Reversing the Effects of Tokenisation Attacks against Content-Based Spam Filters

Igor Santos, Carlos Laorden, Borja Sanz, Pablo G. Bringas
S^3Lab, DeustoTech - Computing
Deusto Institute of Technology, University of Deusto
Avenida de las Universidades 24, 48007, Bilbao, Spain
Email: {isantos, claorden, borja.sanz, pablo.garcia.bringas}@deusto.es

***Abstract*—Spam has become a major issue in computer security because it is a channel for threats such as computer viruses, worms and phishing. More than 85% of received e-mails are spam. Historical approaches to combating these messages, including simple techniques like sender blacklisting or the use of e-mail signatures, are no longer completely reliable. Many current solutions feature machine-learning algorithms trained using statistical representations of the terms that most commonly appear in such e-mails. However, there are attacks that can subvert the filtering capabilities of these methods. *Tokenisation attacks*, in particular, insert characters that create divisions within words, causing incorrect representations of e-mails. In this paper, we introduce a new method that reverses the effects of tokenisation attacks. Our method processes e-mails iteratively by considering possible words, starting from the first token and compares the word candidates with a common dictionary to which spam words have been previously added. We provide an empirical study of how tokenisation attacks affect the filtering capability of a Bayesian classifier and we show that our method can reverse the effects of tokenisation attacks.**

I. Introduction

Spam has become a significant problem for e-mail users in the past decade; an enormous amount of spam arrives in people's mailboxes every day. At the time of this writing, 87.6% of all e-mail messages were spam, according to the Spam-o-meter website[1]. Spam is also a major computer security problem that costs billions of dollars in productivity losses [1]: it is the medium for phishing (i.e., attacks that seek to acquire sensitive information from end-users) [2] and for spreading malicious software (e.g., computer viruses, Trojan horses, spyware and Internet worms) [1].

The simplest methods for filtering junk e-mail are usually blacklisting or signature-based [3]. Blacklisting is a simple technique that is broadly used in most filtering products; such systems filter out e-mails from certain senders. In contrast, whitelisting systems [4] deliver messages only from designated senders to reduce the number of misclassified legitimate e-mails (also known as 'ham' by the spam community). Another popular variant of these so-called banishing methods entails DNS blacklisting, in which the host address is checked against a list of networks or servers known to distribute spam [5], [6].

In contrast, signature-based systems create a unique hash value (i.e., a message digest) for each known spam message [7]. The main advantage of these methods is that they rarely produce false positives. Examples of signature-based spam filtering systems are Cloudmark[2], a commercial implementation of a signature-based filter that is integrated with the e-mail server, and Razor[3], a filtering system that uses a distributed and collaborative technique to spread signatures [3].

However, these simplistic methods have several shortcomings. First, blacklisting methods produce a high rate of false positives, making them unreliable as a standalone solution [8]. Second, signature-based systems are unable to detect spam messages until they have been identified, properly registered and documented [3].

A large amount of research has been dedicated to finding better spam filtering solutions. Machine-learning approaches have been effectively applied to text categorisation problems [9], and they have been adopted for use in spam filtering systems. Consequently, substantial work has been dedicated to the naïve Bayes filtering [10]; several studies on its effectiveness have been published [11], [12], [13], [14], [15]. Another broadly embraced machine-learning technique is the Support Vector Machine (SVM) method [16]. The advantage of SVM is that its accuracy is not diminished even when a problem involves a large number of features [17]. Several SVM approaches have been applied to spam filtering [18], [19]. Likewise, decision trees, which classify samples using automatically learned rule-sets (i.e., tests) [20], have also been used for spam filtering [21]. All of these machine-learning-based spam filtering approaches are known as statistical content-based approaches [22].

Machine-learning approaches model e-mail messages using the Vector Space Model (VSM) [23]. VSM is an algebraic approach for Information Filtering (IF), Information Retrieval (IR), indexing and ranking. This model represents natural language documents mathematically by vectors in a multidimensional space where the axes are terms within messages. VSM requires a pre-processing step in which messages are divided into tokens by separator characters (e.g., space, tab, colon, semicolon, or comma). This step is known as tokenisation [24] and it is mandatory for generating content-based spam filtering tools. Attacks against this process called *tokenisation*

[1]http://www.junk-o-meter.com/stats/index.php

[2]http://www.cloudmark.com
[3]http://razor.sourceforge.net

attacks, insert separators inside words within a message (e.g., 'via.gra'), permitting spammers to bypass filtering systems while the body of the message is still readable by e-mail users [25].

We propose the first method for reversing the effects of tokenisation attacks to recover the filtering capabilities of content-based methods. Our method reconstructs a message by processing each token iteratively, analysing the possible words surrounding the tokens. Our main points are as follows:

- Presentation of a new method capable of removing the effects of tokenisation attacks.
- An empirical study of the effects of tokenisation attacks against machine-learning spam filters.
- An empirical demonstration of the ability of our method to recover the filtering rate of machine-learning classifiers, thus transforming the obfuscated e-mails back to their original form.

The remainder of this paper is organised as follows. Section II describes the attacks against tokenisation in spam filtering tools. Section III introduces and describes a new method capable of reversing the effects of the attacks on tokenisation. Section IV provides an empirical study of how tokenisation affects machine-learnin spam filtering systems and evaluates the proposed method. Section V presents the main limitations of the proposed method and proposes possible enhancements. Finally, Section VI concludes the study and outlines the avenues for future work.

II. Machine Learning Spam Filtering and Tokenisation attacks

Spam filtering software attempts to accurately classify email massages into 2 main categories: spam or not spam (also known as 'ham'). To this end, we use the information found within the body and subject of an e-mail message and discard every other piece of information (e.g., the sender or time-stamp of the e-mail). To represent messages, we start by removing stop-words [26], which are words devoid of content (e.g., 'a','the','is'). These words do not provide any semantic information and add noise to the model [27].

