

# Managing intrusion detection rule sets

Natalia Stakhanova  
Information Security Centre of Excellence  
University of New Brunswick  
Fredericton, Canada  
natalia@unb.ca

Ali A.Ghorbani  
Information Security Centre of Excellence  
University of New Brunswick  
Fredericton, Canada  
ghorbani@unb.ca

## ABSTRACT

The prevalent use of the signature-based approach in modern intrusion detection systems (IDS) emphasizes the importance of the efficient management of the employed signature sets. With the constant discovery of new threats and vulnerabilities, the complexity and size of signature sets reach the point where the manual management of rules becomes a challenging (if not impossible) task for the system administrators. While the automated support of signature management is desirable, the main difficulty that arises in this context is the diversity in syntactical representations of signatures generally allowed in IDS. In this paper, we focus on the automated approach to signature management. Specifically, we propose a model for signature analysis that brings out the semantic inconsistencies in the IDS rule sets. To address the syntactical diversity of the signatures, we use the strengths of a nondeterministic automaton (NFA) and model the individual rules as finite machines to analyze their equivalence. The effectiveness of the proposed approach is evaluated on two collections of attack signatures: the rule sets of the open source Snort IDS and Bleeding Edge Threats.

## 1. INTRODUCTION

Intrusion detection has been at the center of intense research in the last decade owing to the rapid increase of sophisticated attacks on computer systems. Generally, the signature-based approach is the most common technique implemented in commercial intrusion detection systems (IDS). It aims to detect attacks by analyzing the event stream against a predefined set of rules, i.e., attack signatures.

Although the signature-based IDS provides many benefits including accurate detection of known attacks and indication of suspicious behavior, it is highly dependent on expert knowledge in developing and maintaining the detection rules. Constant discovery of the software vulnerabilities and novel threats demands timely addition of new attack signatures, which significantly increases the complexity of the set resulting in overlapping and often redundant signatures.

This situation is aggravated by the fact that in large-scale systems a variety of intrusion detection tools are often bundled together to provide comprehensive security coverage of the individual hosts and network. In addition, most of the off-shelf IDS products include prewritten signatures that over time are customized and periodically updated by different administrators. This results in a complex structure with a cumbersome set of attack signatures.

Traditionally, the management of IDS signatures is a manual process assumed to be a part of system security administrator's responsibility. While this approach might be feasible for small sets of rules (e.g., OSSEC IDS [18]), its inefficiency becomes clear when we consider, for example, a widely used Snort IDS [2] and one of its latest VRT Certified Rules Release v2.8 which includes more than 9000 rules.

While the automated support for such sets is highly beneficial, this process is challenging mainly due to the diverse nature of IDS rules that offer a variety of checks from simple matching of the IP addresses and checks for permission change on file access to complex string matching with regular expressions. Although this diversity provides the administrator with great flexibility in the rule customization process, it allows a variety of syntactical representations of the same rule content. For example, in Snort rules the search for the URL string "hex.cgi" can be defined using three different options: content, urlcontent or pcre expression. The selection of a particular option varies among administrators and essentially depends on the administrator's experience, knowledge and preferences. This diversity presents the main challenge during signature set updates leading to wasted time, redundant and overlapping signatures.

In this work, we focus on a method that can significantly improve the security management tasks, on the one hand, guiding administrators during the attack signature writing process, and on the other hand, helping to manage already existing IDS rule sets.

In this paper, we propose a novel approach to the analysis of intrusion detection rules for semantical inconsistencies based on a non-deterministic finite automaton (NFA). The key component of developing a successful technique for automated analysis is a uniform representation of IDS rules that can effectively accommodate the diversity in a rule format and exhibit the rule semantics. To address this, we utilize a finite automaton as a modeling ground that allows one to represent the semantics of the individual rules uniformly. Based on the traditional notion of automata equivalence, we introduce a method for the equivalence analysis of NFA-based rules' representations that analyzes relationships

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 200X ACM X-XXXXX-XX-X/XX/XX ...\$10.00.

among IDS rules and brings out the potential inconsistencies.

