Consistency verification of stateful firewalls is not harder than the stateless case

LEVENTE BUTTYÁN, GÁBOR PÉK, TA VINH THONG

Laboratory of Cryptography and Systems Security
Budapest University of Technology and Economics, Department of Telecommunications
{buttyan, pek, thong}@crysys.hu

Keywords: ?????

Firewalls play an important role in the enforcement of access control policies in contemporary networks. However, firewalls are effective only if they are configured correctly such that their access control rules are consistent and the firewall indeed implements the intended access control policy. Unfortunately, due to the potentially large number of rules and their complex relationships with each other, the task of firewall configuration is notoriously error-prone, and in practice, firewalls are often misconfigured leaving security holes in the protection system. In this paper, we address the problem of consistency verification of stateful firewalls that keep track of already existing connections. For the first sight, the consistency verification of stateful firewalls appears to be harder than that of stateless firewalls. We show that, in fact, this is not the case: consistency verification of stateful firewalls can be reduced to the stateless case, and hence, they have the same complexity. We also report on our prototype implemetation of an automated consistency verification tool that can handle stateful firewalls.

1. Introduction

Firewalls are the cornerstones of improving the security of enterprise networks. Simple packet filter firewalls work in a stateless manner: they inspect the packets passing through the perimeter of the network as independent objects, and decide to accept or deny them according to a predefined static ruleset. Modern firewalls, however, are more complex and perform stateful packet inspection: they keep track of the already existing connections and they decide about the fate of a packet based on both its header information and the state of the connection that it belongs to.

In practice, firewalls are often misconfigured. Misconfiguration errors result in inconsistencies in the firewall [1,7]. An example for a critical inconsistency is when all the packets that are intended to be denied by a given rule of the firewall are accepted by some preceding rules. This is called shadowing, because the intended effect of the deny rule is cancelled by the preceding accept rules. Shadowing is a critical inconsistency, because it is likely that the deny rule is there to stop some well-known malicious traffic, however, due to the shadowing, that traffic is not actually stopped by the firewall. Such inconsistencies can easily occur when the firewall has a distributed implementation and/or when it is managed by multiple administrators; both being frequent cases in large organizations.

Checking a large firewall (i.e., hundreds of access control rules) for inconsistencies is difficult and prone to errors when it is done in an ad-hoc, non-systematic manner. Thus, several formal methods and automated tools have been proposed in the literature [1-12], but essentially all of them were designed for finding inconsistencies in stateless firewalls. However, stateful firewalls are much more broadly used nowadays due to their connection tracking feature. Finding inconsistencies in stateful firewalls has been considered to be harder than the stateless case due essentially to the potentially very large size of the state space. A first attempt to model stateful properties of firewalls is presented in [13], but that work does not propose any method to find inconsistencies in stateful firewalls.

In this paper, we propose a modeling technique for states and, for the first time, a systematic method for detecting inconsistencies in stateful firewalls. We model a state as a particular subset of the firewall ruleset that consists of all the static rules and those dynamic rules that are relevant in the given state. We show that the number of inconsistencies in any state cannot be larger than the number of inconsistencies in the designated state that includes all dynamic rules. Hence, it is sufficient to check that single designated state for inconsistencies: if no inconsistency is found in that state, then no other state can contain any inconsistencies. Moreover, if the designated state is not free of inconsistencies, then this fact proves that the firewall is inconsistent in at least one state (the designated one). Therefore, we essentially reduce the consistency verification of a stateful firewall to the consistency verification of a single static ruleset.

In order to automate the verification, we implemented a software tool in C#, which is capable of finding inconsistencies in the configuration of stateful firewalls. Our tool is based on FIREMAN [7], an approach which was originally developed for stateless firewalls. We note, however, that the real power of our approach is that *any* stateless tool could have been used. We used the

FIREMAN approach beacuse it uses Binary Decision Diagrams (BDD) for handling IP range set operations, such as intersection and union, and BDDs are conceptually simple and very efficient.

The rest of the paper is organized as follows: We define inconsistencies and inefficiencies in Section 2. We give a brief overview of the connection-tracking feature of contemporary firewalls in Section 3. In Section 4 we introduce our main theorems and their proofs, while Section 5 reports on our implementation. Finally, we conclude the paper and give some future plans in Section 6.