Afterwards, we represent the e-mails using an IR model. Formally, let the IR model be defined as a 4-tuple $[\mathcal{E}, \mathcal{Q}, F, R(q_i, e_j)]$ [24] where $\mathcal{E}$, is a set of representations of e-mails; F, is a framework for modelling e-mails, queries and their relationships; $\mathcal{Q}$, is a set of representations of user queries; and, finally, $R(q_i, e_j)$ is a ranking function that associates a real number with a query q_i ($q_i \in \mathcal{Q}$) and an e-mail representation e_j, so that ($e_j \in \mathcal{E}$).

As $\mathcal{E}$ is the set of text e-mails e, $\{e : \{t_1, t_2, ...t_n\}\}$, each comprising n terms $t_1, t_2, \ldots, t_n$, we define the weight $w_{i,j}$ as the number of times the term t_i appears in the e-mail e_j if $w_{i,j}$ is not present in e, $w_{i,j} = 0$. Therefore, an e-mail e_j can be represented as the vector of weights $\vec{e_j} = (w_{1,j}, w_{2,j}, ...w_{n,j})$.

On the basis of this formalisation, spam filtering systems commonly use the Vector Space Model (VSM) [23], which represents e-mails algebraically as vectors in a multidimensional space. This space consists only of positive axis intercepts. E-mails are represented by a term-by-document matrix, where the $(i,j)^{th}$ element illustrates the association between the $(i,j)^{th}$ term and the j^{th} e-mail. This association reflects the occurrence of the i^{th} term in e-mail j. Terms can represent different textual units (e.g., words or phrases) and can also be individually weighted, allowing the terms to become more or less important within a given mail message or the e-mail collection $\mathcal{E}$ as a whole. Once documents are represented as vectors, classifiers can be applied.

A. Machine Learning algorithms

1) Bayesian approaches: The naïve Bayes is a probabilistic classifier based on the Bayes' theorem [28], with strong independence assumptions. From a finite set of classes $\mathcal{C}$, the classifier assigns to an instance the most probable classification:

$$c_{NB} = \underset{c \in \mathcal{C}}{\text{argmax}} P(c|\vec{v}) \tag{1}$$

where c_{NB} is the class assignation, c is each possible classification (e.g., spam or ham), $\vec{v}$ is the vector of term weights and $P(c|\vec{v})$ is the probability of an e-mail belonging to class c when the attributes $v_i \in \vec{v}$ occur. Applying Bayes' theorem [28] we obtain:

$$c_{nb} = \underset{c \in \mathcal{C}}{\text{argmax}} P(c)(\vec{v}|c) \tag{2}$$

which allows us to estimate posterior probabilities, $P(\vec{v}|c) = P(v_1, v_2, ..., v_n|c)$ using a training data set. We can then introduce the naïve assumption (which assumes that every variable depends only on the class):

$$P(v_1, v_2, ..., v_n|c) = \prod_i P(v_i|c) \tag{3}$$

in which we apply equation 1, obtaining:

$$c_{nb} = \underset{c \in C}{\text{argmax}} P(c) \prod_i P(v_i|c) \tag{4}$$

which is the naïve Bayes classifier. Through this classifier we can estimate the probability of an e-mail being spam given its vector of term weights.

Extending the naïve approach, Bayesian Networks [29], which are based on the *Bayes Theorem*, are defined as graphical probabilistic models for multivariate analysis. Specifically, they are directed acyclic graphs that have an associated probability distribution function [30]. Nodes within the directed graph represent problem variables (they can be either a premise or a conclusion) and the edges represent conditional dependencies between such variables. Moreover, the probability function illustrates the strength of these relationships in the graph [30].

2) Decision Trees: Decision Tree classifiers are a type of machine-learning classifiers that can be graphically represented as a tree. The internal nodes represent conditions of the problem variables, and their final nodes (or leaves) constitute the final decision of the algorithm [20].

Formally, a decision tree graph $G = (V, E)$ consists on a not empty set of finite nodes V and a set of arcs E. If the set of arcs is composed of ordered bi-tuples (v, w) of vertices, then we say that the graph is *directed*. A *path* is defined as an arc sequence of the form $(v_1, v_2), (v_2, v_3), ..., (v_{n-1}, v_n)$. Paths can be expressed by origin, end and distance (i.e., the minimum number of arcs from the origin to the end). In addition, if (v, w) is an arc within the tree, v is considered the *parent* of w. Otherwise, w is defined as the *child* node of v. The single node with no parent is defined as the *root node*. Every other node in the tree is defined as an *internal node*. In order to build the representation of the tree, a set of binary questions (yes-no) are answered.

There are several training algorithms that are typically used to learn the graph structure of these trees by means of a labelled dataset. In this work, we use *Random Forest*, which is an ensemble (i.e., combination of weak classifiers) of different randomly-built decision trees [31]. Further, we also use *J48*, the *Weka* [32] implementation of the *C4.5* algorithm [33].

3) K-Nearest Neighbours: The *K-Nearest Neighbour* (KNN) [34] classifier is one of the simplest supervised machine-learning models. This method classifies an unknown specimen based on the class of the instances closest to it in the training space by measuring the distance (in our work, we have used the Euclidean distance) between the training instances and the unknown instance.

Even though several methods to choose the class of the unknown sample exist, the most common technique is to simply classify the unknown instance as the most common class amongst the K-nearest neighbours.

