

Classification and discovery of rule misconfigurations in intrusion detection and response devices

Natalia Stakhanova¹ Yao Li² Ali A. Ghorbani¹

¹*Faculty of Computer Science
University of New Brunswick
Fredericton, NB Canada
{natalia, ghorbani}@unb.ca*

²*Plato Consulting Inc.
yli@platoconsulting.com*

Abstract

The signature-based intrusion detection is one of the most commonly used techniques implemented in modern intrusion detection systems (IDS). Being based on a set of rules, i.e., attack signatures, the accuracy and reliability of IDS detection heavily depend on the quality of the employed rule set. In this context, any conflicts that arise between rules create ambiguity in classification of network traffic or host events, not only affecting the performance of IDS, but also putting the system in a vulnerable position. Currently existing techniques for conflict detection focus primarily on the security policy of the network devices: IPSec, routers, firewalls.

In this paper we address the conflict detection in host and network-based intrusion detection and response devices and present a rule management framework that allows rule set analysis for potential conflicts. We demonstrate the advantages of the proposed approach on three collections of attack signatures: the set provided by the vendor of the commercial IDS and the rule sets of the open source Snort IDS and Bleeding Edge Threats. Our analysis reveal conflicts in each of them.

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, signature-based approach is the most common technique implemented in commercial intrusion detection

systems (IDS). It aims to detect attacks by analyzing event stream against a predefined set of rules, i.e., attack signatures, that generally describe the IDS-specific set of network features (e.g. destination port, payload content) or system events (e.g., privileged file access, login time) and the response action to be deployed in case the observed behavior matches the description.

Although, the signature-based IDS provides many benefits including accurate detection of the known attacks and indication of suspicious behavior, it is highly dependent on the expert knowledge in developing and maintaining the detection rules. Constant discovery of the software vulnerabilities and exploits demands timely addition of new attack signatures which significantly increases the complexity of set resulting in overlapping and often redundant signatures. This situation is aggravated by the fact that in the large-scale systems a variety of intrusion detection tools are often bundled together to provide comprehensive security coverage of the network. Most of these off-shelf IDS products include prewritten signatures that over time are customized and periodically updated by different administrators. As a consequence, the set of attack signatures becomes unreliable, not only affecting performance of the system, but also putting it in a vulnerable position.

Although analysis of the security policy has been at the center of research attention in the past decade, the majority of these studies focused on the performance issues [8, 10, 18, 17]. A significantly less attention was paid to the conflict-free management of the policy rules. The existing research has mainly explored the configuration policy of firewalls, router lists and IPSec devices [7, 21, 6, 4, 11, 12].

Intrusion detection rules can be considered as a special case of the network security policy rules, however, they possess several unique characteristics that require additional research. One of the challenges in IDS rules' analysis is the absence of the common format. While firewall, IPSec rules employ a small set of commonly used fields of the network packets [15], the format of the IDS rules varies across different intrusion detection devices and often supports the use of logical operators: AND NOT, OR. As a result, the IDS rule sets benefit from the hierarchical representation which allows to escalate the importance of attack combinations and model multi-step intrusions. Similarly, if the firewall rules generally trigger either "deny" or "accept" action, the IDS rules employ a variety of response strategies, e.g., trace network connection, kill process, alert administrator, enable/disable additional firewall rules. Finally, the representation of the firewall, IPSec policy as sequential lists often puts a special importance on the rules ordering in the defined policy to avoid potential conflicts among related rules. While the IDS rules can be represented as access lists, majority of the intrusion detection products ignore the rule ordering, intentionally or unintentionally allowing several rules to match a considered system event.

Solution Methodology. In light of the above, we propose an approach to the analysis of the intrusion detection rules for potential conflicts. Our approach takes advantage of the hierarchical structure of IDS rules modeling their relations in the graph structure. Based on the traditional graph traversal algorithms we present a set of algorithms that allow automatic discovery of potential conflicts in IDS rule sets.

The contributions of our work can be summarized as follows:

- *Classification of the intrusion detection rule misconfigurations:* the developed classification is general and applicable to attack signatures of network and host-based IDS.
- *Algorithms for misconfiguration discovery:* we describe a set of algorithms to examine IDS attack signatures for potential conflicts.
- *Efficient implementation:* we provide an implementation of our approach which we employ to analyze the collections of attack signatures of the commercial IDS, Snort IDS and Bleeding Edge Threats.

The primary focus of the proposed approach is the analysis of the rule set's efficiency, in other words, we are interested to determine whether the rules are doing

the intended task in the most economical way, rather than to analyze the rules compliance with the intended security policy.

