

# Variable-length Signatures for Intrusion Detection

Zhuowei Li<sup>†</sup>, Amitabha Das<sup>‡</sup>, Jianying Zhou<sup>§</sup> and Jagdish C. Patra<sup>‡</sup>

<sup>†</sup>Center for Software Excellence, Microsoft.

<sup>‡</sup>Nanyang Technological University, Singapore.

<sup>§</sup>Institute of Infocomm Research, Singapore.

## Abstract

Intrusion detection has become a basic infrastructure to guarantee the security of most internetworking applications. With more internetworking applications in the Internet nowadays, enormous volume of audit trails are produced for the analysis within intrusion detection. For this reason, it is critical to reduce the detection computation of intrusion detection to meet the realtime detection requirement. In this paper, using a formal intrusion detection framework, we propose a new concept of *variable-length signature*, along with feature selection, to compress the behavior models of our intrusion detection system, USAID[7], that achieves promising detection performance. Intuitively, compact behavior models will make the detection process computationally much cheaper. Our experimental results show that the proposed technique will degrade the detection rate of unknown intrusions, and fortunately, that it achieves a high detection rate for known intrusions with a significantly reduced false alarm rate. As a result, compared to USAID, the size of the behavior model is decreased by 99.52%, and the detection computation is cut down by 81.15% at least.

## 1 Introduction

Intrusion detection is an important defense mechanism to meet the security requirements of networking applications. In general, there exist two approaches for detecting intrusions into computer systems or networks: *signature-based intrusion detection* (SID, a.k.a. misuse detection), where an intrusion is detected if it matches well-known intrusion signatures, such as Snort[11], and *anomaly-based intrusion detection* (AID), such as MADAM ID[5], where an intrusion is detected if a system behavior deviates significantly from normal behaviors. Both of them suffer from their principle and practical problems respectively in the past deployments.

In principle, SID works reliably on well-known intrusions, but it is incapable of detecting new intrusions. In addition, it is also limited by several practical problems: *signature updating bottleneck*, *intrusion variation detection* (Rubin et al.[12]), and *too many false alarms* (Julisch[3]). Besides that it can not identify intrusions in principle, AID suffers from *higher false alarm rate*, *the difficulty*

in determining whether the anomalies are caused by intrusions (Li et al.[6]), concept drifting problem, and mimicry attacks (Wagner et al.[13]). Most importantly, the *computational cost* of intrusion detection must be reduced considerably before it can be usefully employed in practical systems. This is more so as the size of the audit trails keep on growing with more networking applications.

In our earlier work [7], a framework is proposed for intrusion detection (*Section 2*), in which we analyze these abovementioned fundamental and practical problems for intrusion detection, and consequently design a new intrusion detection technique using the analysis results, called *Unifying Signature-based and Anomaly-based Intrusion Detection (USAID)*. Unlike SID and AID techniques utilizing partial knowledge of a resource, USAID makes effective use of all the available knowledge about the intrusive and normal behaviors in the historical data, and all the existing features are applied in it to detect (normal/intrusive) behaviors in the detection phase. In our experimental results, it achieves promising detection performance in comparison with other intrusion detection techniques.

However, USAID suffers from the well known problem of *intensive computation* in detecting behaviors that plagues most intrusion detection techniques. The computational cost becomes even more significant as the size of audit trails for intrusion detection keep on growing with more networking applications conducted. In this paper, our aim is to improve the detection performance of USAID by reducing the computational cost in the detection process. First we evaluate the significance of every feature in our proposed framework (*Section 2*). Unlike the feature selection done by Mukkamala et al. [10] and Lee et al. [5], which use the overall detection performance in the *test* audit trails as the selection criteria, our feature evaluation is done by maximizing the distinguishing ability between the normal and anomalous signatures only using the *training* audit trails (*Section 3.2*). At the same time, we introduce the concept of *variable-length signature* to further decrease the detection computation (*Sections 2 and 3.3*). In our experiments, matching with a variable-length signature will lead to less than 6 feature value comparisons, being reduced from 41 ones in USAID. Finally, a new intrusion detection technique is proposed to alleviate the detection computation using variable-length signatures (*Section 3*).

In summary, we make the following contributions in this paper. First, a formal framework is introduced, and the new concept, called variable-length signature, is proposed. Secondly, we present the USAID system with variable-length signatures to reduce the computational cost, as well as a new criteria which is used to rank the features for intrusion detection. Lastly, we design and perform the experiments to evaluate our methodology.

The remaining parts are organized as follows. In *Section 2*, a framework for intrusion detection is described briefly, and the variable-length signature is defined. The variable-length USAID (**varUSAID**) is introduced in *Section 3*. In *Section 4*, several experiments are done to show the effectiveness of varUSAID. Lastly, we discuss the related work and conclude the paper.

## 2 A Framework for Intrusion Detection

In our framework for intrusion detection, we build behavior models like SID and AID, which use a *feature vector*  $FV = \{F_1, F_2, \dots, F_m\}$ , where  $F_i$  is a feature in the feature set. In general, a feature  $F_i$  can be categorized into *nominal*, *discrete* or *continuous* one. For example, ‘name’ is nominal, ‘TCP port number’ is discrete and ‘SYN rate within 2 seconds’ is continuous. A feature vector can contain any number of nominal, discrete, and/or continuous features. Besides that, we assume that there are *training audit trails*, which are constituted with labeled **normal** audit trails and **intrusive** audit trails. Every element in the training audit trails is an instance of  $FV$ , named as a feature vector instance.

### 2.1 Definitions

For any feature  $F$ , there is a meaningful domain  $Dom(F)$ , which is called the feature space. Any value occurring in the audit trails is a *feature value*  $v_F$ . With respect to the training audit trails, a feature value  $v_F$  will be labeled as *normal*, *suspicious* or *anomalous*. Specifically, if it only occurs in the normal audit trails, it is *normal*. If it only occurs in the anomalous audit trails, for example, in the intrusion signatures, it is *anomalous*. Otherwise, it is labeled as ‘*suspicious*’. We will refer to the labels ‘normal’, ‘suspicious’, or ‘anomalous’ as the **NSA label** of the feature value  $v_F$ , denoted as  $L(v_F) \in \{‘N’, ‘S’, ‘A’\}$  using the first letter of the label.

