The scam filter that fights back

Final Report

by

Aurél Bánsági (4251342)
Ruben Bes (4227492)
Zilla Garama (4217675)
Louis Gosschalk (4214528)

BACKFIRE

Project Duration: April 18, 2016 – June 17, 2016
Project Owner: David Beijer, Peeeks
Project Coach: Geert-Jan Houben, TU Delft

TU Delft

Preface

This report concludes the project comprising of the software development of a tool that makes replying to scam mails fast and easy. This project was part of the course TI3806 Bachelor Project in the fourth quarter of the acadamic year of 2015-2016 and was commissioned by David Beijer, CEO of Peeeks. The aim of this report is documenting the complete development process and presenting the results to the Bachelor Project Committee. During the past ten weeks, research was conducted and assessments were made to come to a design and was then followed by an implementation.

We would like to thank our coach, Geert-Jan Houben, for his advice and supervision during the project.

Aurél Bánsági *(4251342)*
Ruben Bes *(4227492)*
Zilla Garama *(4217675)*
Louis Gosschalk *(4214528)*
June 29, 2016

Abstract

Filtering out scam mails and deleting them does not hurt a scammer in any way, but wasting their time by sending fake replies does. As the amount of replies grows, the amount of time a scammer has to spend responding to e-mails increases. This bachelor project has created a revolutionary system that recommends replies that can be sent to answer a scam mail. This makes it very easy for users to waste a scammer's time and destroy their business model. The project has a unique adaptation of the Elo rating system, which is a very popular algorithm used by systems ranging from chess to League of Legends. A Google Chrome plugin was developed with user friendliness in mind, to let people reply to scam mails quickly. To gather replies, a crowdsourcing website, which also trains the Elo system, was set up. Finally, a Python server connects all the parts, allowing for cross platform communication.

Contents

Appendices

Introduction

This document is the final report of *the scam filter that fights back* project and will document the orientation, aspects of the system and development process.

The Assignment

The project started with the client, David Beijer, who became interested in fighting scam after watching a TED talk by James Veitch[1]. He wanted to make a system that would automatically reply to scam mails for people, and thus have an automated conversation with scammers.

However, prior to the start of the project, the bachelor-end project coordinators expressed their worries concerning privacy issues that the project may encounter. The coordinators were afraid that the scammers would retaliate against the creators of the program that are ruining their business.

This resulted in the assignment changing to a system that will suggest possible replies to the user. The user would still choose the reply and click on send. This way, a larger group of people is responsible, making it more difficult for the scammers to retaliate.

The Solution

With this assignment, a system was designed that consists of four major parts. The system needs to be able to process emails, and recommend replies that the system thinks are fit for the email. The system finds these replies through a recommender algorithm based on the Elo rating system. To get good replies and to build a training set, a crowdsourcing platform was developed as well. Finally, to make the system usable by the users, a plugin was developed for Google Chrome and Gmail. The concept of the solution will be explained more extensively in section 1.3.

The key selling points of the resulting product can be listed as the following headlines:

- (section 1.3) Revolutionary new theory on scam fighting
- (section 2.3) A unique take on and use of the Elo rating system
- (section 3.1) Python server for cross-platform communication
- (subsection 3.1.2) Python processing server call handling
- (subsection 5.1.2) User friendliness
- (subsection 5.1.1) Email body processing

[1] `https://www.youtube.com/watch?v=_QdPW8JrYzQ`

Report Structure

This report is split into three distinct parts, the context, the system, and the process. The context contains an orientation, which will give a formal description of the problem, list existing technologies and offer a concept of the solution.

The next section, the system, describes the subsystems of the solution mentioned above. First the server will be described, starting with the recommender system and then describing the other aspects of the server. Then an overview of the crowdsourcing platform will be given, before ending with a description of the plugin.

The final section concerns the process itself, starting with describing unforeseen circumstances during the development. As the system needs to be very user-friendly, several user tests were done, the feedback and changes resulting from this are described in the next chapter. To conclude, some final remarks will be made, including an evaluation and discussion.