As such, the proposed approach can be beneficial as an addition to security policy management tools for periodic monitoring of the IDS rule sets, as well as a guidance tool for the administrator to help in the attack signature writing process.

2 Related Work

The analysis of the security policy 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 configuration of the router list and network IDS [11, 20] to the modeling of the conflict-free firewall [13, 5, 21].

The initial definition and classification of the security policy conflicts were developed by Hamed and Al-Shaer [14]. They distinguished four primary types of access lists conflicts: shadowing, redundancy, correlation and exception. The proposed classification was the first attempt to establish a theoretical foundation for the security policy analysis, and thus, was widely used by other researchers [21]. As opposed to these studies, our work focuses specifically on the conflicts of the network and host-based IDS rule sets.

In the field of conflict discovery, majority of the developed methods focused on the evaluation of the firewall configuration [6, 13, 21, 20, 7]. Some of these studies [20, 7] consider a general setting of network security policy, and thus, allow to apply the proposed algorithms to the network IDS rules. However, these approaches are not specifically designed for attack signatures, thus relying on the network topology they are limited to the discovery of errors in the detection policy. More general frameworks allowing to manage security policy of the IPSec device, firewall, router, etc. were introduced by [12, 11, 9]. The focus of our approach in this work is the set of techniques for efficient discovery of the rule misconfigurations that can be employed to enhance the capabilities of the above-mentioned security-policy tools.

3 Rule Representation

Generally, in the context of intrusion detection a rule can be defined as a set of conditions to be met for the rule to be triggered, which are referred to as a *conditional part* of the rule, and an *action* that defines measures to be taken in case the conditional part is satisfied. Table 3 presents examples of the IDS rules. Formally, an IDS rule can be defined as follows:

Definition 1 (IDS Rule) An intrusion detection rule is a tuple $R_i = (C_i, A_i)$, where C_i is a predicate defined on a set of conditions c_1, c_2, \dots, c_n and A_i is an action. A condition $c \in C_i$ is defined as a (f, v) tuple, where $f \in F_i$ is a field monitored by IDS (e.g., IP protocol, TCP flag, file access, etc.) and v represents the numerical or descriptive value (or range of values) assigned to the corresponding feature. $F = \bigcup F_i$ denotes the set of all fields represented in the rule collection.

Following this definition, if the conditional part of R is satisfied, i.e., $R(C)$ is true, then $R(A)$ is deployed.

Depending of the organization of the conditional part of the rule, we distinguish **simple rules**, i.e., rules independent of the output of other rules, and rules relying on such output, **interdependent rules**.

Definition 2 (Interdependent Rule) An IDS rule R_i is interdependent if $\exists c_i \in C$ such that c_i is true iff $R_j(C)$ is true.

That is, rule R_i is interdependent if it contains conditions that can be viewed as boolean function $Boolean(R_j)$ returning true if some rule R_j is satisfied or false otherwise.

The use of interdependent rules effectively creates hierarchical relations between rules. Such hierarchical relations can be represented through rule options (e.g., via `if_sid` field in OSSEC IDS [19]) or using direct reference to the other rules (QRadar [2]). For example, in Snort rules such dependency is modeled through `flowbits` rule option that allows to track the state of the application protocol [3]. In this case, the command “set” followed by the name of the state, sets the start state of the processed network flow, and the command “isset” checks if the specified state was previously set for the considered flow. As an example, consider the modified Snort rules,

```
Rule 1:  If protocol:tcp dstPort:995
flow:to_server,established; content:"|16 03 00|"
=> flowbits:set,ssl3.client_hello.request
```

```
Rule 2:  If protocol:tcp srcPort:995
flow:to_client,established;
flowbits:isset,ssl3.client_hello.request;
content:"|16 03 00|" => Alert
```

In this example, the Rule 2 triggers an alert only if Rule 1 is satisfied, that is, the `flowbits` field is set to “`ssl3.client_hello.request`”.

Rule Relations. One of the essential steps in the intrusion detection rules analysis is to understand the relationships among the rules. Following the classification of the rule relationships based on the assumption of

common rule format [6], we redefine the proposed types of relations for individual fields of the considered rules:

Consider rule fields f_i and f_j such that $R_x(f_i) = R_y(f_j)$ and $R_x \neq R_y$, then the field values are **disjoint** if $R(v_i) \cap R(v_j) = \emptyset$. The values are **matching** if $R(v_i) = R(v_j)$, the values are **inclusive** if $R(v_i) \subsetneq R(v_j)$, the values **intersect** if $R(v_i) \cap R(v_j) \neq \emptyset$, while $R(v_i) \neq R(v_j)$ and $R(v_i) \not\subset R(v_j)$ and $R(v_i) \not\supset R(v_j)$.

