT table:		
3.	s=10.192.29.[1-255]	SNAT 202.67.34.[6-10]
4.	s=10.192.28.[1-255]	SNAT 202.67.34.[1-5]
F2 filtering table:		
5.	s=202.67.34.[6-10]	DROP

(b)

$Z1.abs =$ < [10.192.29.1-255 : *true*], [10.192.29.1-255 : *true*] >
$Z2.abs =$ < [202.67.34.6-10 : 10.192.28.1-255], [10.192.29.1-255 : 10.192.28.1-255] >

(c)

Figure 2: (a) Example network (b) Firewalls configuration (c) Reached abstract packets, with Z1 as origin

Z2; p_2 gets refined as shown in the second row of Part (c) of the figure when it reaches Z2. Let's call this abstract packet p_3. Note that the destination address component of *both* the "curr" formula and the "orig" formula of p_3 have gotten refined to Z2's address range. (If, on other hand, some firewall between F1 and Z2 had DNATed p_2's destination address to Z2's address range, then p_3's "orig" formula's destination address component would have remained as it was in p_1 and p_2, because even packets originally not addressed to Z2 could end up reaching Z2 due to this DNATing.)

4. APPLICATION: INFERRING HIGH LEVEL POLICY OF A NETWORK

Real-life networks can be large, with 5-500 intermediate routers [13]. Configuring these routers correctly is a complex and error-prone task. In a study of 37 real firewalls Wool [12] found that each one of them was misconfigured, and had security vulnerabilities. Therefore, it is important for network administrators to have access to tools that infer a compact, high-level policy from a network that has already been setup, to help them debug and validate the configuration. Tongaonkar et al [11] and Horowitz et al [4] have proposed inferring a policy from a *single* firewall. In both these approaches the initial step is to find the rules that have overlapping guards, and then to present a transformed, or differently organized version of the ruleset. While Tongaonkar et al *flatten* the ruleset, by eliminating all overlap between them, Horowitz et al organize the rules hierarchically, with rules with weaker guards placed "above" rules with stronger guards. These ideas do not extend cleanly to the setting of multiple firewalls connected as a network. Because different sets of rules may be correlated along different

paths in a network, it is not clear that rule correlations can be presented in a natural, compact manner in this setting.

Our hypothesis is that in many cases it would help the administrator if for each zone z, they are simply given an "accept" formula that characterizes the set of packet headers that leave z that eventually reach *some* other zone, and a "reject" formula that characterizes the set of packet headers leaving z that get dropped by some rule. The two sets may, in general, be overlapping; a non-empty overlap should be a matter of concern to the administrator, because packets matching both these formulas may reach some zone, or none at all, depending on the (non-deterministic) route they take through the network. This pair of formulas for zone z is a *high-level* policy, in the sense that it is compact, and conveys useful end-to-end information whose representation is not tied to the actual way in the which the network configuration has been set up.

The first step in determining this high-level policy is to run our analysis treating z as the "originating" zone z_0. Then, the "accept" formula for zone z is simply

$$\bigvee_{z_i \in \mathbb{Z} - \{z\}} z_i.abs.orig$$

If the set of all filtering rules in the network with $DROP$ as the action is represented by D then the "drop" formula for z is

$$\bigvee_{r \in D} r.dropped\_packets$$

where $r.dropped\_packets$ is defined as follows, and is intuitively the set of packets (in their original form) that match (and are hence dropped by) rule r.

$$\bigvee_{p \mid p \in r.abs \wedge p.curr \wedge r.grd} p.orig$$

Note that for this application the algorithm as shown in Figure 1 needs to be extended slightly to record the set of abstract packets $r.abs$ that reach each rule r in each firewall, in addition to recording the packets that reach each node n.

In the example in Fig. 2, the "accept" formula for origin zone Z1 is:

[10.192.29.1-255:¬{10.192.29.1-255, 209.85.153.85, 202.65.23.2}]

which corresponds to, $Z2.abs.orig \vee Z4.abs.orig$.