In our work we choose to employ an NFA due to two primary considerations. An NFA-based solution alleviates the memory storage problem and allows a compact representation of the variable nature of rules. Generally, the number of NFA states is on the order of the number of characters in the input rule. Such compactness is also beneficial for succinct representation of regular expressions frequently used in many rule sets. One shortcoming of an NFA is its time complexity that can increase significantly in the case of matching patterns leading to multiple NFA states being active in parallel. Although an alternative solution based on a deterministic finite automaton (DFA) has been more widely used in practice, a DFA-based approach might be infeasible for the representation of rules with complex regular expressions essentially causing an exponential number of DFA states [6].

The contributions of our work can be summarized as follows:

- *A novel approach for semantic IDS rule analysis:* we introduce an NFA-based approach that can be employed for IDS rule analysis on a daily basis.
- *Efficient implementation:* we provide an implementation of our approach which we use to analyze the collections of attack signatures of open-source Snort IDS and Bleeding Edge Threats.

## 2. RELATED WORK

The analysis of security rules has been at the center of research attention over the last two decades resulting in a number of studies ranging from discovery of errors in the configurations of the router lists and network IDS [9, 20] to the modeling of conflict-free firewalls [11, 3, 22].

In the field of conflict discovery, the majority of the proposed approaches focused on the evaluation of the firewall configurations [4, 11, 22, 20, 13, 5]. Some algorithms and tools were proposed to merely highlight potential risks [21], while others provided a deeper analysis of configuration policy attempting to discover inconsistencies and redundancies [13, 4, 11]. More general frameworks allowing one to manage the security policy of the IPsec devices, firewalls, routers, etc. were introduced by [10, 9, 7]. Several works proposed a formal validation of security policy correctness for firewalls and IPsec devices [12, 8].

Several studies [20, 5] have considered a more general setting of network security policy. As such Uribe and Cheung [20] developed an approach for verification of security configurations of firewall and network IDS. Specifically, based on the network topology the primary goal of their method is to check whether the network is configured correctly. Alfaro et al. [5] focused on policy anomalies in the distributed network setting.

Similar to the methods for firewall configuration, all these approaches are limited to the filtering rules format, thus they can only discover inconsistencies bound to the syntactical representation of the rules.

We choose to employ an NFA for uniform modeling of IDS rules. The finite state machines have found application in many areas of computer science. In the network security domain the use of automata has been primarily applied to the optimization of pattern matching process. Various automata extensions were proposed for attack signature rep-

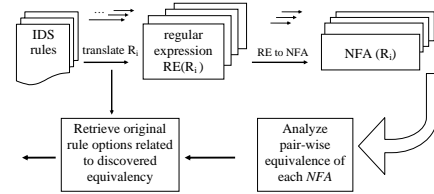


Figure 1: An analysis flow of IDS rules.

resentation mainly to improve the time or space efficiency. Smith et al. developed an extended finite automaton (XFA) that essentially extends a DFA approach [17], Becchi and Crowley introduced a hybrid finite automaton, that combines the strengths of an NFA and a DFA in one [6], while Sidhu employed an NFA based on conventional FPGA construction [16]. While these methods offer significant performance improvements, they are limited in their application abilities. As such, XFA [17] is not capable of representing complex regular expressions owing to a potential exponential memory blow-up, while a hybrid automaton [6] relies on manually constructed regular expressions. Since the memory and time performance requirements are less critical in the inconsistency discovery process, we utilize an NFA approach for modeling attack signatures.

## 3. NFA-BASED APPROACH TO RULE ANALYSIS

One of the intuitive ways for detecting inconsistencies among rules is the analysis of the rule relations. While this analysis might be straightforward for simple rules (for example, rules only limited to the IP address and port information), its ineffectiveness becomes clear when we consider more complex rules sets incorporating character counts and regular expressions (REs). In the past, IDS rules, and in particular, regular expressions have been successfully modeled using various extensions of an automaton [6, 17]. We follow this direction and model IDS rules as a nondeterministic automaton (NFA) that allows one to capture the semantics of the rule. We then analyze the NFA representation of the rules to detect semantic inconsistencies. The overall rule analysis flow is presented in Figure 1.

### 3.1 Rule representation

The format of rules, their configuration and rule sets organization varies across different IDS vendors. In general, any individual IDS rule can be viewed as a set of conditions to be met for the rule to be triggered, which is referred to as a *conditional part* of the rule, and a set of measures to be taken in case the conditional part is satisfied, which is denoted as an *action*.

