DeTypo

Detection, Visualization and Prevention of Social Engineering Attacks on E-Mails by Using Machine Learning Techniques
Bachelor-Thesis von Heinrich-Alexander Engels aus Darmstadt
Juli 2012

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Fachbereich Informatik
Fachgebiet Sicherheit in der
Informationstechnik

CASED

DeTypo
Detection, Visualization and Prevention of Social Engineering Attacks on E-Mails by Using Machine Learning Techniques

Vorgelegte Bachelor-Thesis von Heinrich-Alexander Engels aus Darmstadt

Prüfer: Prof. Dr. Michael Waidner
Betreuer: Marco Ghiglieri

Tag der Einreichung:

Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 23. Juli 2012

(Heinrich-Alexander Engels)

Abstract

E-mail driven communication opens up new opportunities for social engineering attacks like the ‘doppelganger mail attack’. Our goal is to lessen the impact of such attacks by employing machine learning techniques. In order to do so we determine if incoming mail by unknown addresses can be matched to other known contacts and thereby verify if an impersonation attack is being carried out. At the same time doppelganger mail will not work on our setup, since they will be regarded as unknown e-mails.
For that purpose, we develop an extension for the e-mail client Thunderbird and test several scenarios, showing that our approach can successfully counter most doppelganger mail based social engineering attacks. Although our approach is successful, there are still some attacks which cannot be detected. We highlight these attacks in our work and propose ways of detecting them in future implementations.

Die Kommunikation über E-Mail eröffnet Angreifern neue Wege für Social Engineering Angriffe, wie zum Beispiel dem ‚Doppelgänger Mail Angriff‘. Ziel dieser Arbeit ist es, die Auswirkungen solcher Angriffe unter Einsatz von maschinellen Lerntechniken abzuschwächen. Um dies zu erreichen, bestimmen wir, ob unbekannt eingehende E-Mails zu bekannten Kontakten passen und verifizieren darüber, ob ein Nachahmungsangriff auf uns durchgeführt wurde. Gleichzeitig wird der Doppelgänger Mail Angriff verhindert, da die Doppelgänger E-Mail-Adresse als unbekannt eingestuft wird. Um dies umzusetzen haben wir eine Erweiterung für den E-Mail Client Thunderbird entwickelt und mehrere Testszenarien durchgeführt. Unser Ansatz kann die meisten Formen von Social Engineering Angriffe die auf Doppelgänger-Emails basieren erfolgreich abwehren. Es bestehen dennoch weitere Angriffsformen, die unser System noch nicht entdecken kann. Wir weisen in unserer Arbeit auf diese Lücken im System hin und zeigen Perspektiven auf, wie diese zukünftig ebenfalls geschlossen werden können.

Table of Contents

1. Introduction

Phishing attacks on the Internet via e-mail are an important issue, especially as of today. This work tries to identify and mitigate or even avoid certain phishing attacks with special emphasis on social engineering attacks.

1.1. Motivation

In modern times e-mail communication has reached great importance as a means of communication. It is not only used for private communication, but also for conversations within businesses. In 2012 the average daily global e-mail volume exceeds 100 billion messages and can reach numbers of 150 billion. This number might seem very high at first, but it is important to note that spam messages account for 85% of the e-mail volume [Cis12]. Regarding this background, users request that e-mails can be trusted and reliably assigned to corresponding contacts.

The main problem consists of attacks where the user cannot determine who the sender of the message is and therefore is not able to react accordingly. If, for example, two contacts have very similar e-mail addresses the recipient could happen to confuse them. If an attacker intentionally creates a very similar e-mail address, we speak of a ‘doppelganger mail address’. Employing such an e-mail address the assailant can then carry out a so called ‘doppelganger mail attack’. Doppelganger mail attacks are social engineering attacks: The assailant tries to imitate the e-mail address of a person known to the receiver in hopes of being met with unearned trust. This trust can then be misused to retrieve passwords and sensible information or infiltrate the system by sending malware. This poses a threat to companies, identical to the insider threat. Enterprises can be damaged considerably if insider knowledge is leaked. A gathering of insider threat damages can be found in the report by Marissa et al. [Mar05]. Other attacks include simple impersonation attacks, by name or by a statement in the mail body. An example of a doppelganger mail attack can be seen in Figure 1.1, where an assailant tries to imitate the e-mail address of Dieter Berg by substituting the ‘e’ in Berg for an ‘o’. Without highlighting the difference is very hard to detect, even if the e-mail address is displayed in the client. This happens because humans read words in parallel instead of sequentially. This means that we read words as a whole rather than by single characters [Mah11].

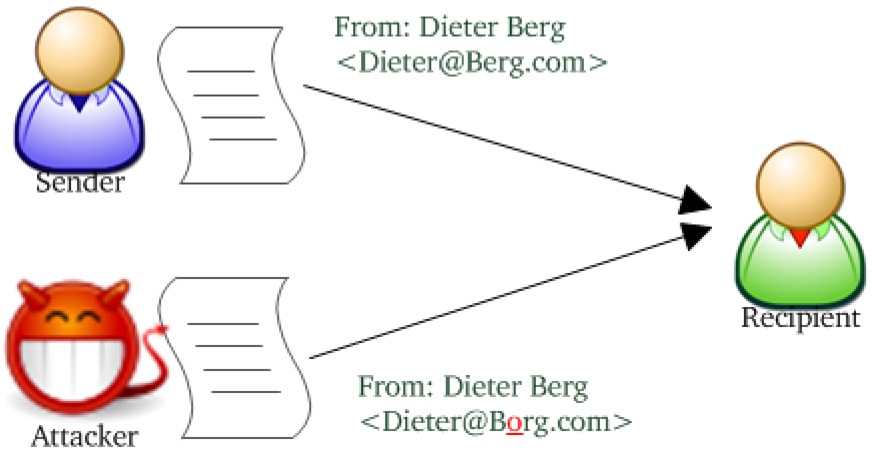


Figure 1.1 Example of a doppelganger mail attack

The main target of our research is to avoid doppelganger mail attacks. At the time of this writing no attempt of detecting doppelganger mail is known to us, while potential countermeasures against these attacks already exist, but are not widely used. They require that trusted third parties prove that the communication partner is indeed trustworthy. This is incorporated in the idea of digital signatures, the 'Pretty Good Privacy' (PGP) software[1] and 'Web of Trust' applications in general. PGP offers digital signatures and encryption on a Web of Trust basis, which means that the certification of digital signatures happens on a trust chain basis. A trust chain is a path of trusted contacts: from a user to his trusted contacts, through their trusted contacts, up to the new communication partner [Abd96].

As a part of this thesis we develop an extension for the popular e-mail client Thunderbird[2], which is named DeTypo. The name DeTypo is derived from 'Detect Typo squatting'. Its purpose is to raise the awareness of the user upon incoming mail from people, with whom he/she never interacted. The term 'typo squatting' denotes a practice of slight modification of resource identifiers, such as URLs and E-mail addresses, to confuse the victim into trusting the authenticity of the source. It is a more general term for 'doppelganger mail addresses'.

We do not intend to detect hacked accounts and therefore do not filter incoming messages for spam or check external content for integrity. We also do not try to avoid man-in-the-mailbox scenarios, but offer possible solutions for these at the end of this writing. Man-in-the-mailbox scenarios are such cases where the attacker controls the e-mail account of a trusted party and can send messages from that e-mail address [Ada10]. Alternatively the attacker establishes contact with two victims employing doppelganger mail or other authentic appearing mail. He then forwards messages between both victims to fool them into believing that they indeed are communicating with each other. This allows the attacker to monitor all communication and to interfere at some point. This scenario, although possible, is not very likely to happen and even harder to detect, since it is a very specialized attack.

Another motivation in this work derives from the assumption that there is no authentication of senders in mail communication. This does not always hold true, because sophisticated measures such as PGP and digital signatures do exist, but are not widely distributed. This is caused by high costs of digital signatures and the great complexity of measures such as PGP. In addition, social users often do not request heightened privacy. The only authentication we assume is the integrity of the domain and of the e-mail sender. Spoofing of the 'from' header of an e-mail is easily detectable by looking up the IP-address of the domain and comparing it to the sender's IP-address. We therefore propose combining our work with a sender policy framework, if the integrity of the 'from' header is in question. In modern E-mail clients this should pose no problem, since spoofed 'from' headers are an obvious criterion for classification of spam or phishing attacks.

1.2. Related work

While no work concerning doppelganger mail is known to us, there are approaches to authenticate message content as not harmful.

The first means were developed as far back as 1998. Sahami et al. were the first to propose classification of messages based on the content of the message itself. They developed the Bayesian filter which classifies messages based on words contained. The likelihood of words being in spam messages is derived from manual classifications the user made before [Sah98]. They also suggested classification based on other features, such as interactions with the sender or at sending time. This attempt found wide reception in practical use and is utilized in a lot of modern anti-spam softwares and e-mail clients. Examples include "Spam Assassin" [Spa12] and "Mozilla Thunderbird" [Thu12].

[1] http://www.pgpi.org/

[2] www.mozilla.org/de/thunderbird/

Doppelganger mail attacks belong to the field of spear phishing attacks, because the assailant knows at least whom the victim would trust. Spear phishing attacks are characterized as targeted attacks where the attacker incorporates knowledge about the victim, like names, social connections and job position into his attack [Mah11]. They are harder to detect by Bayesian filtering, because the attacks are not generic and often address the receiver with targeted concerns rather than spam or advertisements. That is why more recent studies are more closely related to our field of spear phishing attacks, as assailants employ more sophisticated measures.

In order to mitigate spear phishing attacks Mahmoud et al. proposed the “Anti-Spear phishing Content-based Authorship Identification” framework (ASCAI) [Mah11]. ASCAI is used to authenticate the author of an E-mail based on his writing style. We will incorporate this idea in our own work for determining possible senders of e-mails from unknown contacts. ASCAI uses the Naïve Bayes algorithm to classify a message based on n-grams. N-grams are n characters long substrings of the message body. The success rate of the implementation Mahmoud et al. developed exceeds 80% for n-grams of length 7 to 10, showing that this approach is very promising even when the data are noisy.

The authors also proposed three different classification strategies. The first is the non-greedy approach where at most L n-grams are collected and, if the last position was tied so that the size would exceed L, all entries tying for last were excluded. The second is the greedy variant, where all the tying n-grams would also be included. The third is the strict variant, where exactly L n-grams are used and in case of a tie random decisions are met. They found that the strict classification performed very poorly while greedy and non-greedy classifications performed about the same with greedy being slightly better. The results are also highly dependent on a suitable choice for L, which is usually lower for the greedy variant. Since we will not use n-grams in our implementation, we decided to renounce an upper bound for learned features.

Similar to ASCAI the Thunderbird extension MailClassifier [Ema12] employs the Naïve Bayes algorithm in a multi-level Bayesian filter. However, instead of classifying e-mails by senders this extension aims to classify messages to folders which have different topics.

An older add-on for Thunderbird called “Sender Verification Anti-Phishing Extension” [Jos12] helps to detect spoofed ‘from’ headers in e-mails. The main idea is now introduced in most e-mail clients, so the add-on is not necessary anymore. The detection of spoofed headers is accomplished via the Sender Policy Framework (SPF) which, by using an entry in the DNS-zone, verifies whether a computer was allowed to send a message under a certain top level domain. These entries do not necessarily exist, but if they do, detecting unauthorized computers is fast and accurate. Additionally, the add-on provides reputation information on domains.

A related work from the sender side is the recipient recommendation by Carvalho et al. [Car07]. Based on their work and with help from Balasubramanyan, they developed the Thunderbird extension “Cut Once” [Bal]. Cut Once tries to detect information leakage, for example by sending sensible information to a wrong recipient. At the same time Cut Once recommends recipients based on the e-mail content and who else is receiving the mail. In their work Carvalho et al. define several classification models and find that their “k-nearest neighbors model” is best suited to the task. The k-nearest neighbors model finds the k most similar e-mails, which the user sent before composing the new one and compares already chosen recipients with the recipients in old e-mails. This approach can still be improved with combined classification strategies.

1.3. Outline of the Thesis

This paper starts out with the theoretical basics of the developed extension DeTypo. It explains the mechanics of receiving and sending e-mails when DeTypo is active. We will then present a description of the e-mail client Thunderbird followed by explanations of the Naïve Bayes classification algorithm and the Levenshtein word distance algorithm.

After building a theoretical basis for the user, we elaborate on the implementation of the extension DeTypo in greater detail. In that section, we explain the structure of the underlying database, receiving and sending messages and information on how to raise user's awareness of potential social engineering attacks. At the end, we address the GUI and usability concerns.

In the next chapter, we evaluate DeTypo's performance and show whether set goals and functionality are achieved. We therefore first describe a test environment and then document our findings. We also interpret these results to better understand if goals were met and find new implications for our work. These include improvements and new approaches for future work. Since the main focus of this work lies on the implementation of DeTypo, the evaluation is not aiming to be statistically significant. However, the evaluation will be a good indication to determine, whether or not DeTypo is in a state where it can be used for further research. We show a clear trend of DeTypo being successful in the way it should work and also work out its weaknesses.