4 Classification of IDS rule misconfigurations

Intrusion detection systems often employ large repositories of rules to ensure accurate detection of the potential attack attempts and their correct labeling according to the defined security policy. These repositories are populated and frequently updated with new rules as novel threats are identified, this often leads to various misconfigurations among rules resulting in conflicting classification of the network or host behavior. Broadly, the rule misconfigurations can be divided into conflicts between rules, *inter-rule misconfigurations* and conflicts within a rule, *intra-rule misconfigurations*.

4.1 Inter-rule misconfigurations

According to the level of rule dependencies, there can be distinguished: non-hierarchical and hierarchical misconfigurations.

Non-hierarchical misconfigurations occur between rules without direct hierarchical relationship. As an example, consider rules $R1$ and $R2$ in Table 3. The non-hierarchical misconfigurations can be classified into two categories:

Inefficiencies: although inefficiency does not directly create a conflict among rules, it overburdens the system resulting in slower performance and makes it challenging for the administrator to manage the collection of IDS rules. There are several types of inefficient rules:

- **Redundant rules:** the rule R is considered redundant if all events that can match this rule also match a rule R' and both rules have the same action. In this situation, removal of the redundant rule does not affect classification of the network/host behavior. However, it should be also noted that often partially redundant rules are created intentionally to emphasize certain types of intrusive behavior to the administrator. This message is generally provided through the rules’ annotations.
- **Verbose rules:** the rules are verbose if they have the same action and their conditional part can be ef-

Simple rules						
Rule #	Protocol	Attack Context	SrcPort	DstPort	Other fields	Action
R1	udp	RemoteToLocal	-	65535	payloadSize>268	create alert "Suspicious event"
R2	tcp	RemoteToLocal	-	515	Payload:" 24 7B 49 46 53 7D "	alert "Exploit attempt"
R3	udp	RemoteToLocal	-	65535	payloadSize>268	log an attempt
R4	udp	RemoteToLocal	-	515	Payload:" 24 7B 49 46 53 7D "	alert "Exploit attempt"
R5	tcp	-	0	1024-3072	RSTFlag=1	create alert "Port scan", set magnitude=8
R6	icmp	-	-	-	Type:"Redirect"	activate ICMPBlock in Firewall
R7	tcp	-	0	6	dstIP: 172.234.15.5	create alert "Suspicious event"
R8	-	-	-	0-1024	dstIP: 172.234.15.5	log an attempt
Interdependent rules						
R9	(DstPort=65535 AND NumOfAttempts > 5) OR alerttype:"Aggressive attack attempts"					create alert "Suspicious event"
R10	alerttype:"Suspicious event" OR "Port scan"					dispatch alert "Attack attempts"
R11	alerttype:"Exploit attempt" OR "Attack attempts"					create alert "Aggressive attack attempts"
R12	alerttype:"Aggressive attack attempts" OR magnitude>7					block srcIPaddress

Table 1. The examples of IDS rules

ficiently combined in one rule without affecting the classification of any event. In practice, the verbose rules appear in the large rulesets over time, as a result of periodic updates by the administrator [21]. While the verbosity is often considered as undesirable, it has several benefits. Generally, the excessively long rules are broken down into smaller sets to allow easy readability and management on one hand, and encourage reusability, on the other hand. As an example of verbose rules, consider rules $R4$ and $R2$ in Table 3. Although the values of the protocol field are disjoint (udp and tcp), these rules can be effectively combined.

- **Irrelevant rules:** The rule R is considered irrelevant if the event stream with the properties indicated in the rule cannot reach the IDS device. This situation can arise if the settings of the IDS devices are not configured properly.

Inconsistencies: inconsistency between rules is generally a good indicator of a potential conflict.

- **Correlated rules:** the rule R is correlated with the rule R' if there is at least one event that matches both rules. The rules are considered correlated only if they provide different actions and do not have superset/subset relations, i.e., relationships where the event stream that can potentially match R represents superset/subset of the events that can be possibly matched by R' . For example, rule $R5$ is correlated with rule $R8$ (see Table 3).

