Communications Research Centre Canada
An Agency of Industry Canada

Centre de recherches sur les communications Canada
Un organisme d'Industrie Canada

Automatic Generation of Intrusion Detection Verification Rules

Report on research progress to September 1, 2008

Frederic Massicotte[1], Yvan Labiche[3] and Lionel C. Briand[2,3]

[1] Communications Research Centre Canada
[2] Simula Research Laboratory and University of Oslo
[3] Software Quality Engineering Laboratory Department of Systems and Computer Engineering, Carleton University

Ottawa, September 2008

Communications Research Centre Canada
CRC Note No. VPNT2008/02

Canada

CRC

Automatic Generation of Intrusion Detection Verification Rules

Report on research progress to

September 1, 2008

Frederic Massicotte[1], Yvan Labiche[3] and Lionel C. Briand[2,3]

[1] *Communications Research Centre Canada*

[2] *Simula Research Laboratory and University of Oslo*

[3] *Software Quality Engineering Laboratory Department of Systems and Computer Engineering, Carleton University*

CRC Note No. VPNT2008/02

Ottawa, September 2008

Abstract

An Intrusion Detection System (IDS) is a crucial element of a network security posture. One class of IDS, called signature-based network IDSs, monitors network traffic, looking for evidence of malicious behavior as specified in attack descriptions (referred to as signatures).

Many studies have reported that IDSs can generate thousands of alarms a day, many of which are false alarms. The problem often lies in the low accuracy of IDS signatures. It is therefore important to have more accurate signatures in order to reduce the number of false alarms. One part of the false alarm problem is the inability of IDSs to verify attacks (i.e. distinguish between successful and failed attacks). If IDSs were able to accurately verify attacks, this would reduce the number of false alarms a network administrator has to investigate.

In this note, we demonstrate the feasibility of using a data mining algorithm to automatically generate IDS verification rules. We show that this automated approach is effective in reducing the number of false alarms when compared to other widely used and maintained IDSs.

Résumé

Les systèmes de détection d'intrusion réseau (SDI) sont cruciaux pour la sécurité des réseaux. Une classe de SDI, les SDI réseaux utilisant des signatures, analyse le trafic du réseau pour trouver des preuves de comportements illégaux et malicieux. Ces comportements sont spécifiés à l'aide de signatures.

Plusieurs études ont identifié que les SDI peuvent générer des milliers d'alarmes par jours et plusieurs d'entre elles sont de fausses alarmes. Le problème est dans la piètre précision des signatures. Il faut donc être en me sure de générer des signatures plus précises qui réduiraient le nombre de fausses alarmes. Une partie du problème des fausses alarmes réside dans l'incapacité des SDI de vérifier le résultat des attaques (i.e. distinguer entre les attaques réussies et ceux qui n'ont pas fonctionné). Si les SDI seraient en mesure de précisément vérifier les attaques ceci réduirait le nombre de faux positifs que les administrateurs réseaux auraient à investiguer.

Dans cette note, nous démontrons la possibilité d'utiliser une approche d'apprentissage automatique pour générer des règles de vérification pour les SDI. Nous démontrons que cette approche automatique est efficace à réduire les faux positifs comparativement à des SDI connus.

Table of Contents

Introduction

Many studies report that IDSs, including signature-based network IDSs, can generate thousands of alarms a day, and that a large proportion of these are false alarms [11, 19]. These false alarms cannot all be checked by network administrators in a reasonable amount of time, and IDSs are therefore inefficient at accomplishing their network protection role.

The false alarm problem can be divided into two sub-problems: the Detection Problem and the Verification Problem. The Detection Problem refers to the ability of an IDS to distinguish attack traffic from the rest of the traffic (i.e. normal or abnormal). We refer to attack traffic as any traffic that violates the organizational security policies, normal traffic as legitimate traffic that does not contain attack attempts, and abnormal traffic as everything else (e.g., traffic that does not respect protocol specifications). On the other hand, the Verification Problem refers to the ability of an IDS to distinguish between successful and failed attack attempts. For instance, in many situations, IDSs provide the same alarms whether the attack is successful or not. As a result, it is difficult for network administrators to distinguish between alarms related to cases where the target system is compromised from other cases.

Five different complementary approaches are now recognized to address the verification problem. The first approach is to use the network configuration in context with the attack. This approach is based on the hypothesis that an attack can only be successful if it is able to reach the target system. For example, the time to live (TTL) of an IP packet can be used to verify attacks [15]: When the TTL of an attack packet is too low for the packet to reach the target system, the attack is impossible, and the attack can be ignored or an alarm can be raised by the IDS with the proper context (i.e. the attack cannot reach its destination). The second approach uses the protocol specification to identify whether the attack can be successful or not. This approach is based on the hypothesis that protocol implementations always behave according to their specification and that an attack is only possible when it is sent at the proper state of the communication protocol, e.g., for a (attack) TCP packet to be accepted by a target system, it has to be part of an open TCP session. The stateful improvement to IDSs (e.g. [5, 15]) is a good example of a protocol specification approach that seeks to address the verification problem. The third approach is to rely on the knowledge of the target system products (e.g., name and version) and vulnerability databases (e.g., SecurityFocus[1]) to determine whether the target system is indeed vulnerable or not to an attack (i.e. whether the attack can be successful or not) [3]. The fourth approach is to rely on a vulnerability assessment of the target system using a dedicated tool (e.g. Nessus[2]) to more accurately verify attacks than by using the target system products [9]. The fifth approach relies on the target system reaction to an attack and is used in well-known and used IDSs (e.g. Snort [5, 15, 21], Bro [5, 15, 21] and Snort UC Davis [5, 15, 21]). This approach is to look at messages (e.g. an error message) or behaviors (e.g. the target system does not respond) before and after the attack to decide whether or not the attack has been successful. This approach, as implemented in those IDSs, has been shown to partially address the verification problem [13].

In this note, we address the verification problem using the target reaction approach. To do so we employ a data mining technique and real attack scenarios that allow us to automatically generate IDS verification rules that predict the success or failure of attacks. Then these rules were used with a modified version of Snort and the results were compared with Bro, Snort and Snort UC Davis. The results indicate that our approach (1) can be used to automatically learn IDS verification rules, (2) can speed up the process of generating these rules, and (3) reduces human errors that could result in incorrect rules (e.g., as identified for Bro). The contribution of this note is two-fold: (1) We propose an approach that uses a standard data mining algorithm to automatically generate IDS verification rules, thus reducing the effort and errors related to the manual generation of such rules; (2) We report on a case study that shows that our verification rules improve the accuracy of Snort and Bro.

The rest of this note is structured as follows. Section 1 describes related work, and Section 2 provides some background information on data mining technologies. Section 3 presents our approach and tool. Section 4 describes the IDS we used to test and evaluate the accuracy of our IDS verification rules. Section 5 describes a case study. Conclusions are drawn in Section 6.

1. Related Work

In our survey of techniques used to automate the process of generating IDS rules [1, 6, 8, 10, 11, 19], we did not find any approach addressing the automatic generation of IDS verification rules. In fact, most of the approaches reviewed in this section generate IDS rules that distinguish normal/abnormal traffic from attack traffic. As a result, these approaches are related to the detection problem and not to the verification problem.