In the last chapter we review our findings and interpretations and draw a conclusion. Based on this conclusion, we suggest research areas for future work. These research areas can help improve the performance of DeTypo and further on show that DeTypo is a needed and a working concept.

2. Theoretical basics

For our work we extend the e-mail client Thunderbird, which we describe in chapter 2.1, with an extension which we call DeTypo. Our approach at fighting doppelganger mail attacks consists of creating a database of known contacts and assigning trust values to each. Another functionality we want to provide with the database is a black and a white list. The list allows assigning a trust value directly to a domain, so that new incoming contacts are classified automatically corresponding to their domain's trust value. A contact can either be trusted, untrusted or neutral. Figure 2.1 explains the foundation of DeTypo. When e-mails come in, we check whether or not the contact is already in our database and if not apply a trust test on him. The results will then be shown to the user in an information dialogue. If the contact is already in the database the incoming message is color coded according to the trust value.

The test consists of three sub-processes. First the name is checked against possible entries in the database. If matching names are found, the corresponding contact is displayed in a 'similar names' box. Similarity of names is determined by the Levenshtein algorithm as described in chapter 2.3. Accordingly in the second test the e-mail address is compared to existing ones and on resemblance a warning is issued in another part of the information dialogue, because we most likely detected a doppelganger mail. The last part compares the writing style of the message to that of known contacts and similar writers are suggested as senders of the message. For writing style comparison the Naïve Bayes algorithm as explained in chapter 2.2 is used.

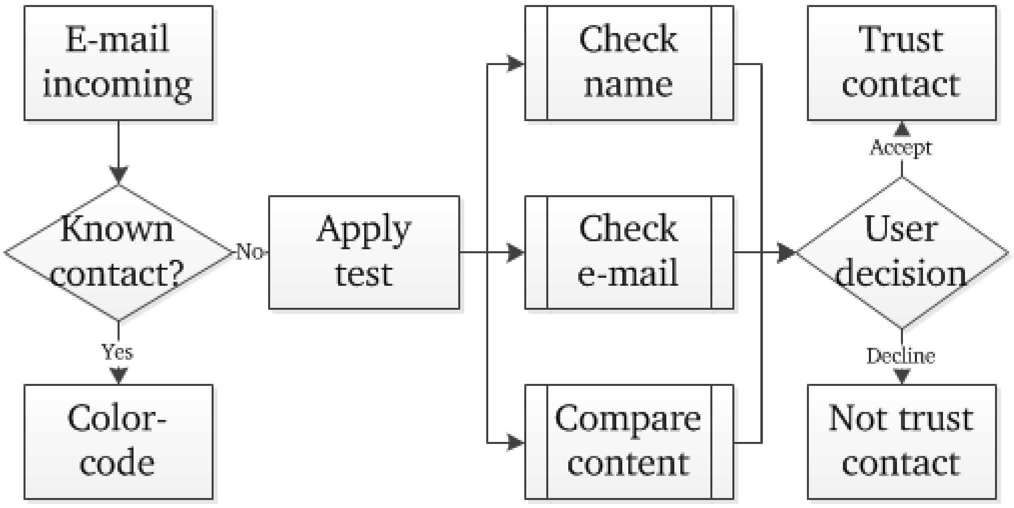


Figure 2.1 Theoretical basis of DeTypo - Receiving

So far we only looked at the receiving side of e-mails. Another part of the problem lies with the sending side. Figure 2.2 illustrates how DeTypo intercepts sending e-mails. First we check that all addressed contacts are known to us. If someone is unknown we offer the user to save the contact as a trusted contact for the future. Next we check whether the user sends e-mails to untrusted contacts. In If he does, we want to warn the user of potential information loss, for example, because he chose the wrong e-mail address when suggested an e-mail and its counterpart doppelganger mail. In that case the user can still proceed in sending the e-mail by confirming a corresponding dialogue option.

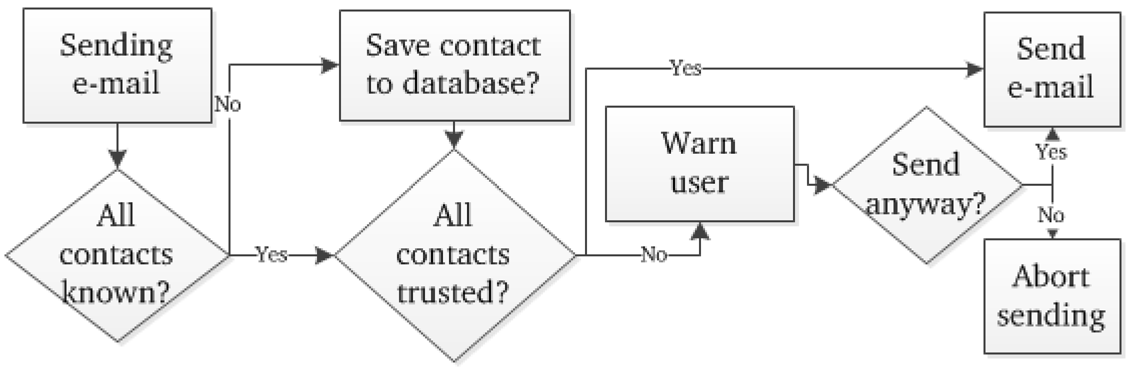


Figure 2.2 Theoretical basis of DeTypo - Sending

To achieve a broader understanding of how our extension works one must first look into the theoretical basics behind the used algorithms and how to implement them. In the following we discuss the e-mail client Thunderbird and how to extend it, the Naïve Bayes classifier and the Levenshtein algorithm which was used and adopted as explained in chapter 3.2 to make string comparison possible.

2.1. Thunderbird

Thunderbird is a free, open-source desktop e-mail client. It is available for many platforms including Microsoft Windows, UNIX, Linux distributions and Mac OSX. It was developed by the Mozilla Foundation and can be downloaded at *http://www.mozilla.org/de/thunderbird*. It is licensed under Mozilla's own software license the MPL[3] combined with other licenses such as GPL[4] and LGPL[5] [Moz12].

Thunderbird's standard functionality includes checking e-mails, sorting them into folders, spam detection and grouping of related messages such as ongoing conversations. Apart from e-mails, there is also support for RSS and other feeds. Other features include searching through one's e-mails and feeds, saving messages locally, encryption and an address book. For more capabilities the user can download add-ons which either manipulate the appearance or enhance the functionality. The first are called themes while the latter are called extensions. Also available are plug-ins which are add-ons written in platform specific code, which interact with Thunderbird only through its external interfaces. Some representatives of such plug-ins are video codecs, Flash and Java.

Thunderbird is closely related to Firefox, which is a web browser also developed by the Mozilla Foundation. They share a lot of technologies and are extensible in a similar fashion. In the following, we describe how to extend Thunderbird. Interested readers will find additional information in the Mozilla Developer Network[6].

Thunderbird runs on the base of a XUL-runner engine. This means that all user interfaces are defined in the XML User Interface Language[7] (XUL). XUL is similar to dynamic HTML and even incorporates many of its capabilities. Wrapped around that is the C++ code which provides even more interfaces. Extensions can interact with both the XUL-runner interfaces and the Thunderbird specific code via JavaScript.

[3] http://www.mozilla.org/MPL/2.0/
[4] http://www.gnu.de/documents/gpl.de.html
[5] http://www.gnu.de/documents/lgpl-3.0.de.html
[6] https://developer.mozilla.org/en/Extensions/Thunderbird
[7] https://developer.mozilla.org/En/XUL

To create an extension one must have a specific folder structure. On the top level the files install.rdf and chrome.manifest have to be available. Install.rdf tells Thunderbird the name of the add-on, which type of add-on it is dealing with and other add-on specific data. The chrome.manifest specifies where the custom content can be found and which native files should be overlayed. The rest of the folder can contain other folders which point to content such as code, custom XUL-Files, localization files or style sheets. The XUL-runner engine allows you to overlay the XUL files used for the display of Thunderbird. This means that the XUL elements in the overlay file either overwrite existing elements or add elements of their own at a specified entry point. One can bind scripts to these overlays to enhance functionality or modify the appearance of the interface. It is also possible to implement style sheets[8] and have common and Mozilla specific style definitions.

In order to access Thunderbird objects within the JavaScript code, one can utilize the features of the global Components class. It allows access to Thunderbird interfaces, classes and utility functions. The Components class allows casting objects which is useful if native Thunderbird functions return objects of a certain kind. It is also possible to overwrite Thunderbird function definitions, as we do to intercept outgoing messages before they are sent. Lastly JavaScript DOM functions also work on XUL elements and allow for programmatical changes of the user interface and procession of user input.

For persistent storage Thunderbird offers two possibilities. One is preference strings, which should only be used for settings or very small datasets. The other one is build-in SQLite[9] support. The SQLite Database is a file that is put into the user’s profile directory. The database can be addressed with standard SQL queries, which are given to Thunderbird routines.
In addition to Thunderbird standard functions we import functions from the Thunderbird stdlib project, which aims to simplify the native Thunderbird API [std]. For testing, the build-in error console was used along with a command console where custom messages can be output.

2.2. Naïve Bayes Classifier

Classification is an important part of modern day machine learning. A program has to determine if an item belongs to a certain class and can then react appropriate according to a rule set. It is also useful to filter text messages for spam and other applications.

In the following section bold capital letters will denote vectors, indexed small letters stand for elements in a vector and P(*) denotes the probability of *. * can be a placeholder for either a variable or a variable given a condition which follows after a pipe (|). We assume that there is a function g($\mathbf{X}$) which tells us the exact class of a feature vector $\mathbf{X}=(x_1, x_2, \dots, x_n)$. This function g can be deterministic, but does not have to be. Since it does not have to be deterministic the same feature vector can lead to different classes. For example the same e-mail could have been written from more than one e-mail address. The classifier tries to imitate this function g by guessing a function h($\mathbf{X}$). The function h($\mathbf{X}$) however, is deterministic, making it impossible to approximate a non-deterministic function g.

The Naïve Bayes classifier tries to match a class to a feature vector $\mathbf{X}=(x_1, x_2, \dots, x_n)$. The possible classes come from a set domain $\mathbf{C}=(c_1, c_2, \dots, c_m)$. It uses the Bayes Theorem (Formula 2.1) to guess the probabilities of all classes in the domain and pick the most likely one.

$$P(c_i\,|\boldsymbol{X}) = \frac{P(c_i) * P(\boldsymbol{X}|c_i)}{P(\boldsymbol{X})} \tag{2.1}$$

The observant reader may notice here that the denominator does not depend on the class C and can therefore be omitted in practical implementations of the Naïve Bayes classifier. The probabilities for a

[8] http://www.ietf.org/rfc/rfc2318.txt
[9] http://www.sqlite.org/

certain class can still be calculated by normalizing the results. Determining the probability of a class P(c_i) is straight forward by counting the total amount of times a class c_i is observed in past classifications and then divided by the total amount of past classifications. This is why the Naïve Bayes classifier first needs to learn, before it can be used. The second problem the Naïve Bayes classifier needs to calculate is an approximation to $P(\boldsymbol{X}|c_i)$.

Naïve Bayes works under the 'naïve' assumption that all features in **X** are independent of each other in a given class c_i. From that follows that the probability to get a specific feature Vector **X** given class c_i is equal to the product over all probabilities to get a feature x_j given class c_i (Formula 2.2).

$$P(\boldsymbol{X}\,|c_i) = \prod_{j=1}^{n} P\big(x_j|c_i\big) \tag{2.2}$$

This is not applicable if the probabilities for features given class c_i are not provided. One must first carry out a learning process, which provides the occurrences of features in the classes. These are then normalized per class to determine $P\big(\boldsymbol{x_j}\,|\boldsymbol{c_i}\big)$. If a feature $\boldsymbol{x_j}$ is not present in a class, then its probability $P\big(\boldsymbol{x_j}\,|c_i\big)$ would be zero. This is problematic, since in formula 2.2 all probabilities for a feature given class c_i are multiplied, thereby setting the whole probability for **X** containing $\boldsymbol{x_j}$ to zero. This is not desirable, so the probability of $P\big(\boldsymbol{x_j}\,|c_i\big)$ is set to a very small value. One can then use this knowledge to calculate $P(\boldsymbol{X}\,|c_i)$, because the probabilities $P\big(\boldsymbol{x_j}\,|c_i\big)$ are now estimated. Obviously, larger training data will yield better results, but Naïve Bayes can sometimes already provide good results for small sets of data.

We have learned about the two restrictions Naïve Bayes is limited by. The first is based on a deterministic algorithm and the second is ignorance of dependencies among features. The result is possible classification errors. In the case where one regards only two classes or regards one class as the observed one and aggregates the rest as 'other classes' one can speak of false positives and false negatives. False positive means that the algorithm classified an item to the observed class, but in reality that item does not belong to it. On the contrary, a false negative is a rejection of the observed class, which in reality belongs to the item.