For any discrete/continuous feature  $F$ , the interval between two feature values  $v_F^1$  and  $v_F^2$  is defined as a *feature range*  $R_F = [v_F^1, v_F^2]$ . For the sake of uniformity, each nominal feature value is also referred to as a feature range. Thus,  $R_F \subseteq Dom(F)$ . The concept of **NSA labels** can be extended to the feature range as follows:

$$\begin{aligned} L(R_F) = ‘N’ &\Leftrightarrow \forall v_F \in R_F, L(v_F) = ‘N’ \\ L(R_F) = ‘A’ &\Leftrightarrow \forall v_F \in R_F, L(v_F) = ‘A’ \\ L(R_F) = ‘S’ &\Leftrightarrow \exists v_F^1, v_F^2 \in R_F, L(v_F^1) = ‘N’ \wedge L(v_F^2) = ‘A’ \end{aligned}$$

Where, all the feature values occur in the training audit trails.

Next, for the feature  $F$ , we can collect a series of feature ranges from  $Dom(F)$ :  $\{R_F^1, R_F^2, \dots, R_F^m\}$ , such that  $R_F^i \neq R_F^j$  and  $L(R_F^i) \neq L(R_F^{j+1})$  ( $1 \leq i, j \leq m$ , and  $i \neq j$ ). Specific for different intrusion detection techniques, the collection strategy is different but the final set of feature ranges should comply the above requirements. Furthermore, using the following rules, we can partition the feature space  $Dom(F)$  into three feature subspaces:  $N(F)$ ,  $S(F)$  and  $A(F)$ .

$$\begin{aligned} N(F) &= \{R_F^j \mid 1 \leq j \leq m, L(R_F^j) = ‘N’\} \\ S(F) &= \{R_F^j \mid 1 \leq j \leq m, L(R_F^j) = ‘S’\} \\ A(F) &= \{R_F^j \mid 1 \leq j \leq m, L(R_F^j) = ‘A’\} \end{aligned}$$

We also define  $\Omega(F) = N(F) \cup S(F) \cup A(F)$ , so that it is a collection of all feature ranges found in the training audit trails.

## 2.2 Compound features

**Definition 1 (compound feature)** *A compound feature  $F_{12}$  is an ordered pair  $\{F_1, F_2\}$ , such that  $\Omega(F_{12})$  is a subset of the cartesian product of  $\Omega(F_1)$  and  $\Omega(F_2)$ , and each compound feature range in  $\Omega(F_{12})$  represents at least one feature vector instance in the training audit trails. For the sake of convenience, we use the expression  $F_{12} = F_1 \times F_2$ .*

Intuitively, similar to its component features, a feature range of  $F_{12}$  has an NSA label with respect to its representative feature vector instance(s), and the compound feature space can also be partitioned into three feature subspaces, i.e.,  $\Omega(F_{12}) = N(F_{12}) \cup S(F_{12}) \cup A(F_{12})$ . Note that the suspicious compound feature ranges can potentially shrink with respect to component feature ranges as the combinations of two ‘suspicious’ feature ranges may be ‘normal’ or ‘anomalous’.

In summary, the compound feature built from two atomic features shows similar behaviors as any of its component atomic features. For this reason, we can treat the compound feature as an atomic one to build higher order compound features. Using this recursive procedure, the feature vector  $FV$  for intrusion detection can be converted into an equivalent n-order compound feature  $F_{1\dots n}$  with feature subspaces  $N(F_{1\dots n})$ ,  $A(F_{1\dots n})$  and  $S(F_{1\dots n})$ .

## 2.3 Variable-length signatures

In USAID, some feature ranges of a (*atomic/compound*) feature have the NSA label ‘normal’ and ‘anomalous’. Thus, without using all the features in the feature vector, the feature instance falling into these feature ranges can be detected/identified correctly, which is also the task of intrusion detection. This special characteristics of the feature can be utilized to reduce the computational cost in the detection phase.

**Definition 2 (Signature)** *Assume that  $FV = \{F_1, F_2, \dots, F_n\}$ , and the feature ranges of every feature are determined beforehand. A **signature** is a compound feature range of  $F_{1\dots n}$ . In other words, the signature is the combination of feature ranges of all features in the feature vector.*

**Example 1** *For example, there is a feature vector  $FV = \{ip\text{-}addr, port, service\}$ , and the feature ranges of every feature are:  $R_{ip\text{-}addr} = \{10.10.1.2, 10.10.2.2\}$ ,  $R_{port} = \{22, 23, 80, 443\}$ ,  $R_{service} = \{telnet, http, ftp, ssh\}$ . A signature of the feature vector is  $(10.10.2.2, 23, telnet)$ . Obviously, the total number of possible signatures in this example is  $|\Omega(R_{ip\text{-}addr})| * |\Omega(R_{port})| * |\Omega(R_{service})| = 2 * 4 * 4 = 32$ .*

**Definition 3 (Variable-length Signature)** *Given  $FV = \{F_1, F_2, \dots, F_n\}$  with labelled feature ranges for every feature, a **variable-length signature** is a feature range of  $F_{1\dots i}$ , such that its NSA label is either ‘normal’ or ‘anomalous’.*

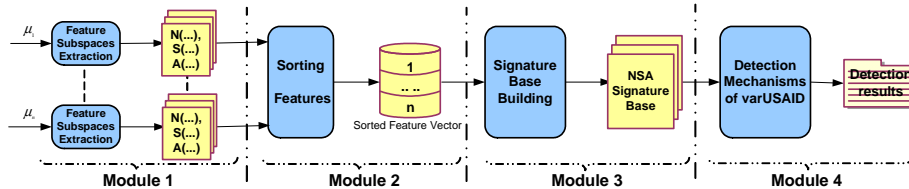


Figure 1: A general framework for varUSAID.

The signature is called variable length as the number of features involved in it can vary from 1 to  $n$ , i.e.,  $1 \leq i \leq n$ .