[1] www.secuirtyfocus.com
[2] www.nessus.org

Play	Outlook	Temperature	Windy
Yes	Sunny	Hot	No
Yes	Sunny	Hot	Yes
Yes	Overcast	Hot	No
No	Rainy	Mild	No
No	Rainy	Cold	No
No	Rainy	Hot	Yes
Yes	Overcast	Hot	Yes
Yes	Sunny	Mild	Yes
Yes	Sunny	Cold	Yes
No	Rainy	Mild	Yes
Yes	Sunny	Mild	No
No	Overcast	Mild	Yes
Yes	Overcast	Hot	Yes
No	Rainy	Cold	Yes

(a) Training Data Set

```
if Outlook = Sunny then
   Yes
else if Temperature = Hot then
   if Outlook = Overcast then
      Yes
   else
      No
else
   No
```

(b) Rule

	Play	Not Play
Play	8	0
Not Play	0	6

(c) Confusion Matrix

Figure 1 Example of use of a data mining technique

The authors of [6-8] focus on distinguishing between IDS events related to network problems such as misconfigured equipment and IDS events related to real attacks. For instance, in [8], the authors use a data mining technique (specifically a clustering technique) to learn under which conditions alarms are kept or discarded by network administrators thereby generating rules that automate the classification of IDS events. In [19], the authors use a data mining approach to infer which IDS events are related to the same attack. This process is called *IDS events correlation*. This approach is used to reduce the number of IDS events by grouping those that are in the same attack scenarios. This provides a better understanding of attacks to network administrators. Currently, IDSs are only able to see an attack as a unit (i.e. one attack exploits one vulnerability) and they are unable to consolidate attacks into an attack scenario. For example, an attacker will first scan a system for vulnerabilities, and will then try to exploit several vulnerabilities until he gets control of the target system. Each scan and vulnerability exploitation raises IDS events, but IDSs are currently unable to relate them.

Different data mining algorithms are used in [10], and later in [1, 11], to automatically generate detection rules. Using events from the system event logs of the target system, data mining algorithms generate IDS rules that represent normal user behavior [10]. The generated IDS rules are event sequences that represent normal user behaviors. An event log sequence that does not match any rule is interpreted by the IDS as an abnormal behavior of the user (i.e. potential attack) and an alarm is raised by the IDS.

The authors of [1, 11] used different data mining algorithms that can all be classified as clustering learning algorithms [20]. These algorithms seek to group objects so that those within a given group (i.e. cluster) are similar. The authors of [1, 11] used algorithms that can be classified as association learning algorithms [20]. These algorithms seek to identify the implications (i.e. associations) between objects. We believe that the detection and verification problems are more classification problems [20] than clustering problems, thus requiring classification algorithms (Section 3.3.1). The authors of [10] used a classification algorithm (i.e., Ripper [2]). However, their approach was for an anomaly host-based IDS (as opposed to our signature network-based approach) and it is difficult to assess whether or not it is practical or feasible to learn every possible normal behavior of a user (i.e., anomaly detection).

We believe that data mining approaches such as the one used in [10] are applicable to the verification problem for two reasons. First, the detection and verification problems are similar problems from a data mining point of view since they are both classification problems. For the detection problem, the objective is to classify normal/abnormal traffic and attack traffic, whereas for the verification problem, the objective is to classify successful attacks and failed attack attempts. Second, the algorithms used to address the detection problem are generic classification techniques that can be used in various application contexts. It is therefore relevant to investigate whether they can address the verification problem.

2. Background on Data Mining Technology

This section introduces data mining terminology to facilitate the understanding of our approach. For example, consider the (over) simplified problem illustrated in Figure 1, where one wants to identify under which conditions we can play outside given the values of three characteristics, namely Outlook, Temperature and Windy. The input to the data mining algorithm is a table (Figure 1 (a)), called the training data set, where rows and columns are referred to as instances and attributes, respectively. Each row is an instance of values of the attributes. Rows can be used by a data mining algorithm to predict the values of an attribute (e.g., Play) using the values of the other attributes. In the data

Figure 2 Automatic Generation of IDS Verification Rules

mining literature, this is called a *classification problem* [20]. The input of Figure 1 (a) contains 14 instances, characterized by four attributes.

The output of a data mining algorithm is called a *prediction model* and takes the form of a rule involving attribute values that predict on attribute's values. Figure 1 (b) shows one such prediction model for the training data set of Figure 1 (a) when we use the C4.5 [16] data mining algorithm (this algorithm is further discussed in Section 3.3.1). The rule has three predicates: `Outlook = Sunny`, `Temperature = Hot` and `Outlook = Overcast`; which are used to specify whether we can go play outside. This rule specifies that we can play outside when it is sunny or when it is overcast and hot, otherwise we cannot go play outside.

Note that in general, the set of predicates that appear in a rule is a subset of the set of attributes or attribute values since not all attribute (values) in the input table are necessarily used to predict the predicted attribute values.

To evaluate the accuracy of generated rules, it is common practice to use a confusion matrix [20], such as the one of Figure 1 (c): a confusion matrix is in general provided along with the rule by the data mining algorithm. A confusion matrix shows the number of instances for which the actual values of the attribute are the rows (i.e., the value from the training data set) and the predicted values of that attribute are the columns (i.e., the values when using the rule). Figure 1 (c) shows that all the instances are correctly classified, i.e., the predicted value equals the value in the training data set. Assuming a larger training data set was used and that the rule misclassified 10 instances, whereby the predicted value is Play whereas the actual value is Not Play, the confusion matrix would have a value of 10 in the Not Play row and the Play column.

When the data set is large enough, or two different sets are available, it is common to build a prediction model using one part of the data and to verify its accuracy on the other part of the data: thus the wording of training data set and testing data set. When the data set is not large, a standard technique called n-folds cross-validation [20] is used to randomly create training and testing data sets from the data set to assess and evaluate the prediction models. It is common practice to using this technique with n=10 [20], which is what we do in this note.

3. A Data Mining Approach for Automatic Generation of IDS Rules

Our IDS Verification Rule Generator (IDS-VRG) tool was developed with two requirements in mind: using the target reaction approach to automatically improve the accuracy of current IDS verification rules and to identify new IDS verification rules. Thus, in this note, the prediction models are also referred to as IDS Verification Rule to simplify the description. The approach is iterative, as illustrated in Figure 2. First, the Capture System (step 1) captures and/or generates labeled traffic traces using various sources (e.g. malware execution, vulnerability exploitation program execution, corporate network traffic) and stores them in a *labeled data set*. Labels indicate whether traffic traces contain successful attacks, failed attacks, or are normal/abnormal traffic traces. Note that normal/abnormal traffic traces are not needed when actually verifying whether an attack has succeeded. However, as discussed in Section 3.3.2, this is required when generating rules to be used for verification. The *IDS Verification Rule Generator* (step 2) then executes the data mining algorithm to generate rules, reports on the accuracy of the generated rules, and requests the user to check (step 3) whether the generated rules are semantically sound and syntactically complete (i.e., the verification rules make sense for a network security expert to use them in an IDS to verify a specific attack) (Section 3.3). A network, signature-based IDS can then be used to compare existing rules (referred to as "Current IDS Rules" in Figure 2) and the generated rules by using both sets of rules on attack traffic (i.e., the labeled data set). If the generated rules are deemed better than existing rules, they can be used instead. The procedure is iterative since, when new attack traces are available, new rules can be generated by the IDS Verification Rule Generator and compared to existing ones.

