Analysis of Computing Policies Using SAT Solvers (Short Paper)

Marijn J. H. Heule, Rezwana Reaz, H. B. Acharya, and Mohamed G. Gouda

The University of Texas at Austin, United States
{marijn,rezwana,acharya,gouda}@cs.utexas.edu

Abstract. A computing policy is a sequence of rules, where each rule consists of a predicate and a decision, and where each decision is either "accept" or "reject". A policy P is said to accept (or reject, respectively) a request iif the decision of the first rule in P, that matches the request is "accept" (or "reject", respectively). Examples of computing policies are firewalls, routing policies and software-defined networks in the Internet, and access control policies. A policy P is called *adequate* iff P accepts at least one request. It has been shown earlier that the problem of determining whether a given policy is adequate (called the policy adequacy problem) is NP-hard. In this paper, we present an efficient algorithm that use SAT solvers to solve the policy adequacy problem. Experimental results show that our algorithm can determine whether a given policy with 90K rules is adequate in about 3 minutes.

Keywords: Policies, Firewalls, Access Control, Routing Policies, SAT

1 Introduction

A computing policy is a filter that is placed at the entry point of some resource. Each request to access the resource needs to be examined against the policy to determine whether to accept or reject the request.

Examples of computing policies are firewalls in the Internet, routing policies and software-defined networks in the Internet, and access control policies. Early methods for the logical analysis of computing policies have been reported in [6], [5], and [4].

The decision of a policy to accept or reject a request depends on two factors:

1. The values of some attributes that are specified in the request and
2. The sequence of rules in the policy that are specified by the policy designer.

A policy is a sequence of rules where a rule in a policy consists of a predicate and a decision, which is either "accept" or "reject". To examine a request against a policy, the rules in the policy are considered one by one until the first rule, whose predicate matches the values of the attributes in the request, is identified. Then the decision of the identified rule, whether "accept" or "reject", is applied to the request.

A rule in a policy is defined as a pair, one predicate and one decision, written as follows:

$$\langle \text{predicate} \rangle \rightarrow \langle \text{decision} \rangle$$

A rule whose decision is "accept" is called an accept rule, and a rule whose decision is "reject" is called a reject rule.

A predicate is of the form: $\big((u_1 \in X_1) \wedge \cdots \wedge (u_t \in X_t)\big)$, where each u_i is an attribute whose value is taken from an integer interval denoted $D(u_i)$, each X_i is an integer interval that is contained in $D(u_i)$, and each $\wedge$ denotes the logical AND or conjunction operation.

A request is a tuple $(b_1, \ldots, b_t)$ of t integers, where t is the number of attributes and each integer b_i is taken from the domain $D(u_i)$ of attribute u_i. A request $(b_1, \ldots, b_t)$ is said to match a predicate $\big((u_1 \in X_1) \wedge \cdots \wedge (u_t \in X_t)\big)$ iff each integer b_i in the request is an element in the corresponding integer interval X_i in the predicate.

A request is said to match a rule in a policy iff the request matches the predicate of the rule. A policy P is said to accept (or reject, respectively) a request rq iff P has an accept (or reject, respectively) rule r such that request rq matches rule r and does not match any rule that precedes rule r in P.

2 The Policy Adequacy Problem

A policy P is said to be adequate iff there is a request rq that is accepted by P. The policy adequacy problem is to design an efficient algorithm that can take as input any policy P and determine whether P is adequate.

It has been shown in [2] that the time complexity of the policy adequacy problem is NP-hard. In [7], the authors present an algorithm that uses SAT solvers, for example `Glucose` [1], to solve the policy adequacy problem. Unfortunately, the algorithm in [7] is based on rule predicates of a form that is different from the form of the rule predicates described in the current paper. Therefore, the presented algorithm in [7] cannot be applied efficiently to solve the policy adequacy problem described in the current paper.

3 Solving the Policy Adequacy Problem using SAT Solvers

In this paper, we present an algorithm, named Algorithm 1, that uses any SAT solver to solve the policy adequacy problem that is described in this paper. Because of space limitation, our presentation of Algorithm 1 is restricted to the case where Algorithm 1 is applied to the following example policy P:

$$\big((u \in [3,5]) \wedge (v \in [4,4])\big) \rightarrow \text{accept}$$
$$\big((u \in [2,4]) \wedge (v \in [4,4])\big) \rightarrow \text{reject}$$
$$\big((u \in [2,5]) \wedge (v \in [4,4])\big) \rightarrow \text{accept}$$

This example policy has 2 attributes u and v whose value domains are as follows: $D(u) = [1, 4]$ and $D(v) = [1, 4]$. Note that this example policy has 2 accept rules and 1 reject rule.

