Detecting Known and New Salting Tricks in Unwanted Emails

André Bergholz, Gerhard Paaß, Frank Reichartz, Siehyun Strobel
Fraunhofer IAIS
Schloß Birlinghoven
53754 St. Augustin, Germany
firstname.lastname@iais.fhg.de

Marie-Francine Moens
Katholieke Universiteit Leuven
Department of Computer Science
Celestijnenlaan 200A
3001 Heverlee, Belgium
sien.moens@cs.kuleuven.be

Brian Witten
Symantec Corporation
12801 Worldgate Drive, Suite 800
Herndon, VA 20170, USA
brian_witten@symantec.com

Abstract

Spam and phishing emails are not only annoying to users, but are a real threat to internet communication and web economy. The fight against unwanted emails has become a cat-and-mouse game between criminals and people trying to develop techniques for detecting such unwanted emails. Criminals are constantly developing new tricks and adopt the ones that make emails pass spam filters. We have developed a systems that enables the detection of certain common salting tricks that are employed by criminals. Salting is the intentional addition or distortion of content. In this paper we describe a framework to identify email messages that might contain new, previously unseen tricks. To this end, we compare the simulated perceived email message text generated by our hidden salting simulation system to the OCRed text we obtain from the rendered email message. We present robust text comparison techniques and train a classifier based on the differences of these two texts. In simulations we show that we can detect suspicious emails with a high level of accuracy.

1 Introduction

In the last years email traffic has shown a rapid expansion of spam and phishing. Spam and phishing emails are not only annoying to users, but a real threat to internet communication and web economy. With the introduction of automatic email spam filtering a cat-and-mouse game between the spammers and the filter developers has been initiated. The spammers are developing tricks that they hope will get their emails through the filters, and the filter developers are adapting their filters. One popular method that spammers are using is to include a high percentage of "real" text in their spam messages to fool text-based email filters, a method known as "good word attack" [9].

Salting is the intentional addition or distortion of content, usually in spam emails, aimed to obfuscate or evade automatic filtering. There exists surface salting, e.g., images containing random pixel dots, and hidden salting, e.g., text in invisible color. We have developed a hidden salting simulation model based on cognitive theory [2]. For an input email the model simulates which text will actually be seen by the user based on a fixed set of known salting tricks. In addition, the model indicates how many times each of the known salting tricks was used in the email.

In this paper we close the loop and address the problem of identifying when our hidden salting simulation system fails, most likely because some spammer has come up with a yet unknown salting trick. To this end, we try to detect differences between the simulated perceived text as generated by our hidden salting simulation system and the message text as obtained by applying OCR to the email message rendered by some rendering engine.

It has to be pointed out that the methods proposed in this paper take too much runtime to be employed at an online policy enforcement point. Rather, they should be employed at an offline analysis system. The methods proposed in this paper enable the detection of suspicious emails, but human intervention is necessary to review the detected emails and perform necessary updates to existing filtering software and the hidden salting simulation system.

Specifically, this paper makes the following contributions:

1. We describe frequent types of hidden salting and our automatic hidden salting simulation system, which generates a simulated perceived email message text.

2. We develop robust text comparison metrics that allow us to compare the simulated perceived message text to the true message text obtained by applying OCR to the rendered email.

3. We use the distances computed with these metrics' as features to train a classifier. An outlier as detected by this classifier would be a candidate email for containing a new, unseen salting trick. We automatically determine the threshold of when to consider an email as outlier.

4. We perform experiments to simulate the detection of a new trick by disabling the detection of a known trick. On publicly available email data we are able to achieve good results for the tested tricks.

This paper is organized as follows. In Section 2 we discuss related work. Section 3 summarizes our system for automatic detection of hidden salting tricks. Section 4 introduces robust metrics to compare the two texts we extract from email messages. In Section 5 we describe the experimental setup of the outlier detection classifier, the automatic classification threshold detection, and evaluation measures. Our empirical results are presented in Section 6, and Section 7 concludes the paper.