The selection of the IDS that we use in step 3 is discussed in Section 3.1. The initial selection of network traffic (step 1) is discussed in Section 3.2. The tasks performed by the IDS Verification Rule Generator (Step 2) are discussed in Section 3.3.

3.1 Selection of an IDS and IDS Version

Our approach requires an IDS rule database (i.e., the Current IDS Rules in the first iteration of Figure 2) and an IDS to be tested (step 3 in Figure 2). We selected the Snort [5] rule database since it is one of the most widely used and maintained open source IDS. We also used it for two other reasons. First, previous work has shown that work still needs to be done to improve the Snort verification accuracy [13, 21]. Second, another widely used and maintained IDS, namely Bro [15], and a proposed solution to the verification problem based on Snort, namely Snort UC Davis [21], generate their verification rules using the Snort rules. It is important to mention that these approaches do not use data mining techniques to generate their verification rules. Instead, verification aspects are manually added to the detection rules. Thus, for comparison purposes, the Snort rules are good starting points to assess our approach, and to identify whether it is feasible to automatically generate equivalent or better verification rules than the ones contained in the Bro and Snort UC Davis rule databases.

We selected the Snort 2.3.0[3] rule database as an initial set of IDS rules because it is the one used by Snort UC Davis. Bro is independent of the Snort rule database version it uses since any Snort rule database can be automatically converted into Bro rules using *s2b*.

3.2 Labeled Traffic Traces

In order to obtain meaningful and accurate verification rules using a data mining technique, we cannot simply use the traffic traces, such as inputs related to the attack, the target, output on success or failure. Our experience with data mining techniques is that using raw data typically leads to meaningless and inaccurate rules because the data mining algorithm cannot learn what input or output properties are potentially of interest but only which ones matter once they are defined. In other words, without some additional guidance, the data mining algorithm is unlikely to find the precise conditions under which an attack succeeds. This guidance, in our context, comes in the form of attributes that characterize traffic traces. We first specify those attributes (Section 3.2.1), and then discuss how we generated the training data set using those attributes (Section 3.2.2).

3.2.1 Attributes

The input to the data mining algorithms we used is a table where the rows represent *instances*, i.e., traffic traces, and the columns represent *attributes* of those instances, i.e., characteristics of the traces. We used three groups of attributes: the attack result, the rule(s) triggered by the attack traces, and the target reactions identified in the traffic traces. We discuss these attributes below.

Result is the predicted value of the verification rules. The possible values are `success`, `failure`, and `normal/abnormal`.

The *rule attributes* are the Snort rules (*Snort(i):m* where *i* is the rule number and *m* the message associated with this rule) that are triggered on each traffic trace. The possible values are `true` and `false`.

The *target reaction attributes* are indicators of failure or success of attacks. We build upon the approaches of Bro [17] and Snort UC Davis [21], where standard protocol messages (e.g. HTTP 200 OK) are monitored after an attack is detected to infer the success or failure of the attack. We also added our own reaction *indicators*. We identified that TCP/UDP port status and the host status of the target system can also be indicators of success or failure of an attack. These target reaction *indicator* attributes are referred to as:

- *StandardMessage(protocol,i)*: tells whether or not a standard message *i* from a *protocol* (e.g. HTTP, FTP, SMTP, IMAP or POP) is in the traffic traces. The values are `true` and `false`.
- *HostPortState(host,port,i,j)*: tells whether or not a *port* of a *host* (i.e. target, attacker) has changed from state *i* (i.e. closed, open) to state *j* (i.e. closed, open). The values are `true` and `false`.
- *HostState(host,i)*: tells whether or not a *host* (i.e. target, attacker) is in a particular state *i* (i.e. up, down). The values are `true` and `false`.

3.2.2 Training Data Set Generation

We selected the labeled traffic traces presented in [13] as our data set. We made the effort to make this data set similar to attacks perpetrated on the Internet. We carefully picked different Vulnerability Exploitation Programs (VEPs) that exploit vulnerabilities in the most popular key services (e.g. HTTP, FTP, SMTP) that are running on the most common operating systems (e.g., Linux, Windows, FreeBSD). Using 124 VEPs, covering a total of 57 vulnerabilities, and 108 target systems every possible combination of VEP, VEP configuration and target system was exercised. This resulted in a data set of 10,000 traffic traces, labeled with the information on whether the attack was successful (label *success*) or not (label *failure*): see [13] for details. This data set is limited in two aspects. First, the number of VEPs

[3] This Snort rule database was published in January 2005.

available (i.e. which can be downloaded) and usable (i.e. the code compiles and it is possible to make it work) is substantially smaller than the number of documented vulnerabilities. Similarly, only a subset of possible target systems has been used. As this data set becomes larger, we will be able to go through new cycles of our iterative process. This data set is however the most appropriate set of labeled traffic traces available to date for attack traffic, and contains more recent attacks than the DARPA data sets which have been used for a long time by the research community. An instance is generated for each traffic trace from the VEP labeled data set. The *Result* attribute (either `success` or `failure`) is provided in the labeled data set, as discussed previously. The *Snort(i)* attributes are generated for each traffic trace using Snort 2.3.0. For the target reaction indicators, we created another Snort rule database since the default Snort rule database does not contains rules related to all the target reaction indicators. We used Snort with this new rule database to generate the target reaction indicator attributes.

We are aware that recent techniques can be used to circumvent the target reaction approach (e.g. [18]), whereby the attacker ensures that when an attack is successful the target reacts as if the attack had failed. The VEP data set we used does not contain traffic traces using these techniques. We plan to use such traffic traces in our future work.

In our context, another limitation of this VEP data set is that it does not contain normal/abnormal traffic. We thus added simulated instances to the training data sets for normal/abnormal traffic instead of using real normal/abnormal traffic. For this approach to be useful, we have to consider two issues: how to simulate normal/abnormal traffic instances and how many simulated instances are needed in the training data sets. To simulate normal/abnormal traffic instances, we looked, one by one, at every attribute contained in the training data sets. When the attribute could be part of normal/abnormal traffic, we assigned it the `true` value and otherwise the `false` value. We assumed that no attack detection rule from Snort is triggered on normal/abnormal traffic. Thus, every *Snort(i)* attribute that corresponds to a rule that detects an attack has a value of `false` for a simulated instance. We know this is a strong hypothesis. However, we were able to generate accurate verification rules using this hypothesis as long as the detection rule is accurate (Section 5.2). For instance, suppose a training data set with the two attributes `Snort(971):WEB-IIS ISAPI .printer access` (i.e. an attack attempt) and `StandardMessage(HTTP,200)` (i.e. a target reaction indicator). The corresponding simulated instance is `normal/abnormal, false, true` since the .printer attack attempt cannot be part of normal/abnormal traffic (based on our hypothesis) and for an HTTP 200 OK message, it is possible. With regard to the number of simulated instances used in the training data set, our choice of heuristic was motivated by the necessity to ensure that the data mining algorithm still provides a verification rule (i.e. a rule that distinguishes between successful and failed attack), which focuses us to be greedy on the number of simulated instances we add. As a heuristic, we added $n/2 + 1$ simulated normal/abnormal traffic instances in the data set (the result is truncated), where n is the smallest number of instances between those that have the `success` and `failure` values for their result attribute. For instance, if we have 91 instances with the `failure` value and 9 with the `success` value then 5 simulated normal/abnormal traffic instances are added.