Since in this work we aim at discovering inconsistencies based on rule relations, there are three types of relations between rules that might present interest to us: *fully matching rules*, i.e., rules that only accept the same events; *partially overlapping rules*, i.e., rules that accept a common subset of events, while also accepting events outside of this common subset; and *inclusive rules*, i.e., rules where one rule can potentially satisfy only a subset of all events accepted by another rule.

### 3.2 Rule translation to RE

Essentially, an IDS rule is the combination of various conditions that range from a simple check of the network protocol or file access to complex pattern matching of the payload content. Although the rule conditions and their format vary significantly from one IDS to another, the rule content can be uniformly represented using regular expressions. The translation to regular expressions, although a semi-automated process, requires the domain knowledge to clarify the rule format. For example, the rule option *content* in a Snort rule set may be described as a string consisting of binary data and mixed text, while an option *distance* specifies the number of bytes to be ignored relative to the previous pattern match. For example, the following Snort rule snippet: `content:USER, offset:3, content:00, distance:1` can be translated to the following regular expression: `.{3}USER.{1}00`. We have developed a translator for converting Snort-based rules into regular expressions. Depending on the rule structure and format the translator generates a single or a set of regular expressions for each rule.

### 3.3 Building NFA

The set of generated regular expressions is compiled together to an NFA representing a specified rule. The logical combination of regular expressions for NFA compilation depends on the specifics of the IDS rule format. We thus, employ the commonly used approach, which assumes that all rule conditions are *AND*ed together and the *OR* clause is not used. Based on this assumption, we combine regular expressions in a conjunctive form.

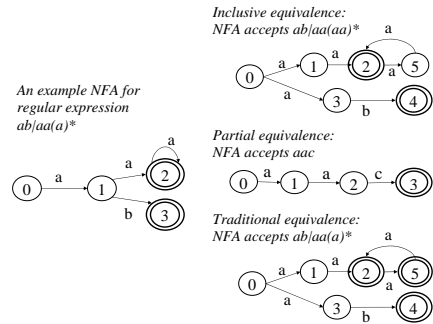
The generation of an NFA follows a traditional Thompson construction method [19]: parsing regular expressions, building an NFA and  $\epsilon$ -transitions elimination. However, we do not proceed to DFA construction and minimization steps for several reasons. One of the primary reasons is the modified structure of an automaton after its conversion and minimization that although allows us to produce the equivalent machine, does not preserve possible redundancies in a form of inclusive and partially overlapping states and transitions that might indicate potential rule inconsistencies. Such a changed automaton structure also makes it very challenging, if not impossible, to indicate the parts in the rule options that cause the problem. Since the proposed approach aims to provide guidance for administrators throughout the rule writing and configuration process, this aspect becomes especially important. In addition to this, the use of a DFA might not be feasible for many regular expressions found in IDS rule sets due to the potential for state explosion [6].

Thus, given a set of regular expressions, the compilation process generates an NFA, that can be formally defined as follows [14]:

**Definition 1 (NFA).** A nondeterministic finite automaton (NFA)  $M$  is 5-tuple  $(S, \Sigma, T, s_0, F)$  where  $S$  is a finite set of states,  $s_1, \dots, s_n$ ,  $\Sigma$  is a finite set of input alphabet,  $T$  is the state transition table that describes the mapping from  $S \times \Sigma$  to  $S$ ,  $s_0$  is a starting state, and  $F \in S$  is the set of final states.

### 3.4 Redefining equivalence for rule analysis

The NFA-based approach allows us to parse the high-level encoding of the rule conditions into a uniform representation



**Figure 2: The examples of traditional, inclusive and partial equivalences. The double circle denotes final accepting state.**

at the level of searched characters. The resulting NFAs can be then compared through the notion of *equivalence* to determine their relations to each other. The traditional definition of finite machine equivalence requires two machines to generate the same sequence of output symbols when given the same input. Note that this does not necessarily require the machines to be equal. Essentially, the machine equivalence can be determined through the *equivalence of states*. The states are said to be equivalent if for every input, the transitions from one state can lead to an accepting state iff the transitions from the second state can result in an accepting state. Then, the machines are equivalent if every state in one machine has the equivalent state in another machine. Formally [14],

**Definition 2 (EQUIVALENCE).** An NFA  $M_1 = (S_1, \Sigma, T_1, q_0, F_1)$  and an NFA  $M_2 = (S_2, \Sigma, T_2, p_0, F_2)$  are equivalent, if for all  $q$  and  $p \in S_1 \cup S_2$  and all  $a \in \Sigma$ ,  $q$  equivalent to  $p$ , written as  $q \equiv p$ , implies  $T_1(q, a) \equiv T_2(p, a)$ . States  $q \in S_1$  and  $p \in S_2$  are equivalent if all  $a \in \Sigma$ ,  $T_1(q, a) \in F_1 \cup F_2$  iff  $T_2(p, a) \in F_1 \cup F_2$ .