2 Related Work

Current email filtering systems mainly rely on lexical and structured features extracted from the email header or from the body of the texts. The features extracted from the header are well-known (e.g., the header fields title, from, to). The structured features from the body are often suspicious domain names of links (e.g., [8]). When unstructured contents of the emails are considered, state-of-the-art tools rely on character sequences (n-grams), the words of the emails, or word tuples composed of two or more words possibly separated by wildcards (e.g., [1]). But, the filters ignore features based on the layout of the emails, syntactic structure, and semantic or topical classifications of the content. They completely neglect hidden salting tricks, which accomplish that the filter considers different content from what the human user of the email perceives. In addition, they ignore the dynamic adaptation of the feature extraction methods to new scams.

Fumera et al. exploit the textual information embedded into images and detect willful obfuscations in images attached to emails [7]. Their contribution lies in image processing and is complementary to our work. Breuel et al. indicate possible solutions to phishing and search engine spam based on analyzing the OCR text of an HTML page instead of the HTML source itself [3]. To the best of our knowledge they do not provide an implementation or experimental results. In their well-known work, Cavnar et al. demonstrate that robust text classification in the presence of OCR errors is possible when using n-gram-based features [4]. On the other hand, recent work by Taghva et al. emphasizes that OCR errors can have a negative impact on text categorization [11].

3 Hidden Salting

Visible salting is commonly found in spam mails; its hidden variant, where deliberately content is hidden from the user, is more dangerous as it is difficult to detect, although it misleads the content filtering. Hidden text salting can be applied to any medium, e.g., text, images, audio, and to any content genre, e.g., emails, Web pages or MMS messages, and is common in fraudulent emails such as phishing mails that aim to steal personal information from users in order to commit identity theft.

We summarize here our approach for the detection of hidden salting based on a hidden salting simulation model. The details can be found in [2].

Given an email as input, a text production process, e.g., a Web browser, creates a parsed, internal representation of the email text and drives the rendering of that representation onto some output medium, e.g., a browser window. Our proposed method takes place during this rendering process, and is composed of two steps:

1. We tap into the rendering process to detect hidden content (i.e., manifestations of salting).

2. We feed the intercepted, visible text into a cognitive hidden salting simulation model, which returns the simulated perceived text.

During the first step, we intercept requests for drawing text primitives, and build an internal representation of the characters that appear on the user screen. This representation is a list of attributed glyphs (i.e., positioned shapes of individual characters, with rendering attributes and any concealing shapes). Glyphs are listed in their compositional order (i.e., the order in which they are drawn according to the source text). Then, we test for glyph visibility (i.e., which glyphs are seen by the user) according to the following glyph visibility conditions:

Clipping The glyph is drawn within the physical bounds of the drawing clip.

Concealment The glyph is not concealed by other glyphs or shapes.

Font Color The glyph's fill color contrasts well with the background color.

Glyph Size The glyph's size and shape is sufficiently large.

Failure to comply to any condition results in an invisible glyph and is an indication of hidden salting. We eliminate all invisible glyphs (i.e., retain only what is expected to be perceived by the user).

In the second step, given the set of retained (visible) glyphs, we define the order in which glyphs are most likely read by humans, i.e., the reading order. We cannot assume that the reading order equals the compositional order, since spammers are known to exploit this (e.g., text can be coded in the source as columns but actually read by humans in lines). Then, given a set of visible, positioned glyphs covering a specific area, e.g., an email or page, we use a cognitive model of text perception to define the glyph reading order as perceived by humans. The task of the cognitive model is to find a coherent partitioning of the page, with proper and coherent reading directions assigned to the each partition. A reading order is detected based on a layout characteristic where we expect that glyphs of parallel lines are aligned. In addition, we measure the compliance of the text with the language specific distributions of character n-grams, common words and word lengths obtained from a reference corpus.