It is important to notice that a specific attack only exploits a specific vulnerability and that only a specific group of rules are used in IDSs to identify an attack. Thus, a verification rule has to be generated for each vulnerability to reflect the reality of the relation between IDS detection rules, attacks and vulnerabilities. Consequently, we grouped the traffic traces (i.e. instances) into multiple training data sets: one per vulnerability. We only used the vulnerabilities for which Snort has corresponding rules and for which the VEP data set has successful and failed attack attempts. Indeed, the data mining algorithm needs a set of instances where all the possible attribute values are used to produce accurate rules. In this case, 16 training data sets have been created to generate verification rules for 16 of the 57 candidate vulnerabilities of the VEP data set (see Section 5.2 for the list of the 16 vulnerabilities we used).

3.3 IDS Verification Rule Generator

In this section, we discuss our choice of data mining algorithm(s) (Section 3.3.1) and the need for normal/abnormal traffic traces (Section 3.3.2). We then discuss evaluation criteria for generated IDS verification rules (Section 3.3.3).

3.3.1 Generation of IDS Verification Rules

Once the input data we have just described is available, classification algorithms or decision tree algorithms can be considered to solve our classification problem [20].

To solve a classification, some techniques, like C4.5 [16], partition the training data set in a stepwise manner using complex algorithms and heuristics to avoid over-fitting the data with the goal of generating models that are as simple as possible. Others, like Ripper [2], are so-called covering algorithms that generate rules in a stepwise manner, removing observations that are "covered" by the rule at each step so that the next step works on a reduced set of observations. With covering algorithms, predicates in a rule are interdependent in the sense that they form a "decision list" where predicates are supposed to be applicable in the order in which they were generated. One important issue is that the order

Result	HTTP Attack	HTTP 200 OK
success	true	true
failure	true	false
normal/abnormal	false	true

(a) Training Data Set

```
if HTTP 200 OK = true then
   success
else
   failure
```

(b) Rule without normal/abnormal instances

```
if HTTP Attack = true then
  if HTTP 200 OK = true then
     success
  else
     failure
else
   normal/abnormal
```

(c) Rule with normal/abnormal instances

Figure 3 Justifying the need for normal/abnormal traffic: an example

in which predicates have to be evaluated when evaluating the rule may not be the order in which the information used to evaluate the predicates is available, i.e., the order in which packets are observed. We therefore decided to use C4.5.

However, the usage of classification and decision tree algorithms prevents IDS-VRG from automatically generating *IDS ready* verification rules. For example, these algorithms do not generate rules with temporal constraints between the predicates of the verification rules. In the case of IDS verification rules, we know that each predicate represents information arriving at different periods of time. The order in which the predicates should be monitored by an IDS is not specified in the verification rules generated by classification or decision tree algorithms. The temporal constraints are not required by the IDS to verify attacks. However, without them the verification rules would not be as accurate (i.e., it would raise false positives). Therefore the output of those algorithms has to be post-processed by a human before including them in an IDS rule database. A network administrator would have enough knowledge, in terms of network administration, communication protocols and vulnerabilities, to find the order in which the predicates composing a rule have to be evaluated. For instance, in many IDS verification rules, the predicate that detects the attack has to be triggered before the predicate that verifies the reaction of the target system. For instance, in the case of denials of service, the attack arrives first and then the reaction of the denial of services is noticed by the IDS on the target system. Consequently, from the verification rules generated by a classification or decision tree algorithm, a network security expert can easily infer the proper temporal order of the predicates (Section 5.2.2). An algorithm that would properly order the predicates in the verification rules would be desirable (e.g. a state machine learning algorithm). However, it is unclear at this stage whether this would lead to better automation and accuracy results. Thus, every verification rule that was generated using the classification and decision tree algorithms was manually modified to reflect the temporal constraints intended among the predicates contained in the verification rules.

3.3.2 Normal/Abnormal Traffic

Normal/abnormal traffic is not required by definition for addressing the verification problem. However, to complete the specification of the verification rules the IDS behavior on normal/abnormal is also required.

Consider for instance the training data set in Figure 3 (a). Figure 3 (b) and (c) show verification rules generated with the C4.5 data mining algorithm [16]. For Figure 3 (b) normal/abnormal traffic is not used (i.e., the last row of the training data set is not used), whereas for Figure 3 (c) normal/abnormal traffic is used. The verification rule in Figure 3 (b) has one predicate: `HTTP 200 OK = true`, and specifies that when the IDS sees a HTTP 200 OK message, this indicates a successful attack. (Recall from Section 2 that a prediction model does not necessarily involve all the attributes, when some attributes do not help classify instances.) This verification rule is incorrect because the `HTTP 200 OK = true` can also hold for normal/abnormal traffic. The verification rule in Figure 3 (c) is what a network security expert would expect as a verification rule where the two predicates `HTTP Attack = true` and `HTTP 200 OK = true` are used together to respectively detect the attack and distinguish between a successful and failed attack attempt.

3.3.3 Evaluation of IDS Verification Rules

We evaluate the IDS verification rules generated by the classification and decision tree algorithms using two criteria: accuracy and usability. Accuracy relates to the generation of false positives and false negatives, and is evaluated with a confusion matrix (recall Section 2): we used a 10-fold cross-validation approach. Usability relates to whether rules are semantically sound and syntactically complete (i.e. the verification rules make sense for a network security expert to use them in an IDS to verify a specific attack), as discussed below.

We say that a verification rules is *usable* if it meets the two following criteria: it has to be syntactically complete, i.e., it has to include all three possible values of the *Result* attribute (i.e. `success`, `failure` and `normal/abnormal`) (criterion 1); and it has to be semantically sound, i.e., it must contain at least one predicate referring to the vulnerability used to create the training data set, i.e., one Snort rule number (criterion 2). These criteria are systematic, can be used by a system to automatically verify the usability of a verification rule, and do not depend on the user of IDS-VRG.

To illustrate this, Figure 4 presents an example of an unusable (on the left) and a usable (on the right) IDS verification rule generated using the training data set for the vulnerability corresponding to Bugtraq ID (BID) 2674 (which is detected by Snort rule number 971). The verification rule on the left specifies that if Snort raises an alarm for rule number 1292, then the attack is successful, otherwise the attack failed. This verification rule is syntactically incomplete because it does not provide a conclusion for all predicted values: `normal/abnormal` is missing (criterion 1). Moreover, this verification rule is not semantically sound because it does not contain a Snort rule to detect the exploitation of BID 2674 (criterion 2): Snort rule number 1292 specifies a target reaction (i.e. `directory listing`) that could be related to an attack.