In essence, the notion of machine equivalence indicates a certain redundancy, and in our case, the potential inconsistency between two rules. However, the traditional definition of equivalence places a very strong requirement on the testing input, and is too strict to determine possible relations between the automata. Thus, we propose to weaken the notion of equivalence to detect the rule relationships defined in Section 3.1. As such we introduce an inclusive and partial equivalence.

Inclusive equivalence is intended to indicate rules that present a case of subset/superset relations. For example, consider the following Snort rules:

```
Rule 1:alert tcp any any->any 111 (pcre:"$/.{2}SS$");
Rule 2:alert tcp any any->any 111 (content:"PASS");
```

Clearly, rule 1 is more general, as it will match all strings ending with 'SS' including string PASS covered by rule 2. These type of exceptions are often used in IDS rule sets to exclude some events from certain actions allowing better flexibility and less redundancy in the rule management. At the same time, excessive use of this approach can lead to the ambiguous and difficult to manage rule sets. Thus, to

ensure accuracy of the rule set update, it is important to provide a system administrator with all of such cases.

We refer to an automaton as inclusively equivalent to another automaton if it accepts the subset of the inputs accepted by another machine and rejects other inputs. Formally,

**Definition 3** (INCLUSIVE EQUIVALENCE). *NFA  $M_1 = (S_1, \Sigma, T_1, q_0, F_1)$  is inclusively equivalent to NFA  $M_2 = (S_2, \Sigma, T_2, p_0, F_2)$ , if given  $A$  and  $B$  such that  $A \subset B \subset \Sigma$ ,  $M_1$  accepts all  $a \in A$ , where  $M_2$  accepts  $b \in B$ .*

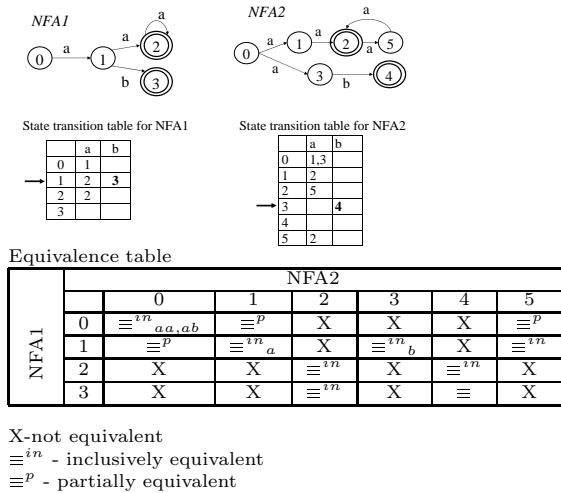
Partial equivalence might be considered as a weaker equivalence compared to inclusive in terms of its requirements. Partial equivalence indicates the partial overlap between two machines and is essentially targeted at IDS rules that have overlapping conditions allowing both rules to match some events. As an example, consider the following Snort rules:

```
Rule 1:alert tcp any any->any 111 (content:"USER";
content:"PASS"; distance:0;)
Rule 2:alert tcp any any->any 111 (content:"PASS";
content:"ANY"; distance:0;)
```

Both rules would match events with a string 'USERPASSANY', and thus, would be redundant in this part. There might be cases when the rules might be intentionally created to overlap to indicate a possibility of both attacks specified by rules or cover uncertainty in the traffic classification. However, one can easily see a potential for a conflict situation, when both rules are configured with the opposite response actions. Thus, it important that an administrator is fully aware of such cases.