This model is, however, vulnerable for the appearance of new salting tricks that make certain content invisible, hence our focus is on the detection of such new tricks in this paper.

4 Text Distance for Change Detection

Our hidden salting simulation system provides the simulated perceived message text when considering a fixed set of salting tricks. Let us call this text the *simulated perceived text* of an email. At the same time we extract a second text from the email by rendering it and then putting the resulting image through an OCR engine. Let us call this text the *true message text* of an email. The reasoning would be that if these two texts differ significantly then our hidden salting simulation system disregards some techniques that were employed by the email author. Note that we discard any external images from consideration for both texts.

Our goal now is to detect emails for which the simulated perceived text and the true message text differ. Many string distance techniques have been proposed in the past, see [6] for a good overview. They generally fall into two categories:

1. Edit-distance like measures: These measures usually add up penalties for editing operations that transform one string into the other one. Examples for these distance measures include the Levenshtein distance, the Monger-Elkan distance, and the Jaro-Winkler distance.
2. Token-based measures: These measures relate the tokens that appear in both strings to the tokens that appear in either string. Examples are the Jaccard similarity, Tf-Idf, and the Jensen-Shannon divergence.

Measures from both of these categories have drawbacks in our scenario. Edit-distances take the order in the texts into account. This is problematic in our setting. Our hidden salting simulation system uses layout analysis to identify text blocks and images in HTML emails. OCR engines may or may not use similar techniques, so that the message text may be produced in a different order. Token-based measures take only exact token matches into account, which is inappropriate as OCR engines produce a significant amount of errors.

4.1 Three Measures for Change Detection

We propose three measures to compare the simulated perceived text and the true message text of an email:

1. Length: The difference in normalized text length. We normalize whitespaces and simply calculate the difference between the two lengths. This is simple and robust with respect to text order changes and OCR errors.
2. TOC: The tolerant overlap coefficient, a combination of token-based and edit-distance like measures. We use an edit-distance to match tokens with some degree of freedom, then use a token-based measure for overall text distance.
3. Complexity: Difference of "information content": The Kolmogorov complexity is a measure of the computational resources needed to specify the string. We use Lempel-Ziv compression for its approximation.

In more detail, the tolerant overlap coefficient TOC is calculated as follows:

1. Tokenize both texts, eliminate short tokens. In our setting, we eliminate tokens consisting of less than 4 characters.

2. Compute the number of tokens that appear in both texts. A token x_i appears in the other text, if a token y_j in the other text can be found, so that the editing distance (Levenshtein) between x_i and y_j is below some threshold θ_{x_i,y_j}. In our experiments, this threshold depends on the token's length and is at least 2, i.e., $\theta_{x_i,y_j} = \max(2, \alpha \max(|x_i|, |y_i|))$, where $\alpha = 0.1$. The parameter α defines the tolerance level relative to the token's length.

3. The similarity of the texts is the number of common tokens divided by the number of tokens in the longer text.

For the third measure COMPLEXITY we rely on an information theoretic motivated approach. The motivation behind is that the amount of information in a string or the complexity of a string is robust with respect to distortion. The introduction of noise does not change the information contained within a string. The Lempel-Ziv coding is a scheme for the universal compression of data which can be used to describe the complexity of a string in terms of contained information [13]. It is basically the count of all contained identical sub-phrases in a string, which can be efficiently computed [12]. Although developed for the compression this coding asymptotically approximates the entropy rate of the unknown source of the string, which can be used as a complexity measure.

Let $c(s)$ denote the Lempel-Ziv coding count for a binary string s, then $l_c(s) = c(s) * (\log_2(c(s) + 1)$ gives the length of the compressed string. Let $l_u(s)$ denote the length of the uncompressed string s, then $r(s) = l_c(s)/l_u(s)$ gives the compression ratio for s. For two strings s_1 and s_2 we can now define the complexity-based distance measure as the relative difference of the two compression ratios:

$$\frac{|r(s_1) - r(s_2)|}{\max(r(s_1), r(s_2))}$$

4.2 An Example

As an example consider the email depicted in Figure 1. On the left you see the image of the rendered email and on the right its HTML source. This email contains font color tricks implemented by the two occurrences of `<font color="ffffff">drywall</font>`. Figure 2 shows both the simulated perceived text as generated by our hidden salting simulation system and the true message text as the OCR text of the rendered email. Note that the detection of font color tricks had been disabled in the salting simulation, otherwise the two occurrences of `drywall` would not appear in the simulated perceived text.