The assumption of independent features does not hold true in most cases. However the Naïve Bayes classifier produces good results in many fields such as image recognition and text classification. Rish analyzed the optimality of Naïve Bayes classifiers under certain circumstances and found that Naïve Bayes performed best when g was deterministic or functional dependencies between the features x_i existed [Ris01]. Optimality was proven for a two class concept where only one feature led to class 0 while the others all resulted in classification to class 1. Optimality was also shown for classifications where equal class priors P(c_i) were given and for each feature x_i, i=2, ..., n, a functional dependency $x_i = f_i(x_1)$ existed [Ris01].

To calculate the optimality of the classifications Rish defined the risk of a classification error as P(h(**X**) ≠ g(**X**)). In deterministic concepts optimality was achieved with a zero classification error. In nondeterministic scenarios however optimality was defined as the classification error being as low as the general Bayes error. The general Bayes error is the error induced by the noise in the data. It can be calculated as P(h*(**X**) ≠ g(**X**)), where h* is a classifier which does consider dependencies among features, *id est* works with the covariance matrix. In consideration of this approach we will also evaluate our work by identifying the percentage of misclassifications, but will not try to determine the general Bayes error.

2.3. Levenshtein Word Distance

The classic Levenshtein algorithm makes it possible to determine how many insertions and deletions it takes to transform one string into another. However we also want to consider substitutions of characters, which can be done with the generalized Levenshtein. We will outline the general way in which the generalized Levenshtein algorithm works, but will not go into detail about the related longest common subsequence problem, from which Levenshtein evolved. A good explanation on further details of the algorithm can be found in Dan Hirschbergs “Serial computations of Levenshtein distances”[Dan97].

First we define three transformations on characters, namely insertion, deletion and substitution, and their respective costs $\delta(\lambda,\sigma)$, $\delta(\sigma,\lambda) and\, \delta(\sigma_1,\sigma_2)$. Note that λ is a placeholder for the empty word, while σ denotes a random symbol. In our implementation the cost for each operation was equal and therefore one. We are then given two strings $x = x_1, x_2 \dots x_n$ and $y = y_1, y_2 \dots y_m$.
We can now calculate the generalized Levenshtein distance as follows:

$$D(i,j) = \begin{cases} 0, & if\ i = 0\ and\ j = 0 \\ D(0,j-1) + \delta(\lambda, y_j), & if\ i = 0\ and\ j > 0 \\ D(i-1,0) + \delta(x_i, \lambda), & if\ i > 0\ and\ j = 0 \\ D(i-1,j-1), & if\ x_i = y_j \\ min\begin{Bmatrix} D(i,j-1) + \delta(\lambda, y_j), \\ D(i-1,j) + \delta(x_i,\lambda), \\ D(i-1,j-1) + \delta(x_i, y_j) \end{Bmatrix}, & if\ x_i \neq y_j \end{cases} \tag{2.3}$$

We will work with a distance matrix D, which has the dimensions (n+1)x(m+1). The first row and column are only dependent on the insertion and deletion costs, respectively. We initialize D by setting the first entry (0, 0) to 0. The first row is then formed by adding the cost of one insertion to the preceding entry. The first column is calculated likewise, but instead of insertion costs the cost of deletion is added. Now that D is initialized, we loop through our strings x and y and compare them char-wise. If we encounter the same character, we do not add to the distance since no transformation is needed. If we encounter different characters, we will try to minimize distances by choosing an operation which yields the smallest distance. Possible transformations are adding a character and thereby adding the cost of an insertion operation to the entry right above the new one, deleting a character for the cost of a deletion added to the left entry and lastly editing a character and thus adding the cost of a substitution to the upper left entry. When we are done, the generalized Levenshtein distance will be in field (n, m).

3. Implementation

This chapter deals with the development environment and the structure of our extension which can be summarized as follows:
There are two different times at which the extension is triggered. The first one is upon receiving e-mails and the second one before sending them. Besides these there are GUI elements and a menu which can be accessed at all times. In order to achieve a deeper understanding of what is happening in the background we first have to look at the adaption of the Levenshtein algorithm. Then we also need to inspect the underlying database which from this point on will be referred to as ‘trustbase’. The latter part of the chapter will then deal with the mechanics of the implementation.

3.1. Development Environment

For the development process we set up a new folder and put the required files in it. We choose not to employ the localization features of Thunderbird and hardcode all strings in English. This is not an issue, because apart from the user information screen and dialogues there are only a few text elements and we only work on a proof-of-concept extension. We then set up a Thunderbird client in version 11.0.1 and disable automatic updates. Next, we define a new developer profile in the client and set custom development configuration options. These include disabling caching of windows, enabling dumping into the console, and enabling most error reporting preferences for the error console. We put a pointer file into the extension directory of our developer profile, which points to our extension folder and start Thunderbird from a shortcut. We utilize the –p switch for the profile, the –no-remote switch for avoidance of interference with other instances of Thunderbird on the developing machine and the –console switch for an additional console, where our debug symbols are displayed.

We set up our development folder with four subdirectories, namely content, defaults, modules and skin. The content folder holds all our XUL files and their related JavaScript files. The defaults folder is used to define the standard preferences upon the first launch of the extension. The modules folder holds importable JavaScript modules, which can also be used as a shared resource. This folder also holds the stdlib project. Lastly, the skin folder holds cascading style sheets to define the look of our elements and overwrite the look of some existing ones.

3.2. Adoption of the Levenshtein Algorithm

For our work, we use the JavaScript variant from Wikibooks [Wik]. However, the generalized Levenshtein algorithm is not enough to determine similarity of names, since the length of the strings is not taken into consideration. We therefore implement an ‘isSimilar’ function which utilizes the Levenshtein algorithm and the length of the shorter string. We also want to allow names to only be partially similar, so that people with similar first or last names would match the conditions. This proves beneficial if a contact has a nickname which he only uses on his private e-mail address and an official name, which he uses on other E-mail addresses. The case when the first and last names are switched around is also matched.

The code below assumes that two strings are not dissimilar, unless proven otherwise. In a first step the names are seperated into single words and put in an array. These arrays are then iterated through and each name is checked against the others. Then the core comparison is carried out. We determine the length of the smaller word and divide the Levenshtein distance by that length. The result is checked against a threshold value.

```
isSimilar = function(str1, str2, threshold){
        if(str1 == "" || str2 == "")
                return false;
        let similar = false;
        names1 = str1.split(" ");
         names2 = str2.split(" ");
         //try to combine all parts of the names with each other
        for(var i in names1){
                if(names1[i] == "")
                        continue;
                for(var j in names2){
                        if(names2[j] == "")
                                 continue;
                         let _length = Math.min(names1[i].length, names2[j].length);
                        //threshold will allow for more (higher) or less (smaller) names to be accepted
                        if(Learner.levenstheim(names1[i], names2[j]) / _length  <= threshold)
                                similar = true;
                        }
                }
        return similar;
}
```

For names a threshold value of 0.4 is used, which is equivalent to 2 alterations for every 5 characters. We use this value because many names may be shortened by some characters. We also use this algorithm to check for doppelganger mail and find a threshold value of 0.2 is better suited for this application, since e-mails often share domain names and we only want to detect intentional doppelganger mail. The same domain name means that a large proportion of an address is already similar and therefore an upper bound of 0.4 would be too high. Given that a long domain name is the same for 2 addresses then only the part in front of the '@' symbol would matter for detecting differences. Assuming the name is as long as the domain name then 4 in 5 characters could be different in the names and the mail would on average still be accepted as lower than the 0.4 threshold value. Obviously this is not intended, so a smaller threshold value was utilized. This is not a problem, since doppelganger mail try to imitate the original e-mail address as closely as possible. According to one's needs the threshold can be adjusted to allow for more or less transformation operations by setting it higher or lower.

3.3. Trustbase

For structural purposes, from this point on, we make a distinction between known and unknown contacts, while most logic interaction happens with unknown contacts. Naturally we will have to analyze unknown contacts more closely than known ones, because these are the ones who will possibly try to win our trust and use social engineering attacks on us.

In chapter 2 we already mentioned that our extension would build up a database. That aforementioned database is implemented as the trustbase. The trustbase keeps track of contacts, their e-mail addresses, interactions, names and trust values. Additionally, white list functionality is provided by saving domains and their trust values.

The trustbase is made up of three tables in a SQLite Database and is stored, for performance reasons, in a shared resource called 'Trustcenter' at the start of each session. The Trustcenter is a Thunderbird module, which holds an object in which associative arrays can be stored. The details of these arrays will be explained along with their counterpart tables in the trustbase. All modifications on the

Trustcenter are write-through to the persistent trustbase. The rest of this section covers each table in more detail.

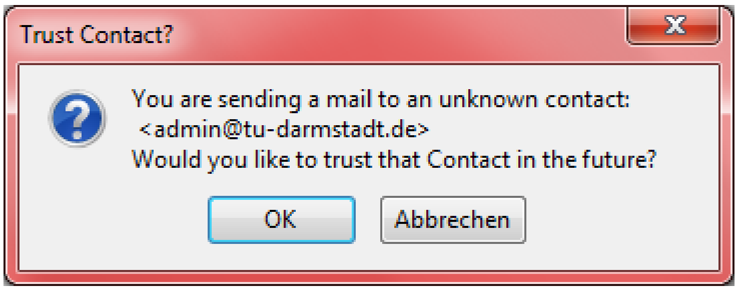
The first table in the trustbase is named 'whitelist'. It keeps track of trusted and untrusted domains by saving the domain's name and a Boolean which indicates whether or not to trust someone sending from that domain. It is also capable of allowing sub-domains. It is worth mentioning that entries in this table are only made by the user and are not checked for their validity. The system however makes sure that for 'domains only' the entries are prefixed with a '@' character, while for domain and sub domain allowance two entries are created, where the first one is preceded by '@' and the second one by a dot ('.').
In the Trustcenter we have a corresponding array, which is also called 'whitelist'. Its key is the domain name and its value is the trust indicating Boolean.

The second table named 'mails' keeps track of the most important data for our extension, namely the contact information. Its columns represent the mail address of a sender, whether to trust him or not and how often we interacted with this particular person. The extension regards every received or send message from or to this address as one interaction.
The Trustcenter arrays 'interactionHashMap' and 'trustHashMap' also associate these values to the e-mail addresses.

Lastly, the third table stores information about known names. It is updated whenever a new name for an e-mail address becomes known. Its counterpart is the 'nameHashMap' which also matches an array of names to an address.

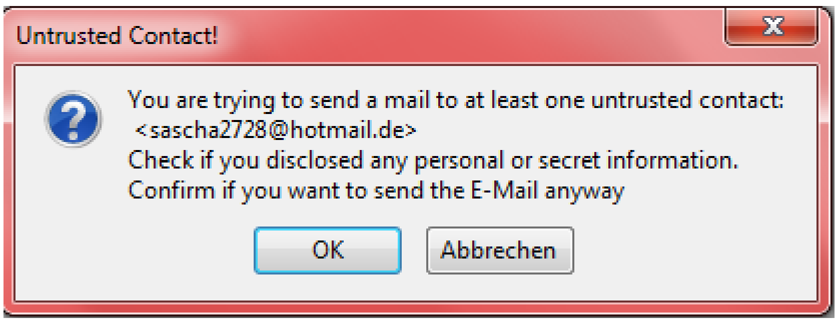
3.4. Sending e-mails

The process of sending an e-mail is implemented straightforward. When the user clicks the send button, the sending gets intercepted and our code checks whether the recipients' addresses are known. In case of one or more unknown contacts, the user gets offered the option of adding them to his trustbase as trusted contacts as shown in Figure 3.1.


Trust Contact?

You are sending a mail to an unknown contact:
<admin@tu-darmstadt.de>
Would you like to trust that Contact in the future?

Figure 3.1 Sending interception dialogue for unknown contacts


Untrusted Contact!