The "reject" formula for Z1 is:

[10.192.29.1-255:{202.65.23.2, 209.85.153.85}]

which corresponds to: $1.dropped\_packets \vee 5.dropped\_packets$, where 1 and 5 are rule numbers in Fig. 2. Rule 1 drops all packets with destination address 209.85.153.85. Of the remaining packets originating from Z1, F1 forwards only the packets that have Z3's address (i.e., 202.65.23.2) as their destination field towards F2 (following the information in its routing table, which is not shown in the figure); before forwarding these packets to F2 F1 NATs their source address field to 202.67.34.6-10. F2 drops *all* these packets. Hence the above reject formula (which represents all packets dropped by F1 or by F2, in their original form when they left Z1).

5. CONCLUSIONS AND FUTURE WORK

We presented a novel, precise, data-flow analysis for computing packet reachability in IP networks. We also presented an application of this analysis to inferring a high-level policy from a distributed firewall. In the future we would like to extend the analysis to address more complex issues such as connection-oriented routing (i.e., stateful filters), and also to answer (restricted) forms of temporal properties of packet flows.

6. REFERENCES

[1] E. Al-Shaer, W. Marrero, A. El-Atawy, and K. ElBadawi. Network configuration in a box: towards end-to-end verification of network reachability and security. In *ICNP: Proc. IEEE Int. Conf. on Network Protocols*, pages 123–132, 2009.

[2] O. Andreasson. Iptables tutorial 1.2.2. *http://www.frozentux.net/iptables-tutorial/iptables-tutorial.html*.

[3] A. El-Atawy, T. Samak, Z. Wali, E. Al-Shaer, F. Lin, C. Pham, and S. Li. An automated framework for validating firewall policy enforcement. In *POLICY: IEEE Int. Workshop on Policies for Distributed Systems and Networks*, pages 151–160, 2007.

[4] E. Horowitz and L. Lamb. A hierarchical model for firewall policy extraction. In *Int. Conf Advanced Information Networking and Applications*, pages 691–698, 2009.

[5] A. Jeffrey and T. Samak. Model checking firewall policy configurations. In *POLICY '09: IEEE Int. Symp. on Policies for Distributed Systems and Networks*, pages 60–67, 2009.

[6] G. Kildall. A unified approach to global program optimization. In *POPL '73: Proc. ACM Symposium on Principles of Programming Languages*, pages 194–206, New York, NY, USA, 1973.

[7] R. Komondoor, K. V. Lakshmi, D. P. Seetharam, and S. Balodia. Packet flow analysis in IP networks via abstract interpretation. *CoRR*, abs/1111.6808, Nov. 2011.

[8] R. M. Marmorstein and P. Kearns. An open source solution for testing nat'd and nested iptables firewalls. In *LISA: Proc. Conf. on Systems Administration*, pages 103–112, 2005.

[9] P. Matousek, J. Ráb, O. Rysavy, and M. Svéda. A formal model for network-wide security analysis. In *ECBS: IEEE Conf. and Workshop on Engg. of Computer Based Systems*, pages 171–181, 2008.

[10] A. J. Mayer, A. Wool, and E. Ziskind. Offline firewall analysis. *Int. J. Inf. Sec.*, 5(3):125–144, 2006.

[11] A. Tongaonkar, N. Inamdar, and R. Sekar. Inferring higher level policies from firewall rules. In *LISA: Large Installation System Administration Conf.*, pages 17–26, 2007.

[12] A. Wool. A quantitative study of firewall misconfiguration errors. *IEEE Computer*, 37(6), 2004.

[13] G. G. Xie, J. Zhan, D. A. Maltz, H. Zhang, A. G. Greenberg, G. Hjálmtýsson, and J. Rexford. On static reachability analysis of ip networks. In *INFOCOM: Annual Joint Conf. of the IEEE Computer and Communications Societies*, pages 2170–2183, 2005.

[14] L. Yuan, J. Mai, Z. Su, H. Chen, C.-N. Chuah, and P. Mohapatra. Fireman: A toolkit for firewall modeling and analysis. In *S&P: IEEE Symp. on Security and Privacy*, pages 199–213, 2006.