The example illustrates the following scenario. Suppose we were unable to detect font color tricks in our hidden salting simulation system. Then it is our goal to mark this email as suspicious, because the two texts differ significantly. Our features produce the following values for this email: LENGTH $= -13$, TOC $= 0.875$, and COMPLEXITY $= 0.026$. The negative value for LENGTH is an indication that the simulated perceived text contains more text than it should, i.e., that in reality some characters are hidden by a trick not yet covered.

5 Experimental Design and Evaluation Criteria

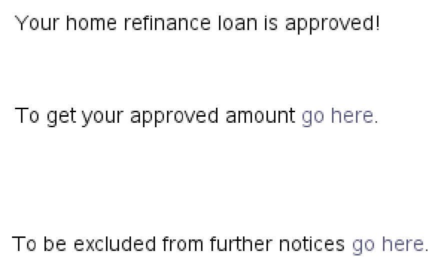
In our experiments we simulate the detection of a new salting trick by disabling the detection of one of the tricks implemented in our salting simulation system. As described in Section 3 the system can detect the following tricks: font size trick, font color trick, concealment, and clipping.

The simulated perceived text is generated by our hidden salting simulation system. We obtain the true message text of an email by rendering it using standard Java rendering and then OCRing the rendered email using the publicly available OCR engines gocr [1] and ocrad [2]. For the latter tool we set the option `-s 5` for scaling the image to obtain better results.

As a classifier, we train a one-class SVM [10]. The setting is as follows. The training set contains only emails, which do not contain the disabled trick. They represent the "one class" of so far "normal" emails, i.e., emails that contain no or only known salting tricks. The test set then contains both emails with and without the disabled trick. The classifier marks emails as outliers, which indicates that they are not in the same class as all the emails from the training set. In other words, these emails are "unnormal" or suspicious, which indicates that they may contain a previously unseen salting trick.

In reality, the classifier produces some real-valued output, where a large value indicates that an email belongs to the one class and a small value indicates that an email is an outlier. Hence, we need a threshold that lets us mark an email as outlier. It would be undesirable to fix this threshold manually as it can vary for different tricks. Usually, classification parameters are estimated using a separate validation set. In our setting we do not have data labeled as outlier in advance. Thus, we need to apply the trained model to the training set to estimate the threshold:

[1] http://jocr.sourceforge.net/
[2] http://www.gnu.org/software/ocrad/



Your home refinance loan is approved!

To get your approved amount go here.

To be excluded from further notices go here.

```
<html>
<body>
<font color="ffffff">drywall</font>
<p>Your home refinance loan is approved!<br></p><br>
<p>To get your approved amount
<a href="http://www.mortgagepower3.com/">go here</a>.</p>
<br><br><br>
<p>To be excluded from further notices
<a href="http://www.mortgagepower3.com/remove.html">go here</a>.</p>
<font color="ffffff">drywall</font>
</body>
<font color="ffffff">1gate
</html>
5297gdqK6-498jyxl3033RafD3-195RTcz6485obQU9-615LOLg9149
```

Figure 1: An example email together with its HTML source text

```
drywall
Your home refinance loan is approved!
To get your approved amount go here.
To be excluded from further notices go here.
drywall
```

```
Your _ome refinance loan is approve_!
To get your approve_ amount _o_o _ere.
To De exclu_e_ from furt_er notices __o _ere.
```

Figure 2: Simulated perceived and true message text for the example email

1. After training the classifier we apply the trained classifier to all emails in the training set and record the real-valued outputs it produces.