2. Inconsistencies and inefficiencies in firewalls

The configuration of a firewall consists in the ruleset that the firewall uses for filtering the traffic. A stateless rule is represented in the form *<P, action>*, where *P* corresponds to a predicate describing the criteria that a packet has to meet to match the rule, and *action* is the corresponding action that is executed when there is a match to the rule. In case of stateless rules, predicate *P* can be represented as a 5-tuple *(prot, srcaddr, srcport, dstaddr, dstport)*, where *prot* refers to a protocol (tcp, udp, icmp), *srcaddr* is the source IP address range, *srcport* is the source port range, *dstaddr* is the destination IP address range, and *dstport* is the destination port range that should be matched by a packet. In addition, an action can be *accept* or *deny*, the meaning of which should be intuitively clear.

The ruleset of a firewall may be inconsistent and/or inefficient. In this paper, we consider three types of inconsistencies: *shadowing, generalization,* and *correlation;* and one inefficiency: *redundancy*. These have also been considered in prior works of others [1,7], but in a stateless environment. In Section 4, we show that they can be defined in the case of stateful firewalls as well. In this section, we give the definitions of these inconsistencies and inefficiencies.

We use the following notation: Let *R* be a ruleset that consists of stateless rules $r_i = <P_i, action_i>$, where $P_i = (prot_i, srcaddr_i, srcport_i, dstaddr_i, dstport_i)$. We say that $(Pj \subseteq Pi)$ if $(prot_i \subseteq prot_j) \wedge (srcaddr_i \subseteq srcaddr_j) \wedge (srcport_i \subseteq srcport_j) \wedge (dstaddr_i \subseteq dstaddr_j) \wedge (dstport_i \subseteq dstport_j)$. Similarly, $(P_j \cap P_i \neq 0)$ if $(prot_i \cap prot_j \neq 0) \wedge (srcaddr_i \cap srcaddr_j \neq 0) \wedge (srcport_i \cap srcport_j \neq 0) \wedge (dstaddr_i \cap dstaddr_j \neq 0) \wedge (dstport_i \cap dstport_j \neq 0)$.

A rule is shadowed by a preceding rule if it is a subset of the preceding rule; and the two rules define different actions:

Definition (Shadowing) *Rule* $(r_i = <P_i, action_i>) \in R$ *shadows rule* $(r_j = <P_j, action_j>) \in R$ *if and only if* $(i < j) \wedge (P_j \subseteq P_i) \wedge (action_i \neq action_j)$, where *i* and *j* denote the order of rules in a ruleset *R*.

A rule is a generalization of a preceding rule if it is a superset of the preceding rule and the two rules define different actions:

Definition (Generalization) *Rule* $(r_i = <P_i, action_i>) \in R$ *is the generalization of rule* $(r_j = <P_j, action_j>) \in R$ *if and only if* $(i > j) \wedge (P_j \subseteq P_i) \wedge (action_i \neq action_j)$.

Two rules are correlating if their intersection is not empty, they are not related by the superset or subset relations, and they define different actions. Packets that match the intersection will take the action of the preceding rule:

Definition (Correlation) *Rule* $(r_i = <P_i, action_i>) \in R$ *and* $(r_j = <P_j, action_j>) \in R$ *are correlating if and only if* $(P_j \cap P_i \neq 0) \wedge (P_j \not\subseteq P_i) \wedge (P_i \not\subseteq P_j) \wedge (action_i \neq action_j)$.

A rule is redundant if the removal of it would not affect the operation of the firewall. In case of masked redundancy (defined below) the successor rule is unnecessary, while in case of partially masked redundancy (also defined below) the preceding rule is unnecessary:

Definition (Redundancy) *Rule* $(r_i = <P_i, action_i>) \in R$ *is redundant with respect to rule* $(r_j = <P_j, action_j>) \in R$ *if and only if at least one of the following two conditions are satisfied:*

Masked redundancy:

$$(P_i \subseteq P_j), \text{ where } (i > j) \wedge (action_i = action_j)$$

Partially masked redundancy:

$$(P_i \subseteq P_j), \text{ where } (i < j) \wedge (action_i = action_j)$$

Note that not all these inconsistencies and redundancies are equally critical. Usually, only shadowing is considered to be a configuration error, while generalization and correlation are in fact often used by firewall administrators to make a ruleset compact. Nevertheless, it may be the case that some of the generalizations and correlations are not intentional, in which case, it is useful to detect them and let the administrator decide if they are harmful or not. Redundancy is not considered a serious configuration error either, but redundant rules are clearly useless, therefore, it is worth identifying and removing them, and increasing the efficiency of filtering by doing so.

3. Connection-tracking with iptables

In order to understand the model described in the next section, we shortly review how connection-tracking works in iptables, a stateful firewall that we used in our work. Other stateful firewalls work in a similar manner.

Iptables defines tables and chains to complete certain operations on packets at different points of the checking. We consider only the input and output chains, and the filter table for demonstration purposes. An extensive description of iptables can be found in [15].