The Context

1

Orientation

In this chapter an overview of the problem will be given. First, an analysis of the problem will be given, and then the existing technologies will be discussed. By keeping these two topics in mind, a concept of the solution will be offered.

1.1. Problem Analysis

In this section a formal description of the problem will be given. This description will then be further extended by stating prerequisites set by the product owner and setting goals for when the product is seen as completed.

1.1.1. Problem Description

Scammers try to make a living by baiting people into sending them money. They reach out to these people by sending unsolicited emails. While only about 1% of people reply to these emails, about 1 out of 20 people who reply wire an average of $7.500 USD.

This analysis of the scammers' methods shows a vulnerability. Their system depends on the fact that only a small portion of the people reply, and the people who do reply have a fair chance of sending money. With such a success rate, scamming will remain lucrative and will most likely not simply cease to exist.

Stopping this way of scamming is difficult and a perfect solution is yet to be found. The take of the project on this is that scammers can be forced to stop their activities by diminishing their success rate. The plan is to semi-automatically send human-like responses to scam mails.

1.1.2. Prerequisites

The product owner has not stated any formal prerequisites, but would prefer the project to be made in Python and running on the Amazon Web Services platform as he and his company Peeeks have prior experience with this.

1.1.3. Target Audience

At the moment there are only a few communities devoted to scam baiting (as explained in subsection 1.2.1). These communities are part of the target audience of the project, but the focus also lies on getting people who normally would ignore a scam mail to use the plugin. To make the system easy to use even for people without technical skill is not part of the project scope, but the future results of the product greatly increase if its widely usable by lesser skilled users. For now, the target audience is people with technical understanding (the ability to install a plugin and configure an alias in Gmail) and an interest in scam baiting.

1.1.4. Goals

The goal of this project is to reduce the damages done by scammers by building an extension that allows users to quickly respond to scam mails with the intent to flood their inbox and making it difficult to assess which response is real and which is sent by our tool. If scammers can not discern whether a response is sent by a victim or our tool, they will waste their time by responding to the wrong mails, making their business model less effective.

1.2. Existing Technologies

In this section existing technologies will be discussed. It is vital to know what already exists, so unnecessary work can be prevented. First scambaiting itself will be looked at, and then email autorepliers will be discussed. For further information, see appendix E.

1.2.1. Scambaiting

There are two websites that are the main source of information on scam baiting, namely 419eater[1] and thescambaiter[2]. Here scambaiters can find information, ask questions and post entire conversations with scammers. However everything is done by hand as there are no tools on these websites that can help you with this. To the best of our knowledge there only exists one tool that can do that. This tool, called Autobait[3], is a bot that automatically replies to scam emails trying to keep the scammers talking. This tool is however private and the source code can thus not be used or researched. It is sadly not possible to research this tool.

1.2.2. Email auto repliers

An email auto replier is a tool that will either reply mails by itself, or offer possible replies from which the user can select the one that they find suitable.

The first way fully automates the process, and requires little more effort than clicking a button to start the process. An example of such a system is SlightlyCyborg[4], a system in which the user first has to label an email as ”annoying”. This application will then pick up this email and make an automatic response for it.

[1] www.419eater.com
[2] www.thescambaiter.com
[3] www.autobait.com
[4] www.github.com/SlightlyCyborg/Email-Autoresponder

The second way is used by Google's SmartReply from Inbox by Gmail. It provides three different replies to an email which the user can then choose from. This way the chance of errors are reduced, and the input of the user can be used to further train the system.

1.3. Concept

Given the lack of results of the search for existing technologies, it can be said fairly safely that the project idea as described in the project description does not exist yet. This section will describe the structure of the product by means of its technical components.

1.3.1. General Idea

The idea of the product is to automate conversations with scammers. This is something that has been done before, but all previous ideas have been executed by creating an artificial intelligence bot which makes up its own replies. What makes the BackFire idea unique is the fact that its replies are not generated but gathered through crowdsourcing, which gives the opportunity to create a totally different system than a mailing bot. Having a distinct crowdsourcing website existing solely for gathering scam mails and replies makes it possible to build a system based on independent components. Creating such independent components makes a system easier to maintain.