**Definition 4** (PARTIAL EQUIVALENCE). *Two NFAs  $M_1 = (S_1, \Sigma, T_1, q_0, F_1)$  and  $M_2 = (S_2, \Sigma, T_2, p_0, F_2)$  are partially equivalent, if given  $A$  and  $B$  such that  $A \subset \Sigma$ ,  $B \subset \Sigma$  and  $A \neq B$ ,  $A \cap B \neq \emptyset$ ,  $M_1$  accepts all  $a \in A$ , where  $M_2$  accepts  $b \in B$ .*

The examples of traditional, inclusive and partial equivalences given in terms of automata are illustrated in Figure 2.



**Figure 3: An example of equivalence test on two NFAs.**

### 3.5 Testing for equivalence

The common approach to checking automata equivalence is based on the conversion of an NFA to a DFA and often requires the minimization of an automaton, i.e., removal of equivalent states, before the equivalence check can be performed. As we discussed it before, this defeats the purpose of our work, and thus we need a technique that allows us to work with nondeterministic transitions. The method that we chose to adopt to our purposes is the table-filling approach originally presented by Hopcroft, Motwani and Ullman [15].

The pseudocode for the testing algorithm is given in Figure 4. The strategy of the algorithm is to analyze all states of two NFAs to determine the type of equivalence, if any, on the defined inputs. Essentially, if any pair of states is determined to be preliminary equivalent, i.e., to have transitions on the given inputs, then the predecessor states, states to which the preliminary equivalent states have transitions should be checked for equivalence. At the end, the existence of equivalence and its type is determined by the type of equivalence of the start states.

The illustration of the equivalence test for two NFAs given in Figure 2 is presented in Figure 3. The transition function for both automata is given as a *state transition table* that helps to determine the next state given the current state and the input symbol. We start filling out the equivalence table by marking all entries associated with final states as stated in Lines 2-3 (Algorithm 1). If both states are final and have no outgoing transitions, we mark the corresponding equivalence table entry as equivalent, since both states will accept input  $\epsilon$ . In case one state is final and another state is not, we consider these states as not equivalent, since on input  $\epsilon$  only one state will accept. Finally, if both states are final with outgoing transition(s), label the corresponding entry as inclusively equivalent, since both states will accept on input  $\epsilon$ , and at least one state may accept on some other input. We proceed with filling the table starting at the lowest right entry, working our way to the beginning of the table. As such, consider a marking process for entry  $t[1, 3]$  (Figure 3).  $T_1$  contains two transitions from state 1:  $T_1[1, a] = 2$  and  $T_1[1, b] = 3$ , while  $T_2$  contains only one transition  $T_2[3, b] = 4$ , thus only input  $a$  is defined for both states and the preliminary status is set to inclusively equivalent. The predecessor states for the input  $a$  are 3 and 4. Since  $t[3, 4]$  is marked as equivalent, current entry is marked by the highest equivalence type, not exceeding the preliminary status, which is inclusively equivalent. The remaining table is marked in a similar manner. At the end, the relations between two automata are determined by the entry corresponding to the start states. Since the entry  $t[0, 3]$  is marked as inclusively equivalent, two automata are inclusively equivalent.

**Performance optimization.** Since the algorithm does not specify how to determine the processing order of pairs of states, we might choose to process all table entries one-by-one or follow a dependency list created through the transition function, i.e., a list of states that has to be processed before others. However, it can be seen that with the former approach the time complexity of the algorithm becomes  $O(n^4m)$ , where  $n$  is the number of states and  $m$  is the number of defined inputs in both automata. Although it is the worst case performance, since not all pairs of states need to be checked, the latter approach allows to reduce the run