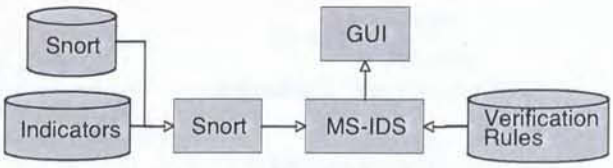

Figure 5 MS-IDS Overview

Thus, this verification rule is not *usable* because it does not specify whether identifying a directory listing is part of normal/abnormal traffic (criterion 1). If a directory listing is actually part of normal/abnormal traffic, this verification rule is unable to distinguish normal/abnormal traffic from successful attacks; and if it is not part of normal/abnormal traffic this verification rule is unable to distinguish failed attacks from normal/abnormal traffic. Moreover, this verification rule is not *usable* because a directory listing is not an attack against vulnerability BID 2674 (criterion 2). The IDS verification rule presented on the right, on the other hand, is usable. It provides all possible conclusions (criterion 1) and a predicate describing the attack attempt that refers to the exploitation of BID 2674, specifically Snort rule 971 (criterion 2).

For the semantically sound criterion (criterion 2), it is also crucial to verify whether there are attributes that are equivalent. We say that two attributes a_1 and a_2 are equivalent (with respect to a training data set *tds*) if there is a bijection f between the sets of values of a_1 and a_2 such that for all instances x in *tds*, we have $f(a_1(x)) = a_2(x)$. In other words, a_2 can be seen as a renaming of a_1 for *tds*. For example, suppose that for all instances in the training data set for vulnerability BID 2674 (detected by Snort rule 971), the values of the Snort rule 1292 attribute are equal to the values of the Snort rule 971 attribute. Since the C4.5 algorithm only provides one verification rule as an output, and because it only uses one attribute for classification purposes when there are several equivalent attributes in the training data set, the generated rule could either contain the `Snort(1292)` or `Snort(971)` attributes. Therefore, if the verification rule only contains the predicate `Snort(1292) = true` the verification rule is not *usable* (criterion 2): this is the wrong rule for BID 2674. To ensure a semantically sound verification rule, we use an algorithm that identifies attribute equivalences. To do so, one possible approach could be to identify and remove equivalent attributes in the training data (i.e., before generating the verification rule) and only keep the attributes that satisfy this criterion 2. One issue though (recall the definition of equivalence) is that two attributes can be equivalent for a training data set, i.e., for a vulnerability, but may not be equivalent for another training data set (i.e., another vulnerability), which would prevent any a-priori identification and removal of equivalent attributes. In IDS-VRG, we therefore removed equivalent attributes after the generation of the verification rules. We plan to study this issue in more depth in future work.

4. Multi-Session IDS

IDS verification rules using the *HostPortState(host,port,i,j)* or *HostState(host,i)* target reaction indicators (Section 3.2.1) require a multi-session IDS engine (i.e., multiple transport layer sessions have to be analyzed to trigger such verification rules). Verification rules only using the *StandardMessage(protocol,i)* indicators can be implemented in Snort because they can be monitored using one session. Snort relies on a multi-packets IDS engine that is only able to monitor packets within one transport layer session to identify attacks, whereas some of the IDS verification rules generated with IDS-VRG require the IDS engine to monitor packets in multiple sessions to distinguish between successful and failed attacks. Moreover, the absence of any reaction from the target after an attack, required by some verification rules (Section 5.2.1), also prevents these verification rules from being implemented using the Snort IDS engine. To overcome these problems, we developed a Multi-Session IDS (MS-IDS) [12] as the IDS to test the generated verification rules.

In this section, we outline the architecture of MS-IDS. A more detailed presentation is available in [12]. MS-IDS has

```
If Snort(1292):dir listing = true then
  success
else
  Failure
```

```
if Snort(971):WEB-IIS .printer access = true then
  if StandardMessage(HTTP,response) = true then
    success
  else
    failure
else
  normal/abnormal
```

Figure 4 Verification Rule Examples for BID 2674

	Likely-To-Succeed	Likely-To-Fail
Successful	TP	FN
Failed	FP	TN

Figure 6 Verification Confusion Matrix

$$VA = \frac{TP + TN}{TP + FN + FP + TN}$$

Equation 3 Verification Accuracy

$$TP\ rate = \frac{TP}{FN + TP}$$

Equation 3 True Positive Rate

$$TN\ rate = \frac{TN}{FP + TN}$$

Equation 3 True Negative Rate

a multi-session IDS engine that can monitor complex communication patterns involving multiple packets in multiple sessions. In particular, this allows MS-IDS to capture the reaction of the target during an attack. MS-IDS is composed of two IDS engines and three IDS rule databases (see Figure 5).

The Snort IDS engine identifies relevant packets by using two IDS rule databases: the default Snort rule database and the Indicators. (Recall from Section 3.2.1 that we developed a set of Snort rules to provide the target reaction indicators required by the IDS verification rules.) All the events generated by the Snort IDS engine are sent to the MS-IDS engine. The MS-IDS engine then uses the database containing our verification rules to verify the attack attempts. The resulting events are displayed in a GUI and stored in a file.

Based on results provided in [14], we decided to use interval logic (a form of temporal logic) as a paradigm for our multi-session MS-IDS engine. Since we did not have access to an interval logic IDS engine such as the one presented in [14], we decided to develop one in Prolog. Prolog offers an ideal programming environment because it is a rule-based language that

facilitates the implementation of the interval logic rules [14] that are the basis of the interval logic.

5. Case Study

In this section, we describe the IDS verification rules obtained using IDS-VRG. Section 5.1 presents the design of this case study. Section 5.2 presents the verification rules that we obtained using IDS-VRG and a comparative analysis against Bro and Snort UC Davis when our rules are used with MS-IDS. Section 5.3 discusses the limitations of IDS-VRG.

5.1 Design

We used IDS-VRG to generate verification rules for 16 vulnerabilities (Section 3.2.2) and manually added temporal constraints to make them IDS ready (Section 3.3.1). To evaluate those rules we compared them (i.e., we executed MS-IDS) to Bro and Snort UC Davis on the attack scenarios of the VEP data set. We selected Bro and Snort UC Davis because they also contain verification rules (manually) derived from the default Snort rule database. To compare these three IDSs, we used verification metrics to assess their ability to distinguish between successful and failed attack attempts. These metrics use a confusion matrix (Figure 6) where the actual values are *Successful* attack and *Failed* attack (as reported in labeled traces in the VEP data set) and the predicted values (i.e., as reported by the IDSs) are *Likely-To-Succeed* and *Likely-To-Fail*. The cells of the matrix indicate the number of True Positives (TP), False Negatives (FN), False Positives (FP), and True Negatives (TN). To analyze the results presented in Section 5.2, we used the verification accuracy (VA) measure (Equation 3), the true positive and the true negative rates (Equation 3, Equation 3).