The key selling points of the resulting product can be listed as the following headlines:

- (section 1.3) Revolutionary new theory on scam fighting
- (section 2.3) A unique take on and use of the Elo rating system
- (section 3.1) Python server for cross-platform communication
- (subsection 3.1.2) Python processing server call handling
- (subsection 5.1.2) User friendliness
- (subsection 5.1.1) Email body processing

1.3.2. Component Description

Breaking the product down into its core pieces would result into four different components, namely the MongoDB server, the Python processing server, the crowdsourcing platform and the email plugin. MongoDB is purely used for data storage, it has the possibility to store functions which can then be called but this appeared to only support core functions such as adding and subtracting. Functions are therefore being executed on a standalone Python processing server which communicates with the other components through a websocket. This python processing server subsequently processes the data, e.g. extracts keywords or calculates ratings, and inputs this in the database.

The crowdsourcing platform is created to gather mails and replies and to train the rating system on this gathered data through a page where users can rate a reply in relation to a scam mail. This works in such a way that users simply connect to a website and choose to train the system.

Thereafter they receive a random mail and reply from the MongoDB server and give this combination a compatibility rating of ’inapplicable’, ’applicable’ or ’perfect’.

Lastly, the plugin is created as the user interface, enabling a user to create, semi-automatic conversations. The plugin is downloaded from the Chrome Web Store and allows the user to apply an alias for conversing and generate a fake name. After selecting a mail, the plugin gathers four replies from the database which are recommended for the chosen mail.

The System

2

Rating system

The goal of the system is to recommend replies to scam mails. In order to do this efficiently a system is needed that can decide whether a certain reply is ”good” or ”bad” for this email. First the keywords that categorize the scam mails will be discussed. Second the Elo Rating system will be explained and the way the Elo Rating system was adapted to suit the needs of the system will be discussed. Finally, the way the recommender system retrieves replies will be explained.

2.1. Keywords

The system needs to be able to categorize emails such that the system knows which reply works well for which type of scam mail. The system does this by extracting keywords from a scam mail every time it is submitted to the database. The Natural Language ToolKit (NLTK) is used to categorize each word by its grammatical type. The system then selects the nouns because they are crucial for the meaning of the sentence and they have only two conjugations (singular and plural).

However, before the system can start extracting keywords, everything is converted to lowercase characters and email addresses, numbers, special characters and words smaller than 3 characters are removed. This is because scammers often write emails in full caps and NLTK recognizes everything that is capitalized and not at the beginning of the sentence as names. NLTK also does not support special characters and when you do insert them, the toolkit crashes. Email addresses, numbers and words smaller than 3 characters are never suitable keywords so it is useful to just remove them immediately.

2.2. Elo Rating System

The Elo rating system is a method for calculating the relative skills of professional chess players. However it is used in many other multiplayer competition games as well. Even Facemash, the precursor of Facebook, used it. This system is known to be really fair and even though it does not seem like it at first, suits our needs very well. It works as follows:

After, for instance a chess game, a player's score has to be updated. Let us call this player player A. Its new rank R'_A is the old rank R_A plus a weight (K) times the scored points (S_A) minus the expected points scored (E_A). In equation form this will be:

$$R'_A = R_A + K(S_A - E_A) \tag{2.1}$$

The expected score is calculated using the following equation:

$$E_A = \frac{1}{1 + 10^{R_B - R_A/400}} \tag{2.2}$$

Here, R_B denotes the rank of player A's opponent, let us call this opponent player B. With this system, when a player with a high rank loses to a player with a low rank, the player with the low rank will receive a much higher rank than before and the player with the high rank will not lose too many points. If a player with a high rank wins from a player with a low rank than the winner will not receive too many points and the loser will not lose too many points because this outcome was expected. This ensures that a player quickly attains the rank suited for his skill level.