---

**Algorithm 1:** EquivalenceTest(NFA  $M_1$ , NFA  $M_2$ )

---

```
1 Construct  $n_1 \times n_2$  table  $t$ , where  $n_1 = |S_1|$  and  $n_2 = |S_2|$ ;  
2 Mark all entries in  $t$  as equivalent, if both states are final without outgoing transitions;  
3 Mark all entries in  $t$  as inclusively equivalent, if both states are the final states and at least one with outgoing transitions;  
4 Mark all entries in  $t$  as not equivalent, if one of the states is the final state without outgoing transitions and another is not final state;  
5 forall unmarked entries  $t[q, p]$  do  
6   Step1: check the relations of current states;  
7   if  $\forall a \in \Sigma$  where  $T_1(q, a)$  is defined,  $\exists T_2(p, a)$  then  
8     preliminary: current states are equivalent;  
9   else if  $\exists a \in \Sigma$  where  $T_1(q, a)$  and  $T_2(p, a)$  are defined then  
10    preliminary: current states are inclusively equivalent;  
11   else  
12     mark entry  $t[q, p]$  as not equivalent;  
13     goto Step1;  
14   Step2: check the equivalence of predecessors:  $t[T_1(q, a), T_2(p, a)]$ ;  
15   if the predecessors  $\neq$  not equivalent then  
16     mark entry  $t[q, p]$  by the type of equivalence given in  $t[T_1(q, a), T_2(p, a)]$  (not higher than preliminary);  
17     if  $T_1(q, a)$  or  $T_2(p, a)$  contains several states use entry in  $t$  with the highest priority;  
18     update overlapping input for  $t[q, p] = a +$  overlapping input from  $t[T_1(q, a), T_2(p, a)]$ ;  
19   else if  $T_1(q, a) \in F_1$  or  $T_2(p, a) \in F_2$  then  
20     mark entry  $t[q, p]$  as partially equivalent;  
21     update overlapping input for  $t[q, p] = a +$  overlapping input from  $t[T_1(q, a), T_2(p, a)]$ ;  
22   else  
23     mark entry  $t[q, p]$  as not equivalent; /* none of the predecessors is final state */
```

---

Figure 4: The pseudocode for NFA equivalence test.

time to  $O(n^2)$  by only processing the pairs of states that depend on each other on a particular input.

### 3.6 Tracking the conflicting conditions

The final phase of the rule analysis process is to restore the original conditions that caused the conflict. Essentially, we need to retrace the equivalent states starting at the start state until we reach a final state. However, to reduce the space and time complexity, we keep track of the input symbols during the marking process if two states demonstrate some type of equivalence. To ensure the continuity of the input, the input associated with the equivalence of the states is combined with the input recorded at the predecessor entry (see Lines 18 and 21, Algorithm 1).

For example, the entry  $t[0, 0]$  marked as inclusively equivalent in the example given in Figure 3 will produce two overlapping inputs “ $aa$ ” and “ $ab$ ”. The multiplicity of input comes from the fact that  $t[0, 0]$  identified as inclusively equivalent on input  $a$  corresponds to two predecessor states:  $t[1, 1]$  with overlapping input  $a$  and  $t[1, 3]$  with overlapping input  $b$ . Thus, the combination of the inputs produces strings “ $aa$ ” and “ $ab$ ”.

## 4. EXPERIMENTAL RESULTS

To determine the effectiveness of our approach, we analyzed 15 rule sets from the latest collection of VRT Certified Rules for Snort v2.8. [2] and 3 sets of Bleeding Edge Threats[1]. From our analysis we excluded rules with lookahead and lookbehind assertions as they are used in test-driven engines, while our approach is finite automata driven. Table 5 presents the discovered inconsistencies.

As the results show, among the discovered redundancy conflicts dominates the inclusive equivalence (124 cases out of 164 conflicts). In practice, this generally means that sev-

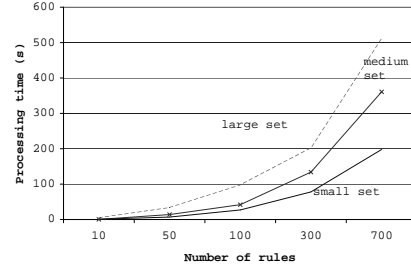
eral rules can match a network packet. Compared to the inclusive redundancy cases, the number of the discovered traditionally and partially equivalent conflicts is relatively small. We have identified four cases of traditional equivalence which essentially means that eight rules accept exactly the same set of network packets and thus four of these rules can be safely removed without affecting the traffic classification. The 36 cases of partial equivalence also indicate the overlap in network traffic classification. This type of inconsistency is generally less visible during the manual rule inspection, thus automatic identification of such cases is important. The examples of the discovered inconsistencies are given in Table 1.

Figure 5 presents the results of a series of experiments on evaluation of the algorithm performance which was conducted using several sets of rules. Our experiments were performed on an Intel(R) Pentium(R) 4 with CPU of 2.40GHz and 1.5GB of RAM. To study the performance of the proposed method, we generated three sets of Snort rules to reflect different processing requirements. Specifically, the first set, labeled as small, incorporated rules that were translated into an NFA with an average number of states equal to 7 and average number of transitions equal to 259. The second set, a medium set, presented an NFA with 15 states and 259 transitions on average, and finally, the largest set, had 40 states and 288 transitions. The distribution of rules within these sets were as follows: majority of processed rules (75%) fell into a medium set, with 22% in large and 3% in small sets, respectively.