2. We calculate the inter-quartile range IQR of the real valued outputs and subtract it from the first quartile value Q_1 to obtain the threshold θ: $\theta = Q_1 - \beta IRQ$, where we set $\beta = 2.0$. A larger value for β means that the cutting threshold will be smaller, i.e., that fewer email messages will be considered as outlier.

In our experiments a classification of *true* indicates that the email is of the same type as the emails in the training set whereas *false* indicates that the email is an outlier and thus potentially contains a new hidden salting trick. For the evaluation we report the false positive rate, the false negative rate and the F2-measure with respect to the outlier class. The false positive rate indicates how many outliers were missed whereas the false negative rate indicates how many regular emails were considered to be outliers. In our scenario a false positive might be considered a more severe error than a false negative. The F2-measure with respect to the outlier class combines precision and recall of the outliers, i.e., the emails classified as false. In accordance with the above observation the F2-measure weights outlier recall twice as high as outlier precision.

For a more detailed analysis we also show the ROC graphs. The ROC graph has the false positive rate on the x-axis and the true positive rate on the y-axis. One classifier only defines one point in this ROC space. However, by considering all meaningful cutting thresholds between the positive and the negative class we obtain a set of classifiers and hence a complete graph. The ROC graph illustrates that one can trade a lower false positive rate for a higher false negative rate and vice versa. An attractive property of ROC graphs is that they are insensitive to changes in class distribution, i.e., to the proportion of positive and negative examples.

6 Data and Empirical Results

To evaluate our system for detecting new salting tricks we compile corpora from publicly available sources. We use the SpamAssassin corpus and the phishing email corpora compiled by Nazario as our primary sources. SpamAssassin provides us with a total of 6951 ham messages and 2154 spam messages, and Nazario provides us with a total of 4559 phishing messages compiled during different time periods. Table 1 summarizes our data together with a statistics about how many email messages contain a certain salting trick as detected by our hidden salting simulation system. Note that the "Any trick"-row is not equal to the sum of the others as some emails contain more than one trick.

	Ham	Spam	Phishing	Total
Full corpus	6951	2154	4559	13664
Font color	6	91	430	527
Font size	40	376	954	1370
Clipping	0	0	1	1
Concealment	1	17	0	18
Any trick	43	422	1314	1779

Table 1: Summary of the available email data

For both the font size and the font color trick we created a specific corpus in the following manner. For training, we randomly selected 800 messages that do not contain the respective trick. For testing, we randomly selected 100 messages that do and 300 messages that do not contain the respective trick. The random selection did not take into account whether the emails came from the ham, the spam, or the phishing subcorpus. Unfortunately, the other tricks appear not frequently enough to design meaningful experiments for their detection. Of course, it would be the ultimate challenge to detect the one single new email that would contain an unknown trick.

6.1 Feature Processing and Parameter Settings

Our features produce values of a very different scale. The LENGTH-feature can, for outliers, easily have values in the hundreds or even thousands. On the other hand, the TOC-feature is always between 0 and 1 as is the COMPLEXITY-feature. Hence, we scale the values of the LENGTH-feature to the interval $[-1, 1]$, where all values above 1000 are transformed to 1 and all values below -1000 to -1.

As one-class SVM implementation we use the libSVM library [5]. We fix the parameters of the SVM to the following setting: RBF kernel, $\nu = 0.5$, $\gamma = 0.1$. We fix the parameter of our automatic threshold detection to $\beta = 2$.

6.2 Detection of the individual hidden salting tricks

Table 2 shows the results we obtain for the detection of two salting tricks, font color and font size.