Our algorithm Algorithm 1 consists of the following 4 steps:

Step 1. In the first step of Algorithm 1, P is encoded into the following Boolean formula FP such that a request rq is accepted by P iff rq makes the value of FP true:

$$FP = \big(\mathrm{ac}(1) \vee \mathrm{ac}(2)\big) \wedge \mathrm{ar}(1) \wedge \mathrm{ar}(2) \wedge \mathrm{rr}(1) \wedge LP$$

Each $\mathrm{ac}(i)$, where $i \in \{1, 2\}$, is a Boolean variable denoting that the i-th accept rule in P is matched by rq. Each $\mathrm{ar}(i)$, where $i \in \{1, 2\}$, is a predicate whose value is true iff $\mathrm{ac}(i)$ is false or request rq matches the i-th accept rule in policy P. Each $\mathrm{rr}(j)$, where $j \in \{1\}$, is a predicate whose value is true iff the j-th reject rule in policy P is preceded by some i-th accept rule where $\mathrm{ac}(i)$ is true or request rq does not match the j-th reject rule in policy P. Predicate LP is discussed below.

Step 2. In the second step of Algorithm 1, we introduce into formula FP Boolean variables that we will use in the third step of the algorithm to encode the predicates $\mathrm{ar}(1)$, $\mathrm{ar}(2)$, $\mathrm{rr}(1)$ and LP.

For each interval $[y, z]$, of an attribute w, that occurs in any rule in policy P, introduce into FP two Boolean variables named $\mathrm{le}(w, y-1)$ and $\mathrm{le}(w, z)$. Therefore, we end-up introducing the following six Boolean variables in this case: $\mathrm{le}(u, 2)$, $\mathrm{le}(u, 5)$, $\mathrm{le}(u, 1)$, $\mathrm{le}(u, 4)$, $\mathrm{le}(v, 3)$, and $\mathrm{le}(v, 4)$.

Step 3. In the third step of Algorithm 1, we use the introduced "le" Boolean variables to encode the predicates $\mathrm{ar}(i)$, $\mathrm{rr}(j)$, and LP as follows.

Let the i-th accept rule in policy P be of the form:

$$u_1 \in [y_1, z_1] \wedge \cdots \wedge u_t \in [y_t, z_t] \rightarrow \text{accept}$$

In this case, predicate $\mathrm{ar}(i)$ can be encoded as follows:

$$\big(\overline{\mathrm{ac}}(i) \vee \overline{\mathrm{le}}(u_1, y_1 - 1)) \wedge (\overline{\mathrm{ac}}(i) \vee \mathrm{le}(u_1, z_1)\big) \wedge$$
$$\cdots$$
$$\big(\overline{\mathrm{ac}}(i) \vee \overline{\mathrm{le}}(u_t, y_t - 1)\big) \wedge \big(\overline{\mathrm{ac}}(i) \vee \mathrm{le}(u_t, z_t)\big)$$

Let the j-the reject rule in policy P be of the form:

$$u_1 \in [y_1, z_1] \wedge \cdots \wedge u_t \in [y_t, z_t] \rightarrow \text{reject}$$

and assume that there are k (note that k can be 0) accept rules that precede the j-th reject rule in P. In this case, predicate $\mathrm{rr}(j)$ can be encoded as follows:

$$\big(\mathrm{ac}(1) \vee \cdots \vee \mathrm{ac}(k) \vee \mathrm{le}(u_1, y_1 - 1) \vee \overline{\mathrm{le}}(u_1, z_1) \vee \cdots \vee \mathrm{le}(u_t, y_t - 1) \vee \overline{\mathrm{le}}(u_t, z_t)\big)$$

Predicate LP in formula FP describes some expected restrictions on the values of the "le" Boolean variables introduced into FP. For example, for the two Boolean variables $\text{le}(u, 2)$ and $\text{le}(u, 5)$ that are introduced into FP, predicate LP should include the clause $(\overline{\text{le}}(u, 2) \vee \text{le}(u, 5))$.

Therefore, we encode the predicates $\text{ar}(1)$, $\text{ar}(2)$, $\text{rr}(1)$, and LP as follows:

$$\text{ar}(1) = \big(\overline{\text{ac}}(1) \vee \overline{\text{le}}(u, 2)\big) \wedge \big(\overline{\text{ac}}(1) \vee \text{le}(u, 5)\big) \wedge \big(\overline{\text{ac}}(1) \vee \overline{\text{le}}(v, 3)\big) \wedge \big(\overline{\text{ac}}(1) \vee \text{le}(v, 4)\big)$$

$$\text{ar}(2) = \big(\overline{\text{ac}}(2) \vee \text{le}(u, 1)\big) \wedge \big(\overline{\text{ac}}(2) \vee \text{le}(u, 5)\big) \wedge \big(\overline{\text{ac}}(2) \vee \overline{\text{le}}(v, 3)\big) \wedge \big(\overline{\text{ac}}(2) \vee \text{le}(v, 4)\big)$$

$$\text{rr}(1) = \big(\text{ac}(1) \vee \text{le}(u, 1) \vee \overline{\text{le}}(u, 4) \vee \text{le}(v, 3) \vee \overline{\text{le}}(v, 4)\big)$$

$$\begin{aligned} LP = {} & \big(\overline{\text{le}}(u, 1) \vee \text{le}(u, 2)\big) \wedge \big(\overline{\text{le}}(u, 2) \vee \text{le}(u, 4)\big) \wedge \\ & \big(\overline{\text{le}}(u, 4) \vee \text{le}(u, 5)\big) \wedge \big(\overline{\text{le}}(v, 3) \vee \text{le}(v, 4)\big) \end{aligned}$$

Step 4. In the fourth step of Algorithm 1, we use any SAT solver, for example `Glucose` [1], to determine whether the above formula FP is satisfiable. The above policy P is adequate iff formula FP is satisfiable.