Connection-tracking is the basis of stateful firewalls. It refers to the ability to maintain state information about a connection as an entry in a state table. Entries are inserted in and removed from the state table according to the packets the firewall is examining. For instance, we demonstrate how connection-tracking tracks a TCP connection establishment.

Suppose we have the following rules in the output and input chains of the filter tables, respectively:

1. *iptables –A OUTPUT -p tcp -m state –state NEW, ESTABLISHED -j ACCEPT*;
2. *iptables –A INPUT -p tcp -m state –state ESTABLISHED -j ACCEPT*.

Connection-tracking classifies each packet as being in different states: *NEW* (if the packet initiates a new connection), *ESTABLISHED* (if the packet is associated with a connection that has encountered packets in both directions), *RELATED* (if the packet initiates a new connection, but also associated with an already established connection), *INVALID* (not part of an existing connection). For instance, the second rule above means that only packets that belong to an established connection are permitted to enter the network.

Once a *syn* packet that initiates a TCP connection is sent in the output chain, and accepted by the first rule above that allows a *NEW* connection, the following connection table entry is created:

tcp 6 54 SYN_SENT src=10.0.0.1 dst=154.32.43.44
sport=1506 dport=22 [UNREPLIED]
src=154.32.43.44 dst=10.0.0.1 sport=22 dport=1506 use=1

Here, *tcp* refers to the protocol of the connection (and 6 is its numerical form), the remaining time before removal of this entry is 54 seconds, *SYN_SENT* is the tcp state of the connection, *src* and *dst* are the source and destination IP addresses, *sport* and *dport* are the source and destination ports of the connection, and *UNREPLIED* refers to the connection-tracking state of the connection. In the following, the addresses and ports are listed in reverse order for the response traffic.

When a *syn+ack* packet arrives, the entry in the connection tracking table is modified as follows:

tcp 6 60 SYN_RCVD src=10.0.0.1 dst=154.32.43.44
sport=1506 dport=22
src=154.32.43.44 dst=10.0.0.1 sport=22 dport=1506 use=1

One can see that the TCP connection state changes to *SYN_RCVD*, while the tracked connection-state changes from *NEW* to *ESTABLISHED*. Note that the tracked connection states (*NEW, ESTABLISHED,* etc.) are different from the TCP connection establishment states (*SYN_SENT, SYN_RCVD,* etc.).

Finally, when the last part of the three-way TCP connection establishment handshake, an *ack* packet arrives from the server, the connection-tracking entry becomes:

tcp 6 43 1995 ESTABLISHED src=10.0.0.1 dst=154.32.43.44
sport=1506 dport=22 [ASSURED]
src=154.32.43.44 dst=10.0.0.1 sport=22 dport=1506 use=1

The TCP state of the connection is altered to *ESTABLISHED* and the connection-tracking state of the connection is modified to *ASSURED. ASSURED* connections are not dropped from the state table when the connection is overloaded. Note that the remaining time value is increased to a previously defined timeout value.

4. Verification of stateful firewalls

In case of a stateless firewall, inconsistencies and inefficiencies between rules can be detected by means of static analysis of the ruleset. In case of a stateful firewall, the detection appears to be harder, because the static analysis has to be performed in all possible states of the firewall in order to be sure that the ruleset always remains consistent. In this section, we show that this is indeed not the case, and it is sufficient to verify a single designated state for inconsistencies in order to prove that the firewall's rule set is consistent in all possible states.

For doing so, we must first introduce the notion of *firewall state*:

Definition (Stateful rules) A firewall rule is said to be stateful if it defines state information, and is presented in the form *<P, action, stateinfo>*.

Definition (Firewall state) *The states of a firewall includes all the static firewall rules and those dynamic (stateful) rules that have an associated entry in the connection-tracking table.*

As an example, let us consider the following rule set, where the first three rules are dynamic (stateful) rules and the fourth rule is a static rule:

```
Rule 1: iptables –A OUTPUT –s 10.0.0.1
–dport 80 –m state – state NEW, ESTABLISHED –j ACCEPT
Rule 2: iptables –A OUTPUT –s 10.0.0.1
–dport 443 –m state – state NEW, ESTABLISHED –j ACCEPT
Rule 3: iptables –A OUTPUT –s 10.0.0.1
–dport 22 –m state – state NEW, ESTABLISHED –j ACCEPT
Rule 4: iptables –A OUTPUT –s 10.0.0.1
–dport 22 –j DROP
```