Trick	OCR	FPR	FNR	F2
Font color	gocr	7.53%	17.85%	84.15%
Font size	gocr	9.18%	10.10%	87.08%
Font color	ocrad	20.43%	11.41%	77.08%
Font size	ocrad	30.61%	10.85%	69.11%

Table 2: Results for salting trick detection

The table suggests that we achieve better results using the gocr engine than using the ocrad engine. Though the evaluation in f-measure does not differ very much we can observe significant differences in the false positive and the false negative rate. As we pointed out earlier, the false positive rate is of particular importance. Figure 3 depicts ROC graphs for both tricks and both OCR engines, and supports the claim that the results for the gocr engine are better. Currently we are setting up a commercial OCR engine, which promises much better character recognition results.

Figure 3 also lets us reason about the optimal operating point and whether our automatic threshold detection comes close to finding it. If we consider the closest

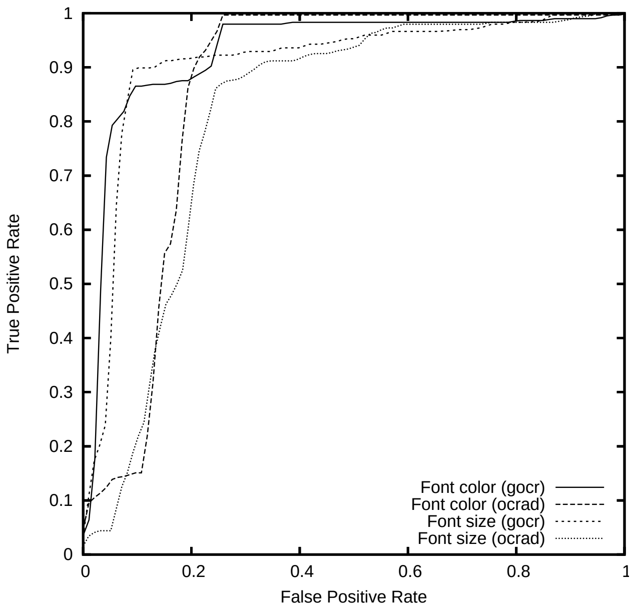


Figure 3: ROC graphs for two tricks and two OCR engines

point to (0,1) as the optimal operating point then we can observe that for font color and gocr the optimal operating point is at a false positive rate of about 9% and a true positive rate of about 86% (which corresponds to a false positive rate of about 14%). If we compare that to the result as depicted in Table 2 then we can observe that we come close to that point. For the font size trick the optimal operating point is at a false positive rate of about 9% and a true positive rate of about 90%, and the result in Table 2 is indeed very close to that point.

On the other hand, one can define the optimal operating point in a cost-based manner, i.e., in our scenario by penalizing false positives higher than false negatives. That would lead to larger values for θ or, equivalently, smaller values for β as defined in Section 5.

When we look at the misclassified emails we can observe two things. False positives are typically messages that contain very few tricks relative to the length of the message text, i.e., very few characters are salted. This is the price that we have to pay for the robustness of our system. It is a small price though, because such an email would then only contain very few salted "good words" to confuse a traditional email filter. Second, false negatives usually occur due to OCR errors. If the OCR quality is too bad then our features will indicate that the two compared texts are different. One particular observation here is that the OCR often misses horizontal lines explicitely written out using the '-'- or the '_'-character in text emails.

Another potential source of errors comes from the correlation of salting tricks. In some emails some text is not only written in a tiny size but also with low color visibility at the same time. That could lead to false positives. For example, in our experiments the hidden salting simulation system should miss the font color trick for a character (if that detection is disabled) and include it in the simulated perceived text, but fails to do so, because it recognizes the font size trick for the same character. Then the email that should be an outlier is classified as true. It remains largely a philosophical question whether that is a problem. In the extreme case, if a new unknown salting trick correlates 100% with an already known trick, we could of course never detect it as such, but still the hidden salting simulation system would always produce the correct simulated perceived text. In our original corpus of 13664 emails there are 123 emails that contain both font color and font size tricks.

6.3 Results for the different distance measures

In this subsection we look into the individual distance metrics proposed in Section 4. Table 3 summarizes the classification results for both salting tricks and each individual distance metrics.