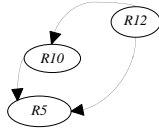
Usually, the correlated IDS rules are considered as a warning. The potential conflict can arise when traffic matching both rules triggers two conflicting response actions, for example, *activate the firewall rule to allow the connection* and *send a fake "host unreachable" reply*. However, the rules might be intentionally created to overlap to indicate a possibility of both attacks specified by rules or cover

uncertainty in the traffic classification. The fully overlapping rules, although also considered as correlated rules, create a conflict condition. For example, rules $R1$ and $R3$ (see Table 3).

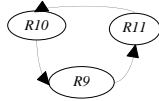
- **Exception rules:** the rule R is considered an exception of a rule R' if the rules have subset/superset relations and different actions. Exceptions are often used in IDS rulesets to exclude some events from certain actions allowing better flexibility and less redundancy in the rule management. At the same time, excessive use of this approach can lead to the ambiguous and difficult to manage rulesets. For example, rule $R7$ is an exception of more general rule $R8$ (see Table 3).

Hierarchical misconfigurations refer to the conflicts between rules with hierarchical relationships, i.e., between simple and interdependent rules or between interdependent rules. Generally, such rule hierarchies are used in IDS to escalate the severity of the alerts in the presence of additional conditions or to correlate alerts into meaningful attack scenarios.

- **Redundant dependency:** refers to the situation when several interdependent rules rely on the output of a single rule. This situation is common among IDS rules as certain attack steps are prerequisites for more advanced intrusions, e.g., *multiple login attempts* and *port scanning*. Thus, referring to the common rule allows to reduce rule redundancy and not necessarily causes the conflict situation. However, such dependency may create inefficiency if several interdependent rules depending on each other's output refer to the one child. As an example of this inefficiency, consider dependency among a simple rule $R5$ and two interdependent rules $R10$ and $R12$ depicted below (dependency is indicated by arrows).



- **Circular dependency:** occurs among rules that contain reference to each other in their conditional parts. As such a parent rule depends on the output of child rule which in turn requires the output of parent rule. This misconfiguration is a critical error. Alternatively, the rule may be a part of circular dependency if it refers not to a direct parent, but to a predecessor rule. For example, rules *R9* creates a chain of circular dependency with rules *R10* and *R11* (see Table 3) shown below (dependency is indicated by arrows).



4.2 Intra-rule misconfigurations

Inefficiencies:

- **Redundant condition:** refers to the repetitive condition within a rule. The condition is considered redundant if the rule contains fields with matching or inclusive values. While the redundancy is not considered as a critical error, it contributes to the rule matching time, and consequently, affects the IDS performance. For example, consider the following rules:

```

Rule 1: If alertType: ACL Deny AND dstPort=
1808-1810 log srcIP address
Rule 2: If Category is Access.ACL Deny AND
destination is Database server AND dstPort is
1520-2430 create alert ``ACL Deny``
  
```

Since the destination port is defined in simple rule *R2*, redefinition of this condition in rule *R1* is redundant.

- **Verbose condition:** refers to the conditions that can be efficiently combined in one. As such the conditions are verbose if the rule contains fields with disjoint or intersecting values.
- **Complicated condition:** represents the condition that contributes to complicated and long structure of the rule. The excessive conditions overburden the rule, making it difficult for the administrator

to update it and consequently, leading to potential conflicts. While it is challenging to determine the optimal number of conditions within a rule, if the administrator finds it difficult to follow the rule structure, it is important to modify the rule. For example, Snort rule 9799 from 'web-client.rules' set contains 8 conditions, two of which 'content' and 'pcre' are excessively long: 'content' field has 180 characters, while 'pcre' contains regular expression of length equal to 391.

- **Needless condition:** is the condition that only becomes true if the other conditions of the rule are satisfied. Due to this reason, this condition is unnecessary and can be removed without any change in the event stream classification.

Inconsistencies:

- **Contradicting condition:** is the condition that contradicts the other rule conditions. For example, in the following case destination port is the conflicting condition.

```

Rule 1: If alertType: ACL Deny AND dstPort=
1808-1810 log srcIP address
Rule 2: If Category is Access.ACL Deny AND
destination is Database server AND NOT dstPort
is 1808-1810 create alert ``ACL Deny``
  
```

The primary source of this type of inconsistency is the extensive use of the negation operator. While this operator is considered as a convenient tool for excluding certain parts of event stream from the classification, it often complicates the ruleset contributing to potential conflicts.

- **Irrelevant condition:** is the condition that is irrelevant for event stream classification by this rule, e.g., due to inability of IDS device to process the condition, mistake in the condition statement.