**Example 2** As in Example 1, for the signature  $(10.10.2.2, 23, telnet)$ , if the NSA label of the feature range  $(10.10.2.2, 23)$  is ‘normal’ or ‘anomalous’,  $(10.10.2.2, 23)$  is a variable-length signature.

The relation between signatures and variable-length signatures is subtle but important. A signature with NSA label ‘normal’ or ‘anomalous’ can be reduced in size by omitting some component feature ranges if that does not affect its NSA label. Thus, it can become a variable-length signature. On the other hand, if a signature bears the label ‘suspicious’, then it cannot shed any feature ranges, so that it remains of the fixed maximum length.

### 3 varUSAID: USAID with variable-length signatures

The architecture of varUSAID consists of four modules (Figure 1). The first module extracts the three feature subspaces for every feature. The second module does the feature ranking and selection. The signature building module constructs the variable-length signature base, named as NSA signature base. The last module incorporates the detection mechanisms of varUSAID.

#### 3.1 Feature subspaces extraction

**Step 1: feature value collection.** In the labeled training audit trails, all feature values are collected for every feature. The status (*normal and/or intrusions depending on whether it occurs in normal or intrusive audit trails*) of every feature value is also collected and accumulated in its status list. Based on its status list, each feature value is assigned an NSA label. Note that this step is applied to nominal, discrete and continuous features, but the following two steps are only applicable to discrete and continuous features.

**Step 2: feature value clustering.** The objective of this step is to form initial feature ranges for every feature by clustering the neighboring feature values. For a discrete feature, two feature values  $x_1$  and  $x_2$  are *neighboring*

Table 1: Feature ranges of the discrete feature “hot”.

INDEX	NSA	RANGE	INTRUSIONS AND THEIR FREQUENCY
0	S	[0,7]	“normal:971107”, “perl:3”, “neptune:1072017”, “smurf:2807886”, ....., “rootkit:10”, “loadmodule:9”, “buffer_overflow:30”, “phf:4”
1	N	[8,9]	“normal:14”
...	...	...	...
8	S	[27,29]	“normal:5”, “warezclient:274”
9	N	[30,77]	“normal:285”

if  $|x_1 - x_2| = 1$ <sup>1</sup>. For a continuous feature, two feature values  $x_1$  and  $x_2$  are neighboring if  $|x_1 - x_2| \leq \delta$ . If several neighboring feature values have the same NSA label, they will be combined to form an *initial feature range*. As a special case, if, for a feature value, its neighbors have different NSA labels from itself, it forms an initial feature range whose upper and lower bounds are both equal to the feature value itself. Every initial feature range thus formed inherits the NSA label of the feature values falling within it.

**Step 3: feature range generalization.** However, under most scenarios, the initial feature ranges of a feature will not cover all of its feature space. Any feature subspace outside the labelled feature ranges is named as an *uncovered space*. Two feature ranges are called *neighboring* if there is no other feature range between them. The uncovered space between any two neighboring feature ranges is treated as follows: if the two neighboring feature ranges have the same NSA label, a new larger feature range is formed that includes the two feature ranges as well as the intervening uncovered space. This new range is assigned the NSA label of the ranges at the two edges. Otherwise, if the two neighboring ranges have different NSA labels, the scheme divides uncovered feature spaces equally into two halves, one merges with normal range and another with normal range without knowing the actual nature of the halves, i.e., generalization. The NSA labels of the initial feature ranges will be inherited by the newly extended or combined feature ranges. Finally, all the *known* feature space of every feature will be covered by well-defined feature ranges.

Table 1 is an example of feature range set generated in the experiments for the discrete feature “hot” (“*number of ‘hot’ indicators [1]*”). In this table, for the feature range [27,29] with lower bound 27 and upper bound 29, there are 5 instances labelled as ‘normal’ and 274 instances labelled as ‘warezclient’ in the training audit trails.

### 3.2 A Sorted Feature Vector

In this section, a function is first defined to measure the distinguishing ability of a feature, and then it is used to sort all features in the feature vector. To achieve it, the attack feature subspace is further split into two feature subspaces.

<sup>1</sup>If the distance is larger than 1, there is an uncovered space  $|x_1 - x_2 - 1|$ .

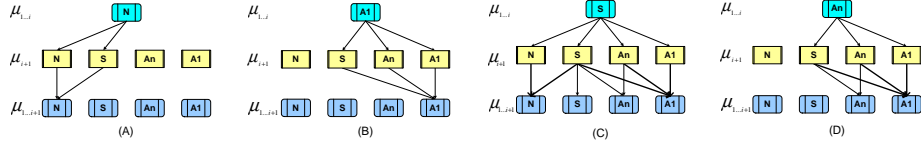


Figure 2: Four compounding scenarios.

### 3.2.1 Two Further Feature Subspaces

Consider a compound feature  $F_{1\dots i}$ , with  $N(F_{1\dots i})$ ,  $S(F_{1\dots i})$  and  $A(F_{1\dots i})$ , and an atomic feature  $F_{i+1}$  with  $N(F_{i+1})$ ,  $S(F_{i+1})$  and  $A(F_{i+1})$ . Basically, it is possible that a feature range labelled ‘*anomalous*’ is caused by several intrusions. The feature range that is labelled ‘*anomalous*’ because of a single intrusion is extremely valuable for identifying that intrusion, and is thus highly desirable. Thus, the anomalous feature subspace  $A(F_{1\dots i})$  is split into two feature subspaces  $An(F_{1\dots i})$  and  $A1(F_{1\dots i})$ . Specifically, if the status list of the feature range in  $A(F_{1\dots i})$  only has one intrusion, it is classified as  $A1(F_{1\dots i})$ , otherwise, as  $An(F_{1\dots i})$ . As a result, there are now four feature subspaces for  $F_{1\dots i}$ , namely  $N(F_{1\dots i})$ ,  $S(F_{1\dots i})$ ,  $An(F_{1\dots i})$ , and  $A1(F_{1\dots i})$ , and for  $F_{1\dots i+1}$ :  $N(F_{1\dots i+1})$ ,  $S(F_{1\dots i+1})$ ,  $An(F_{1\dots i+1})$ , and  $A1(F_{1\dots i+1})$  respectively.