Trick	Feature	FPR	FNR	F2
Font color	all	7.53%	17.85%	84.15%
Font color	LENGTH	10.75%	15.49%	82.83%
Font color	TOC	6.45%	16.16%	85.80%
Font color	COMPL.	8.60%	10.77%	86.91%
Font size	all	9.18%	10.10%	87.08%
Font size	LENGTH	48.98%	11.45%	52.52%
Font size	TOC	11.22%	9.09%	85.97%
Font size	COMPL.	46.94%	5.72%	56.40%

Table 3: Results for salting trick detection

We can observe that the TOC-feature is clearly the most useful one. For the font color trick the classifier based on this feature alone even outperforms the classifier based on all features. On the other hand, the LENGTH and COMPLEXITY features are useful for the font color trick detection but not at all useful for the font size trick detection.

One could suspect a bad cutting threshold there, but looking into more detail confirms the initial impression. Figures 5 and 4 depict the ROC graphs for the font size and the font color tricks. They include the graph based on all features as well as the graphs based on each individual feature.

Again, the graphs clearly indicate that the TOC-

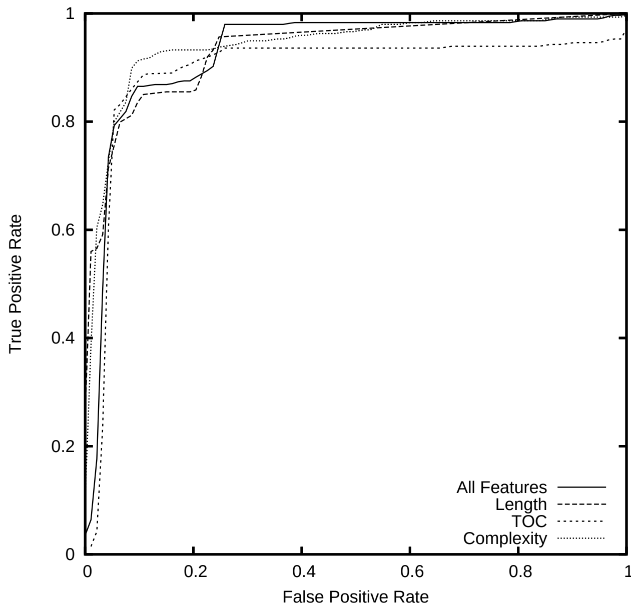


Figure 4: Detailed ROC graph for font color trick

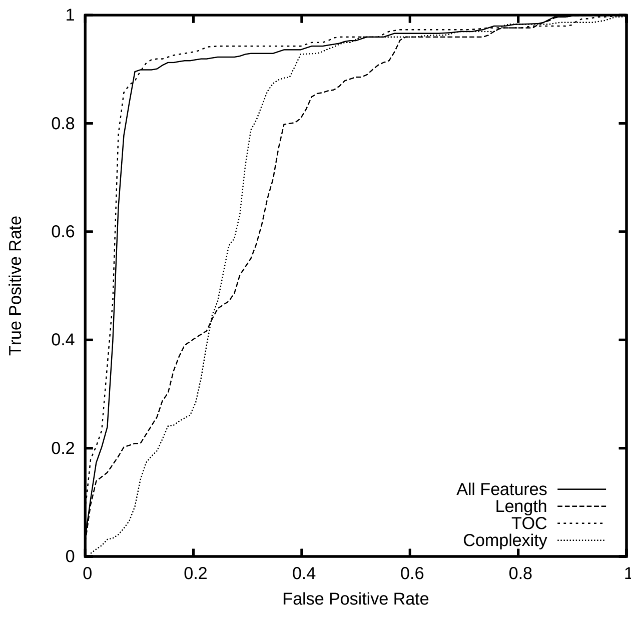


Figure 5: Detailed ROC graph for font size trick

feature that combines token-based and edit-distance measures contributes most to the overall performance.