The *Verification Accuracy* is defined as the ratio of properly predicted values (i.e., TP and TN) over the total number of test cases. The TP rate is defined as the ratio of properly predicted values for the successful test cases (i.e., TP) over the total number of successful test cases. The TN rate is defined as the ratio of properly predicted values for the failed test cases (i.e., TN) over the total number of failed test cases. These measures have to be used together to compare the accuracy of different verification rules. VA is not sufficient since it hides the fact that false positives and false negatives do not have the same importance. Even if it is desirable to minimize false positives, this should not be done at the cost of increasing false negatives. An accurate verification rule is therefore one that has no false negatives (i.e., TP rate = 1) and that minimizes the number of false positives (i.e., TN rate ~ 1). Thus, it is a verification rule that only has true positives and maximizes the true negatives.

5.2 Results

Table 1 presents the verification accuracy (VA), the true negative (TN) rate and the true positive (TP) rate obtained when using Bro, Snort UC Davis, and MS-IDS on the attack scenarios of the 16 training data sets (Section 3.2.2). Since Bro and Snort UC Davis remain silent (i.e., do not raise an IDS event) when an attack fails, we interpret the *silent* IDS

event as a *Likely-To-Fail* IDS event, i.e., as a true negative. The BID column identifies the training data set (i.e. test case group) using the BugtraqID (BID) (i.e. vulnerability identifier) related to the vulnerability exploited in the training data set.

5.2.1 Accuracy

Table 1 shows that MS-IDS is either more accurate than or equivalent to Bro and Snort UC Davis for all the 16 vulnerabilities in the training data sets, achieving perfect (100%) True Positive and True Negative rates. We conducted a manual semantic analysis of our verification rules and compared them to Bro and Snort UC Davis to better understand the root causes of the differences between MS-IDS, Bro and Snort UC Davis. We drew a number of conclusions from this analysis.

First, our verification rules are semantically equivalent to the correct verification rules of Bro and Snort UC Davis, i.e., the rules for BID 2708 and 1806: the three IDSs achieve 100% VA, 100% TN and 100% TP.

Second, our rules improved the accuracy over Bro and Snort UC Davis in a number of cases. For example, consider our rule for BID 8035. We obtained a more accurate verification rule than Bro and Snort UC Davis because our verification rule used the port state indicators instead of the standard message of the protocol to verify the attacks.

Third, IDS-VRG also generated verification rules that could replace the incorrect verification rules in Bro for BID 2674 and 3335: both Bro and Snort UC Davis have a 0% TP rate, i.e., they did not detect successful attacks. For BID 3335, Bro is waiting for the wrong reaction from the target system (Section 5.2.2). (Note that the verification rule for BID 3335 is not instrumented in Snort UC Davis.) For BID 2674, Bro waits to evaluate whether or not the target system is a Microsoft IIS Server. Bro uses the HTTP reply message (e.g. HTTP 200 OK) from the server to gather this information. However, the server does not provide an HTTP reply message when the attack against BID 2674 is successful. Thus, Bro is unable to detect a successful attack against this vulnerability. This explains the TP rate of 0% for these two BIDs when using Bro.

Fourth, we identified problems with Snort UC Davis for BID 2674 and 4482. The verification rules that detect attacks related to these vulnerabilities are correct from a logical point of view, but they require a *not* operator to verify successful attack, which the Snort engine interprets differently than what was meant by the authors of the Snort UC Davis verification rules. For example, consider our usable verification rule for BID 2674 (i.e., the one on the right) in Figure 4. It specifies that the IDS should first identify the attack specified by Snort rule 971 and, if the attack is not followed by any HTTP response message from the target system, then the attack is successful, otherwise, the attack has failed. Figure 7 shows how Snort detects and verifies attacks for BID 2674 using the `flowbits` mechanism. Figure 7 is a simplified version of the one found in Snort UC Davis for the purpose of illustrating the problem (note that the third rule is not in Snort UC Davis, and that it does not distinguish between normal/abnormal traffic and failed attack attempts). Figure 7 and Figure 4 are very similar: Snort UC Davis rule 971 (i.e. a modified version of Snort rule 971 with the `flowbits` plug-in) is first used and then an HTTP answer from the target is either detected (third rule in Figure 7) or not (second rule in Figure 7): the last two rules in Figure 7 correspond to the two alternatives of the rule in Figure 4 It is the reaction of the target system that allows verification, and the meaning of *no HTTP message response should be sent by the target system* is therefore critical. To trigger rule 10971 (i.e. verify that the attack is successful), a packet

BID	Attack Type	MS-IDS			Bro			Snort UC Davis		
		VA	TN	TP	VA	TN	TP	VA	TN	TP
2708	WA	100%	100%	100%	100%	100%	100%	100%	100%	100%
1806	WA	100%	100%	100%	100%	100%	100%	100%	100%	100%
3335	WA	100%	100%	100%	71%	96%	0%	71%	96%	0%
2674	WA	100%	100%	100%	92%	100%	0%	92%	100%	0%
8035	WA	100%	100%	100%	96%	96%	50%	72%	71%	100%
4482	DOS	100%	100%	100%	5%	0%	100%	95%	100%	0%
514	DOS	100%	100%	100%	96%	100%	0%	96%	0%	100%
1163	DOS	100%	100%	100%	37%	0%	100%	37%	0%	100%
5556	DOS	100%	100%	100%	45%	0%	100%	45%	0%	100%
10115	DOS	100%	100%	100%	53%	0%	100%	53%	0%	100%
7106	AA	100%	100%	100%	1%	0%	100%	1%	0%	100%
7294	AA	100%	100%	100%	6%	0%	100%	6%	0%	100%
10108	AA	100%	100%	100%	18%	0%	100%	18%	0%	100%
9633	AA	100%	100%	100%	22%	0%	100%	22%	0%	100%
10116	AA	100%	100%	100%	28%	0%	100%	28%	0%	100%
8205	AA	100%	100%	100%	19%	0%	100%	19%	0%	100%
WA = Web Attack DOS = Denial of Service AA = Admin Attempt										

Table 1 IDS Verification Rules compared to Bro and Snort UC Davis

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS
(msg:"WEB-IIS ISAPI .printer access attempt";
flow:to_server,established; flowbits:set,flag; flowbits:noalert;
uricontent:".printer"; nocase;
sid:971; rev:9;)

alert tcp $HTTP_SERVERS $HTTP_PORTS -> $EXTERNAL_NET any
(msg:"WEB-IIS ISAPI .printer access success";
flow:to_client,established; flowbits:isset,flag; flowbits:unset,flag;
content:!"HTTP";
sid:10971; rev:1;)