### 3.2.2 Measuring and Sorting Features

In general, more feature ranges in  $N(1\dots i)$  and  $A1(1\dots i)$  lead to better efficiency of varUSAID in detecting intrusions. In Figure 2, the four compounding scenarios are illustrated to design a measure for feature ranking. As indicated in (A), the compounding between feature ranges in  $N(F_{1\dots i})$  and feature ranges in  $\Omega(F_{i+1})$  will result in feature ranges in  $N(F_{1\dots i+1})$ . Thus, there is no improvement in the distinguishing ability for intrusion detection when we add more features to a compound feature range that is already labelled as ‘*normal*’. Similarly, the compounding in (B) will not lead to any benefit. For this reason, we will not consider the compounding scenarios (A) and (B) in our feature ranking. In contrast, in the compounding scenarios (C) and (D), the feature ranges in  $S(F_{1\dots i})$  and  $An(F_{1\dots i})$  will increase the distinguishing ability of the new feature  $F_{1\dots i+1}$  as some of their resultant feature ranges fall into  $N(F_{1\dots i+1})$  or  $A1(F_{1\dots i+1})$ . In other words, to increase the distinguishing ability of the new feature, the increment due to compounding scenarios (C) and (D) should be maximized.

In addition, as our objective is to reduce the computation overhead in the detection phase, it is useful to consider frequency information of feature ranges in the training audit trails. The more frequently a variable-length signature occurs in the training audit trails, the smaller its length should be. To justify it, we assume that the training audit trails have the same frequency distribution as the test audit trails, and thus the detection computation can be reduced. In summary, the number of instances falling into  $S(F_{1\dots i})$  and  $An(F_{1\dots i})$  before the compounding with  $F_{i+1}$  but falling into  $N(F_{1\dots i+1})$  and  $A1(F_{1\dots i+1})$  after the

compounding, which represents the distinguishing ability of the feature  $F_{i+1}$ , should be maximized within the context of varUSAID. For the convenience of expression, the following notations are defined. With respect to the compounding operation between  $F_{1\dots i}$  and  $F_{i+1}$ , in the training audit trails,

- $\#S2N(F_{1\dots i}, F_{i+1})$  is the number of instances in  $S(F_{1\dots i})$  that moves to  $N(F_{1\dots i+1})$ .
- $\#S21(F_{1\dots i}, F_{i+1})$  is the number of instances in  $S(F_{1\dots i})$  that moves to  $A1(F_{1\dots i+1})$ .
- $\#n21(F_{1\dots i}, F_{i+1})$  is the number of instances in  $An(F_{1\dots i})$  that moves to  $A1(F_{1\dots i+1})$ .

With the above, we define the new identification capacity  $\Theta(F_{i+1}|F_{1\dots i})$ :

$$\Theta(F_{i+1}|F_{1\dots i}) = \#S2N(F_{1\dots i}, F_{i+1}) + \#S21(F_{1\dots i}, F_{i+1}) + \#n21(F_{1\dots i}, F_{i+1}) \quad (1)$$

---

**Algorithm 1** Feature ranking.

---

**Require:** Feature Vector  $FV$ , Number of Features  $n$ , Training audit trails  $\Sigma_{tr}$ ;

- 1: Selecting one feature  $F_{max}$  with the maximal size of instances in  $\Sigma_{tr}$  falling into  $N(F_k)$  and  $A1(F_k)$  ( $1 \leq k \leq n$ );
- 2: Inserting  $F_{max}$  into  $FV_{sort}$ ;
- 3: **for**  $i = 2$  to  $n$  **do**
- 4:      $\Theta_{max} = -\infty$ ;  $F_{\Theta_{max}} = -1$ ;
- 5:     **for**  $j = 1$  to  $n$  **do**
- 6:         **if**  $F_j \in FV_{sort}$  **then** continue;
- 7:         Building the compound feature by compounding  $F_j$  and the features in  $FV_{sort}$ ;
- 8:         Calculating  $\Theta(F_j|F_{FV_{sort}})$  from  $\Sigma_{tr}$ ;
- 9:         **if**  $\Theta(F_j|F_{FV_{sort}}) > \Theta_{max}$  **then**  $\Theta_{max} = \Theta(F_j|F_{FV_{sort}})$ ;  $F_{\Theta_{max}} = j$ ; **endif**
- 10:     **end for**
- 11:     Inserting  $F_{\Theta_{max}}$  into  $FV_{sort}$ ;
- 12: **end for**
- 13: Output  $FV_{sort}$ ;

---

Therefore, in our feature ranking procedure, the feature that maximizes the value of *newIdentify* will be selected in step  $i$ , and then the feature is sorted in a so-called sorted feature vector  $FV_{sort}$ . The feature ranking is shown in Algorithm 1 using a greedy strategy. After that, the sorted feature vector  $FV_{sort}$  will be applied in the following two modules.

### 3.3 Building NSA signature base

**Definition 4 (Signature Base)** *Assuming that there exists a labeled training audit trails, and its feature vector is  $FV$ , a signature base  $\Omega$  consists of signatures and variable-length signatures built from the training audit trails, so that every instance in the training audit trails will match one of its signatures or variable-length signatures. Specifically, the signature base consists of normal and anomalous variable-length signatures, and suspicious signatures built from the training audit trails.*

The NSA signature base building process starts with the first feature in  $FV_{sort}$ , and includes one additional feature in each step in the sorted order. At  $i$ -th step, only the compounding feature ranges in  $N(F_{1\dots i})$  and  $A1(F_{1\dots i})$  are left as variable-length signatures in the NSA signature base, and the compound feature ranges in  $S(F_{1\dots i})$  and  $An(F_{1\dots i})$  are carried forward for the subsequent

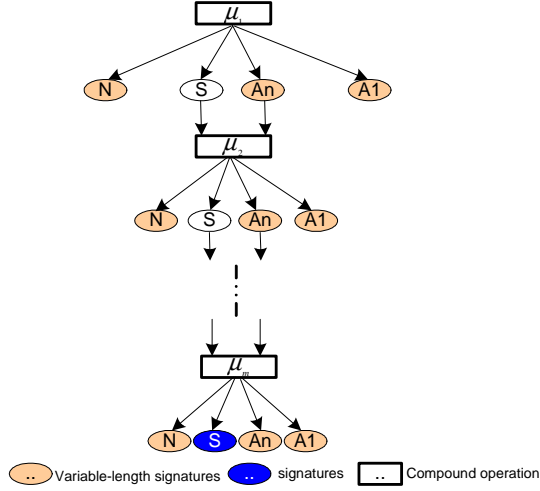


Figure 3: NSA signature base in varUSAID.