We looked again into the details and found that indeed for the font size trick the LENGTH- and COMPLEXITY-feature have larger values for the training set data where in fact those values should be close to zero. Hence, the trained classifier model is of a lower quality in this case. Further analysis will be necessary here.

7 Conclusion

In this paper we presented a framework for detecting both known and new salting tricks in email messages. Our previously developed hidden salting simulation system detects the presence of a number of known tricks employed by spammers and generates a simulated message text as perceived by the user. Here we developed a robust mechanism to detect new tricks not yet covered. To this end, we compare this simulated perceived text to the true message text as obtained by applying OCR to the rendered email and feed features based on this comparison into an outlier detection classifier. In simulation experiments we show that tricks can be detected with a high level of accuracy.

There are a number of technical issues that could when resolved further improve our system. First and foremost, a better OCR engine, such as the commercial Abbyy Fine Reader engine, will most definitely be beneficial. We also observed that different OCR engines tend to make different kinds of mistakes systematically. For example, gocr tends to split tokens whereas Tesseract [3] tends to merge them. Secondly, we employ some OCR preprocessing, such as binarization, that could be improved further. Thirdly, segmentation and reading order are significant challenges both for our salting detection system and the OCR of the rendered email. Any breakthrough in these areas will most certainly benefit our system.

8 Acknowledgments

This paper is based upon work performed within the FP6-027600 project AntiPhish (http://www.antiphishresearch.org/). The authors would like to thank the European Commission for partially funding the AntiPhish project as well as all the AntiPhish project partners for their interest, support, and collaboration in this initiative.

[3]http://code.google.com/p/tesseract-ocr/

References

[1] F. Assis, W. Yerazunis, C. Siefkes, and S. Chhabra. Crm114 versus Mr. X: CRM114 notes for the trec 2005 spam track. In *Proceedings of the Text Retrieval Conference (TREC)*, 2005.

[2] J. D. Beer and M.-F. Moens. Challenging hidden text salting in digital media. Submitted for publication.

[3] T. M. Breuel and D. Keysers. Round-trip html rendering and analysis for testing, indexing, and security. In *7th IAPR Workshop on Document Analysis Systems (DAS)*, Nelson, New Zealand, February 2006. Extended abstract.

[4] W. B. Cavnar and J. M. Trenkle. N-gram-based text categorization. In *Proceedings of the Annual Symposium on Document Analysis and Information Retrieval (SDAIR)*, pages 161–175, Las Vegas, US, 1994.

[5] C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at http://www.csie.ntu.edu.tw/˜cjlin/libsvm.

[6] W. W. Cohen, P. Ravikumar, and S. E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *Proceedings of the Workshop on Information Integration on the Web (IIWeb)*, pages 73–78, Acapulco, Mexico, August 2003.

[7] G. Fumera, I. Pillai, and F. Roli. Spam filtering based on the analysis of text information embedded into images. *Journal of Machine Learning Research*, 7:2699–2720, 2006.

[8] E. Kirda and C. Krügel. Protecting users against phishing attacks. *Computer Journal*, 49(5):554–561, 2006.

[9] D. Lowd and C. Meek. Good word attacks on statistical spam filters. In *Proceedings of the Conference on Email and Anti-Spam (CEAS)*, Stanford, CA, USA, July 2005.

[10] B. Schölkopf, J. C. Platt, J. C. Shawe-Taylor, A. J. Smola, and R. C. Williamson. Estimating the support of a high-dimensional distribution. *Neural Computing*, 13(7):1443–1471, 2001.

[11] K. Taghva, R. Beckley, and J. S. Coombs. The effects of ocr error on the extraction of private information. In H. Bunke and A. L. Spitz, editors, *Document Analysis Systems*, volume 3872 of *Lecture Notes in Computer Science*, pages 348–357. Springer, 2006.

[12] T. A. Welch. A technique for high-performance data compression. *Computer*, 17(6):8–19, 1984.

[13] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5):530–536, 1978.