In addition, let us suppose that the following two entries have been created in the connection-tracking table (as the result of processing some packets earlier):

```
1. tcp 6 54 SYN_SENT src=10.0.0.1
   dst=154.32.43.44 sport=6322 dport=80 [UNREPLIED]
   src=154.32.43.44 dst=10.0.0.1 sport=80 dport=6322 use=1
2. tcp 6 432 ESTABLISHED src=10.0.0.1
   dst=154.32.43.44 sport=1506 dport=443
   [ASSURED] src=154.32.43.44 dst=10.0.0.1 sport=443
   dport=1506 use=1
```

As one can see, in this state, Rules 1 and 2 have associated entries in the connection-tracking table, while Rule 3 has no such entry. This means that in this state, no packet can match Rule 3, and therefore, it can be ignored. At the same time, packets may match Rules 1 and 2, due to the entries in the connection-tracking table, and packets may also match Rule 4, as it is a static rule (i.e., independent of any states). For this reason, Rules 1, 2, and 4 must be considered in this parti-

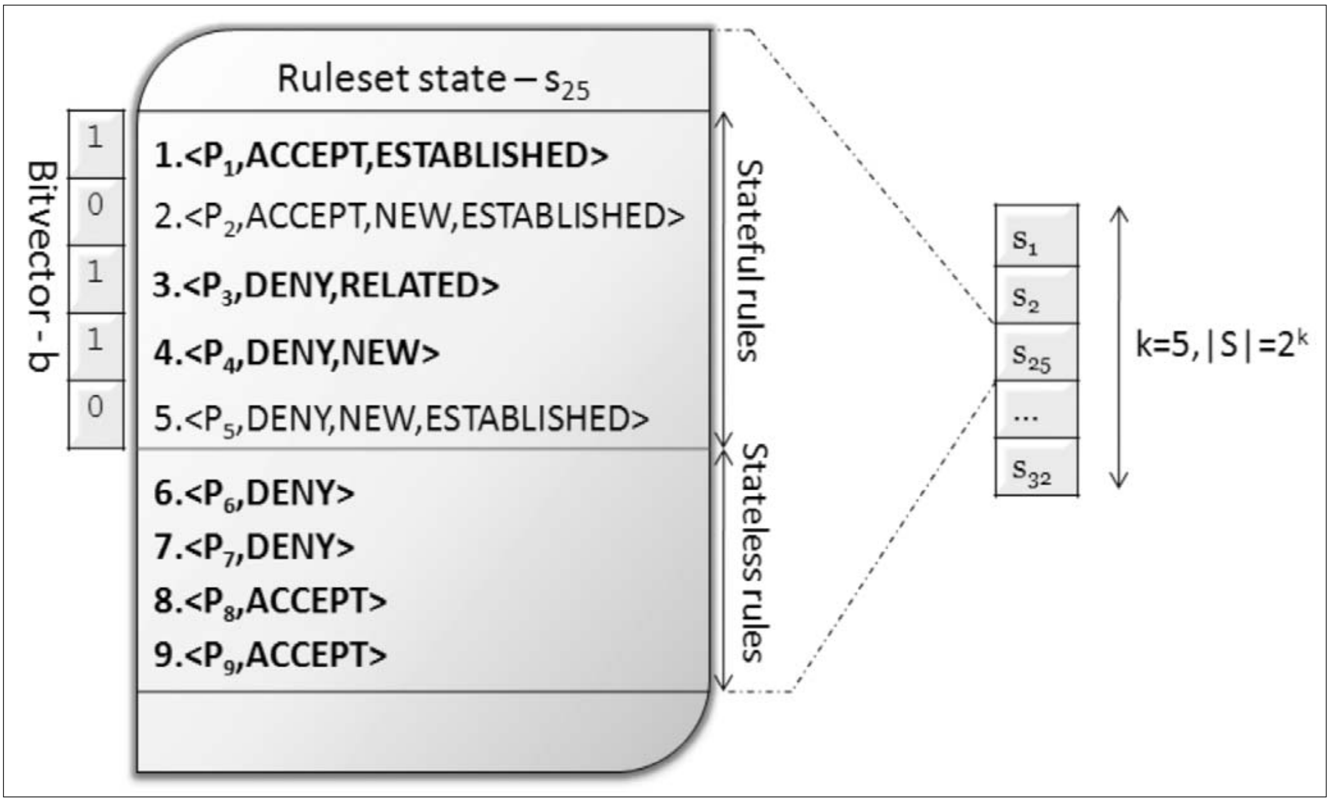


Figure 1. Encoding the firewall state as a binary vector

cular state. This means that, essentially, the state of the firewall can be represented by those three rules.