2.3. Recommendation system

The Elo Rating system was adapted to suit the needs of the system, because the system is not hosting a competition but is instead a recommendation system for email replies. The system recommends four replies to a scam mail. When a user chooses to send a certain reply, this reply ”wins” over the other replies. So instead of two players the system has four ”players” (replies) competing against each other. Because the system know which reply wins and which lose, the system can compute the ranks of each reply as if the winning reply played a match against every other reply. The ranks are different for every keyword and thus it is actually the keywords of the winning reply that compete against the keywords of the losing replies.

A low rank for a certain keyword means that the reply lost a lot of competitions for this type of email. This reply should thus not be recommended for this type of email. One could say that it is a ”bad” reply. Similarly, a high rank means that it is a ”good” reply and should be recommended.

When a reply does not contain a keyword from the scam mail, the keyword will be added to the reply. The initial rank of this keyword is set to 1200 as a balancing point for our system. This beginning rank is deliberately not low, but in the middle, as else a new reply would be marked as a ”bad” reply right away. That would mean that it would not be recommended and new replies would not get the chance to prove themselves. If the server would give the new keyword a high number then all new replies would be marked as ”good” replies immediately. This would mean that ”better” replies may not be recommended only because someone wrote a new reply.

2.4. Retrieving replies

After the Elo system was adapted to suit the needs of the system, a retrieval algorithm had to be developed. A key point of this algorithm was that it had to be diverse enough to allow newly created replies to get a proper ranking, but also stable enough to always return fitting replies as well. In order to find a balance, the choice was made to split evenly between the best replies, and

completely random replies.

When the server receives an email, it will first generate a list of keywords from the email, based on the rules mentioned in section 2.1. It will then send the keywords to the recommender system, asking for a number of replies.
The recommender system will then first find half the number of best replies. It does this by first getting all replies that have any ranking for any keyword that was found in the email. It will then calculate the mean score of all the keywords that the reply has a ranking for, and sorts the list. The top replies are then returned. In case the system found less than the required number of elements, it will also return a number indicating how many elements are missing.
When the best possible replies have been found, the system will then find random replies. Ensured is that the replies found earlier are excluded from this, so that there will be no duplicate entries. The amount of random replies is by default half the number of elements needed, but can be greater in case there were less best elements than were needed.

After the recommender system has retrieved the replies, it will return them back to the server for further processing.

3

Storage & Processing Servers

While the plugin is the most important part for the users, the Python server containing all the logic is the heart of the project. In this chapter, the different parts of the server will be discussed. First a general explanation of the structure will be given, then each layer will be explained and finally the database will be highlighted.

3.1. Python Server

The server must be capable of a number of things. Most importantly, it has to be connected to the plugin and crowdsourcing platform and requires a clean way of sending data back and forth. Next, incoming data must be processed and the correct data has to be returned to the plugin or crowdsourcing platform. This data can be a request for replies, but it could also contain emails that must be inserted into the database. This will be decided in the JSON processor, after which the managers handle all logic and operations needed before the database can be accessed. A high level overview can be found in figure 3.1.

3.1.1. Communication

Finding a clean way of communication between plugin or crowdsourcing platform proved to be a struggle. Since there must be more projects that require such a connection, a search for existing solutions was started to avoid having to reinvent the wheel.

Eventually it became quite clear that WebSockets[1] would be the solution to this non-trivial problem. This made it possible to open a connection between the different systems. While JavaScript supports WebSockets out of the box, the same can not be said for Python. At first, an attempt was made to implement WebSockets from scratch. This meant that a handshake and other functions had to be developed on our own. It quickly became apparent that this would take a lot of effort, so a different solution that would work out of the box had to be found. After some searching, Autobahn[2] was found, which provides an open source implementation of the WebSockets protocol.