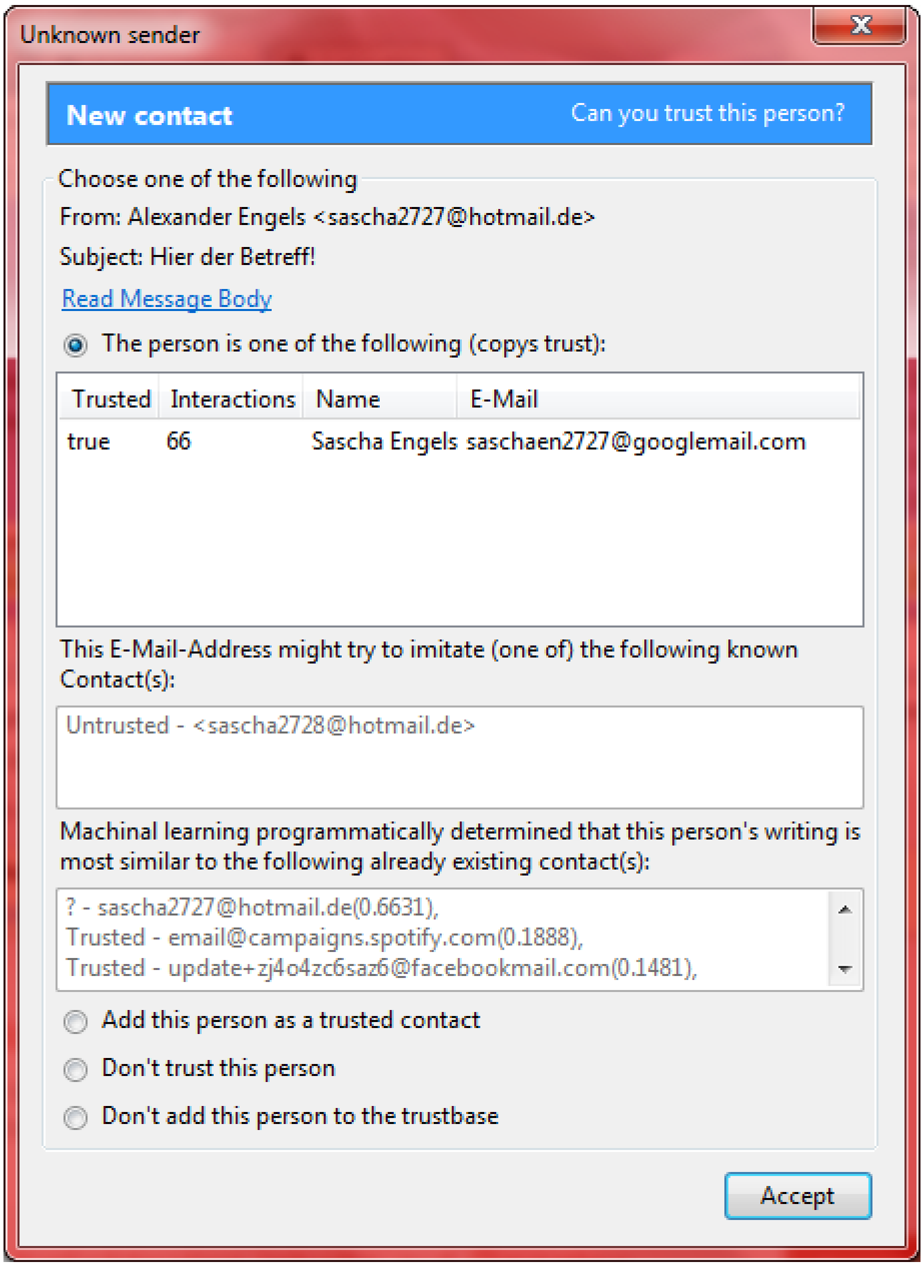
You are trying to send a mail to at least one untrusted contact:
<sascha2728@hotmail.de>
Check if you disclosed any personal or secret information.
Confirm if you want to send the E-Mail anyway

Figure 3.2 Sending interception dialogue for not trusted contacts

If he refuses the interaction will not be counted in the future and the contact will not be added to the trustbase. If however an e-mail is send to a known contact, the extension will register a new interaction for each known recipient. If one or more of the recipients are untrusted, the user is issued an alert stating the dangers of sending e-mail to untrusted contacts. This dialogue can be seen in Figure 3.2. He can however dismiss the dialog and still send the message.

3.5. Receiving e-mails

On the receiving site the underlying logic is more complex. There is a special case when the incoming e-mail address matches one of our white listed domain names. In that case, a predefined trust is transferred to the new e-mail address. An e-mail address is regarded as matching if it ends with a known domain name. The predefined trust value comes from the entry the user made for the specific domain. Not white listed domains are handled as follows.



Choose one of the following

From: Alexander Engels <sascha2727@hotmail.de>

Subject: Hier der Betreff!

[✓ The person is one of the following (copys trust):]

Trusted	Interactions	Name	E-Mail
true	66	Sascha Engels	saschaen2727@googlemail.com

This E-Mail-Address might try to imitate (one of) the following known Contact(s):

Untrusted - <sascha2728@hotmail.de>

Machinal learning programmatically determined that this person's writing is most similar to the following already existing contact(s):

? - sascha2727@hotmail.de(0.6631),
Trusted - email@campaigns.spotify.com(0.1888),
Trusted - update+zj4o4zc6saz6@facebookmail.com(0.1481),

Add this person as a trusted contact

Don't trust this person

Don't add this person to the trustbase

Figure 3.3 A possible information dialogue

For known users the interaction counter is raised and the user is made aware of whether or not to trust the e-mail by GUI elements which will be discussed further ahead. Also if the sender added an unknown name in the header, the name is associated with this e-mail address.

For unknown senders the user will be presented with a rich collection of information on whether to trust this person or not. Since we focus on avoiding social engineering and especially impersonation attacks like the doppelganger mail, the information concentrates on related fields. A possible information dialogue can be seen in Figure 3.3.

First off, the user is made aware that he is interacting with a new contact. He is given header information such as the sender's name and e-mail address, the subject of the mail and the possibility to read through the body of the e-mail, which is stripped of HTML and possible scripts.

If the sender specified his name, the user can find a listbox below the message body. It features contacts from the trustbase with similar names, which are determined by our adopted Levenshtein algorithm as described in chapter 3.2. If the user knows that the person has two e-mail accounts, he can then proceed to directly link their addresses and thereby copy the trust and interactions the first known address had. Unfortunately, there is no deep binding of the addresses to an identity, but it is generally possible and could be adopted to improve results of the Bayes classification.
Next, the user may be informed about impersonation of e-mail addresses. This is the core element in the fight against doppelganger mail addresses since all close e-mail addresses, as again determined by our adopted Levenshtein, are listed and confusion between those is mitigated.

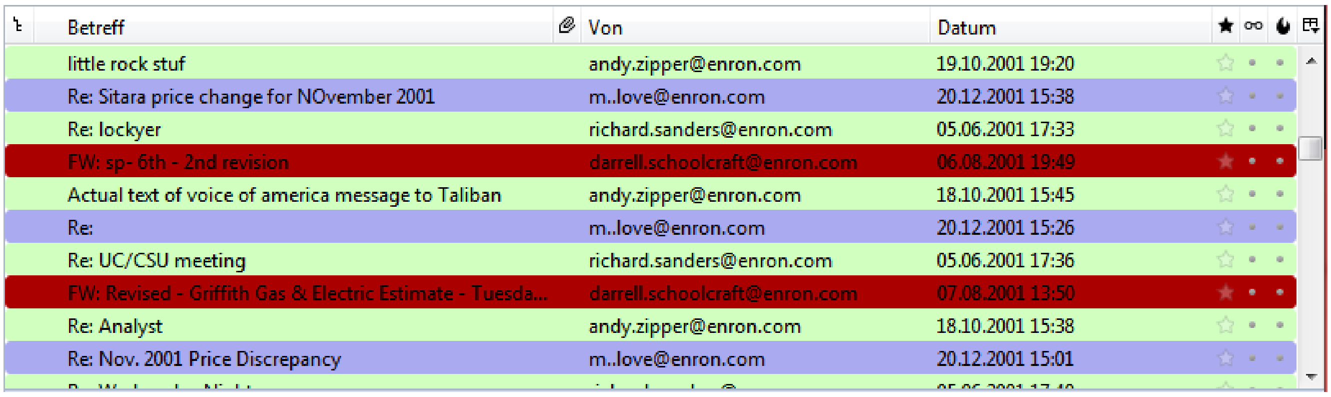
At last, the user gets presented with information about the likelihood that the received message is similar to messages written by contacts that sent him e-mails in the past. That likelihood is calculated via the Naïve Bayes algorithm as specified in chapter 2.2. In order to archive the classification the Naïve Bayes is trained with all messages in the inbox and then fed the body of the new e-mail. In our scenario e-mail addresses suffice as classes and words found in the body of e-mails are the given features. In the learning process the Naïve Bayes classifier collects and counts all the words written from an e-mail address (class). The message bodies are cleaned of common words and symbols beforehand, so the result is not distorted. The possible candidates are then ordered by their respective likelihood and if possible information about their trusted state is also displayed. The trusted state cannot always be determined, because unlike as with the former approaches this time we utilize all possible contacts which send messages to our inbox.

Both, the similar names and similar e-mail address information, will be omitted if no similar entities are found. Likewise the Bayes classification can inform the user that it is unlikely that the received message is similar to any known message collection.

After consuming all of the above information, the user can then decide whether or not to trust the person or disregard classification by canceling or choosing the corresponding option.

3.6. User interface

The extension has two main interface enhancements apart from its dialogues. These are the modifications of the e-mail listing and the settings menu.
The first adds a new column to the list of received e-mails, which is called 'Trusted'. Here the user is presented with 'Y', 'N' and 'U' entries, which respectively mean that the user is trusted, untrusted or unknown. Additionally, the rows are colored green, red or blue to represent the same information even if the column is hidden. The choice of colors is oriented on typical psychological symbolism they carry in central Europe such as 'danger' for red, 'good' for green and 'neutral' for blue. Figure 3.4 depicts what a typical inbox could look like.

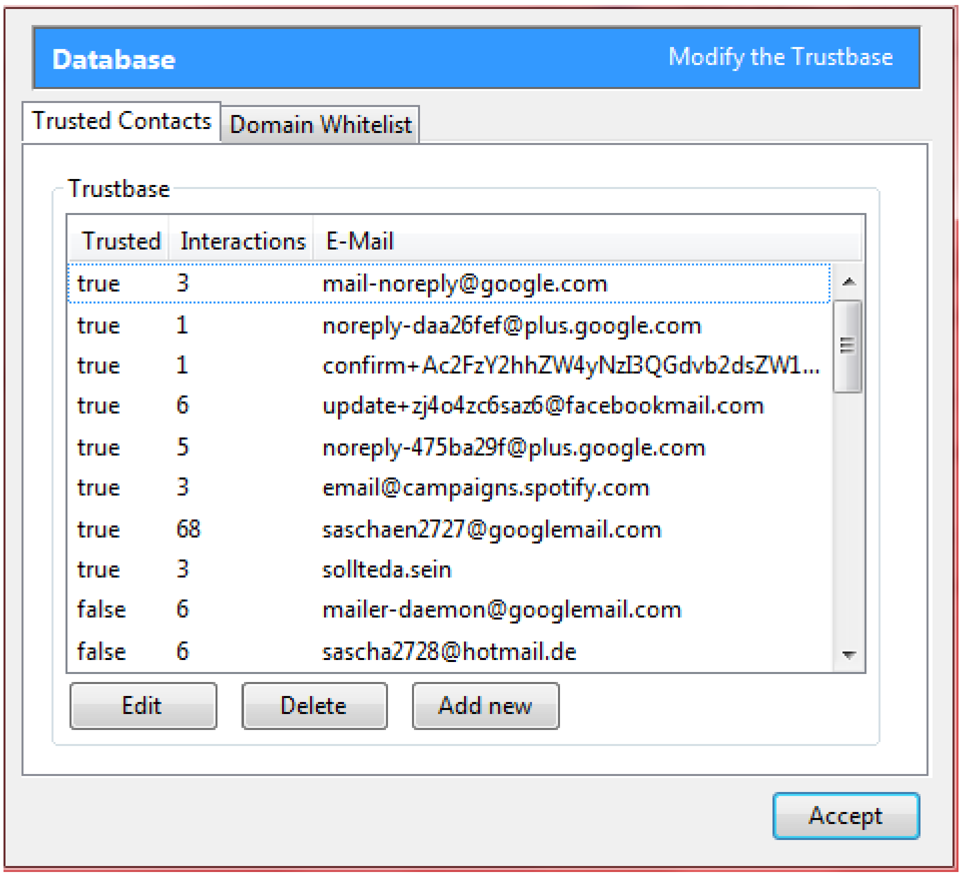


Betreff	Von	Datum
little rock stuf	andy.zipper@enron.com	19.10.2001 19:20
Re: Sitara price change for NOvember 2001	m..love@enron.com	20.12.2001 15:38
Re: lockyer	richard.sanders@enron.com	05.06.2001 17:33
FW: sp- 6th - 2nd revision	darrell.schoolcraft@enron.com	06.08.2001 19:49
Actual text of voice of america message to Taliban	andy.zipper@enron.com	18.10.2001 15:45
Re:	m..love@enron.com	20.12.2001 15:26
Re: UC/CSU meeting	richard.sanders@enron.com	05.06.2001 17:36
FW: Revised - Griffith Gas & Electric Estimate - Tuesda...	darrell.schoolcraft@enron.com	07.08.2001 13:50
Re: Analyst	andy.zipper@enron.com	18.10.2001 15:38
Re: Nov. 2001 Price Discrepancy	m..love@enron.com	20.12.2001 15:01
[illegible]	[illegible]	[illegible]