It is natural to encode such a firewall state as a binary vector the length of which is equal to the number *k* of the dynamic (stateful) rules in the rule-set. This is illustrated in *Figure 1*. It trivially follows that the number of all possible firewall states is 2^k.

We can now introduce a partial ordering ≤ on the set *S* of all possble states:

Definition (Partial ordering of firewall states) *Let s and s' be two states of the same firewall (i.e., two binary vectors of the same length). We have s ≤ s' if all the dynamic rules that are included in s are also included in s'. In other words, if the i-th element of the binary vector corresponding to s is 1, then the i-th element of the binary vector corresponding to s' is also 1.*

The key idea of our work is that we show that whenever *s* ≤ *s'*, the number of inconsistencies in *s* cannot be larger than the number of inconsistencies in *s'*. The next theorem states this for the number of shadowings:

Theorem (Shadowing) *Let s and s' be two states of the same firewall such that s ≤ s'. The number of shadowings in s cannot be larger then the number of shadowings in s'.*

Proof: Without loss of generality, we can assume that the *i*-th stateful rule r_i of the firewall rule-set is included in *s'*, otherwise *s'* contains no stateful rules, which means that *s* = *s'*, and the statement of the theorem follows trivially. Let us denote by *s"* the state that we obtain from *s'* by removing rule r_i.

Now, if there exists a (stateful or stateless) rule r_j of the same firewall, such that either r_i shadows r_j or r_j shadows r_i, then removing r_i from *s'* and obtaining *s"* surely decreases the number of shadowings. Otherwise, if no rule shadows r_i and no rule is shadowed by r_i, then removing r_i from *s'* and obtaining *s"* does not affect the number of shadowings.

As state *s* can be obtained from *s'* by iteratively removing from *s'* the dynamic rules that are not contained in *s*, the statement of the theorem can be obtained by iteratively using the above argument. ♣

Similar theorems can be stated and proven in the same way for the other types of inconsistencies and inefficiencies (see [14] for details). This leads to the following main theorem:

Theorem (Reduction to the stateless case) *Let s_{all-1} be the state that contains all dynamic rules of the rule set. If no inconsistencies and inefficiencies exist in state s_{all-1}, then all states are free from inconsistencies and inefficiencies, and hence, the firewall configuration is correct.*

Proof: Immediately follows from the fact that $s \leq s_{all-1}$ for any state *s* of the firewall. ♣

The consequence is that it is sufficient to verify the firewall in state s_{all-1} for inconsistencies, and this can be done by using any static analysis tool.

Note that for the sake of this static analysis, the dynamic rules are converted to static rules by ignoring those parts of their predicate that refer to some state information.

5. Implementation

In our implementation, we used the approach called FIREMAN [7], which applies static analysis techniques to check misconfigurations, such as policy violations, inconsistencies and inefficiencies in individual firewalls as well as in distributed firewalls using symbolic model checking and Binary Decision Diagrams (BDD). Based on the concepts of FIREMAN, we implemented the methodology of stateful verification described in the previous chapter as a software tool. In the rest of this section, we briefly explain the operation of FIREMAN, and hence, our tool.