4) Support Vector Machines (SVM): SVM classifiers consist of a hyperplane dividing a n-dimensional-space-based representation of the data into two regions. This hyperplane is the one that maximises the *margin* between the two regions or classes (in our case spam or not spam). Maximal margin is defined by the largest distance between the examples of the two classes computed from the distance between the closest instances of both classes (called *supporting vectors*) [16]. Formally, the optimal hyperplane is represented by a vector w and a scalar m in a way that the inner products of w with vectors $\phi(X_i)$ from the two classes are divided by an interval between -1 and $+1$ subject to m:

$$(w, \phi(X_i)) - m \geq +1 \tag{5}$$

for every X_i that belongs to the first class (in our case spam) and

$$(w, \phi(X_i)) - m \leq -1 \tag{6}$$

for every X_i that belongs to the second class (in our case not spam).

The optimisation problem that involves finding w and m can be formulated solely in terms of inner products $\phi(x_i)$, $\phi(x_j)$ between data points when this problem is stated in the dual form from quadratic programming [35]. In mathematics, an inner product space is a vector space with the additional structure called an inner product. This structure relates each pair of vectors in the space with a scalar quantity known as the inner product of the vectors. An specific case of inner product is the well-known dot product between vectors.

A kernel is thus defined by $\alpha(\phi(X_i), \phi(X_j)) = x_i \cdot \phi(X_j)$ forming the linear kernel with inner products. Generally, instead of using inner products, the so-called *kernel functions* are applied. These kernel functions lead to non-linear classification surfaces, such as polynomial, radial or sigmoid surfaces [36].

B. Tokenisation

Tokenisation is the process of breaking the stream of text into tokens, which are the minimal units of features [27]. This process is performed to construct the VSM representation of a document [27], [24] and it is required for the learning and testing of the naïve Bayes classifier.

In particular, tokenisation is an important step for statistical content-based spam filtering tools. These methods model e-mail messages through the *Vector Space Model* (VSM) [23] that represents documents in a mathematical manner using vectors in a multidimensional space.

However, attacks against tokenisation modify key features of the message by splitting words up using spaces or HTML layout tricks [25]. For example, consider a message $m = \{buy\ viagra\}$ that has been stored in a training set $\mathcal{E}$. In the VSM, the features that compose the vector correspond to words within the message. Because both buy and $viagra$ are common spam terms and there are already similar messages within $\mathcal{E}$, the message will be flagged as spam. To circumvent this detection, spammers can modify the message by inserting spaces into words, creating a *tokenised e-mail* $m' = \{bu\ y\ via\ gra\}$. The new vector cannot be classified as spam unless there are messages in the training set with the words composing the e-mail m'. These techniques allow spammers to bypass spam filtering systems while the text remains understandable by the e-mail recipient.

III. A NEW METHOD FOR REVERSING TOKENISATION

Common tokenisation attacks insert separators (e.g., dots, commas, semicolons, spaces, tabs) into words making the content-based spam filtering systems incorrectly select the terms for the *term vector model* [23] (also known as the Vector Space Model). This attack allows a spammer to evade filtering. To solve this issue, we developed a new algorithm called *JURD* (Joiner of Un-Readable Documents). JURD is capable of reversing most of the effects of tokenisation attacks in e-mail messages.

Formally, we define an e-mail $\mathcal{M}$ as a set composed of n terms (tokens), i.e. t_i, $\mathcal{M} = \{t_1, t_2, \ldots, t_{n-1}, t_n\}$. We use a dictionary resource $\mathcal{D}$ that includes every word w_i within a language $\mathcal{L}$ such as $\forall w_i \in \mathcal{D} \ : \ w_i \in \mathcal{L}$. Because our scope is spam filtering, we add to $\mathcal{D}$ a set of words $\mathcal{S}$ composed of common spam terms where $\exists w_i \in \mathcal{S} \;:\; w_i \notin \mathcal{D}$ (e.g., named entities like 'viagra' or 'cialis'). Thus, a new dictionary, $\mathcal{D}' = \mathcal{D} \cup \mathcal{S}$, is formed.

Algorithm 1 shows the pseudo-code of our method. We extract ℓ possible terms for each token t_i within the original message $\mathcal{M}$. Each of these possible terms $\mathcal{P}_i$ is the result of the concatenation of the next 0 to $\ell - 1$ number of tokens in the original message $\mathcal{M}$ to t_i. In other words, we obtain the different possible combinations by extending a window (i.e., n-gram) of size 1 to ℓ from a token t_i within the original e-mail $\mathcal{M}$. Thereafter, we determine whether the possible terms are words in the dictionary $\mathcal{D}'$. If a term $\mathcal{P}_i$ cannot be found in $\mathcal{D}'$, we retrieve the most similar word in $\mathcal{D}'$. To this end, we use the Levenshtein distance [37], i.e., the minimum number of edits needed to transform one word into another, where insertion, deletion, and substitution are the possible edits. If the possible term is either in $\mathcal{D}$ or a term exists in $\mathcal{D}$ with a Levenshtein distance to $\mathcal{P}_i$ lower or equal than t, then we add it to the dis-tokenised message, and we update the index in order to repeat the process with the next token in the original message $\mathcal{M}$ immediately after the last one within $\mathcal{P}_i$.

```
input : A message $\mathcal{M}$, the dictionary of words $\mathcal{D}'$, the maximum length of the
        window $\ell$, and the similarity threshold $t$
output: A dis-tokenised message $\mathcal{M}'$
$\mathcal{M}' = \emptyset$;
for $i \leftarrow 1$ to $|\mathcal{M}|$ do
    // $\mathcal{P}$ is the set of possible composing terms within
       $\mathcal{M}'$ of an actual word. Its size when full is $\ell$
    $\mathcal{P} = \emptyset$;
    // This loop extracts all the possible token
       combinations, starting from the $i^{th}$ token,
       composed of a number $\ell$ of tokens
    for $j \leftarrow 1$ to $\ell : (i + j) < |\mathcal{M}|$ do
        $\mathcal{P}_j \leftarrow$ Concatenation($\mathcal{P}'_{j-1}$,$t_{i+j}$);
    end
    if $\mathcal{P} \neq \emptyset$ then
        // $b$ indicates whether a combination has been
           selected or not
        $b \leftarrow$ false;
        // $n$ indicates whether a combination is similar
           (with a distance lower than $t$) to a word in $\mathcal{D}'$
        $n \leftarrow$ false;
        // This loop evaluates every possible
           combination of tokens.
        for $z \leftarrow |\mathcal{P}|$ down-to 1 do
            // The combination of tokens $\mathcal{P}_i$ was found in
               the dictionary $\mathcal{D}'$
            if $\mathcal{P}_i \in \mathcal{D}'$ then
                $b \leftarrow$ true;
                AddXtoY($\mathcal{P}_i$,$\mathcal{M}'$);
                $i \leftarrow i + z$;
            end
            else if $\neg b \wedge \neg n$ then
                // We retrieve the most similar word to
                   $\mathcal{P}_i$ with an edit distance greater than
                   $t$
                $s \leftarrow$ RetrieveMostSimilarWord($\mathcal{P}_i$,$t$);
                if $s \neq \emptyset$ then
                    $n \leftarrow$ true;
                    AddXtoY($s$,$\mathcal{M}'$);
                    $i \leftarrow i + z$;
                end
            end
        end
    end
end
return $\mathcal{M}'$
```