---

**Algorithm 2** Building NSA signature base in varUSAID.

---

**Require:** Sorted Feature Vector  $FV_{sort}$ , Number of Features  $m$  in  $FV_{sort}$ , Training audit trails;

- 1: NSABase= $\Phi$ ;
- 2: **for**  $i = 1$  to  $m$  **do**
- 3:   Building the compound feature  $F_{1\dots i}$  in  $FV_{sort}$ ;
- 4:   **for** each feature range in  $N(F_{1\dots i})$  and  $A1(F_{1\dots i})$  **do**
- 5:     **if** it does not match the signatures in NSABase **then**
- 6:       Inserting it into NSABase as a variable-length signature;
- 7:     **end if**
- 8:   **end for**
- 9: **end for**
- 10: Inserting  $S(F_{1\dots m})$  and  $An(F_{1\dots m})$  to NSABase;
- 11: Output NSABase;

---

compounding operation to increase the identification ability of the signature base. Finally, after compounding with the last feature, the compounding feature ranges in  $S(FV_{sort})$  and  $An(FV_{sort})$  are left as signatures in the signature base. Figure 3 illustrates the process of constructing the NSA signature base, and Algorithm 2 describes it in detail. Like the Huffman encoding, the NSA signature base thus constituted is compact enough to represent all the behavior identification information in the training audit trails.

Table 2 shows several example variable-length signatures. In every signature, the feature range is denoted by its index, and its NSA label is paired with the index using a pair of square brackets. The compounding of the feature ranges is denoted by ‘+’. For instance, in ‘[2,S]+[214,S]’, ‘[2,S]’ refers to the second range of the first feature in  $FV_{sort}$  (which is ‘source bytes’, please see the full description of  $FV_{sort}$  in the appendix), and ‘S’ indicates that its label is ‘suspicious’. Similarly, ‘[214,S]’ represent the 214th range of the second feature (‘service’) in  $FV_{sort}$ , and the whole expression, along with its NSA label and status, constitutes a complete variable-length signature “[2,S]+[214,S]=>A:buffer\_overflow”.

Table 2: Several example variable length signatures.

VARIABLE-LENGTH SIGNATURES	NSA	INTRUSIONS OR NORMAL
[94,A]	A	portsweep
[1069,N]	N	normal
[2,S]+[214,S]	A	buffer_overflow
[0,S]+[182,S]+[84,N]	N	normal
[0,S]+[86,S]+[0,S]	A	smurf
[0,S]+[277,S]+[239,S]+[14,S]	A	back

Even though the NSA labels of feature ranges are not useful within the context of this paper, they are included in variable-length signatures for future usage, such as the anomaly context identification scheme [6].

Obviously, if there exist ‘suspicious’ signatures (i.e.,  $S(F_{1\dots m}) \neq \Phi$  in Algorithm 2), it implies that the feature vector is not enough to completely distinguish between intrusive and normal behaviors in the audit trails. Thus, these ‘suspicious’ signatures will lead to detection errors (*false alarms or false negatives*) in varUSAID.

**Definition 5 (Signature Variations)** *In the signature base, every element is a signature or a variable-length signature. If two elements A and B in it have the same status list (thus, the same NSA label), A is called a signature variation of B, and vice versa. For example, “[1069,N]=>N:Normal” and “[0,S]+[182,S]+[84,N]=>N:Normal” in Table 2 are signature variations since both are caused by ‘normal’ behavior(s).*

In the signature base, one behavior (*normal or an intrusion*) may lead to several (*variable-length*) signatures. This causes the well-known intrusion variation detection problem of SID. In general, the number of signature variations in the signature base indicates the diversity of the corresponding intrusions.

### 3.4 Detection mechanisms via variable-length signatures

First, we extract an instance of a feature vector from the audit trails. Based on Algorithm.3, the instance is then detected as ‘normal’ or ‘anomalous’. In USAID, a complete signature in the test dataset is computed for every instance before the detection phase, by mapping the feature values into the predefined ranges for all the features in the feature vector. In contrast, in varUSAID, the variable-length signature is generated incrementally by mapping of the feature values one at a time. We then compare the signature thus generated with the variable-length signature base, and if a match is found, the remaining features are not used. This reduces the computation overhead of the detection phase significantly.

### 3.5 Mimicry attacks on varUSAID

The threat to varUSAID from the mimicry attacks (introduced by Wagner et al.[13]) can be explained as follows. In USAID, a mimicry attack achieve its goal

---

**Algorithm 3** Detecting an instance in varUSAID.

---

**Require:** NSABase, Sorted Feature Vector  $FV_{sort}$  and its size  $m$ , An instance in the test audit trails;

- 1: DResult=nil; {Detection Result for the instance}
- 2: VarSig=nil; {Variable-length signature corresponding to the instance}
- 3: **for**  $i = 1$  to  $m$  **do**
- 4:   **if** the feature value of  $F_i$  falls outside its known feature space **then** DResult="Anomalous";  
    break; **end if**
- 5:   Attaching the corresponding feature range of feature  $F_i$  into VarSig;
- 6:   **if** the NSA labels of feature ranges in VarSig are contradictory **then** DResult="Anomalous";  
    break; **end if**
- 7:   **if** VarSig $\in$ NSABase **then**
- 8:     **if** VarSig is identified by an 'anomalous' signature **then**
- 9:      DResult="Anomalous"; break;
- 10:    **else if** VarSig is identified by a 'normal' signature **then**
- 11:     DResult = "Normal"; break;
- 12:    **end if**
- 13:   **end if**
- 14: **end for**
- 15: **if** varSig is identified by a 'suspicious' signature **then** DResult="Anomalous"; **end if**
- 16: **if** DResult is 'nil' **then** DResult="Anomalous"; **end if**
- 17: Output DResult;

---

mimicking all the feature ranges of a 'normal' signature. In contrast, a mimicry attack to varUSAID can be designed more easily than USAID since the short 'normal' variable-length signatures will be the preference to attackers. In this aspect, varUSAID becomes more vulnerable than USAID.