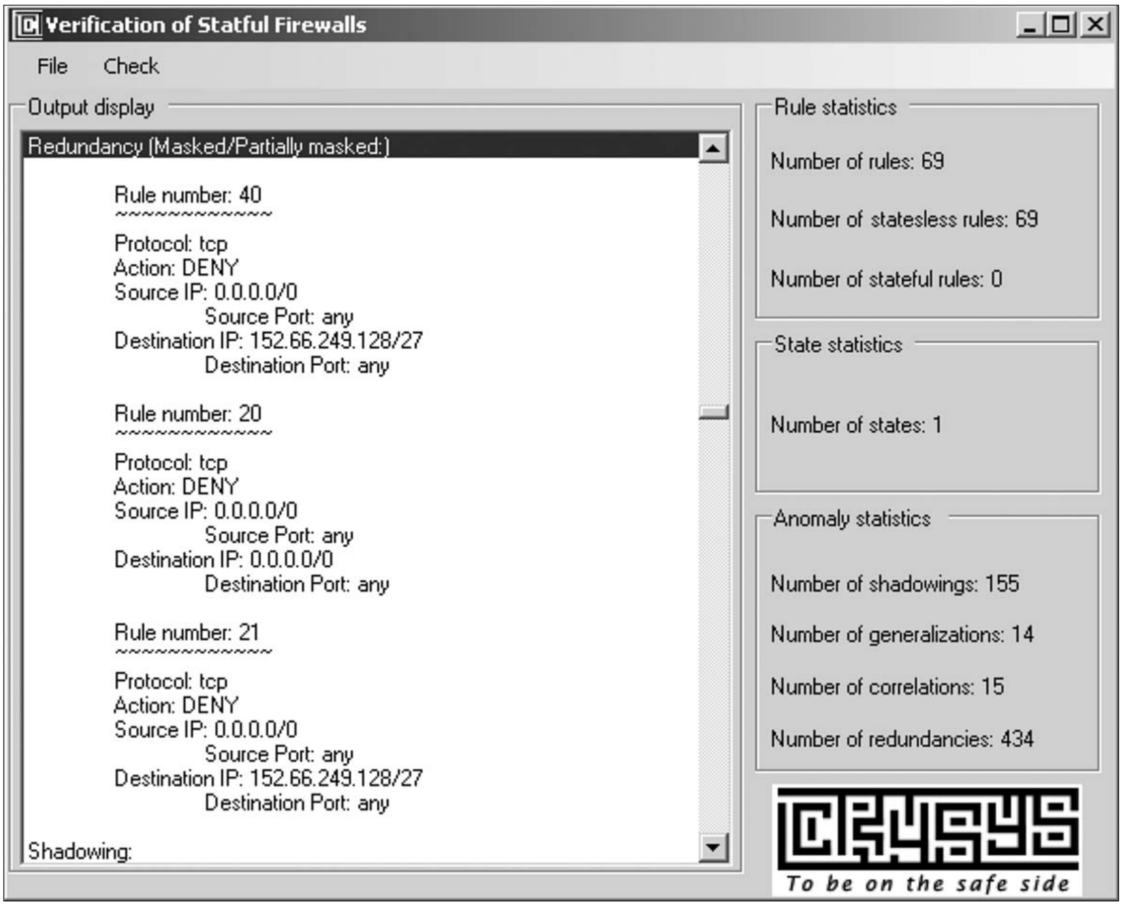
Inspired by the successfully applied software implementations of the previous works [1,7] a new application was implemented in C# that is capable of verifying a stateful firewall configuration. First of all, this tool builds upon the methodology of the aforementioned works, but uses its own Binary Decision Diagram class, to make calculations (union, intersection, subset) on IP ranges as quickly as possible.

Binary Decision Diagram is a data structure which can represent Boolean functions. It is a rooted, acyclic, directed graph which comprises several non-terminal (decision) nodes and terminal nodes with assigned value either 1 (the Boolean function is true) or 0 (the Boolean function is false). When an IP range is presented in BDD form, the number of non-terminal nodes is given by the length of the corresponding netmask. Each decision node is one of the variables of the Boolean function. Interested redears are referred to [7,14] for more details and examples.

In the following, the functioning of the application is presented. First and foremost, a valid iptables rule file has to be opened. Right after it, the application parses the rules of the file one-by-one, and tries to recognize the given parameters and their values. If a suggested parameter is not set, then default values are used instead.

An example is when one does not specify explicitly the destination port in a corresponding rule. In this case, all packets carrying one of the valid ports in the range [1, 65535] are accepted. When one rule is parsed then it is compared against with the already stored preceding rules at once. This is the task of the static analysis method that was previously mentioned. Naturally, the trivial translation of stateful rules into stateless ones is done when the current rule has state information.

According to the definition of firewall state, there is no need to distinguish rules with different state information (*NEW, ESTABLISHED, RELATED,* etc.), so they are handled uniformly. As it was explained previously, there is only one state $s_{all\text{-}1}$, which contains all the stateful (and stateless rules), that has to be checked. There are two internal lists defined, where the parsed rules are put: AcceptList and DenyList, where AcceptList contains rules with action *ACCEPT* and DenyList stores rules with action *DENY*. Note that iptables defines two declining action values: *REJECT* and *DROP*. In fact each of them refers to the internal representation of *DENY*.



```
Redundancy (Masked/Partially masked:)

        Rule number: 40
        ~~~~~~~~~~~~~~
        Protocol: tcp
        Action: DENY
        Source IP: 0.0.0.0/0
                Source Port: any
        Destination IP: 152.66.249.128/27
                Destination Port: any

        Rule number: 20
        ~~~~~~~~~~~~~~
        Protocol: tcp
        Action: DENY
        Source IP: 0.0.0.0/0
                Source Port: any
        Destination IP: 0.0.0.0/0
                Destination Port: any

        Rule number: 21
        ~~~~~~~~~~~~~~
        Protocol: tcp
        Action: DENY
        Source IP: 0.0.0.0/0
                Source Port: any
        Destination IP: 152.66.249.128/27
                Destination Port: any

Shadowing:
```