Fig. 1: JURD algorithm

```
vi.a.gra makes yo.u perform and
fe.el like yo.u are 1.8 again
```

Fig. 2: An example of a tokenised message

Notice that JURD gives priority to terms composed of larger numbers of tokens. To understand this choice, Figure 2 shows an example of a tokenised message. Using an ℓ of 3, we can obtain the following possible combinations starting from the first token: 'vi': 'vi','via', and 'viagra'. In this case, 'vi' is not in the dictionary $\mathcal{D}'$, but 'via' and 'viagra' are. If we had given priority to small combinations, we would have selected 'via' instead of 'viagra', which is actually a common spam term.

IV. Evaluation

A. General Methodology

We employed the *Ling Spam* dataset[4] to serve as the spam corpus. Ling Spam comprises both spam and legitimate messages retrieved from the *Linguistic list*, an e-mail distribution list focusing on *linguistics*. The dataset consists of 2,893 different e-mails, of which 2,412 are legitimate e-mails obtained by downloading digests from the linguistic list and 481 are spam e-mails retrieved from one of the authors' inbox (a more detailed description of the corpus is provided in [13], [38]). Spam represents nearly 16% of the whole dataset, a commonly used rate in experiments [39], [40], [38]. We performed *Stop Word Removal* [26] and *stemming* [41] on the e-mails, generating 4 different datasets:

1) **Bare:** In this dataset, HTML tags, separation tokens and duplicate e-mails were removed from messages.
2) **Lemm:** In addition to the removal pre-process step, a stemming phase was performed. Stemming reduces inflected or derived words to their stem, base or root form.
3) **Stop:** For this dataset, a stop word removal task was performed. This process removes all stop words (e.g., common words like *'a'* or *'the'*).
4) **Lemm_stop:** This dataset uses a combination of both stemming and stop-word removal processes.

We did not use the *lemm* or *lemm_stop* datasets. Additionally, instead of using the *stop* dataset we used the *bare* dataset and we performed a stop word removal based on an external stop-word list[5].

To generate the enhanced dictionary resource $\mathcal{D}'$, we used an English dictionary composed of 236,983 words[6] as $\mathcal{D}$ and we extracted a list of 10,149 words[7] from the spam messages within the Ling spam dataset to act as the common spam word corpus $\mathcal{S}$.

In this work, we want to answer the following research questions:

[4]http://nlp.cs.aueb.gr/software_and_datasets/lingspam_public.tar.gz

[5]http://www.webconfs.com/stop-words.php

[6]Available online: http://free.pages.at/rnbmusiccom/fulldictionary00.zip Alternative link: http://paginaspersonales.deusto.es/isantos/public/englishwords.zip

[7]Available online in http://paginaspersonales.deusto.es/isantos/public/spamwords.zip

1) *How do tokenisation attacks affect common spam filters?*
2) *What is the effect of applying JURD to tokenised messages and how does it affect common Bayesian spam filters?*

To answer the first question, we performed an experiment that compared the results of training several spam filters with non-tokenised e-mails. We then used a set of both tokenised and non-tokenised messages to reveal the differences in the results. For the second question, we applied the JURD algorithm to the set of the tokenised e-mails and compared the results with the results obtained in the previous experiment.

For both experiments, we modelled the messages' original dataset using the VSM [23]. We used the *Term Frequency – Inverse Document Frequency* (TF–IDF) [27] weighting schema, where the weight of the i^{th} term in the j^th document, denoted by $weight(i,j)$, is defined by:

$$weight(i,j) = tf_{i,j} \cdot idf_i \tag{7}$$

where *term frequency* $tf_{i,j}$ is defined as:

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}} \tag{8}$$

where $n_{i,j}$ is the number of times the term $t_{i,j}$ appears in a document d, and $\sum_k n_{k,j}$ is the total number of terms in the document d. The inverse term frequency idf_i is defined as:

$$idf_i = \frac{|\mathcal{D}|}{|\mathcal{D} : t_i \in d|} \tag{9}$$

where $|\mathcal{D}|$ is the total number of documents and $|\mathcal{D} : t_i \in d|$ is the number of documents containing the term t_i.

We used the *Themis* implementation [42], which enabled us to populate a database using the e-mail messages from the Ling Spam dataset.