Figure 3.4 Color-coded inbox

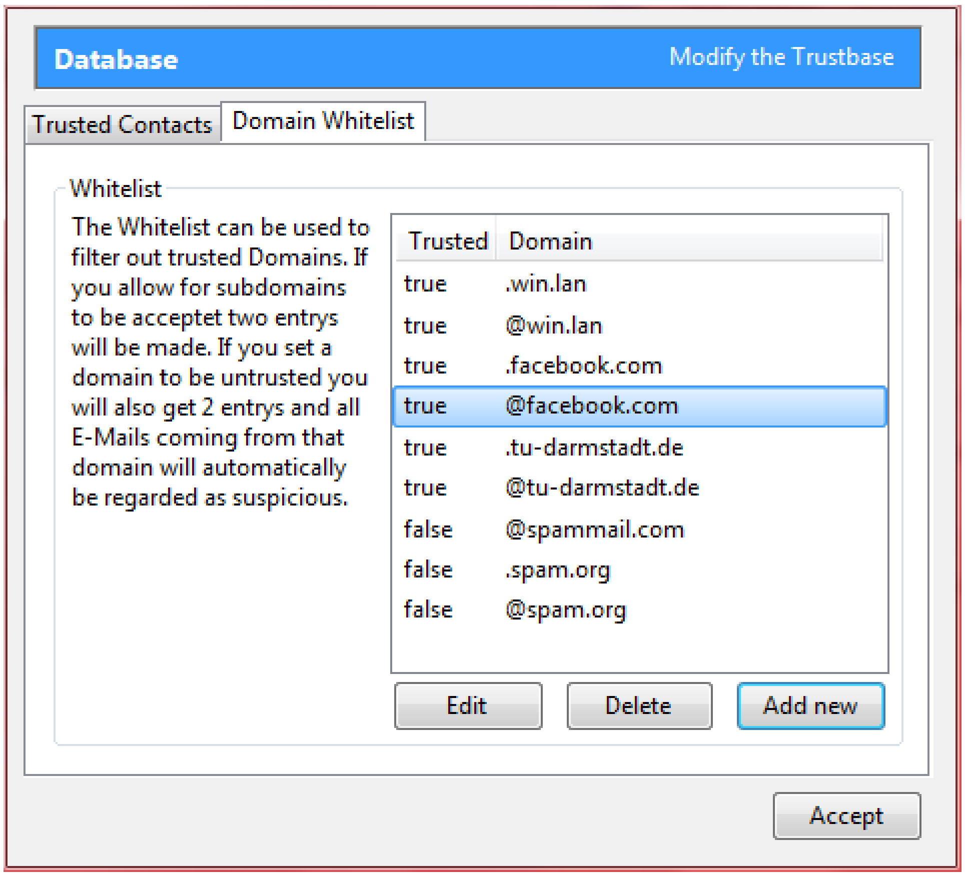
The second GUI element is a window with two tabs. The first tab as shown in Figure 3.5 is used to add, edit or delete known contacts. The modifications allow changing e-mail address, interactions and trust. The second tab selected in Figure 3.6 is similar, but is dedicated to the modification of domains to the whitelist. Here the user can choose between trusting or not trusting a domain and possibly its sub-domains.

Considering user interaction overhead we tried to implement only dialogues considered absolutely necessary. For an average user, who does not have many new contacts writing to him every day, the additional time required for clicking through the information dialogue is very low. It is only necessary to make a decision based on the presented information and click accept. When sending e-mail, there is no overhead unless he sends to an unknown or untrusted contact, in which case he is informed of that action and can dismiss the dialogue with one click per unknown or untrusted contact. We feel that the performance of DeTypo outperforms the necessary time consummation and is therefore appropriate.



Trusted	Interactions	E-Mail
true	3	mail-noreply@google.com
true	1	noreply-daa26fef@plus.google.com
true	1	confirm+Ac2FzY2hhZW4yNzI3QGdvb2dsZW1...
true	6	update+zj4o4zc6saz6@facebookmail.com
true	5	noreply-475ba29f@plus.google.com
true	3	email@campaigns.spotify.com
true	68	saschaen2727@googlemail.com
true	3	sollteda.sein
false	6	mailer-daemon@googlemail.com
false	6	sascha2728@hotmail.de

Figure 3.5 Options window used for editing the trustbase



The Whitelist can be used to filter out trusted Domains. If you allow for subdomains to be acceptet two entrys will be made. If you set a domain to be untrusted you will also get 2 entrys and all E-Mails coming from that domain will automatically be regarded as suspicious.

Trusted	Domain
true	.win.lan
true	@win.lan
true	.facebook.com
true	@facebook.com
true	.tu-darmstadt.de
true	@tu-darmstadt.de
false	@spammail.com
false	.spam.org
false	@spam.org

Figure 3.6 Option window used for editing the whitelist

4. Testing and Evaluation

In this section we describe the way the extension is tested. We have a look at the testing environment as well as the evaluation process. Afterwards we discuss how the results match our expectations.

The evaluation process does not claim to be statistically significant. We only try to show the tendencies towards which the proof of concept strives. As such, the tests will sufficiently demonstrate whether DeTypo is successful in providing the user with relevant information. Especially the Naïve Bayes classification needs adequate testing.

4.1. Testing Environment

The testing environment consists of an e-mail server and a Thunderbird client in version 11. The e-mail storage of the e-mail server can be easily accessed and modified by copying or deleting e-mails into it. The Thunderbird client has a registered account on the server and is notified upon receiving e-mails.

4.1.1. Server

The server, which was generously provided by the CASED Faculty/Chair of Security in information technology from TU Darmstadt, uses Courier Mail[10] Server and Postfix[11] software. This allows for IMAP services as well as SMTP. They both allow copying of e-mails into a dedicated folder to emulate newly arrived e-mails.

The server can be accessed remotely via a secure shell (ssh) tunnel. This made it possible to modify server data, such as e-mails, without physical access to the server. Ssh is a network protocol, which allows for encrypted communication with devices and access to command shells on those. We used the PuTTY[12] client to access the server, but any ssh client would have been sufficient.

4.1.2. Enron e-mail Dataset

The e-mail data was taken from the Enron corpus, which had to be made public during legal investigations concerning the Enron Company. The corpus contains 619,446 messages from 158 users, but many of these are duplicates [Kli04]. The interested reader can find the whole corpus at William Cohen's website at the Carnegie Mellon University [Enr12].

The data are sorted by users and their folders. In our research we use only a few users and copy their inboxes or parts of them into our own inbox. We also modify header data such as the name and address fields.

4.2. Evaluation process

We try to cover as many scenarios as possible, but emphasize phishing and social engineering attacks. The standard tests covers receiving of e-mails from white listed domains and sub domains as well as contacts in the trustbase. These tests also cover attacks with simple doppelganger mail addresses. The whole list of tests carried out can be seen in Table 4.1.

The more important tests cover interactions with unknown e-mail addresses. They are more important because interaction with trusted and untrusted contacts leaves only limited room for errors. We want to see how good the classification of contacts works. We therefore build a base on which to classify on

[10] http://www.courier-mta.org/

[11] http://www.postfix.org/

[12] http://www.putty.org/

by copying messages from multiple contacts into our current e-mail directory. We usually copy about 100 messages per contact and about 5 or more contacts. This small number in relation to the total amount of e-mails in the Dataset is owed to the fact that no automation is used in creating the test inbox. Also the performance is expected to significantly drop with too many e-mails in the inbox. We however test the performance with a great number of e-mails in the Inbox in the following section. We then choose one of these contacts and copy more of its messages into our new e-mail directory. Note that at this point none of these contacts are in the trustbase, but are subject to classification. To ensure that we always see the classification screen we then proceed by not adding the contact to the trustbase.

We also inspect entirely new e-mails and analyze whether or not we get false positives for known contacts. A false positive is regarded as a misclassification to another contact, who did not write the message.

4.3. Expectations and Results

The tests of standard functionality all passed as expected, except for a test where Thunderbird is closed on start-up, right after it receives new messages, but before learning new messages. In that case no 'new message' events are triggered on a second start-up, so our new e-mail listener is not fired and no classification occurs. This only affects unknown contacts and those stacked behind them in the event queue.

In another test we expected a serious slow down of Thunderbird upon start-up with 4000 messages in the inbox. That expectation is based on the fact that our implementation teaches all the messages to our Bayes classifier each time Thunderbird is opened. At the time this test was carried out we learned the message as a whole, as opposed to a later attempt when we only learned the first 100 words. At first this test failed, in that it hindered Thunderbird from even starting for the whole duration of the learning process. But slight changes in the implementation to run the classification on a background thread improved the results significantly. This test result shows that Thunderbird is able to easily asynchronously calculate the classes in a reasonable amount of time, which is just close below 3 minutes for 4000 messages. In that time no new messages are classified, but they are put on a stack and are traversed after the learning process finishes. Note that this outcome of the test is based on implementation decisions and could be improved by persistent storage of learned data. In hindsight of the outcome of the second to last test of this series, where we closed Thunderbird right at start-up, after receiving new e-mails, this however suggests possible information loss to us.

A detailed list of tests, expectations and results can be found in Table 4.1 below. The results suggest that DeTypo works as intended upon development and no major issues are encountered. This forms a solid basis for our further tests of performance in a second set of tests, because we do not have to fear to encounter inconsistencies from the implementation side.

Test	Expectation	Result
Receive e-mail from white listed domain	No learning dialogue, but instead entry in trustbase according to trust of the e-mail	Implementation successful.
Receive e-mail from white listed sub domain	No learning dialogue, but instead entry in trustbase according to trust of the e-mail	Implementation successful.
Receive e-mails from trusted parties	Incrementation of the interaction counter. Green coloring of the header entry in the list of received messages	Implementation successful.
Receive e-mails from not trusted parties	Incrementation of the interaction counter. Red coloring of the header entry in the list of received messages	Implementation successful.
Receive e-mails from doppelganger mail addresses	Detection of a new e-mail. Possible warning of similar names. Identification of a doppelganger mail and warning. Suggestion of known contacts who might have carried out the attack.	Learning dialogue is triggered and if possible similar named contacts are listed. Doppelganger mail always detected. Suggestion of possible attackers limited by knowledge and capabilities of the Naïve Bayes algorithm.
Receive e-mail from known contact with a new name	Interaction counter counts up. Correct coloring occurs. New name is saved in the system.	Implementation successful.
Receive e-mail from unknown contact	Same as receiving E-mails from doppelganger mail. Guess whether this mail could come from an already known contact.	Implementation successful. Success of contact suggestion dependant on, whether contact is already known. Further research in the second stage of the evaluation.
Closing Thunderbird right after or on start-up	Classification on next start.	Loss of new message event, therefore no classification.
Start Thunderbird with 4000 messages in inbox	Serious slow down on start-up.	Learn process taking 3 minutes. First implementation froze Thunderbird completely during that time. Adjustments led to not locking Thunderbird or slowing it down. DeTypo could however not make any suggestions during classification time.

Table 4.1 Evaluation of standard functionality

In the second part of the evaluation we analyze the expressiveness of the e-mail writer suggestions. The first test run provided questionable results, so that we redeveloped our testing process. We found that quotes in messages have a great impact on the classifyability and that long messages, which contain a lot of quotes could not be classified. Of the 18 failed classifications 17 were caused by very long messages and one by a very short message. The 14 correctly classified messages all have lengths in the range of 20 to 100 words.

Another implication from not cleaning up quotes in the messages is possible misclassification, since related users share a lot of common features. We decided to adopt our classification strategy in future tests by only classifying based on the first 100 written words. This however can mean a loss of signatures and similar features which help in classification. A better solution is detection and deletion of quotes within messages, but the Enron corpus made that an impossible task without fitting tools, which to our knowledge do not exist, because the way messages are quoted varies significantly from person to person.

Messages in inbox	Contacts in inbox	Messages by test contact	Messages to test	Successful classifications	Failed classifications	Wrong classifications	% of success
417	7	66	32	14	18	0	43.75

Table 4.2 First evaluation of classification based on whole messages

Table 4.2 shows the result of the first classification and Table 4.3 contains the results of our redeveloped tests. The statistical power of the latter is lowered for two reasons. Firstly we do not include data about the average length of the classified messages and secondly we only use the first 100 words in longer E-mails, because we are not able to delete quotes from the messages. Some e-mails containing many quotes have lengths of above 500 words. These long messages took a significant longer time to learn and classify, before the adjustments were made.

Numbers in brackets in Table 4.3 are interpreted as follows. If the number is listed under successful classifications then the e-mail address of the contact which had to be learned was among the suggestions, but not at the top. If the number is listed in the 'wrong classifications' column the correct contact was suggested, but wrong ones were also in the list, and the correct contact had less than 80% probability.

Test run	Messages in inbox	Contacts in inbox	Messages by test contact	Messages to test	Successful classifications	Failed classifications	Wrong classifications	% of success
1	484	6	99	49	26	23	0	53.06
2	484	6	91	50	35(+3)	10	5(+2)	70
3	583	7	99	45	22	22	1	48.89

Table 4.3 Evaluation of classifications on 100 word classification

Lastly Table 4.4 shows the evaluation of classifications on entirely new contacts. The expected outcome in this test is a 'failed classification'. Any suggestion for a known contact is regarded as a 'wrong classification', although we cannot rule out the possibility that the writer and the suggested person are related, since the relationship between e-mail writers in the Enron corpus is unknown to us.