Number of rules: 69

Number of stateless rules: 69

Number of stateful rules: 0

Number of states: 1

Number of shadowings: 155

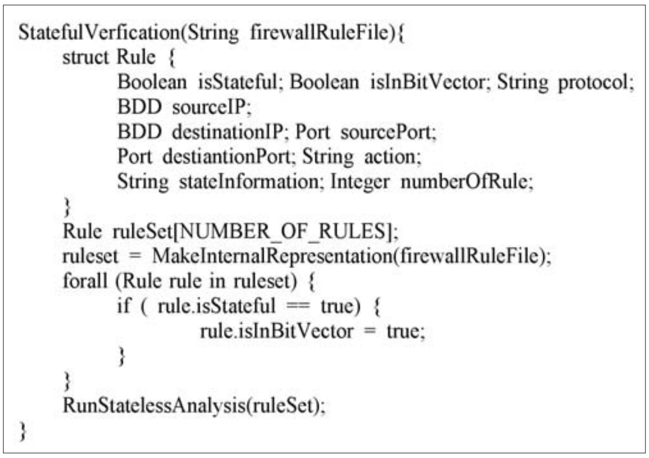
Number of generalizations: 14

Number of correlations: 15

Number of redundancies: 434

Figure 2. Screen shot of the prototype implementation

The pseudo-code in *Table 1* demonstrates the core of stateful verification.



```
StatefulVerfication(String  firewallRuleFile){
      struct Rule  {
            Boolean  isStateful; Boolean  isInBitVector; String  protocol;
            BDD  sourceIP;
            BDD  destinationIP; Port  sourcePort;
            Port  destiantionPort; String  action;
            String  stateInformation; Integer  numberOfRule;
      }
      Rule ruleSet[NUMBER_OF_RULES];
      ruleset = MakeInternalRepresentation(firewallRuleFile);
      forall (Rule rule in ruleset) {
            if ( rule.isStateful == true) {
                  rule.isInBitVector = true;
            }
      }
      RunStatelessAnalysis(ruleSet);
}
```

Table 1. The pseudo code of stateful verification

It is essential to put efforts on the demonstration of the application by verifying firewalls that are used in practice. In order to satisfy these kind of requirements two firewalls at BME are analyzed by means of the implemented tool. The machine that we used for the verification was an IBM Thinkpad R40 notebook with Intel Pentium 4-M processor and 512 MB DDR RAM.

As *Figure 2* shows, many inconsistencies and inefficiencies have been found among the firewall rules. In detail, there are around 70 firewall rules among which there are 155 shadowings, 14 generalizations, 15 correlations and 434 redundancies. The verification time needed to discover all these inconsistencies and inefficiencies required less than 4 sec.

6. Conclusions and future work

So far, formal methods have been considered only for the verification of stateless firewall. In this paper, we proposed, for the first time, a formal verification method for stateful firewalls.

Our contributions are three-fold. First, we introduced a modelling technique for states, and defined the notion of inconsistency in case of the stateful environment. Second, we reduced the problem of verifying a stateful firewall to the problem of verifying a stateless firewall. More specifically, we proved that if the firewall configuration is free from inconsistencies and inefficiencies in a designated state, then it is free from these anomalies in all states. Third, we implemented our approach as a prototype stateful firewall verification tool, and used it for verifying real firewalls used in practice. Our experiments show that the tool is effective and efficient.

Regarding future work, there are many possible improvements that are yet to be done. Our approach could be extended to distributed firewalls, where multiple filters organized in some topology must function together without anomalies. It would also be interesting to extend this approach to the complex chain model of iptables.

Finally, the implementation can be extended to support other firewall products too, such as the Checkpoint FireWall-1 and Cisco ASA.

Acknowledgments

The work presented in this paper has been partially supported by Ericsson through the HSN Laboratory at the Budapest University of Technology and Economics. Apart from this, Ericsson has no responsibility for the content of this paper.

The authors are thankful to Boldizsár Bencsáth for his help in understanding how iptables works and for his useful comments on firewall management in practice.