We constructed a file with the resultant vector representations of the e-mails. We extracted the top 1,000 attributes using *Information Gain* [43], an algorithm that evaluates the relevance of an attribute by measuring the information gain with respect to the class:

$$IG(j) = \sum_{v_j \in R} \sum_{C_i} P(v_j, C_i) \cdot \frac{P(v_j, C_i)}{P(v_j) \cdot P(C_i)} \tag{10}$$

where C_i is the i^{th} class, v_j is the value of the j^{th} interpretation, $P(v_j, C_i)$ is the probability that the j^{th} attribute has the value v_j in the class C_i, $P(v_j)$ is the probability that the j^{th} interpretation has the value v_j in the training data, and $P(C_i)$ is the probability that the training dataset belongs to the class C_i .

For the evaluation of the different machine learning algorithms we used the tool WEKA (Waikato Environment for Knowledge Analysis) [32]. Specifically, the algorithms used in this tool can be seen in Table **??**. In those cases in which no configuration parameters are specified, the configuration used was the default of the tool.

TABLE I: Configuration of the algorithms.

Used Algorithms	**Configuration**
NaïveBayes	N/A
Bayesian Network	K2 and TAN
KNN	K: 1, 3 and 5
SVM	Polynomial kernel Radial Basis Function (RBF) kernel Pearson VII Kernel
C4.5	N/A
RandomForest	N = 10, 30 and 50

After removing the less significant attributes, the resultant file is the training dataset for the classifiers [10]. In this way, we obtained a training set, and 2 testing datasets: a dataset comprising 478 spam messages after performing a tokenisation attack, and 478 junk e-mails obtained after JURD-mediated de-tokenising of the previously tokenised messages.

To assess the results of every option, we measured the *True Positive Ratio* (TPR) which is the number of correctly detected spam messages, divided by the total number of e-mails (shown in equation 11):

$$TPR = \frac{TP}{TP + FN} \tag{11}$$

where TP is the number of correctly classified spam e-mails (i.e., true positives), FN is the number of spam messages misclassified as legitimate mails (false negatives), and TN is the number of legitimate e-mails that were correctly classified.

We divided these two processes into two different experiments. The first experiment examined the effects of tokenisation attacks against statistical spam filters. The second experiment measured the effectiveness of JURD as a countermeasure to this attack.

B. Performance evaluation of JURD

We evaluated the processing overhead introduced by our method. To this end, we measured the times required to process the 478 spam messages over the number of tokens composing the messages with a configuration of $\ell = 10$ and $t = 1$.

Average Time for Message (ms)	**Average Time(ms) for Token**
52,092.36	61.72

TABLE II: Average Time Requirements for Message and Token for $\ell = 10$ and $t = 1$

The dependence between the required time and the number of tokens was linear (refer to Figure 3) while the average time requirements were 52,092.35 ms for a single message and 61.72 ms for each token (Table II).

C. Evaluation of the effects of tokenisation attacks

In order to evaluate the effects of tokenisation attacks against spam filters, we first developed an algorithm that automatically inserts separating characters into words within e-mail messages (shown in Algorithm 4).

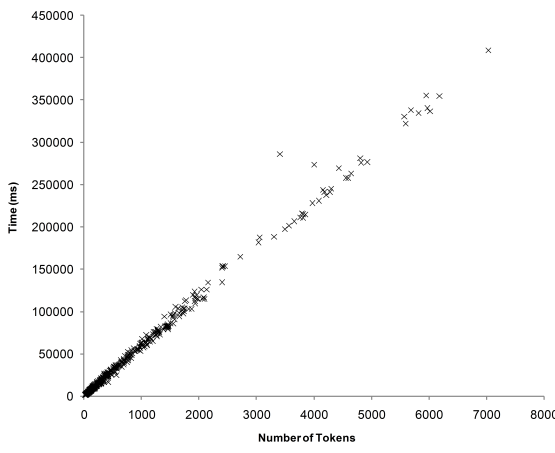

Fig. 3: Performance of JURD depending on the number of tokens with a configuration of $\ell = 10$ and $t = 1$.

```
input  : A message $\mathcal{M}$ and the probability to insert a token $p$
output: A tokenised message $\mathcal{M}'$
$\mathcal{M}' = \emptyset$;
foreach $w_i \in \mathcal{M}$ do
    // We insert separators into a word according the
       given probability $p$
    if (RandomInteger(100)/100) $< p$ then
        // The number of insertions to be performed is
           randomly determined
        $n \leftarrow$ RandomInteger($|M| - 1$);
        for $i \leftarrow 1$ to $n$ do
            // The position of the separator is randomly
               determined
            $r \leftarrow$ RandomInteger(Length($w_i$));
            $s_1 \leftarrow$ SubString($w_i$,0,$r$);
            $s_2 \leftarrow$ SubString($w_i$,$r$,Length($w_i$));
            AddXtoY(Concatenation($s_1$,$s_2$),$\mathcal{M}'$);
        end
    end
end
return $\mathcal{M}'$
```

Fig. 4: A Random Tokenisation Attack

We performed the tokenisation attack with 478 of the Ling Spam dataset messages with a probability of insertion of 95 %. Figure 5 shows a snippet from a spam message before and after this process.

```
the virtual girlfriend
and virtual boyfriend are
artificial intelligence
programs for your ibm pc
or compatible and also for
macintosh. you can watch
them , talk to them ...
```

(a) Before performing the attack

```
the virtua l girlfri end an
d.virtual boyfrien d. a re
ar.t.ificial intellig ence
p rogram;s fo r, you r i bm
pc o r compatible and also
fo r macinto.sh you c a n
watch t hem talk t o.them
...
```

(b) After performing the attack

Fig. 5: Example of the effects of tokenisation attack

We applied this tokenisation algorithm to the 478 spam messages. We then represented the messages in the VSM formed by the 1,000 selected words. Thus, we finally generated the test file for the evaluation of the tokenisation effects.