The results of these experiments given in Figure 5 show that the largest NFA requires the highest processing time. This is an expected scenario as generally complex and diverse rule conditions result in a larger NFA. Since the rule analysis is intended for off-line usage we consider these process time requirements reasonable.

Rule set	Number of rules	Inconsistencies	
		total	type of equivalence
Snort IDS:			
chat	34	7	inclusive-2, partial-5
pop3	35	1	inclusive-1
ftp	75	3	inclusive-3
sql	89	1	inclusive-1
smtp	90	4	inclusive-1, partial-3
icmp-info	93	6	inclusive-6
web-cgi	357	82	inclusive-78, partial-3, traditional-1
web-misc	367	34	inclusive-27, partial-7
web-client	712	6	inclusive-2, partial-2, traditional-2
backdoor	738	11	inclusive-1, partial-9, traditional-1
Total		155	inclusive-122, partial-29, traditional-4
Bleeding Edge threats:			
bleeding-attack_response	38	3	partial-3
bleeding-exploit	257	1	inclusive-1
bleeding-web	125	5	inclusive-1, partial-4
Total		9	inclusive-2, partial-7

(a)



(b)

Figure 5: (a)Inconsistencies detected in IDS rule sets (b)Performance.

Redundancy (traditional equivalence):
1) alert tcp \$EXTERNAL_NET \$HTTP_PORTS -> \$HOME_NET any (msg:"WEB-CLIENT msdds clsid attempt";content:"EC444CB6-3E7E-4865-B1C3-0DE72EF39B3F"; nocase; pcre:"/<OBJECT[s+[*]]*classid[s+[*]]*\[x22\x27]?s*clsid[s+[*]]*x3a[s+[*]]*x7B?s*EC444CB6-3E7E-4865-B1C3-0DE72EF39B3F/si";sid:4132;)
2) alert tcp \$EXTERNAL_NET \$HTTP_PORTS -> \$HOME_NET any (msg:"WEB-CLIENT Microsoft DDS Library Shape Control ActiveX Object Access";content:"EC444CB6-3E7E-4865-B1C3-0DE72EF39B3F"; nocase; pcre:"/<OBJECT[s+[*]]*classid[s+[*]]*\[x22\x27]?s*clsid[s+[*]]*x3a[s+[*]]*x7B?s*EC444CB6-3E7E-4865-B1C3-0DE72EF39B3F/si";sid:4211;)
Redundancy (inclusive equivalence):
1) alert tcp \$EXTERNAL_NET any->\$HOME_NET 110 (msg:"PO3 EXPLOIT x86 Linux overflow";content:" D8 @ CD 80 EB D9 FF FF FF /bin/sh"; sid:288;)
2) alert tcp \$EXTERNAL_NET any->\$HOME_NET 110 (msg:"PO3 EXPLOIT qopper overflow";content:" E8 D9 FF FF FF /bin/sh"; sid:290;)
Redundancy (partial equivalence):
1) alert tcp \$HOME_NET any -> \$EXTERNAL_NET 16661:6668 (msg: "BLEEDING-EDGE ATTACK RESPONSE IRC - Channel JOIN on non-std port";dsize: <64;content:"JOIN "; nocase; offset: 0; depth: 5; tag:session,300,seconds;pcre:"/&# ! /R"; sid:2000348;)
2) alert tcp \$HOME_NET any -> \$EXTERNAL_NET 16661:6668 (msg: "BLEEDING-EDGE ATTACK RESPONSE IRC - channel join on non-std port";content:"JOIN #: "; nocase; offset: 0; depth: 8; tag: session,300,seconds; sid: 2000351;)

Table 1: The examples of the detected inconsistencies.