5 Analysis Algorithms for IDS Rule Misconfiguration Detection

The analysis algorithms aim to discover misconfigurations based on the rule relations. One of the intuitive ways to represent these relations is to model IDS rule set as a graph data structure.

Since the IDS devices may contain multiple sets of attack signatures, we translate all collections into a uniform graph, which we refer to as *IDS rule graph*, *G*. This data structure is a directed graph, where each node denotes a specific rule of the set and the edges represent the relations of this rule with the other nodes. As

Algorithm 1: ruleSet RuleGraphTraversal(G,Node x)

```

1 ruleSet newRule, myRule;
2 visited(x);
3 foreach y ∈ G that has edge (x, y) do
4   setCondition(x);
5   if y is not visited then
6     Path+ = y;
7     newRule=RuleGraphTraversal(G, y);
8     if condition(x) ∈ {AND, OR, NOT} then
9       InterRuleAudit(myRule,newRule);
10      combine(myRule,newRule);
11    else
12      combine(myRule,newRule);
13      IntraRuleAudit (myRule);
14    Path- = y;
15  else /* y is visited */
16    if y ∈ Path then
17      conflict("Circular dependency");
18    else
19      conflict("Redundant dependency");
20      if condition(x) ∈ {AND, OR, NOT} then
21        InterRuleAudit(myRule,newRule);
22      else
23        IntraRuleAudit (myRule);
24 if x is a leaf node then
25   set(myRule, rule(x));
26   IntraRuleAudit (myRule);
27 return myRule;

```

Figure 1. Rule graph traversal.

such a simple rule is represented by a single node and an interdependent rule as a hierarchical component with outgoing edges of a parent node pointing to the nodes representing rules defined in its conditional part. The rules that refer to the same child rule in their condition statement have edges pointing to that child rule. This representation may eventually create loops in the graph representing circular or redundant dependencies among the rules. To create a unified graph, simple rules and components representing hierarchy of simple rules and interdependent rules are connected by an abstract root. Based on the constructed IDS rule graph, we analyze relations between adjacent nodes, traversing the graph following DFS approach. Sequentially visiting every node in the rule graph, we combine the rules of the children nodes at the parent levels, invoking a rule audit routine every time a new rule is added to a parent set.

The Algorithms 1, 2 and 3 present the pseudocode for rule graph traversal and analysis. Although the classification presented in Section 4 defines a broader range of misconfiguration types, detection of some conflicts, e.g., irrelevant rules/conditions, requires additional knowledge of network configuration, system security policy, level of administrator skills, etc. and thus, is not a part of rule audit algorithms at this time.

The procedure RuleGraphTraversal() starting at the root, recursively visits the nodes along one

Algorithm 2: InterRuleAudit(RuleSet, rulesToAdd)

```

1 forall (Rx ∈ RuleSet and R ∈ rulesToAdd) do
2   if inclusiveRelations(Rx, R) then
3     if ∄i, j : Rx(fi) =
4       R(fj) & (Rx(vi) & R(vj) - disjoint) then
5       if ∃i, j : Rx(fi) =
6         R(fj) & (Rx(vi) & R(vj) - intersect) then
7         if Rx(A) = R(A) then
8           conflict("Redundancy");
9         else
10          conflict("Correlation");
11       else if ∃i, j : Rx(fi) = R(fj) and
12         Rx(vi) & R(vj) - inclusive || matching then
13         if Rx(A) = R(A) then
14           conflict("Redundancy");
15         else
16           conflict("Exception");
17     else if matchingRelations(Rx, R) then
18       if ∃i, j : Rx(fi) =
19         R(fj) & (Rx(vi) & R(vj) - disjoint) then
20         if R(vi) & R(vj) continuous &&
21           Rx(A) = R(A) then
22           conflict("Verbosity");
23       else if ∃i, j : Rx(fi) =
24         R(fj) & (Rx(vi) & R(vj) - intersect) then
25         if Rx(A) = R(A) then conflict("Redundancy");
26         if R(vi) & R(vj) continuous then
27           conflict("Verbosity");
28         else conflict("Correlation");
29       else if ∃i, j : Rx(fi) = R(fj) and
30         Rx(vi) & R(vj) - inclusive || matching then
31         if Rx(A) = R(A) then conflict("Redundancy");
32         else conflict("Exception");

```

Figure 2. InterRule relation analysis.

branch of the graph until it reaches the leaf node. Once the leaf node is reached, the procedure calls IntraRuleAudit() algorithm for the analysis of rule's conditions. After that, the procedure backtracks collecting the rule sets of the children nodes at the parent level and invoking the rule audit algorithms. Depending on the logical condition defined at the parent node, the children nodes are processed according to InterRuleAudit() or IntraRuleAudit() procedures. During the rule graph traversal, the function memorizes the current path and processes only not previously visited nodes. If the node has been explored already, which essentially points at the presence of loops in the graph, the current path is a primary indicator of the type of the hierarchical misconfiguration in the rule set. Both IntraRuleAudit() and InterRuleAudit() algorithms are based on the analysis of the relations between rules, i.e., the existence of matching fields in the rules and the relationships between these fields' values. The rule/fields relationships are defined according to the classification given by [6].