To evaluate the effects of tokenisation by comparison with the original results (without performing tokenisation attacks), we evaluated the classifiers through a k-fold cross-validation [44] applied to the training dataset, with $k = 10$. In this way, the dataset was split 10 times into 10 different sets of learning sets (90% of the total dataset) and testing sets (10% of the total dataset). The purpose of this division was to use different spam messages for training and testing to obtain a reliable precision for the method when applied to common messages.

Table III shows the effects of the tokenisation attack in the detection of spam. Because we only tokenised spam messages, the results are show in TPR (i.e., the amount of spam messages correctly detected). Note that although the tokenisation attack was performed in a random fashion, every classifier we tested lost more than 5 units of its detection capability. In particular, the Bayesian classifiers, which are the standard for spam filtering, were affected by the tokenisation in 17.8 units for the naïve approach, 15.9 for the K2 structural learning approach, and 6.5 units for the TAN approach. The TAN approach, which is an algorithm that starts with a na"ive Bayes classifier but that expands that structure by expanding the tree, was the less affected Bayesian classifier. However, it lost more than a 6% of detection ratio. Regarding the instance-based approaches, we tested KNN with different values of K: 1,3, and 5. These classifiers did not perform as well as the Bayesian approaches even with the not tokenized messages, and they also lost this capabilities when dealing with tokenised messages. Concretely, KNN K=1 lost 12.8 units, KNN K=3 25.7 units, and KNN K=5 28 units. We also tested SVM approach with different kernels and they performed similarly when dealing with the tokenised messages. SVM with polynomial kernel decreases its detection ratio in 16.6 units, with RBF kernel in 19.4 units, and the Pearson VII kernel lost 17.6 units. Regarding decision tree based algorithms, we tested C4.5 and different configurations of Random Forests. C4.5 decreases from 84.9 to 72.6 of TPR, while Random Forest N=10 losts 4.2 units, N=30 6.7 units, and N=50 6.9 units.

Note that although the tokenisation attack was performed in a random fashion, every classifier lost an important detection capability. Therefore, if the spammer selected the words to be tokenised (which is very likely to happen) the detection rate would be even lower. These results outline the importance of countering these attacks in order to face custom spam.

D. Evaluation of the JURD algorithm

To counter the tokenisation attack, we applied JURD to the 478 already-tokenised spam messages. To select a window size ℓ, we performed a preliminary study in which we selected the most tokenised spam messages and tried different window sizes. We realised that the higher values for the parameter ℓ would result in better JURD performance. However, higher values for parameter ℓ introduced higher performance overhead. Therefore, we chose an ℓ value of 10, which produced the same results for nearly every spam message. Figure 5 shows a sample from a previously tokenised spam message of the Ling dataset before and after the application of JURD with $\ell = 10$ and $t = 1$.

TABLE III: Effects of Tokenisation Attack in Spam Filtering. Measuring TPR (%).

Algorithm	**Original Dataset**	**Tokenised Dataset**
Naïve Bayes	77.8	60.5
Bayesian Network K2	79.3	63.4
Bayes TAN	95.8	89.3
KNN K=1	68.0	55.2
KNN K=3	61.3	35.6
KNN K=5	54.2	26.2
SVM Polynomial Kernel	85.8	69.2
SVM RBF Kernel	41.6	22.2
SVM Pearson VII Kernel	76.6	59.0
C4.5	84.9	72.6
Random Forest N=10	95.0	90.8
Random Forest N=30	95.6	88.9
Random Forest N=30	96.2	89.3

To select the similarity threshold t, we conducted a similar study. We concluded that a large edit distance changes the meaning of a message, while a distance of 0 is too strict. Thus, we selected a $t = 1$.

```
the virtua l girlfri end an
d.virtual boyfrien d. a re
ar.t.ificial intellig ence
p rogram;s fo r, you r i bm
pc o r compatible and also
fo r macinto.sh you c a n
watch t hem talk t o.them
...
```

(a) Before applying JURD

```
the virtual girlfriend and
virtual boyfriend arear
t ificial intelligence
programs for your ibm pc
or compatible and also for
macintosh you can watch
them talk to them ...
```

(b) After applying JURD

Fig. 6: Example of the effects of JURD

Thereafter, we represented the messages in the VSM formed by the 1000 selected words and we finally generated the test file for the evaluation of JURD.

Table IV shows the effects of JURD against the tokenisation attack. After application of JURD, the spam filtering tools nearly achieved the precision obtained with the original spam messages and in some cases they recover the full detection capability. In this way, the algorithms that performed best were Random Forest, C4.5, SVM with Pearson VII, and Bayes TAN. These result demonstrate JURD's capability to recover the detection ratio of spam filters, when tokenisation attacks are applied.

V. DISCUSSION

The final results show that JURD can reverse the effects of tokenisation attacks that allow spam to evade detection by spam filtering methods. However, there are several topics of discussion regarding the suitability of JURD.

First, JURD must be applied as a pre-processing step before modelling e-mails in the document space $\mathcal{E}$. Currently, there is no method that is able to detect whether a message has been tokenised. Therefore, a filtering system should always employ our method to assure that these attacks are engaged. However, this method introduces a significant processing overhead to spam filtering systems. Therefore, we consider that a method capable of detecting transformations within the e-mail would help to improve the overall performance of spam filtering. Such a method would act as a preliminary step for deciding if JURD is needed.