Messages in inbox	Contacts in inbox	Messages to test	Successful (no) classifications	Wrong classifications	% of success
840+	10+	30	18	12	60%

Table 4.4 Evaluation of contacts which had no interaction so far

4.4. Interpretation

The test data suggests that our main goal, the detection of doppelganger mail, is always successful. We cannot find any scenario in which a doppelganger mail cannot be detected as such. However one other impersonification attack's impact was boosted by DeTypo. This attack employs the way DeTypo learns from message content to confuse DeTypo into believing the sender is a certain person whose writing style is known to the attacker.

In conclusion of the first test we determine that the basic functionality aimed for is successfully implemented. We therefore regard DeTypo as fully capable of raising the awareness of users in all environments.

In case the attacker does not know anything about the writing style of a third party it is very unlikely that he can successfully impersonate it. In that case our Naïve Bayes classifier shows promise. Misclassifications are unlikely to happen, however no classification is a common occurence, but does not confuse the user as much as a misclassification.

The cooperation of different aspects helps a lot to raise the expressiveness of DeTypo. The warning of doppelganger mail together with a failed classification as the pretend-to-be contact and copying of the name highly suggest a doppelganger mail attack. On the other hand DeTypo makes it easier to detect whether a contact uses another e-mail address, by highlighting the similar names and a fitting classification.

The last test shows that attackers are not easily classified as a specific known contact. With a 60% chance in our case the attacker is not classified. If a classification can successfully be made, our extension suggests only one person in most cases. If it is unlikely that this person would have written under another name, or the suggestion does not fit the sender, the attack could easily be detected.

Overall the Naïve Bayes classifier does not perform very well. Although serious failures, where misclassifications occur, are rare, correct classifications are neither seen often. The success rate of the classification is only slightly higher than 50%, suggesting that there is a lot of improvement needed before its expressiveness is high enough. We find this not to be critical, since the classification only serves as a detector of the source of an attack at first. However, since the classification approach shows promise and is gaining popularity with measures such as ASCAI we still tried to improve it by limiting the classification to the first 100 words, which slightly improved the results. We are positive that if a good quote deletion algorithm existed the outcome of the classification could be further improved.

5. Conclusion and Future Work

This analysis clearly shows the great potential of the extension DeTypo to the E-Mail client Thunderbird for future work. Not only is it possible to fully implement the planned features, it is also possible to do it in a reasonable amount of time. However, the documentation of some specific Thunderbird functionalities is very poor and it is not likely that specific parts of the extension are easily portable to other e-mail clients such as Microsoft Outlook or Gmail, but similar extensions are conceivable.

The extension helps to raise awareness of new contacts and possible frauds. Especially the doppelganger mail attack looses impact when DeTypo is employed. Other impersonification attacks can successfully be detected and possible sources can be stated.

Further research of DeTypo is needed to confirm its practicability. New fields of research are usability tests in real live scenarios and further performance confirmation which yields statistical significant data. Especially tests which rate how often a user can be tricked into believing the authenticity of e-mails when he employs DeTypo against cases where he does not employ DeTypo seem promising.

The Enron dataset is limited in that it only contains English messages from people employed in similar fields. Other datasets can provide information on how DeTypo performs with different languages and contexts. Since a large quantity of spam is written in English, it is easier to detect it if the user usually communicates in another language. However, spear phishing attacks are likely to be in an expected language and context, so that DeTypo should perform similar to the Enron dataset evaluation.

One major impact on the capacity of DeTypo is imposed by using e-mail services as an instant messenger. Short messages with many common words could confuse the Naïve Bayes classification. That is why a research employing a dataset with only very short messages is assumed to show interesting results. Our expectation is that DeTypos classification ability will be slightly dulled, but will still work if the attacker sends a regular message, since all messages of one contact as a whole are taken into consideration. If the attacker sends a short message the outcome cannot be predicted by our evaluation, however, we still believe it to be very hard to impersonate a specific contact without knowledge of his writing style.

There are still a lot of possibilities to improve the presented information. For a start Levenshtein and Naïve Bayes with words are not the smartest algorithms in today's time. There are other algorithms which can compute word distances in different ways, but we decided to go with a well-known one. It seems most likely that there is a superior algorithm to find better results for the similarity of names. More important however is the fact that the way we use Naïve Bayes is very straightforward which can be advanced in many ways. One can for example work with n-grams instead of whole words or use more sophisticated approaches and learning algorithms altogether. Frank and Bouckaert suggest that Naïve Bayes performs poorly if given unbalanced classes [Fra06]. They propose ways to improve the performance of the algorithm with various methods such as data transformation and setting attribute priors. These insights can be incorporated into our classifier, since it is highly probable that the classes, which in our case are the e-mail addresses of all messages in the inbox, are very unbalanced.

A minor field of work is a better visualization of information. Possible approaches are preventing untrusted e-mails from being read without confirmation or collecting information on known phishing attacks and combining them with knowledge already gathered. In general a combination of our information with already known phishing detections to suggest a certain type of phishing attack being carried out is an interesting field of work.

Another problem we encounter is the detection of copy content attacks, like the man-in-the-mailbox scenario. So far it is not possible to tell whether an attacker just copies and pastes content he receives from the contact he tries to imitate or if the genuine contact is sending us e-mails. An interesting approach in this area is to combine DeTypo with ASCAI. That means classifying every incoming message, whether known or unknown and, if the result shows a poor likelihood of the message being genuine, the user can be warned. Also a dummy class for phishing attacks can be employed.

Another possibility we do not consider in our research is the scenario where an unintentional doppelganger mail exists. Consider two people with similar names, who work for the same company, but in different areas and levels in the business hierarchy. These people will probably have a similar e-mail address, which might be formed from their last name and their company's name. On the first encounter a mutual customer is warned by DeTypo, but upon trusting both parties he will not be warned any further. The customer can then accidentally confuse these parties and send information to the wrong place. This often does little damage, but there is potential for critical data leaks. To help avoid this, the user could be prompted to reconfirm outgoing messages, when he trusts doppelganger mail like addresses.

6. References

6.1. List of figures

6.2. List of tables

6.3. List of abbreviations

API	Application programming interface
ASCAI	Anti-Spear phishing Content-based Authorship Identification
GPL	GNU General Public License
IMAP	Internet Message Access Protocol
HTML	Hypertext markup language
LGPL	GNU Lesser General Public License
MPL	Mozilla Public License
PGP	Pretty Good Privacy (Software program for Encryption)
RSS	Really Simple Syndication
SMTP	Simple Mail Transfer Protocol
SQL	Structured Query Language (, but also a name on its own)
SSH	Secure Shell
URL	Unified Resource Locator
XUL	XML User Interface Language
XML	Extensible Markup Language

6.4. Bibliography

[Abd96] Abdul-Rahman, A. (1996). *The PGP Trust Model.* London.

[Ada10] Adair, S., Deibert, R., Rohozinski, R., Villeneuve, N., & Walton, G. (2010). *Shadows in the Cloud: Investigating Cyber Espionage 2.0.*

[Bal] Balasubramanyan, R., Carvalho, V. R., & Cohen, W. (2008). *Cut Once - Thunderbird Extension*. Retrieved June 10, 2012, from http://www.cs.cmu.edu/~vitor/cutonce/cutOnce.html

[Car07] Carvalho, V., & Cohen, W. (2007). *Recommending recipients in the Enron email corpus.* Carnegie Mellon Univeristy.

[Cis12] Cisco. (2012). *Cisco IronPort*. Retrieved from http://www.senderbase.org/home/detail_spam_volume?displayed=last12months&action=&screen=&order=

[Enr12] Cohen, W. W. (2009). Retrieved May 20, 2012, from Enron E-Mail corpus: http://www.cs.cmu.edu/~enron/

[Fra06] Frank, E., & Bouckaert, R. R. (2006). Naive Bayes for Text Classification with Unbalanced Classes. *In Proceedings of the 10th European Conference on Principles and Practice of Knowledge Discovery in Databases* , pp. 503-510.

[Dan97] Hirschberg, D. (1997). *Serial Computations of Levenshtein Distances.*

[Kli04] Klimt, B., & Yang, Y. (2004). *The Enron Corpus: A New Dataset for Email Classification Research.* Proceedings of the European Conference on Machine Learning: Springer.

[Wik] Levenshtein, W. (2012, June 10). *Algorithm Implementation/Strings/Levenshtein distance*. Retrieved from http://en.wikibooks.org/wiki/Algorithm_Implementation/Strings/Levenshtein_distance

[Mah11] Mahmoud, K., Iraqi, Y., & Jones, A. (2011). Mitigation of Spear Phishing Attacks: A Content-Based Authorship Identification Framework. *Internet Technology and Secured Transactions (ICITST), 2011 International Conference for*, (pp. 416-421). Abu Dhabi, United Arab Emirates.

[Moz12] Mozilla. (2012). Retrieved June 20, 2012, from Mozilla Thunderbird End-User Software License Agreement: http://www.mozilla.org/en-US/legal/eula/thunderbird.html

[Mar05] Randazzo, M. R., Keeney, M., Kowalski, E., Cappelli, D. M., & Moore, A. P. (2005). *Insider Threat Study: Illicit Cyber Activity in the Banking and Finance Sector.* Pittsburgh, PA.

[Ris01] Rish, I. (2001). An empirical study of the naive bayes classifier. *Proceedings of IJCAI-01 workshop on Empirical Methods in AI"* , pp. 41-46.

[Ema12] Sabellico, E. (2010). *MailClassifier*. Retrieved June 2012, from https://addons.mozilla.org/de/thunderbird/addon/mailclassifier/

[Sah98] Sahami, M., Dumais, S., Heckerman, D., & Horvitz, E. (1998). A Bayesian approach to filtering junk e-mail. *AAAI'98 Workshop on Learning for Text Categorization.*

[Spa12] SpamAssassin. (2004). *Welcome to SpamAssassin*. Retrieved June 11, 2012, from http://spamassassin.apache.org/

[std] stdlib. (2012, June 10). *Thunderbird Standard Library Project*. Retrieved from https://github.com/protz/thunderbird-stdlib

[Jos12] Tauberer, J. (2008). *Sender Verification Anti-Phishing Extension*. Retrieved June 10, 2012, from https://addons.mozilla.org/de/thunderbird/addon/sender-verification-anti-phish/?src=search

[Thu12] Thunderbird. (2012). *Mozilla Thunderbird Features*. Retrieved June 11, 2012, from http://www.mozilla.org/en-US/thunderbird/features/#customize-email

7. Appendix

7.1. Program code

The program code is distributed among many files. The most important code to understand the plug-in will be listed here.

Levenshtein as suggested by:
http://en.wikibooks.org/wiki/Algorithm_Implementation/Strings/Levenshtein_distance#JavaScript