Algorithm 3: void intraRuleAudit (Rule R)

```

1 forall  $f_i \& f_j \in R$  such that  $f_i = f_j$  do
2   if  $c(f_i) \& c(f_j)$  are negated ||  $c(f_i) \& c(f_j)$  are not negated
   then
3     if  $R(v_i) \& R(v_j)$  disjoint || intersected then
4       conflict("Verbose condition");
5     else
6       conflict("Redundant condition");
7   else if  $!(R(v_i) \& R(v_j) \text{ disjoint})$  then
8     conflict("Contradicting condition");

```

Figure 3. IntraRule relation analysis.

6 Experimental results

To evaluate a practical value of our approach, we have implemented the algorithms described in Section 5 and analyzed the effectiveness of the conflict detection process based on the graph approach using the rule collections of the open-source Snort IDS [3], Bleeding Edge Threats[1] and set of rules provided by the vendor of the commercial IDS¹.

Conflict detection Table 2 presents the details of the discovered misconfigurations². The results show that neither the commercial support of IDS nor the support of the security community allow to guarantee the absence of misconfigurations among IDS rules. Although majority of the discovered misconfigurations are inefficiencies that can be considered as warnings, rather than critical errors, they negatively affect IDS performance contributing to the complexity of the rule set and eventually leading to potential problems.

The commercial IDS set together with the inefficiencies, contains inconsistencies, which generally have more serious consequences. As such, we have discovered 1 case of correlated rules. This type of conflict presents a potential problem allowing event stream to match two rules triggering different response actions. We have also identified 6 cases of exception rules. Generally, the exception rules are common among system administrators. However, since the ordering of the rules in IDS is usually not enforced, additional measures are often introduced to distinguish general and specific rules. To ensure the accuracy of the events classification, it is important to notify the administrator about these exception cases.

In addition to the experiments with commercial IDS rules, we have also analyzed rule sets employed by Snort

¹Due to the proprietary nature of this IDS, the rule sets were provided under non-disclosure agreement.

²The details of the discovered conflicts can be found in <http://iscx.ca/results/>.

Rule set	# of rules	Misconfigurations	
		total	details
Commercial IDS product			
	45	17	Exception: 6 Correlation:1 Redundancy:1 Redundant condition:9
Snort IDS			
attack-response	16	1	Verbosity:1
specific-threats	24	0	
ddos	30	2	Verbosity:2
chat	34	7	Redundancy: 7 (inclusive-2 intersect-5)
pop3	35	0	
voip	47	1	Verbosity:1
policy	66	7	Redundancy: 1 (inclusive-1) Verbosity: 6
ftp	75	3	Redundancy: 3 (inclusive-3)
sql	89	23	Redundancy: 1 (inclusive-1) Verbosity: 21
smtp	90	14	Redundancy: 6 (intersect-6) Verbosity:11
icmp-info	93	6	Redundancy: 6 (inclusive-6)
web-cgi	357	83	Redundancy: 82 (inclusive-78 intersect-2 matching-2) Verbosity:1
web-misc	367	41	Redundancy: 34 (inclusive-28 intersect-6) Verbosity:7
web-client	712	8	Redundancy: 5 (inclusive-3 matching-2) Verbosity: 3
Total Snort conflicts: 216			Redundancy:154 (inclusive-122 intersect-27 matching-5) Verbosity:62
Bleeding Edge threats			
bleeding-attack_response	38	3	Redundancy:3 (intersect-3)
bleeding-exploit	257	13	Redundancy:1 (inclusive-1) Verbosity:12
bleeding-web	125	5	Redundancy:5 (intersect-4 inclusive-1)
Total Bleeding Edge conflicts: 21			Redundancy:9 (inclusive-2 intersect-7) Verbosity:12

Table 2. Misconfigurations detected in IDS rule sets

IDS and Bleeding Edge threats. Since Snort and Bleeding Edge Threats rules are configured by default with the same action *alert*, the misconfigurations discovered in these sets were classified as verbosity and redundancy.