Second, we prioritised the selection of words composed of a large numbers of tokens. Although this choice nearly transformed the tokenised messages back to their original form, errors appeared in the transformation. For instance, in the example shown in Figure 6, JURD selected 'arear' instead of 'are', resulting in an incorrect selection of words thereafter. To solve this issue, we can use a dictionary of words based on frequency of usage in natural language, or in spam messages, we can prioritise the average number of tokens composing the word and its frequency of use. We can also extend the dictionary to include slang terms and common spelling mistakes (although the latter issue is nearly solved by the Levenshtein distance [37]).

Finally, though JURD manages to deal with tokenisation attacks, there are other attacks that must be considered. For example, *Good Word Attacks* modify word statistics by appending a set of words that are characteristic of legitimate e-mails, thereby circumventing spam filters [45]. However, we can adopt some of the methods that have been proposed to improve spam filtering, such as *Multiple Instance Learning* (MIL) [46]. MIL divides an instance or vector in traditional supervised learning methods into several sub-instances and it classifies the original vector based on the sub-instances [47]. [48] proposed the adoption of MIL for spam filtering by dividing an e-mail into a bag of multiple segments and classifying it as spam if at least one instance in the corresponding bag is spam.

VI. CONCLUDING REMARKS

Spam is a serious computer security issue that is not only annoying for end-users, but also financially damaging and dangerous to computer security because of the possible spread of other threats like malware or phishing. Classic machine-learning-based spam filtering methods, despite their ability to detect spam, can be defeated by tokenisation attacks.

In this paper, we presented a new method that is able to transform a tokenised message to a form similar to the original message. To this end, we iteratively processed the message looking for possible word candidates starting from a token and comparing the candidates with a dictionary of words and

TABLE IV: Effects of JURD against Tokenisation Attacks. Measuring TPR (%).

Algorithm	Original Dataset	Tokenised Dataset	After applying JURD (%)
Naïve Bayes	77.8	60.5	76.2
Bayes K2	79.3	63.4	76.7
Bayes TAN	95.8	89.3	96.9
KNN K=1	68.0	55.2	99.2
KNN K=3	61.3	35.6	73.6
KNN K=5	54.2	26.2	66.3
SVM Polynomial Kernel	85.8	69.2	90.2
SVM RBF Kernel	41.6	22.2	39.7
SVM Pearson VII Kernel	76.6	59.0	95.4
C4.5	84.9	72.6	90.4
Random Forest N=10	95.0	90.8	97.9
Random Forest N=30	95.6	88.9	99.0
Random Forest N=30	96.2	89.3	99.0

common spam terms. Our experiments show that this approach restores the spam detection rate of the classifiers and thus, handles the tokenisation attacks.

Future work will move in three main directions. First, we will enhance this method by modifying the priorities in the selection of words by adding the frequencies of use of candidate terms. Second, we will focus on attacks against statistical spam filtering systems such as good word attacks. Finally, we plan to develop a fast method for detecting whether an e-mail has been tokenised.

References

[1] A. Bratko, B. Filipič, G. Cormack, T. Lynam, and B. Zupan, "Spam filtering using statistical data compression models," *The Journal of Machine Learning Research*, vol. 7, pp. 2673–2698, 2006.

[2] T. Jagatic, N. Johnson, M. Jakobsson, and F. Menczer, "Social phishing," *Communications of the ACM*, vol. 50, no. 10, pp. 94–100, 2007.

[3] J. Carpinter and R. Hunt, "Tightening the net: A review of current and next generation spam filtering tools," *Computers & security*, vol. 25, no. 8, pp. 566–578, 2006.

[4] S. Heron, "Technologies for spam detection," *Network Security*, vol. 2009, no. 1, pp. 11–15, 2009.

[5] J. Jung and E. Sit, "An empirical study of spam traffic and the use of DNS black lists," in *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*. ACM New York, NY, USA, 2004, pp. 370–375.

[6] A. Ramachandran, D. Dagon, and N. Feamster, "Can DNS-based blacklists keep up with bots," in *Conference on Email and Anti-Spam*, 2006.

[7] A. Kołcz, A. Chowdhury, and J. Alspector, "The impact of feature selection on signature-driven spam detection," in *Proceedings of the 1st Conference on Email and Anti-Spam (CEAS-2004)*, 2004.

[8] G. Mishne, D. Carmel, and R. Lempel, "Blocking blog spam with language model disagreement," in *Proceedings of the 1st International Workshop on Adversarial Information Retrieval on the Web (AIRWeb)*, 2005, pp. 1–6.

[9] F. Sebastiani, "Machine learning in automated text categorization," *ACM computing surveys (CSUR)*, vol. 34, no. 1, pp. 1–47, 2002.

[10] D. Lewis, "Naive (Bayes) at forty: The independence assumption in information retrieval," *Lecture Notes in Computer Science*, vol. 1398, pp. 4–18, 1998.

[11] I. Androutsopoulos, G. Paliouras, V. Karkaletsis, G. Sakkis, C. Spyropoulos, and P. Stamatopoulos, "Learning to filter spam e-mail: A comparison of a naive bayesian and a memory-based approach," in *Proceedings of the Machine Learning and Textual Information Access Workshop of the 4th European Conference on Principles and Practice of Knowledge Discovery in Databases*, 2000.

[12] K. Schneider, "A comparison of event models for Naive Bayes anti-spam e-mail filtering," in *Proceedings of the 10th Conference of the European Chapter of the Association for Computational Linguistics*, 2003, pp. 307–314.

[13] I. Androutsopoulos, J. Koutsias, K. Chandrinos, G. Paliouras, and C. Spyropoulos, "An evaluation of naive bayesian anti-spam filtering," in *Proceedings of the workshop on Machine Learning in the New Information Age*, 2000, pp. 9–17.