Naïve Bayes as suggested by: http://www.dusbabek.org/~garyd/bayes/bayes.js

7.1.1. Overlay on the main window of Thunderbird

(content/overlay.js)

```
Cu.import("resource://gre/modules/Services.jsm");
Cu.import("resource://gre/modules/FileUtils.jsm");
Cu.import("resource://doppelgangermail/stdlib/msgHdrUtils.js");
Cu.import("resource://doppelgangermail/stdlib/compose.js");
Cu.import("resource://doppelgangermail/stdlib/send.js");
Cu.import("resource://doppelgangermail/stdlib/misc.js");
Cu.import("resource://doppelgangermail/dbHelper.js");
Cu.import("resource://doppelgangermail/trustCenter.js");
Cu.import("resource://doppelgangermail/machinalLearner.js");


var DoppelgangerMail = {};


/*
 *  Test function linked to Button in the Toolbar
 **/
doSomething = function(){
   //DBManager.resetDB();
   //DBManager.initTrustCenter();
}

//initialisation function (to be triggered upon load)
function init() {
   var notificationService = Cc["@mozilla.org/messenger/msgnotificationservice;1"].
   getService(Ci.nsIMsgFolderNotificationService);
   notificationService.addListener(newMailListener, notificationService.msgAdded);

   let prefService = Cc["@mozilla.org/preferences-service;1"].getService(Ci.nsIPrefBranch);
   if(prefService.getBoolPref("extensions.doppelgangermail.initial.load")){
      window.setTimeout(buildInitialTrustBase, 1000);
      prefService.setBoolPref("extensions.doppelgangermail.initial.load", false);

   }

   DBManager.openCon();
   DBManager.initTrustCenter();
   DBManager.closeCon();
```

```
    TrustCenter.TBWindow = window;
    TrustCenter.Learner = Learner;

    var ObserverService = Cc["@mozilla.org/observer-service;1"].getService(Ci.nsIObserverService);
    ObserverService.addObserver(ViewObserver, "MsgCreateDBView", false);

    Learner.getLearnContent();

}

//binds the init() function to the "load" Event, which is triggered upon opening TB
addEventListener("load", init, true);


/*
 *  Listener for new mail
 *  Function msgAdded is called when a new e-mail arrives
 */
var newMailListener = {
    msgAdded: function(aMsgHdr) {
        if( !aMsgHdr.isRead && msgHdrIsInbox(aMsgHdr) )
            //new unread message, check Contact
            window.setTimeout(processIncomingMail, 1000, aMsgHdr);
    }
}

function processIncomingMail(aMsgHdr){
    DBManager.openCon();
    DBManager.checkContact(aMsgHdr);
    DBManager.closeCon();
}

var ViewObserver = {
    observe: function(aMsgFolder, aTopic, aData){
        initGUI();
    }
}

/*
 *  Prepares coloring in the message list and appends the column handler to said list
 **/
function initGUI(){

  var columnHandler = {
    getCellText:        function(row, col) {
        var hdr = gDBView.getMsgHdrAt(row);
        var mail = parseMimeLine(hdr.author)[0].email;
        //find out if mail is trusted
        if(typeof(TrustCenter.trustHashMap[mail]) !== 'undefined'){
            if(TrustCenter.trustHashMap[mail] == 'true')
                return 'Y';
            else
                return 'N';
        }else
            return 'U';
    },
    getSortStringForRow: function(hdr) {},
    isString:           function() {return true;},
    getCellProperties:   function(row,col,props){},
    getRowProperties:    function(row, props){
        var hdr = gDBView.getMsgHdrAt(row);
        var mail = parseMimeLine(hdr.author)[0].email;
        if(typeof(TrustCenter.trustHashMap[mail]) !== 'undefined'){
            if(TrustCenter.trustHashMap[mail] == 'true'){
                var aserv=Cc["@mozilla.org/atom-service;1"].getService(Ci.nsIAtomService);
                props.AppendElement(aserv.getAtom("greenBG"));
            }else{
                var aserv=Cc["@mozilla.org/atom-service;1"].getService(Ci.nsIAtomService);
                props.AppendElement(aserv.getAtom("redBG"));
            }
        }else{
            var aserv=Cc["@mozilla.org/atom-service;1"].getService(Ci.nsIAtomService);
            props.AppendElement(aserv.getAtom("blueBG"));
        }
    },
    getImageSrc:         function(row, col) {return null;},
    getSortLongForRow:   function(hdr) {}
  }
  gDBView.addColumnHandler("colTrust", columnHandler);
}


/*
 *  Executes once after a fresh install of the Plugin
 *  Prompts the User whether he wants to trust all his contacts so far or not
 *  if not they will stay uncategorized
 **/
function buildInitialTrustBase(){
    let prompts =  Cc["@mozilla.org/embedcomp/prompt-service;1"].getService(Ci.nsIPromptService);
    if (prompts.confirm(TrustCenter.TBWindow, "Trust all?", "Hello,\nthis is the first time you installed
the doppelganger Plugin.\nWould you like to trust the authors of\nall the mail you recieved so far?"))
{
        //accept
        //initialisator is a helper object ot count the emails
        var Initialisator = {
            interactions : new Object(),
            names : new Object(),
            hdrs : new Object(),
        };

        //loop through all folders
        //not a perfect approach since only top level folders are considered
        var acctMgr = Cc["@mozilla.org/messenger/account-
manager;1"].getService(Ci.nsIMsgAccountManager);
        var accounts = acctMgr.accounts;
        for (var i = 0; i < accounts.Count(); i++) {
            var account = accounts.QueryElementAt(i, Ci.nsIMsgAccount);
            var rootFolder = account.incomingServer.rootFolder; // nsIMsgFolder
            if (rootFolder.hasSubFolders) {
                var subFolders = rootFolder.subFolders; // nsIMsgFolder
                while(subFolders.hasMoreElements()) {
                    var theFolder = subFolders.getNext().QueryInterface(Ci.nsIMsgFolder);
                    var msgArray = theFolder.messages;
                    while( msgArray.hasMoreElements() ) {
                        let msgHdr = msgArray.getNext().QueryInterface(Ci.nsIMsgDBHdr);
                        Initialisator.hdrs[msgHdr.messageKey] = msgHdr;
                    }
                }
            }
        }

        for(var hdrKey in Initialisator.hdrs){
            let hdr = Initialisator.hdrs[hdrKey];
            let mail = parseMimeLine(hdr.author)[0].email;
            let name = parseMimeLine(hdr.author)[0].name;
            if(typeof(Initialisator.interactions[mail]) == 'undefined'){
                Initialisator.interactions[mail] = 1;
                Initialisator.names[mail] = new Object();
                //Using arrays is also possible, but this is easier
                Initialisator.names[mail][name] = true;
            }else{
                Initialisator.interactions[mail]++;
                Initialisator.names[mail][name] = true;
            }
        }

        DBManager.openCon();
        DBManager.resetDB();
        for(var mail in Initialisator.interactions){
            let interactions = Initialisator.interactions[mail];
            //let name = Initialisator.names[mail];
            DBManager.addTrustMail(mail, "", 'true', interactions);
            //add the collected names
            for(var _name in Initialisator.names[mail])
                DBManager.addNametoMail(mail, _name);
        }
        DBManager.closeCon();
    }
}
```

7.1.2. Database interaction

(modules/dbHelper.js)

```
var EXPORTED_SYMBOLS = ["DBManager", "file", "mDBConn", "Learner"];
const {classes: Cc, interfaces: Ci, utils: Cu} = Components;

Cu.import("resource://gre/modules/Services.jsm");
Cu.import("resource://gre/modules/FileUtils.jsm");
Cu.import("resource://doppelgangermail/stdlib/msgHdrUtils.js");
Cu.import("resource://doppelgangermail/stdlib/misc.js");
Cu.import("resource://doppelgangermail/trustCenter.js");

var DBManager = {};
var file;
var mDBConn;



/*
 * Initializes Connections to Database
 */
DBManager.openCon = function(){
        file = FileUtils.getFile("ProfD", ["contactIDs.sqlite"]);
        mDBConn = Services.storage.openDatabase(file);
}

/*
 *  Closes DB Connection
 */
DBManager.closeCon = function(){
        try{
                mDBConn.asyncClose();
        }catch(e){
                //Error can occur when a statement is evaluated and triggers another Database query
        }
}

/*
 *  Resets the Database (important for Developing and maybe for the user, possible to
 *  link it to a GUI Element)
 */
DBManager.resetDB = function(){
        TrustCenter.nameHashMap = new Object();
        TrustCenter.interactionsHashMap = new Object();
        TrustCenter.trustHashMap = new Object();
        TrustCenter.Whitelist = new Object();
        DBManager.openCon();
        try{
        mDBConn.executeSimpleSQL("DROP TABLE contacts");
        mDBConn.executeSimpleSQL("DROP TABLE mail");
        mDBConn.executeSimpleSQL("DROP TABLE whitelist");
        } catch(e){

        }
        mDBConn.executeSimpleSQL("CREATE  TABLE 'contacts' ('mail' VARCHAR NOT NULL, 'name'
VARCHAR NOT NULL)");
        mDBConn.executeSimpleSQL("CREATE TABLE 'mails'  ('id' INTEGER PRIMARY KEY
AUTOINCREMENT, 'mail' VARCHAR NOT NULL, 'trusted' BOOL NOT NULL , 'interactions' INTEGER
NOT NULL)");
        mDBConn.executeSimpleSQL("CREATE TABLE 'whitelist' ('domain' VARCHAR NOT NULL,
'trusted' BOOL NOT NULL)");
        DBManager.closeCon();
}
```

```
/*
 *  Checks the TrustCenter for a Contact (by mail)
 *  @param header of a incomming Mail
 */
DBManager.checkContact = function(aMsgHdr){
        var mail = parseMimeLine(aMsgHdr.author)[0].email;
        var name = parseMimeLine(aMsgHdr.author)[0].name;

        if(typeof(TrustCenter.trustHashMap[mail]) !== 'undefined'){
                //known Contact, increment Interactions entry
                DBManager.incrementInteractions(mail);
                //check if we have to learn a new name
                if(typeof(TrustCenter.nameHashMap[mail][name]) == 'undefined'){
                        DBManager.addNametoMail(mail, name);
                }
                TrustCenter.Learner.teachAs(aMsgHdr);
      //if(TrustCenter.trustHashMap[mail] == 'true'){
        //trusted Contact
   }else{
        //new Contact
        //when on whitelist add it directly
        let listed = false;
        for(var white in TrustCenter.Whitelist){
                if(mail.indexOf(white, mail.length - white.length) !== -1){
                        listed = true;
                        DBManager.addTrustMail(mail, name, TrustCenter.Whitelist[white], 1);
                }
        }
        if(!listed)
        TrustCenter.Learner.tryToLearn(aMsgHdr);
   }
}


/*
 *  Initialises the shared trustCenter objects for a effective way of looking
 *  up if a mail is trusted
 *  @after: trustCenter contains knowledge of trusted and untrusted recipients
 *          and number of interactions
 **/
DBManager.initTrustCenter = function(){
        var statement = mDBConn.createStatement("SELECT * FROM 'mails'");
        statement.executeAsync({
      handleResult: function(aResultSet){
        for (let row = aResultSet.getNextRow(); row; row = aResultSet.getNextRow()) {
                                let mail = row.getResultByName("mail");
                                let trust = row.getResultByName("trusted");
                                let interactions = row.getResultByName("interactions");
                                TrustCenter.trustHashMap[mail] = trust;
                                TrustCenter.interactionsHashMap[mail] = interactions;
                                TrustCenter.nameHashMap[mail] = new Object();
                        }
      },
        handleError: function(aError) {
                alert("Error: " + aError.message);
        },
        handleCompletion: function(aReason) {
        if (aReason != Ci.mozIStorageStatementCallback.REASON_FINISHED)
                alert("Initialisation failed!");
        }
   });

   var stmt = mDBConn.createStatement("SELECT * FROM 'contacts'");
        stmt.executeAsync({
      handleResult: function(aResultSet){
        for (let row = aResultSet.getNextRow(); row; row = aResultSet.getNextRow()) {
                                let mail = row.getResultByName("mail");
                                let name = row.getResultByName("name");
                                if(typeof(TrustCenter.nameHashMap[mail]) != "undefined")
                                         TrustCenter.nameHashMap[mail][name] = true;
                        }
      },
        handleError: function(aError) {
                alert("Error: " + aError.message);
        },
        handleCompletion: function(aReason) {
        if (aReason != Ci.mozIStorageStatementCallback.REASON_FINISHED)
                alert("Initialisation failed!");
        }
   });
   var whitestmt = mDBConn.createStatement("SELECT * FROM 'whitelist'");
        whitestmt.executeAsync({
      handleResult: function(aResultSet){
        for (let row = aResultSet.getNextRow(); row; row = aResultSet.getNextRow()) {
                                let domain = row.getResultByName("domain");
                                let trusted = row.getResultByName("trusted");
                                TrustCenter.Whitelist[domain] = trusted;
                        }
      },
        handleError: function(aError) {
                alert("Error: " + aError.message);
        },
        handleCompletion: function(aReason) {
        if (aReason != Ci.mozIStorageStatementCallback.REASON_FINISHED)
                alert("Initialisation failed!");
        }
   });
}


/*
 * Adds a contact to the trustCenter and to the Database
 * @param: mail and name is a incoming mail header content, trust is whether to trust the person or
not
```

```
 **/
DBManager.addTrustMail = function(mail, name, trust, interactions){
        TrustCenter.trustHashMap[mail] = trust;
        TrustCenter.interactionsHashMap[mail] = interactions;

   let query = "INSERT INTO 'mails' (mail,trusted,interactions) VALUES ('"+mail+"','"+trust+"','"+
interactions+"')";
        try{
                mDBConn.executeSimpleSQL(query);
        }catch(e){dump(e+"\n");}
        TrustCenter.nameHashMap[mail] = new Object();
        DBManager.addNametoMail(mail, name);
}

/*
 * Adds a domain to the trustCenter and to the Database
 * @param: domain is the domain and trust is whether to trust it
 **/
DBManager.addTrustDomain = function(domain, trust){
        TrustCenter.Whitelist[domain] = trust;

   let query = "INSERT INTO 'whitelist' (domain,trusted) VALUES ('"+domain+"','"+trust+"')";
   mDBConn.executeSimpleSQL(query);
}

/*
 * Adds a name to a contact based on id
 **/
DBManager.addNametoMail = function(mail, name){
        if(name == "") return;
        let query = "INSERT INTO 'contacts' (mail,name) VALUES ('"+mail+"','"+name+"')";
        TrustCenter.nameHashMap[mail][name] = true;
   mDBConn.executeSimpleSQL(query);
}

/*
 * Manages intercepted outgoing e-mails
 * and checks whether all contacts are trusted or not
 **/
DBManager.registerOutgoing = function(msgCompFields, cmpWindow){

        var recipient;
        var recipients = parseMimeLine(msgCompFields.to + "," + msgCompFields.cc + "," +
msgCompFields.bcc);
        var untrusted = false;

        // email absenden stoppen mit: return true;
        for(var i=0; i < recipients.length; i++){
                recipient = recipients[i];
                let mail = recipient.email;

                if(typeof(TrustCenter.trustHashMap[mail]) == 'undefined'){
                        //mail send to a new contact
                        //if domain in whitelist add it
                        let listed = false;
                for(var white in TrustCenter.Whitelist){
                        if(mail.indexOf(white, mail.length - white.length) !== -1){
                                listed = true;
                                DBManager.addTrustMail(mail, recipient.name,
TrustCenter.Whitelist[white], 1);
                        }
                }
                if(!listed){
                                let prompts =  Cc["@mozilla.org/embedcomp/prompt-
service;1"].getService(Ci.nsIPromptService);
                                if (prompts.confirm(cmpWindow, "Trust Contact?", "You are sending a
mail to an unknown contact:\n" + recipient.name + " <" + recipient.email +">\nWould you like to
trust that Contact in the future?\n")) {
                                        //add the contact to the list of trusted contacts
                                        DBManager.addTrustMail(mail, recipient.name, 'true', 1);
                                } else {
                                        //do nothing since the user doesnt want to have the contact
added
                                }
                        }
                }else{
                        if(TrustCenter.trustHashMap[mail] == 'true'){
                                //mail send to a trusted contact, increment Interactions
                                DBManager.incrementInteractions(mail);
                        }else{
                                //mail send to a untrusted contact
                                //increment Interactions anyway
                                DBManager.incrementInteractions(mail);
                                untrusted = true;
                                var untrustedContact = recipient.name + " <" + recipient.email + ">";
                        }
                }
        }
        if(untrusted){
                let prompts =  Cc["@mozilla.org/embedcomp/prompt-
service;1"].getService(Ci.nsIPromptService);
                if (prompts.confirm(cmpWindow, "Untrusted Contact!", "You are trying to send a mail
to at least one untrusted contact:\n" + untrustedContact +"\nCheck if you disclosed any personal or
secret information.\nConfirm if you want to send the e-mail anyway"))
                        untrusted = false;
        }
        return untrusted;
}


/*
 *  Increments the interactions counter based on the mail
 **/
DBManager.incrementInteractions = function(mail){
        TrustCenter.interactionsHashMap[mail]++;
        let query = "UPDATE 'mails' SET interactions = interactions + 1 WHERE mail = '" + mail + "'";
        mDBConn.executeSimpleSQL(query);
}

/*
 *  Deletes the old entry for a mail and changes it to the new desired one
 *  Copys all names for the contact to new entrys in the contact database
 **/
DBManager.editContact = function(oldmail, mail, interactions, trust){
        DBManager.delContact(oldmail);
        DBManager.addTrustMail(mail, "", trust, interactions);
        //Copy the names
        let query = "UPDATE 'contacts' SET mail = '"+mail+"' WHERE mail = '" + oldmail + "'";
        mDBConn.executeSimpleSQL(query);
}

/*
 *  edits a entry for a domain
 **/
DBManager.editWhitelist = function(olddomain, domain, trust){
        DBManager.delTrustDomain(olddomain);
        DBManager.addTrustDomain(domain, trust);
}

/*
 *  Deletes a contact, but not its corresponding names
 **/
DBManager.delContact = function(mail){
        TrustCenter.trustHashMap[mail] = undefined;
        delete TrustCenter.trustHashMap[mail];
        TrustCenter.interactionsHashMap[mail] = undefined;
        delete TrustCenter.interactionsHashMap[mail];
        let query = "DELETE FROM 'mails' WHERE mail = '"+mail+"'";
        mDBConn.executeSimpleSQL(query);
}

/*
 *  Deletes an entry from the whitelist Database
 **/
DBManager.delTrustDomain = function(domain){
        TrustCenter.Whitelist[domain] = undefined;
        delete TrustCenter.Whitelist[domain];
        let query = "DELETE FROM 'whitelist' WHERE domain = '"+domain+"'";
        mDBConn.executeSimpleSQL(query);
}

DBManager.copyNames = function(from, to){
        for(var name in TrustCenter.nameHashMap[from]){
                DBManager.addNametoMail(to, name);
        }
}
```