As the results show the redundancy constitutes the majority of the identified conflicts (154 out of 216). Among these are 5 cases of matching rules, i.e., rules with the same fields and values, but labeled with differ-

ent id numbers and alert messages. The rest of redundancy includes inclusive and intersect cases, i.e., rules with subset/superset and intersecting relations, respectively. Some of the found conflicts are given in Table 3. In practice this generally means that several rules can fully or partially match the network packet. To resolve this situation, Snort [16] forces the rule engine to select the best match. However, since each rule processing involves expensive computation, reducing the amount of rule redundancy is one of the ways to potentially optimize rule engine performance.

Compared to the redundancy cases, the number of the discovered verbosity conflicts is relatively small (62 vs. 154). Most verbosity cases in the Snort rules have common pattern, where the content of the rules differs by a value of one of the fields, e.g., the destination/source port number or protocol. Since the processing of each rule requires a separate handling, combining these rules using comma-separated list of values would be more efficient.

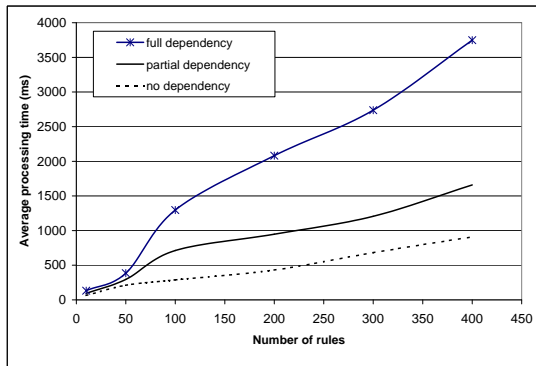


Figure 4. Processing time evaluation

Performance We have also conducted a set of experiments to measure the performance of the conflict detection algorithms on the artificial rule sets with full (all rules are interdependent), partial (50% of interdependent rules) and no dependency (no interdependent rules) among rules. The experiments were run on Intel(R) Pentium(R) 4 with CPU of 2.40GHz and 1.5GB of RAM. The results of these experiments given in Figure 4 show that the analysis of rule set with full dependency might be costly (3.7 sec for 400 rules). However, since the full dependency rule sets rarely exist in IDS, such cases can be processed off-line. The other two plots showing partial and no dependency have reasonable process time requirements that we consider suitable for off-line as well as online analysis.

7 Conclusion and Future work

The prevalent use of signature-based approach in modern IDS emphasizes the importance of efficient and conflict-free management of the employed rule set. This paper describes the problem and presents an effective approach to conflict discovery in the rule sets. Specifically, we presented a classification of intrusion detection rule misconfigurations and propose a set of algorithms to detect these conflicts through the analysis of the rule relations. We experimentally validated our approach on the example of the rule sets provided by the vendor of the commercial IDS, a Snort IDS and Bleeding Edge Threats and present the discovered misconfigurations.

While the current approach primarily focuses on the syntactical analysis of IDS rules, in our future work, we plan to explore the rules' semantic part to discover conflicts hidden behind the use of different syntaxes.

8 Acknowledgement

The authors graciously acknowledge the funding from NSERC and the Atlantic Canada Opportunity Agency (ACOA) through the Atlantic Innovation Fund (AIF) to Ali A. Ghorbani.

References

- [1] Bleeding edge threats. Available from "http://www.bleedingthreats.net/", 2008.
- [2] Qradar. Available from "http://www.q1labs.com/", 2008.
- [3] *SnortTM Users Manual 2.8.2*, 2008. Available from "http://www.snort.org/docs/snort_htmanuals/htmanual_282/".
- [4] T. Abbes, A. Bouhoula, and M. Rusinowitch. An inference system for detecting firewall filtering rules anomalies. In *Proceedings of the ACM symposium on Applied computing*, pages 2122–2128, 2008.
- [5] E. Al-Shaer and H. Hamed. Modeling and management of firewall policies. *IEEE Transactions on Network and Service Management*, 1(1), April 2004.
- [6] 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.
- [7] J. G. Alfaro, F. Cuppens, and N. Cuppens-Boulahia. Analysis of policy anomalies on distributed network security setups. In *Proceedings of the European Symposium on Research in Computer Security*, pages 496–511. Springer, 2006.
- [8] S. Antonatos, M. Polychronakis, P. Akritidis, K. G. Anagnostakis, and E. P. Markatos. Piranha: Fast and memory-efficient pattern matching for intrusion detection. In *Proceedings of the IFIP International Information Security Conference*, pages 393–408, 2005.
- [9] 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*