## 4 Experiments

We have chosen a typical dataset for network intrusion detection from KDD CUP 1999 contest <sup>2</sup> The dataset meets the requirements of varUSAID: labeled audit trails and intrusion-specific feature vector. The size specifications of the dataset are: *training dataset: 4898431 records, test dataset: 311029 records.* The intrusions in the datasets are listed in Table 3, and Table 4 lists the features. For a detailed description of the datasets, please refer to [1]. In addition, for continuous features, we set the threshold  $\delta = 0.01$  for determining the neighborhood of a feature value as the precision in our experimental datasets is 0.01.

Table 3: Intrusions in the training and test datasets.

CATEGORY	MEANING	KNOWN INTRUSIONS IN THE TRAINING DATASET	NEW INTRUSIONS IN THE TEST DATASET
normal	normal	—	—
probe	probe	ipsweep, nmap, portsweep, satan	mscan, saint
DOS	denial of service	back, land, neptune, pod, smurf, teardrop, warezmater	apache, mailbomb, processtable, udp-storm
U2R	user to root	buffer_overflow, loadmodule, multihop, perl, rootkit	httptunnel, sqlattack, xterm, ps
R2L	remote to local	ftp_write, guess_passwd, httptunnel, imap, multihop, phf	named, sendmail, snmpgetattack, snmpguess, worm, xlock, xsnoop

---

<sup>2</sup>Most intrusion detection techniques need a supervised dataset. For example, in a SID technique, it is a dataset with only intrusive behaviors, and in an AID technique, it is a normal dataset with only normal behaviors. However, it is difficult to collect such a dataset. To our knowledge, only this supervised KDD CUP dataset, which is publicly available, meets the requirements of our framework.

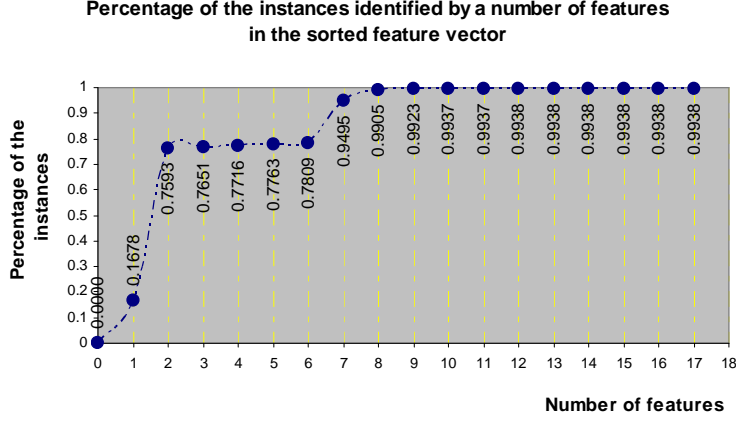


Figure 4: Percentage of instances identified by features in the sorted feature vector.

Table 4: Features in connection instances (n=41).

TYPES	FEATURES
nominal (9)	protocol type, service, flag, land, logged_in, root shell, su_attempted, is_hot_login, is_guest_login
discrete (15)	duration, src_bytes, dst_bytes, wrong_fragments, urgent, hot, num_failed_logins, num_compromised, num_root, num_file_creations, num_shells, num_access_files, num_outbound_cmds, count, srv_count
continuous (17)	error_rate, srv_error_rate, error_rate, srv_error_rate, same_srv_rate, diff_srv_rate, srv_diff_host_rate, dst_host_count, dst_host_srv_count, dst_host_same_srv_rate, dst_host_diff_srv_rate, dst_host_same_src_port_rate, dst_host_srv_diff_host_rate, dst_host_error_rate, dst_host_srv_error_rate, dst_host_error_rate, dst_host_srv_error_rate

First, let us talk about the dataset quality. In the training dataset, there are several illegal records (e.g. instance 4817100). In the test dataset, we found several instances (whose indices are 136489 and 136497) with illegal combination between ‘TCP’ protocol type and ‘ICMP’ service. Therefore, they are discarded in our experiments. Moreover, there is an intrusion ‘spy’ that is not documented properly (in instances 1381226, 1381227).

In the following sections, we will focus on mining the sorted feature vector, building the signature base by variable-length signatures, and comparing detection performance. To save space, we would refer the reader to [7] for the details of the feature ranges as the feature ranges of every feature are the same as USAID.

#### 4.1 Mining a sorted feature vector.

Figure 4 illustrates the percentage of identified instances in the audit trails with more features being added into the sorted feature vector. Obviously, the curve converges to a stable value as more features are included into the sorted feature

Table 5: The constitutions of NSA signature base in USAID and varUSAID.

ITEM	$N(\dots)$	$S(\dots)$	$An(\dots)$	$AI(\dots)$	TOTAL
Size of NSA base in USAID	60371	43	15	2779	63208
Size of NSA base in varUSAID	952	43	15	1072	2082
Average length of variable-length signatures	5.79	17	17	5.53	5.97

vector, i.e., more instances in the training audit trails are identified by these features.

Ultimately, for any further feature  $F_k$  in our applied feature vector,  $\Theta(F_k|F_{FV_{sort}}) = 0$ . That is, there are no instances in the training audit trails that will be identified by adding a further feature in the sorted feature vector. For this reason, the stopping criteria for feature selection in varUSAID is defined as that the value of  $\Theta(F_k|F_{FV_{sort}})$  is zero. As the minimum value of *newIdentify* is zero as well, the stopping criteria does guarantee that there is no additional distinguishability left in the remaining features. Using this criteria, (m=)17 features are selected into the sorted feature vector, i.e., (n-m=)24 features are discarded before continuing with further experiments.