7.1.3. Trustcenter

(modules/trustCenter.js)

```
var EXPORTED_SYMBOLS = ["TrustCenter"];
const {classes: Cc, interfaces: Ci, utils: Cu} = Components;

Cu.import("resource://doppelgangermail/stdlib/msgHdrUtils.js");
Cu.import("resource://doppelgangermail/stdlib/misc.js");


/*
 * Shared ressource contains 2 arrays with data on what email to trust
 * and how many interactions have happened with a contact
 * also some other shared ressources, because of mutual inclusion problems
 **/
var TrustCenter = {
        trustHashMap : new Object(),
        interactionsHashMap : new Object(),
        nameHashMap : new Object(),
        TBWindow : new Object(),
        Learner : new Object(),
        Whitelist : new Object(),
}
```

7.1.4. Processing new e-mail Addresses

```
var EXPORTED_SYMBOLS = ["Learner"];
const {classes: Cc, interfaces: Ci, utils: Cu} = Components;

Cu.import("resource://doppelgangermail/stdlib/msgHdrUtils.js");
Cu.import("resource://doppelgangermail/stdlib/misc.js");
Cu.import("resource://doppelgangermail/trustCenter.js");
Cu.import("resource://doppelgangermail/dbHelper.js");
Cu.import("resource://app/modules/gloda/public.js");

/*
 * Learner provides routines for machinal learning of email bodys and
 * matching them to other contacts
 **/
var Learner = {};
var ready = false;
var headerStack = new Array();

Learner.tryToLearn = function(hdr){
  if(!ready){
    //save message for procession at a later time
    headerStack.push(hdr);
    return;
  }
  var params = {inn:{
    contact : new Object(),
    namesSet: false,
    mailsSet: false,
    names : new Object(),
    mails : new Object,
    bayes : "",
    }
  };

  Learner.findSimilarNames(parseMimeLine(hdr.author)[0].name, params);
  Learner.findSimilarMails(parseMimeLine(hdr.author)[0].email, params);
  params.inn.bayes = Learner.guessContact(hdr);
  params.inn.contact.mail = parseMimeLine(hdr.author)[0].email;
  params.inn.contact.name = parseMimeLine(hdr.author)[0].name;
  params.inn.contact.hdr = hdr;
  TrustCenter.TBWindow.openDialog("chrome://doppelgangermail/content/learner.xul", "",
    "chrome, dialog, modal, centerscreen, resizable=no", params).focus();
}

/*
 *  Teaches all message contents to our bayes classifier on startup
 **/
Learner.getLearnContent = function(){
        //teach all the messages

  //get all the accounts, then the first subfolders and then all messages of these
  //then teach the messages as class "mailaddress"
  var acctMgr = Cc["@mozilla.org/messenger/account-
manager;1"].getService(Ci.nsIMsgAccountManager);
  var accounts = acctMgr.accounts;
  for (var i = 0; i < accounts.Count(); i++) {
    var account = accounts.QueryElementAt(i, Ci.nsIMsgAccount);
    var rootFolder = account.incomingServer.rootFolder; // nsIMsgFolder
    if (rootFolder.hasSubFolders) {
      var subFolders = rootFolder.subFolders; // nsIMsgFolder
      while(subFolders.hasMoreElements()) {
        var theFolder = subFolders.getNext().QueryInterface(Ci.nsIMsgFolder);
        var msgArray = theFolder.messages;
        while( msgArray.hasMoreElements() ) {
          let msgHdr = msgArray.getNext().QueryInterface(Ci.nsIMsgDBHdr);
          Learner.teachAs(msgHdr);
        }
      }
    }
  }
  ready = true;
  while(headerStack.length > 0)
    Learner.tryToLearn(headerStack.pop());
}

//Teaches a message body as belonging to a class (e-mail)
Learner.teachAs = function(msgHdr){
  //teach a certain message
  var doc = msgHdrToMessageBody(msgHdr, true, -1);
  var tokens = scrub(doc);
  for (var i in tokens)
    bayes.teach(tokens[i], parseMimeLine(msgHdr.author)[0].email);
}
```

```
/*
 *  Finds names which are close to the authors name using levensthein
 *  @param the name of the autor which name should be found
 *  @result object containing mail addresses and close names
 *    @like: new object[mail][names]
 **/
Learner.findSimilarNames = function(name, params){
  if(name == "")
    return false;

  for(var mail in TrustCenter.trustHashMap){
    for(var compareName in TrustCenter.nameHashMap[mail]){
      if(Learner.isSimilar(compareName, name, 0.4)){
        params.inn.namesSet = true;
        let i=1;
        while(params.inn.names[compareName]){
          i++;
          if(i==2)
            compareName = compareName + i;
          else{
            //naive annahme von maximal 9 gleichen namen...
            compareName = compareName.substring(0, compareName.length - 2);
            compareName = compareName + i;
          }
        }

        params.inn.names[compareName] = new Object();
        params.inn.names[compareName].trust = TrustCenter.trustHashMap[mail];
        params.inn.names[compareName].interactions = TrustCenter.interactionsHashMap[mail];
        params.inn.names[compareName].mail = mail;
      }
    }
  }
}

/*
 *  Checks whether there are similar E-mail Adresses in our Trustbase
 **/
Learner.findSimilarMails = function(mail, params){
  for(var compareMail in TrustCenter.trustHashMap){
    if(Learner.isSimilar(compareMail, mail, 0.2)){
        params.inn.mailsSet = true;
        params.inn.mails[compareMail] = new Object();
        params.inn.mails[compareMail].trust = TrustCenter.trustHashMap[mail];
        params.inn.mails[compareMail].interactions = TrustCenter.interactionsHashMap[mail];
        //only set Name if an entry exists
        if(TrustCenter.nameHashMap[compareMail])
          for (var name in TrustCenter.nameHashMap[compareMail])
            params.inn.mails[compareMail].name = name;
    }
  }
}
```

```
/*
 * Guesses contact based on naive bayes classifiers
 **/
Learner.guessContact = function(hdr){
  var doc = msgHdrToMessageBody(hdr, true, -1);
  var tokens = scrub(doc);
  var probs = new Array();
  var probSum = 0;
  for (var cls in bayes.classes) {
    if (cls == "__length") continue;
    var prob = bayes.probabilityOfClassGivenDocument(cls, tokens);
    probSum += prob;
    probs[probs.length] = {
      classification : cls,
      probability : prob,
      pc : bayes.probabilityOfClass(cls)
    };
  }
  // normalize
  // the log normalization conversion took me quite a bit of googling to figure out.  I'm still not sure
how
  // accurate the math is.  Probabilities are adding up to 1 though, so I must be nearly right.
  var matchesPc = true;
  for (var i = 0; i < probs.length; i++) {
    if (USE_LOGS)
      probs[i].probability = 1 / (1 + Math.pow(Math.E, probSum-(2*probs[i].probability)));
    else if (probSum > 0)
      probs[i].probability = probs[i].probability / probSum;

    // in really bad cases, we may be given a text containing tokens that have never been seen before
for any
    // classification.  In that cases, bayes craps out (because of the unseen seed value producing a very
unlikely
    // probability) and returns a probability distribution that matches almost exactly the probability
distribution of
    // p(C).  Recognize this bad case and shunt all probabilities to zero where they belong.
    var ratio = probs[i].probability / probs[i].pc;
    if (ratio < 0.95 || ratio > 1.05)
      matchesPc = false;
  }
  if (matchesPc || probSum == 0) {
    return "It was not possible to determine a fitting already known Contact to this message.";
  }
  //teach this message for next time
  for (var i in tokens)
    bayes.teach(tokens[i], parseMimeLine(hdr.author)[0].email);

  // sort them highest first.
  probs.sort(function(a, b) {
    return b.probability - a.probability;
  });
  // convert to a pretty string.
  var str = "";
  for (var i = 0; i < probs.length && i<5 && probs[i].probability.toFixed(4) != 0; i++){
    let trust = "?";
    if(typeof(TrustCenter.trustHashMap[probs[i].classification]) != 'undefined')
      trust = (TrustCenter.trustHashMap[probs[i].classification] == "true") ? "Trusted" : "Untrusted";

    str += trust + " - " + probs[i].classification + "(" + probs[i].probability.toFixed(4) + "), \n";
  }
  return str;
}

/*
 *  Determines whether two strings are similar
 **/
Learner.isSimilar = function(str1, str2, threshold){
            if(str1 == "" || str2 == "")
                        return false;
  let similar = false;
  names1 = str1.split(" ");
  names2 = str2.split(" ");
  //try to combine all parts of the names with each other
  for(var i in names1){
    if(names1[i] == "")
      continue;
    for(var j in names2){
      if(names2[j] == "")
        continue;
        let _length = Math.min(names1[i].length, names2[j].length);
        //adjust the value here to allow for more (higher) or less (smaller) names to be accepted
        if(Learner.levensthein(names1[i], names2[j]) / _length  <= threshold)
          similar = true;
    }
  }
            return similar;
}

//lastly a modification added to the Naïve bayes implementation (note the max depth of 100 words):

function strip_common_words(tokens) {
    var newTokens = new Array();
    var ntPos = 0;
    var depth;
    if(tokens.length > 100)
      depth = 100;
    else
      depth = tokens.length;
    for (var i = 0; i < depth; i++) {  // original: tokens.length; i++) {
      var common = false;
      for (var cw = 0; cw < COMMON_WORDS.length; cw++) {
        if (COMMON_WORDS[cw] == tokens[i]) {
          common = true;
          break;
        }
      }
      if (!common)
        newTokens[ntPos++] = tokens[i];
    }
    return newTokens;
  }
```