Authors

LEVENTE BUTTYÁN received the M.Sc. degree in Computer Science from the Budapest University of Technology and Economics in 1995, and the Ph.D. degree from the Swiss Federal Institute of Technology in Lausanne in 2002. He joined the Department of Telecommunications at the Budapest University of Technology and Economics (BME) in January 2003, where he currently holds a position as an Associate Professor. His current research interests include security and privacy problems in wireless networked embedded systems, and the application of formal methods in security engineering. He has supervised several international projects at BME (e.g., UbiSecSens, SeVeCom, EU-MESH, WSAN4CIP), and he has been teaching courses on network security and electronic commerce in the MSc program. In addition, he is an Associate Editor of the IEEE Transactions on Mobile Computing, Area Editor of Elsevier Computer Communications, member of the editorial board of the Infocommunications Journal in Hungary, and he was a Guest Coeditor of the IEEE Journal on Selected Areas in Communications, Special Issue on Non-cooperative Behavior in Networking. He is a Steering Committee member of the ACM Conference on Wireless Network Security. In the last 6 years, he served on the Program Committee of around 30 international conferences and workshops, most of which were related to wireless network security.

GÁBOR PÉK received the B.Sc. degree in Computer Science in 2009 from the Budapest University of Technology and Economics, where he is currently doing his M.Sc. studies. He has been working on his project laboratories on several fields of security in collaboration with Levente Buttyán and his group for 1,5 years. In addition, he has experience in ethical hacking and forensics and was active on several industrial projects. His work on the analysis of stateful firewalls was awarded at the scientific student conferences (TDK) at the university and at the national levels. He also has internationally accepted publications in robot navigation algorithms. Currently, he is interested in malicious codes, botnets and mobile security.

TA VINH THONG received the M.Sc. degree in Computer Science from the Budapest University of Technology and Economics (BME). Since 2008, he has been working as a PhD student in the Laboratory of Cryptography and System Security (CrySyS), Department of Telecommunications, BME. His research interest is analyzing security systems using formal methods, especially, formal analysis of security protocols. His current research activities are formal verification of MANETs and formal languages.

References

[1] E. Al-Shaer and H. Hamed, "Design and Implementation of Firewall Policy Advisor Tools." Technical Report CTI-techrep0801, School of Computer Science Telecommunications and Information Systems, DePaul University, August 2002.

[2] Ehab S. Al-Shaer and Hazem H. Hamed, Firewall policy advisor for anomaly discovery and rule editing. In: Integrated Network Management, pp.17–30, 2003.

[3] Ehab Al-Shaer and Hazem Hamed, Management and Translation of Filtering Security Polices, IEEE ICC'03, May 2003.

[4] Ehab Al-Shaer and Hazem Hamed, Modeling and Management of Firewall Policies, IEEE Transact. on Network and Service Management, Vol.1, April 2004.

[5] Ehab Al-Shaer, Hazem Hamed, Raouf Boutaba and Masum Hasan, Conflict Classification and Analysis of Distributed Firewall Policies, IEEE Journal on Selected Areas in Communications, Issue 10, Vol. 23, pp.2069–2084, October 2005.

[6] E. Lupu and M. Sloman, "Conflict Analysis for Management Policies." In Proceedings of IFIP/IEEE International Symposium on Integrated Network Management, May 1997.

[7] L. Yuan, J. Mai, Z. Su, H. Chen, C. Chuah and P. Mohapatra, FIREMAN: A toolkit for firewall modeling and analysis. In IEEE Symposium on Security and Privacy, pp.199–213, 2006.

[8] Florin Baboescu and George Varghese, Fast and scalable conflict detection for packet classifiers. Computer Networks 42(6), pp.717–735, 2003.

[9] Venanzio Capretta, Bernard Stepien, Amy Felty and Stan Matwin, Formal correctness of conflict detection for firewalls. In Proceedings of the ACM workshop on Formal Methods in Security Engineering (FMSE'07), pp.22–30, 2007.

[10] D. Eppstein and S. Muthukrishnan, "Internet Packet Filter Management and Rectangle Geometry." In Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), January 2001.

[11] P. Eronen and J. Zitting, "An Expert System for Analyzing Firewall Rules." In Proceedings of the 6th Nordic Workshop on Secure IT-Systems (NordSec 2001), November 2001.

[12] B. Hari, S. Suri and G. Parulkar, "Detecting and Resolving Packet Filter Conflicts." In Proceedings of IEEE INFOCOM'2000, March 2000.

[13] Mohamed G. Gouda and Alex X. Liu, A model of stateful firewalls and its properties. In Proceedings of the IEEE Int. Conf. on Dependable Systems and Networks, Yokohama, Japan, June 2005.

[14] Gábor Pék, Security Verification of Stateful Firewalls, Student Scientific Conference (TDK), Budapest, 2008.

[15] Oskar Andreasson, Iptables tutorial: http://iptables-tutorial.frozentux.net/iptables-tutorial.html

[16] A. Liu, E. Torng, and C. Meiners, Firewall Compressor: An algorithm for minimizing firewall policies. In Proceedings of the 27th Annual IEEE Conference on Computer Communications, 2008.