Here are the details of the sorted feature vector,  $FV_{sort} = \text{Source Bytes, Service, Flag, REJ Error Rate, Destination Host Service REJ Error Rate, Same Service Rate, Destination Host Service Different Host Rate, Different Service Rate, Service Different Host Rate, Destination Bytes, Protocol Type, Service SYN Error Rate, Destination Host Service SYN Error Rate, Logged In, Land, SYN Error Rate, Duration}$ .

## 4.2 Building the NSA signature base.

Table 5 lists the statistics of the variable-length signatures in the NSA signature base. The average length of variable-length signatures in varUSAID is 5.97. Considering that the length of signatures in USAID is 41, varUSAID achieves 85.44% saving in the average length of a variable-length signature. In addition, the ratio of the signature base sizes is given by  $USAID_{base} : varUSAID_{base} = 63208 : 2082 = 30.36$ , i.e., the storage space of NSA signature base in varUSAID is cut down by 99.52% in comparison with USAID. Note that, the size of  $S(\dots)$  and  $An(\dots)$  remain unchanged in USAID and varUSAID. It indicates that variable-length signatures cannot remove the ambiguity in the audit trails. From another aspect, the information in the feature vector is not enough to distinguish all the behaviors in the training audit trails.

Table 6 summarizes the signature variations in the NSA signature base. It is clear that the same intrusion can lead to different variable-length signatures. Therefore, to detect an intrusion, one intrusion signature is not enough to detect all intrusive instances for most intrusions, which is a hard problem of SID techniques. As the signature variations of an intrusion have different lengths, there is no specific feature vector to detect all intrusion variations efficiently, i.e., different feature vectors are needed to detect some intrusion variations. For example, to detect intrusion ‘back’, only 19 intrusion variations are detected

Table 6: Number of signature variations in the training audit trails.

NAME/LENGTH	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
normal	141	214	28	29	62	37	67	4	76	265	1	17	5	2	0	0	8
neptune	1	0	23	45	0	147	3	43	9	0	0	7	3	0	1	1	0
buffer-overflow	1	10	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0
portsweep	2	0	84	48	7	66	2	37	0	2	0	0	0	0	0	1	6
warezclient	3	19	0	1	0	2	7	0	3	37	1	0	0	0	0	0	0
back	19	8	3	10	3	2	0	0	12	0	0	5	0	0	0	0	0
pod	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
teardrop	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
rootkit	0	2	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0
smurf	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
phf	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
warezmaster	0	1	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0
multihop	0	4	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0
satan	0	8	19	102	1	43	1	12	1	13	2	0	0	0	0	0	0
imap	0	3	0	1	0	1	0	0	2	0	0	0	0	0	0	0	0
ftp-write	0	5	0	0	0	0	1	0	0	2	0	0	0	0	0	0	0
ipsweep	0	1	2	1	6	12	20	0	2	8	0	0	0	0	0	0	0
nmap	0	0	52	0	0	1	3	0	1	1	0	0	1	0	0	0	0
guess-passwd	0	0	1	1	8	0	0	0	0	1	0	0	0	0	0	0	0
land	0	0	0	1	0	0	3	0	0	0	0	0	1	0	2	0	0
load module	0	0	0	0	0	2	1	0	0	6	0	0	0	0	0	0	0
spy	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0
perl	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0

Table 7: Efficiency parameters in our experiments.

PARAMETERS	FAR (%)	DR-Known (%)	DR-New (%)	Num-FR
USAID-41	1.451	99.779	98.184	41
USAID-17	1.409	99.778	97.901	17
varUSAID	0.431	95.928	55.172	3.205

if only the first feature ‘source bytes’ is utilized, and the number of detected intrusion variations becomes 30 if the first 3 features are used.

At the same time, it is worth noting that the most diverse behavior is the ‘normal’ behavior, which has the largest number of ‘normal’ signature variations. This phenomenon can be explained as follows. In the normal usage of a computing resource, there are many tasks to do with different objectives, and different tasks lead to different signatures. However, in an intrusion, the objective of the task is to attack and compromise a computing resource, so the intrusive task is not as complicated as normal tasks. Therefore, an intrusion behavior will cause fewer signatures than ‘normal’ behavior.

### 4.3 Detection results.

To show the influence of the feature ranking methodology and variable-length signatures, three experiments are performed using *USAID with 41 features (USAID-41)*, *USAID with the selected 17 features (USAID-17)*, and *varUSAID with the sorted feature vector (varUSAID)* respectively. Similar to [7], two new intrusions, snmpgetattack and mailbomb, are eliminated from the detection results because they have a predominating amount of instances in the test dataset, and most of their instances match the normal signatures. In Table 7, these techniques are compared with respect to four parameters in the detection results: (1)**FAR**: false alarm rate in detecting all intrusions; (2)**DR-Known**: detection rate in detecting known intrusions; (3)**DR-New**: detection rate in detecting new intrusions; (4)**Num-FR**: number of feature ranges in detecting an instance.

In Table 7, USAID-41 and USAID-17 almost have the same detection rate and false alarm rate, which indicates the effectiveness of our feature ranking methodology. The slight difference between USAID-41 and USAID-17 is caused by the similar reason as the one causing the difference between USAID and varUSAID, which is explained below. Even though the detection rate of varUSAID is lower than USAID-41 and USAID-17, the lower false alarm rate and lower number of feature ranges in detection are very encouraging since it achieves much lower false alarm rate and significantly lessens computation overhead.

The lower detection rate in varUSAID can be justified as follows. In USAID and varUSAID, an instance in the test dataset can be formalized as a signature by replacing its feature values with corresponding feature ranges. For every signature in USAID, there is one corresponding variable-length signature in varUSAID, in which some feature ranges are discarded for compactness. However, some new intrusions will cause anomalies in the discarded feature ranges. For example, assuming that one signature in USAID is “[0,N]+[14,S]+[23,S]=>N:Normal” and the corresponding variable-length signature in varUSAID is “[0,N]=>N:Normal”. If the signature of an instance is “[0,N]+[25,S]+[23,S]”, it will be detected as ‘normal’ in varUSAID but as ‘anomaly’ in USAID. For this reason, although the longer signature increases the detection computation, the redundancy in a signature is critical to enhance the detection of novel intrusions.