## 5. CONCLUSION

This paper describes the problem and presents an effective approach to the management of IDS rule sets. Specifically, the work targets IDS rule analysis for inconsistencies. The approach based on an NFA-based representation of the IDS rules allows us to employ the traditional notion of automata equivalence to analyze the semantics of the rules. We experimentally validate our approach on the example of the Snort IDS and Bleeding Edge Threats rule sets and present the examples of the discovered inconsistencies. As the experimental results show reasonable processing requirements for off-line application, our approach can be viewed as a light-weight tool for improving the quality of the rule set. This work is the initial step in the direction of effective rule set management. In our future work, we plan to investigate the formal verification of the rule set status with respect to the existence of inconsistencies.

## 6. REFERENCES

- [1] Bleeding edge threats. Available from "http://www.bleedingthreats.net/", 2008.
- [2] *Snort<sup>TM</sup> Users Manual 2.8.4*, 2009. Available from "http://www.snort.org/assets/82/snort\_manual.pdf".
- [3] E. Al-Shaer and H. Hamed. Modeling and management of firewall policies. *IEEE Transactions on Network and Service Management*, 1(1), April 2004.
- [4] E. Al-Shaer, H. Hamed, R. Boutaba, and M. Hasan. Conflict classification and analysis of distributed firewall policies. *IEEE Journal on Selected Areas in Communications*, 23(10):2069–2084, 2005.
- [5] J. G. Alfaro, F. Cuppens, and N. Cuppens-Bouhalia. Analysis of policy anomalies on distributed network security setups. In *Proceedings of ESORICS*, pages 496–511. Springer, 2006.
- [6] M. Becchi and P. Crowley. A hybrid finite automaton for practical deep packet inspection. In *Proceedings of ACM CoNEXT*, pages 1–12, New York, NY, USA, 2007. ACM.
- [7] J. Burns, A. Cheng, P. Gurung, S. Rajagopalan, P. Rao, D. Rosenbluth, and A. V. Surendran. Automatic management of network security policy. In *Proceedings of the DARPA DISCEX*, pages 1012–1026, 2001.
- [8] V. Capretta, B. Stepien, A. Felty, and S. Matwin. Formal correctness of conflict detection for firewalls. In *Proceedings of ACM FMSE*, pages 22–30. ACM, 2007.
- [9] P. Eronen and J. Zitting. An expert system for analyzing firewall rules. In *6th Nordic Workshop on Secure IT Systems*, pages 100–107, 2001.
- [10] Z. Fu and S. F. Wu. Automatic Generation of IPSec/VPN Security Policies In an Intra-Domain Environment. In *Proceedings of DSOM*, pages 279–290, 2001.
- [11] M. G. Gouda and X.-Y. A. Liu. Firewall design: Consistency, completeness, and compactness. In *Proceedings of ICDCS*, pages 320–327, 2004.
- [12] H. Hamed, E. Al-Shaer, and W. Marrero. Modeling and Verification of IPSec and VPN Security Policies. In *Proceedings of ICNP*, pages 259–278, 2005.
- [13] A. Hari, S. Suri, and G. Parulkar. Detecting and resolving packet filter conflicts. In *Proceedings of INFOCOM*, pages 1203–1212, 2000.
- [14] J. E. Hopcroft and R. M. Karp. A linear algorithm for testing equivalence of finite automata. Technical Report TR71-114, Cornell University, 1971.
- [15] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, 2 edition, 2001.
- [16] R. Sidhu and V. K. Prasanna. Fast regular expression matching using fpgas. In *Proceedings of FCCM*, pages 227–238, 2001.
- [17] R. Smith, C. Estan, S. Jha, and I. Siahaan. Fast signature matching using extended finite automaton (XFA). In *Proceedings of ICISS*, pages 158–172, 2008.
- [18] Third Brigade, Inc. *OSSEC Manual*, 2008. Available from "http://www.ossec.net/main/manual/#rules".
- [19] K. Thompson. Regular expression search algorithm. *Communications of the ACM*, 11(6):419–422, 1968.
- [20] T. E. Uribe and S. Cheung. Automatic analysis of firewall and network intrusion detection system configurations. In *Proceedings of the ACM FMSE*, pages 66–74, 2004.
- [21] A. Wool. Architecting the lumeta firewall analyzer. In *Proceedings of USENIX Security Symposium*, pages 7–7, Berkeley, CA, USA, 2001. USENIX Association.
- [22] L. Yuan, J. Mai, Z. Su, H. Chen, C.-N. Chuah, and P. Mohapatra. Fireman: A toolkit for firewall modeling and analysis. In *Proceedings of SP*, pages 199–213, 2006.