[14] I. Androutsopoulos, J. Koutsias, K. Chandrinos, and C. Spyropoulos, "An experimental comparison of naive Bayesian and keyword-based anti-spam filtering with personal e-mail messages," in *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, 2000, pp. 160–167.

[15] A. Seewald, "An evaluation of naive Bayes variants in content-based learning for spam filtering," *Intelligent Data Analysis*, vol. 11, no. 5, pp. 497–524, 2007.

[16] V. Vapnik, *The nature of statistical learning theory*. Springer, 2000.

[17] H. Drucker, D. Wu, and V. Vapnik, "Support vector machines for spam categorization," *IEEE Transactions on Neural networks*, vol. 10, no. 5, pp. 1048–1054, 1999.

[18] E. Blanzieri and A. Bryl, "Instance-based spam filtering using SVM nearest neighbor classifier," *Proceedings of FLAIRS-20*, pp. 441–442, 2007.

[19] D. Sculley and G. Wachman, "Relaxed online SVMs for spam filtering," in *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, 2007, pp. 415–422.

[20] J. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[21] X. Carreras and L. Márquez, "Boosting trees for anti-spam email filtering," in *Proceedings of RANLP-01, 4th international conference on recent advances in natural language processing*, 2001, pp. 58–64.

[22] L. Zhang, J. Zhu, and T. Yao, "An evaluation of statistical spam filtering techniques," *ACM Transactions on Asian Language Information Processing (TALIP)*, vol. 3, no. 4, pp. 243–269, 2004.

[23] G. Salton, A. Wong, and C. Yang, "A vector space model for automatic indexing," *Communications of the ACM*, vol. 18, no. 11, pp. 613–620, 1975.

[24] R. A. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[25] G. Wittel and S. Wu, "On attacking statistical spam filters," in *Proceedings of the 1st Conference on Email and Anti-Spam (CEAS)*, 2004.

[26] W. Wilbur and K. Sirotkin, "The automatic identification of stop words," *Journal of information science*, vol. 18, no. 1, pp. 45–55, 1992.

[27] G. Salton and M. McGill, *Introduction to modern information retrieval*. McGraw-Hill New York, 1983.

[28] T. Bayes, "An essay towards solving a problem in the doctrine of chances," *Philosophical Transactions of the Royal Society*, vol. 53, pp. 370–418, 1763.

[29] J. Pearl, "Reverend bayes on inference engines: a distributed hierarchical approach," in *Proceedings of the National Conference on Artificial Intelligence*, 1982, pp. 133–136.

[30] E. Castillo, J. M. Gutiérrez, and A. S. Hadi, *Expert Systems and Probabilistic Network Models*, erste ed., New York, NY, USA, 1996.

[31] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[32] S. Garner, "Weka: The Waikato environment for knowledge analysis," in *Proceedings of the New Zealand Computer Science Research Students Conference*, 1995, pp. 57–64.

[33] J. Quinlan, *C4. 5 programs for machine learning*. Morgan Kaufmann Publishers, 1993.

[34] E. Fix and J. L. Hodges, "Discriminatory analysis: Nonparametric discrimination: Small sample performance. technical report project 21-49-004, report number 11," USAF School of Aviation Medicine, Randolf Field, Texas, Tech. Rep., 1952.

[35] C. Bishop, *Pattern recognition and machine learning*. Springer New York., 2006.

[36] S. Amari and S. Wu, "Improving support vector machine classifiers by modifying kernel functions," *Neural Networks*, vol. 12, no. 6, pp. 783–789, 1999.

[37] V. Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals," in *Soviet Physics Doklady*, vol. 10, 1966, pp. 707–710.

[38] G. Sakkis, I. Androutsopoulos, G. Paliouras, V. Karkaletsis, C. Spyropoulos, and P. Stamatopoulos, "A memory-based approach to anti-spam filtering for mailing lists," *Information Retrieval*, vol. 6, no. 1, pp. 49–73, 2003.

[39] L. Cranor and B. LaMacchia, "Spam!" *Communications of the ACM*, vol. 41, no. 8, pp. 74–83, 1998.

[40] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz, "A Bayesian approach to filtering junk e-mail," in *Learning for Text Categorization: Papers from the 1998 workshop*, vol. 62. Madison, Wisconsin: AAAI Technical Report WS-98-05, 1998, pp. 98–05.

[41] J. Lovins, "Development of a Stemming Algorithm." *Mechanical Translation and Computational Linguistics*, vol. 11, no. 1, pp. 22–31, 1968.

[42] A. Polyvyanyy, "Evaluation of a novel information retrieval model: eTVSM," 2007, MSc Dissertation.

[43] J. Kent, "Information gain and a general measure of correlation," *Biometrika*, vol. 70, no. 1, p. 163, 1983.

[44] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *International Joint Conference on Artificial Intelligence*, vol. 14, 1995, pp. 1137–1145.

[45] D. Lowd and C. Meek, "Good word attacks on statistical spam filters," in *Proceedings of the 2^{nd} Conference on Email and Anti-Spam (CEAS)*, 2005.

[46] T. Dietterich, R. Lathrop, and T. Lozano-Pérez, "Solving the multiple instance problem with axis-parallel rectangles," *Artificial Intelligence*, vol. 89, no. 1-2, pp. 31–71, 1997.

[47] O. Maron and T. Lozano-Pérez, "A framework for multiple-instance learning," *Advances in neural information processing systems*, pp. 570–576, 1998.

[48] Y. Zhou, Z. Jorgensen, and M. Inge, "Combating Good Word Attacks on Statistical Spam Filters with Multiple Instance Learning," in *Proceedings of the 19^{th} IEEE International Conference on Tools with Artificial Intelligence-Volume 02*. IEEE Computer Society, 2007, pp. 298–305.