Complexity. The complexity of Algorithm 1 to determine whether a given policy P is adequate is measured by the number of Boolean variables introduced into formula FP in Algorithm 1.

Note that if the given policy P has n rules and t attributes, then formula FP in Algorithm 1 has $\mathcal{O}(nt)$ Boolean variables and the complexity of Algorithm 1 does not depend on the range of values of the different attributes in policy P.

We performed some experiments to evaluate the effectiveness of Algorithm 1. In each experiment, we applied Algorithm 1 to determine whether a given firewall P selected at random, is adequate. The given firewall P is a policy with 5 attributes and between 10K and 90K rules. The value domain of each attribute in firewall P is the integer interval $[0, 2^{16} - 1]$. The state-of-the-art SAT solver `Glucose` version 3.0 [1] was used to check whether the generated Boolean formula FP is satisfiable[1].

Figure 1 shows the relationship between the number of rules in a given firewall P and the execution time of Algorithm 1 when this algorithm is applied to firewall P to determine whether P is adequate. From Figure 1, the execution time of Algorithm 1 is less than 3 minutes when the given firewall P has up to 90,000 rules.

4 Concluding Remarks

In the full version of this paper [3], we show how to extend Algorithm 1 to solve other policy problems beyond policy adequacy. In particular, we show how to solve the problems of policy completeness, policy implication, policy equivalence, and redundancy checking in policies.

[1] Files are available at `http://www.cs.utexas.edu/~marijn/firewall`

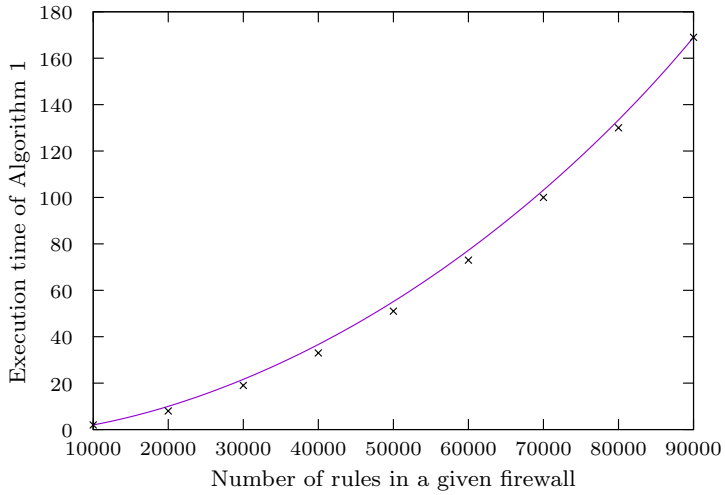

Fig. 1. Execution time (in seconds) of Algorithm 1 to determine whether a given firewall is adequate.

5 Acknowledgements

Research of M. J. H. Heule is supported by DARPA Contract FA8750-15-2-0096 and NSF Award CCF-1526760. Research of M. G. Gouda is supported by NSF Award 1440035.

References

1. Audemard, G., Simon, L.: Predicting learnt clauses quality in modern SAT solvers. In: Boutilier, C. (ed.) IJCAI 2009. pp. 399–404 (2009)
2. Elmallah, E.S., Gouda, M.G.: Hardness of firewall analysis. In: NETYS, LNCS, vol. 8593, pp. 153–168. Springer (2014)
3. Heule, M.J.H., Reaz, R., Acharya, H.B., Gouda, M.G.: Analysis of computing policies using sat solvers. In: Technical Report No. TR-16-14, Department of Computer Science, The Universisty of Texas at Austin (2016)
4. Hoffman, D., Yoo, K.: Blowtorch: a framework for firewall test automation. In: Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE). pp. 96–103. ACM (2005)
5. Kamara, S., Fahmy, S., Schultz, E., Kerschbaum, F., Frantzen, M.: Analysis of vulnerabilities in internet firewalls. Computers & Security 22(3), 214–232 (2003)
6. Mayer, A., Wool, A., Ziskind, E.: Fang: A firewall analysis engine. In: IEEE Symposium on Security and Privacy. pp. 177–187. IEEE (2000)
7. Zhang, S., Mahmoud, A., Malik, S., Narain, S.: Verification and synthesis of firewalls using SAT and QBF. In: Proceedings of the 20th IEEE International Conference on Network Protocols (ICNP). pp. 1–6. IEEE (2012)