<p>Matching redundancy:</p> <pre> alert tcp \$EXTERNAL_NET any->\$HTTP_SERVERS \$HTTP_PORTS (msg:"WEB-CGI alert.cgi access"; flow:to_server,established; uricontent: "/alert.cgi"; nocase; sid:2195;) alert tcp \$EXTERNAL_NET any->\$HTTP_SERVERS \$HTTP_PORTS (msg:"WEB-CGI catgy.cgi access"; flow:to_server,established; uricontent: "/alert.cgi"; nocase; sid:2196;) </pre>
<p>Inclusive redundancy:</p> <pre> alert tcp \$EXTERNAL_NET any->\$SQL_SERVERS 1433 (flow:to_server,established; content:"- 00 - 00 ";sid:2000373;) alert tcp \$EXTERNAL_NET any->\$SQL_SERVERS 1433 (flow:to_server,established; content:"' 00 "; content:"- 00 - 00 "; sid:2000488;) </pre>
<p>Intersect redundancy:</p> <pre> alert tcp any any<>any 5101 (msg:"CHAT Yahoo IM message"; flow:established; content:"YMSG"; depth:4; nocase; sid:2457;) alert tcp \$EXTERNAL_NET 5050->\$HOME_NET any (msg:"CHAT Yahoo IM successful logon"; flow:from_server,established; content:"YMSG";depth:4; nocase; content:" 00 01 "; depth:2; offset:10; sid:2450;) </pre>
<p>Verbosity:</p> <pre> alert tcp \$EXTERNAL_NET any->\$HOME_NET 8181 (msg:"WEB-MISC PIX firewall manager directory traversal attempt"; flow:to_server,established;content:"/../../../../";sid:1498; rev:9;) alert tcp \$EXTERNAL_NET any->\$HOME_NET 4080 (msg:"WEB-MISC iChat directory traversal attempt"; flow:to_server,established; content:"/../../../../";sid:1604; rev:7;) </pre>

Table 3. The examples of detected misconfigurations.

- DARPA Information Survivability Conference and Exposition*, pages 1012–1026, 2001.
- [10] S. Dharmapurikar and J. W. Lockwood. Fast and scalable pattern matching for network intrusion detection systems. *IEEE Journal on Selected Areas in Communications*, 24(10):1781–1792, 2006.
- [11] P. Eronen and J. Zitting. An expert system for analyzing firewall rules. In *6th Nordic Workshop on Secure IT Systems*, pages 100–107, 2001.
- [12] Z. Fu and S. F. Wu. Automatic Generation of IPSec/VPN Security Policies In an Intra-Domain Environment. In *Proceedings of the International Workshop on Distributed System Operation & Management*, pages 279–290, 2001.
- [13] M. G. Gouda and X.-Y. A. Liu. Firewall design: Consistency, completeness, and compactness. In *Proceedings of the International Conference on Distributed Computing Systems*, pages 320–327, 2004.
- [14] H. Hamed and E. Al-Shaer. Taxonomy of conflicts in network security policies. *IEEE Communications Magazine*, 44(3):134–141, March 2006.
- [15] J. Wack, K. Cutler, and J. Pole. Guidelines on firewalls and firewall policy. Technical Report SP 800-41, NIST Recommendations, Jan 2002.
- [16] M. Norton. Optimizing pattern matching for intrusion detection. Available from "http://docs.idsresearch.org/OptimizingPatternMatchingForIDS.pdf", 2004.
- [17] H. Song and J. W. Lockwood. Efficient packet classification for network intrusion detection using fpga. In *Proceedings of the international symposium on Field-programmable gate arrays*, pages 238–245, 2005.
- [18] I. Sourdis, V. Dimopoulos, D. Pnevmatikatos, and S. Vassiliadis. Packet pre-filtering for network intrusion detection. In *Proceedings of the ACM/IEEE symposium on Architecture for networking and communications systems*, pages 183–192, 2006.
- [19] Third Brigade, Inc. *OSSEC Manual*, 2008. Available from "http://www.ossec.net/main/manual/#rules".
- [20] T. E. Uribe and S. Cheung. Automatic analysis of firewall and network intrusion detection system configurations. In *Proceedings of the ACM workshop on Formal methods in security engineering*, pages 66–74, 2004.
- [21] 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 the IEEE Symposium on Security and Privacy*, pages 199–213, 2006.