alert tcp $HTTP_SERVERS $HTTP_PORTS -> $EXTERNAL_NET any
(msg:"WEB-IIS ISAPI .printer access failure";
flow:to_client,established; flowbits:isset,flag; flowbits:unset,flag;
content:"HTTP";
sid:20971; rev:1;)
```

Figure 7 Snort Verification Rule for BID 2674

has at least to match the rule header (i.e. `tcp $HTTP_SERVERS $HTTP_PORTS -> $EXTERNAL_NET any`). Thus, at least a packet has to be seen from the target system after the attack to trigger this rule. In this case, no reply (i.e. packet) is expected (Figure 4). Thus, this Snort specification of the verification rule for BID 2674 is incorrect. The Snort IDS engine interprets Snort rule 10971 as *the next packet should not be an HTTP packet*, whereas the intent of the designer of the rule was *there is no HTTP packet*. This explains the true positive rate of 0% for these two BIDs when using Snort UC Davis. The problem is that it is not possible to specify such a verification, whereby the absence of reaction is to be checked, using the current Snort signature language. Any rule in Snort UC Davis that requires the detection of no reaction from the target system—which is the case of a buffer overflow attack (e.g., BID 2674) and a denial of service attack (e.g., BID 4482)—should have the same problem. Since our MS-IDS engine does not only rely on Snort to evaluate rules, we do not have this problem and we have accurate verification rules.

Fifth, Table 1 shows that a large number of rules in Bro and Snort UC Davis have a detection part but do not have a verification part, as opposed to our rules: the Bro and Snort UC Davis rules classify all the attacks as Likely-to-Succeed (TP rate of 100% and TN rate of 0%). (The corresponding BIDs are highlighted in Table 1.) In other words, IDS-VRG generated verification rules not found in Bro or in Snort UC Davis. There are exceptions though. Snort UC Davis contains verification rules, inherited from Snort 2.3.0, for BID 8205 and 7294. However, Bro has a detection part for BID 514 but it is unable to detect the attack because it is an IGMP attack and Bro is unable to decode the IGMP protocol. In this case, the true positive rate is 0% and the true negative rate is 100% (silence is interpreted as a Likely-To-Fail) because Bro remains silent for all attack attempts.

5.2.2 Usability

We analyzed the usability of the verification rules generated by IDS-VRG. We present the verification rules based on the attack classes (i.e., Web Attack, Admin Attempt and DOS) because we observed that verification rules from the same attack class contain similar predicates. We present the C4.5 verification rules and not the IDS verification rules with temporal constraints to show that the temporal constraints can easily be inferred by a network security expert from these verification rules.

For all training data sets in the Admin Attempt attack class (i.e., for BID 7106, 7294, 10108, 9633, 10116 and 8205), the C4.5 algorithm provides usable verification rules (according to criteria 1 and 2, Section 3.3.3). All the verification rules generated in the Admin Attempt attack class are similar to the verification rule generated from the BID 7294 training data set: what differs is the Snort rule being used for attack detection. To verify the attack, all the verification rules in the Admin Attempt attack class monitor the effect of the attack on the state of ports of the target system. Thus, to simplify the current discussion, we only present in Figure 8 the verification rules for BID 7294 as generated by C4.5.

This verification rule specifies that an attack is successful for BID 7294 when the attack matches the Snort rule 2103 and when a port on the target system other than the attacked port is changing its state from closed to open. The attack fails when the attack matches the Snort rule 2103 and when all ports on the target system other than the attack port

```
if Snort(2103):NETBIOS SMB trans2open buffer overflow attempt = true then
   if HostPortState(target,port,close,open) = true then
      success
   else
      failure
else
   normal/abnormal
```

Figure 8 IDS Verification Rule for BID 7294

```
if  Snort(273):DOS IGMP dos attack = true then
  if HostState(target,up) = true  then
    failure
  else
    success
else
  normal/abnormal
```

Figure 10 IDS Verification Rule for BID 514