## 5 Related work

Even though feature selection is well known as a critical step for intrusion detection [5], most research on AID techniques utilize the expert knowledge to select the features manually [4] [2] [8], and there are only a few reported work relating to feature evaluation for intrusion detection [10] [5]. In [10], the feature ranking is done by evaluating the performance on the test audit trails for intrusion detection. Therefore, to evaluate every feature, the whole intrusion detection procedure, including training and detecting, should be run at a time. In our methodology, the feature evaluation and selection is done by maximizing the feature distinguishability between normal and anomalous signatures only in the training phase. In MADAM ID [5], the statistical patterns are first mined from the network audit data, and then these patterns are used as guidelines to select features for intrusion detection. Unfortunately, the statistical patterns may not reflect the audit trails completely, and there is no guarantee that all feature distinguishability will be used in the detection phase. However, in varUSAID, as the stopping criteria for the feature selection is that the distinguishing ability will not change even with more features, all the features that add to the distinguishability will be included in the sorted feature vector.

The method of variable-length signatures in varUSAID is similar to the generated rules of decision tree [9], and the feature selection is also done in decision tree. However, unlike varUSAID, the feature selection in decision tree is based on the information gain by categorizing the training instances according to feature values of a feature. In varUSAID, feature selection is done by maximizing

the distinguishing ability of a feature, which is measured by the number of additional instances classified into ‘normal’ and ‘anomalous’ feature subspaces caused by only one intrusion. Besides that, there are several significant differences between varUSAID (or USAID) and decision tree,

- if two instances are identical except class labels in the training audit trails, they perform different operations. Decision tree will use all features to build a leaf node, and at last label the node using majority voting. varUSAID will not use the overlapping features since they will not increase the distinguishability of the signatures.
- for nominal features, decision tree needs to know all feature values beforehand. But USAID (varUSAID) can incrementally append the new nominal feature values. This scenario is possible for intrusion detection.
- the original decision tree design (e.g., ID3) is only for discrete features [9], but varUSAID is built on nominal, discrete and continuous features, and they are treated in a uniform way using the concept of feature range.

## 6 Conclusions

In this paper, we try to find a solution to cut down the detection computation, to compact the behavior model, and to lower the false alarm rate, all of which are well known problems plaguing the past research on intrusion detection. To achieve it, a methodology for feature selection and a novel concept, *variable-length signature* are introduced in a new intrusion detection technique called varUSAID. Among our experimental results, we found that our feature selection methodology is effective in incorporating all distinguishing ability in the sorted feature vector. Our experimental results also have shown that the false alarm rate in varUSAID was only 30% of that in USAID, that the storage space of the behavior model in varUSAID was only 0.48% of that in USAID, and that the computation in detecting intrusions has been decreased 92.18% w.r.t. USAID-41, and 81.15% w.r.t. USAID-17. Besides these advantages, the detection efficiency of varUSAID for known intrusions are also encouraging. In addition, in our experiments, an important fact is discovered for the first time that the redundancy in signatures is useful to detect new intrusions, which is contrary to intuition.

## References

- [1] Charles Elkan. Results of the KDD’99 classifier learning contest. <http://www-cse.ucsd.edu/users/elkan/clresults.html>, September 1999.
- [2] Stephanie Forrest, Alan S. Perelson, Lawrence Allen, and Rajesh Cherkuri. Self-nonsel discriminant in a computer. In *Proceedings of IEEE Symposium on Research in Security and Privacy*, pages 202–212. Oakland, CA, 16-18 May 1994.

- [3] Klaus Julisch. Clustering intrusion detection alarms to support root cause analysis. *ACM Transaction on Information and System Security*, 6(4):443–471, November 2003.
- [4] Christopher Kruegel, Thomas Toth, and Engin Kirda. Service specific anomaly detection for network intrusion detection. In *Proceedings of the 2002 ACM symposium on Applied computing*, pages 201–208, Madrid, Spain, March 2002.
- [5] Wenke Lee and Salvatore J. Stolfo. A framework for constructing features and models for intrusion detection systems. *ACM Transactions on Information and System Security*, 3(4):227–261, November 2000.
- [6] Zhuowei Li and Amitabha Das. Visualizing and identifying intrusion context from system calls trace. In *Proceedings of the 20th Annual Computer Security Applications Conference*, pages 61–70, Arizona, USA, December 2004.
- [7] Zhuowei Li, Amitabha Das, and Jianying Zhou. Usaid: Unifying signature-based and anomaly-based intrusion detection. In *PAKDD*, pages 702–712, 2005.
- [8] Matthew V. Mahoney and Philip K. Chan. Learning nonstationary models of normal network traffic for detecting novel attacks. In *Proceedings of ACM SIGKDD*, pages 376–385, July 23-26 2002.
- [9] Tom Mitchell. *Machine Learning*. McGraw Hill, 1997.
- [10] Srinivas Mukkamala and Andrew H. Sung. Identifying significant features for network forensic analysis using artificial intelligent techniques. *International Journal of Digital Evidence*, 1(4):1–17, Winter, 2003.
- [11] Martin Roesch. Snort - lightweight intrusion detection for networks. In *Proceedings of the 13th USENIX conference on System Administration*, pages 229–238, Berkeley, CA, USA, 1999. USENIX Association.
- [12] Shai Rubin, Somesh Jha, and Barton P. Miller. Automatic generation and analysis of nids attacks. In *Proceedings of the 20th Annual Computer Security Applications Conference*, pages 28–38, Tucson, Arizona, USA, December 2004. IEEE Computer Society.
- [13] David Wagner and Paolo Soto. Mimicry attacks on host-based intrusion detection systems. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 255–264, Washington, DC, USA, November 2002.