```
if Snort(1002):WEB-IIS cmd.exe access = true then
  if StandardMessage(HTTP,200) = true then
    success
  else
    failure
else
  normal/abnormal
```

Figure 10 IDS Verification Rule for BID 1806

remain closed. There is no attack when the attack does not match the Snort rule 2103. In the temporal version of that C4.5 verification rule, we added the additional requirement that the attack attempt has to occur before the port state verification.

However, even if this verification rule is correct (i.e., it should not generate false positives), it is not complete (i.e., it may generate false negatives) because the attack may have a different effect than changes to a port state: for instance it could result in a remote shell being opened on the target system by the attacker. The current version of the VEP data set used to generate our training data sets only contains Admin Attempt that open a direct (i.e., the attacker connects to the target system) or reverse (i.e., the target system connects to the attacker) remote shell on the target system when it is successful and exploits the corresponding vulnerability. This illustrates one limitation of our results: a larger, more diverse set of attacks would lead to more complete verification rules. Recall from Table 1 and the discussion in the previous section that Bro and Snort UC Davis do not have verification rules for these attacks.

For all the DOS attack class training data sets (i.e., BID 4482, 514, 1163, 5556 and 10115), the C4.5 algorithm also provides usable verification rules. All the verification rules generated from the DOS attack class training data sets are similar to the verification rules generated for BID 514: what varies is the Snort rule used to detect the attack, and the target state verification (i.e., host down, port closed, application unavailable) after the DOS attack. Thus, we only present the verification rule for BID 514 to simplify this analysis. Figure 10 describes the verification rule generated using the C4.5 algorithm. This verification rule describes that an attack exploiting vulnerability BID 514 is successful if the attack matches the Snort rule 273 and the target system is down. The attack fails if it matches the Snort rule 273 and the target system is up. When the traffic does not match the Snort rule 273, the traffic is normal/abnormal. We specified that the attack attempt has to come before the target system state change in the IDS verification rule. This is what is expected when a denial of service is attempted against a target system. Thus, this verification rule is correct and usable. Again, recall that Bro and Snort UC Davis do not have verification rules for these BIDs, with the exception of BID 4482 for Snort UC Davis. However, we demonstrated in the previous section that the Snort UC Davis verification rule for BID 4482 is incorrectly specified.

For all the Web Attack class training data sets (i.e., for BID 2708, 1806, 3335, 2674 and 8035), all our verification rules are usable and equivalent, in terms of usability, to the correct verification rules of Bro and Snort UC Davis (BID 2708, 1806 and 8035). All the verification rules generated in the Web Attack class are similar to the verification rule of BID 1806: what varies is the Snort rule that detects the attacks and the standard HTTP response message used to verify the attack. Thus, we only present the verification rule for BID 1806 to simplify the analysis of the results. Figure 10 presents this verification rule for BID 1806, generated using the C4.5 algorithm.

The rule specifies that an attack succeeds when the attack matches the Snort rule 1002 and there is an HTTP message 200 OK from the target system. The attack fails when it matches the Snort rule 1002 but there is no HTTP message 200 OK from the target system, and there is no attack against the BID 1806 vulnerability when the traffic does not match Snort rule 1002.

Recall that our verification rules for BIDs 2674 and 3335 can replace the incorrect verification rules in Bro. Bro verification rules also vary in the Snort rule being used to detect the attack, but all use the 200 OK HTTP message for verification purposes. However, for BID 3335, the Bro verification rule is not correct because it is the type of HTTP error message that decides the success or failure of the attack. In the case of BID 2674, as explained previously, it is the absence or the presence of an HTTP message after the attack that defines the success or failure of the attack. The Snort UC Davis does not have this problem. It has the correct verification rule for BID 2674, and the verification rule for BID 3335 was not instrumented in Snort UC Davis. However, as explained in Section 5.2.1 the Snort UC Davis verification rule for BID 2674 does not work because it is not specified correctly in Snort.

5.3 Limitations

The IDS verification rules obtained using our IDS-VRG are more accurate or equivalent to the ones used by Bro and Snort UC Davis. However, as discussed in Section 5.2.2, some of the verification rules are too specific because of the lack of diversity of the attack scenarios contained in the VEP data set and the attributes used to monitor the target reaction behavior. This is however only a limitation of the data set used as input, and not a limitation of the methodology we followed: a larger, more diverse data set would lead to more complete rules.

Moreover, as discussed in Section 3.3.1 the data mining algorithm used does not generate verification rules with temporal constraints: the rule shows predicates but does not indicate in what order the predicates have to be evaluated. However, we have illustrated that identifying the adequate order of evaluation of the predicates would be a straightforward task for a network administrator.

Dreger [4] identified that the state that needs to be kept by the IDS and the per packet analysis time are two major factors that slow down network intrusion detection systems. Thus, using verification rules could introduce processing costs for IDS. In this research, we did not address the processing costs related to these verification rules. However, we know that although some verification rules can be very accurate, they could degrade the performance of an IDS. For instance, the target reaction approach increases the per packet analysis time because the success and error message of protocols that have to be monitored in addition to packets already monitored by the IDS. Moreover, these protocol messages are frequent on a network. Thus, the gathering of this information involves the frequent triggering of rules monitoring these messages and influences the per packet analysis time. However, in this case, the protocol messages only have to be observed after attacks, thus only a subset (i.e. the ones that follow attack attempts) of these messages is required (Figure 10).

Using verification rules also has an impact on the information the IDS has to keep in memory. For instance, our verification rules require that some information (e.g., attack attempt) has to be kept in memory by the IDS while waiting for the target reaction. Thus, verification rules should increase the processing cost because the IDS has to keep the states of the verification rule in memory.

6. Conclusion

In this note, we presented an approach that generates automatically verification rules for Intrusion Detection Systems (IDSs). The overall objectives were to provide techniques that automatically generate IDS rules to improve IDS accuracy by preventing human errors and reducing the number of false positives. This approach relies on labeled traffic traces and a data mining algorithm to generate the verification rules. This is the first time a data mining algorithm is used to generate verification rules and we believe that this has the potential to help improve the accuracy of IDS rules in the future.

We built our IDS Verification Rule Generator using the C4.5 data mining algorithm and its implementation in the Weka framework to automatically generate verification rules. However, our IDS Verification Rule Generator has some limitations as it cannot generate verification rules with temporal constraints (required to more accurately verify the attack). Thus, the temporal constraints were manually derived to create actual IDS verification rules. Verification rules without temporal constraints are less accurate, and thus generate more false positives. However, we have shown that temporal constraints could be easily added to our verification rules by a network administrator.

We used a multi-session IDS, called MS-IDS, to test our verification rules because the current Snort rule specification language (Snort 2.8) does not support multi-session rules. We used our verification rules with MS-IDS and compared their accuracy against Bro and Snort UC Davis on an existing set of attack network traces.

Our analysis showed that our automatically generated verification rules are more accurate or equivalent to the ones used by Bro and Snort UC Davis. We automatically generated rules that are semantically equivalent to some of the correct verification rules of Bro and Snort UC Davis and, furthermore, we were able to generate correct verification rules to replace some of the incorrect verification rules in Bro and Snort UC Davis. New verification rules were also generated that were not included in these IDSs.

Future work will look at generating automatically the temporal constraints in the verification rules, address the diversity problems identified in the VEP data set and tackle the performance issues when using verification rules. We also plan to investigate whether our approach can be used to automatically generate detection rules.

References

[1] Barbara D., Couto J., Jajodia S. and Wu N., "Adam: A testbed for exploring the use of data mining in intrusion detection," *ACM SIGMOD Record*, vol. 30 (4), pp. 15–24, 2001.

[2] Cohen W. W. and Singer Y., "Simple, Fast, and Effective Rule Learner," *Proc. AAAI/IAAI*, pp. 335-342, 1999.

[3] Dayioglu B. and Ozgit A., "Use of Passive Network Mapping to Enhance Signature Quality of Misuse Network Intrusion Detection Systems," *Proc. Int. Symp. on Computer and Information Science*, 2001.

[4] Dreger H., Feldmann A., Paxson V. and Sommer R., "Operational experiences with high-volume network intrusion detection," *Proc. ACM Conf. on Computer and Communications Security*, 2004.

[5] Green C. and Roesch M., "The Snort Project: version 2.1.0,", User Manual, www.snort.org, 2003.

[6] Julisch K., "Mining alarm clusters to improve alarm handling efficiency," *Proc. ACSAC*, pp. 12–21, 2001.

[7] Julisch K., "Clustering intrusion detection alarms to support root cause analysis," *ACM Trans. Inf. Syst. Secur.*, pp. 443–471, 2003.

[8] Julisch K. and Dacier M., "Mining intrusion detection alarms for actionable knowledge," *Proc. ACM International Conference on Knowledge Discovery and Data Mining*, pp. 366–375, 2002.

[9] Krügel C. and Robertson W. K., "Alert verification determining the success of intrusion attempts," *Proc. Workshop the Detection of Intrusions and Malware and Vulnerability Assessment*, pp. 25–38, 2004.

[10] Lee W., Stolfo S. J. and Mok K. W., "A data mining framework for building intrusion detection models," *Proc. IEEE Symposium on Security and Privacy*, pp. 120–132, 1999.

[11] Manganaris S., Christensen M., Zerkle D. and Hermiz K., "A data mining analysis of rtid alarms," *Proc. International Symposium on Recent Advances in Intrusion Detection*, 1999.

[12] Massicotte F., "Passive network monitoring tool extended (pnmt-x)," CRC, Technical Report CRC-VPNT-2007-002, 2007.

[13] Massicotte F., Gagnon F., Labiche Y., Briand L. and Couture M., "Automatic Evaluation of Intrusion Detection Systems," *Proc. Annual Computer Security Applications Conference*, 2006.

[14] Morin B. and Debar H., "Correlation of Intrusion Symptoms: an Application of Chronicles," *Proc. International Conference on Recent Advances in Intrusion Detection*, LNCS 2820, pp. 94-112, 2003.

[15] Paxson V., "BRO: A System for Detecting Network Intrusion in Real-Time," *Computer Networks*, vol. 31 (23-24), pp. 2435-2463, 1999.

[16] Quinlan J. R., *C4.5: Programs for Machine Learning*, Morgan Kaufmann, 1993.

[17] Sommer R. and Paxson V., "Enhancing byte-level network intrusion detection signatures with context," *Proc. ACM Conference on Computer and Communications Security*, 2003.

[18] Todd A. D., Raines R. A., Baldwin R. O., Mullins B. E. and Rogers S. K., "Alert Verification Evasion Through Server Response Forging," *Proc. Recent Advances in Intrusion Detection*, pp. 256-275, 2007.

[19] Treinen J. J. and Thurimella R., "A framework for the application of association rule mining in large intrusion detection infrastructures," *Proc. International Symposium on Recent Advances in Intrusion Detection*, LNCS 4219, pp. 1–18, 2006.

[20] Witten I. H. and Frank E., *Data Mining: Practical Machine Learning Tools and Techniques*, Morgan Kaufman, 2nd Edition, 2005.

[21] Zhou J., Carlson A. J. and Bishop M., "Verify Results of Network Intrusion Alerts Using Lightweight Protocol Analysis," *Proc. ACSAC*, 2005.