[1] `https://www.websocket.org/aboutwebsocket.html`
[2] `http://autobahn.ws/python/`

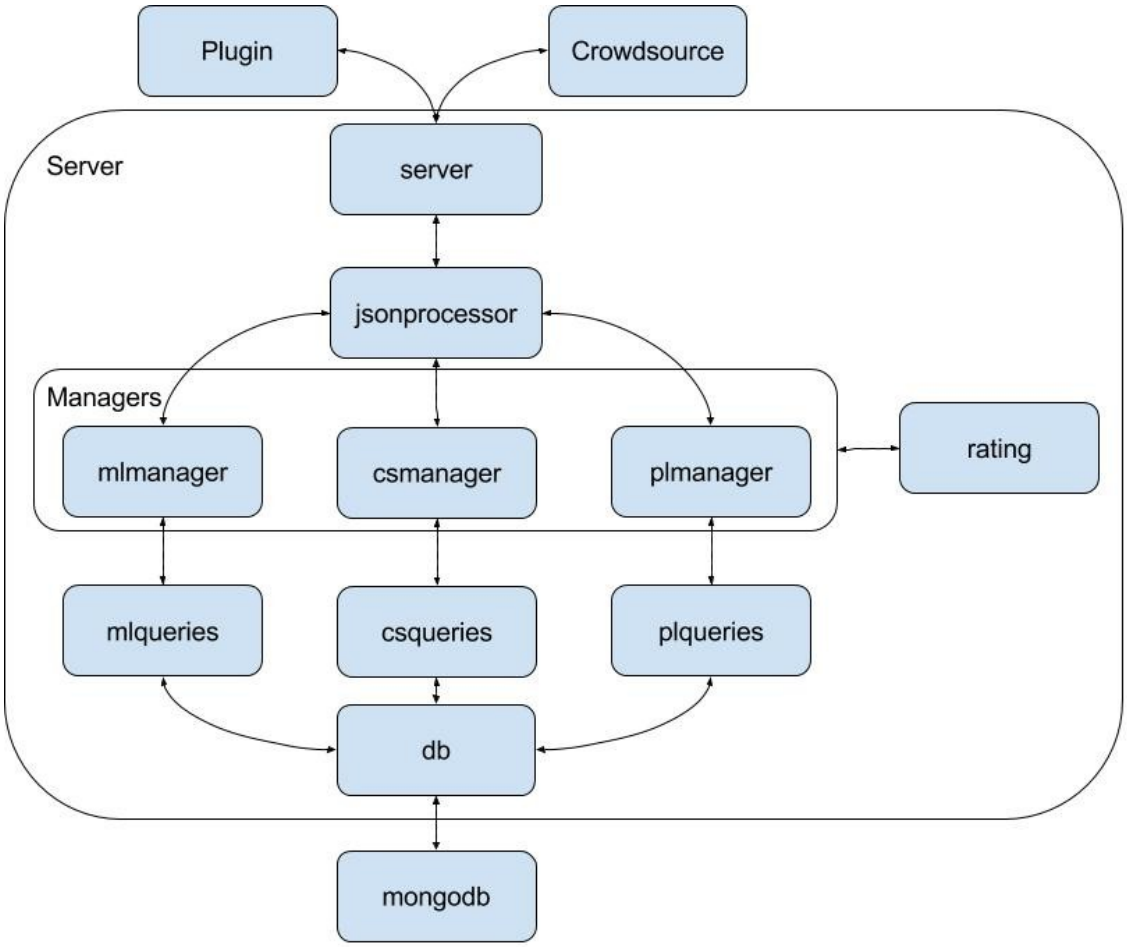


Figure 3.1: The structure of the project

Using Autobahn, all the necessary communication options are available.

The next step was coming up with a flexible way of letting the server know which functions should be called. Since both the plugin and MongoDB use JSON objects, combined with the flexibility of JSON, the decision to use JSON objects for this was not very difficult. Every JSON object sent through the WebSocket was to have a field depicting its type, resulting in a specific function being called.

3.1.2. Processing Incoming Data

As mentioned above, the server communicates by sending and receiving JSON objects, encoded as UTF-8 strings. Depending on the field ”type”, different functions can be called. An example of such an object is shown in listing 1.

```
jsonobject = {”type”: ”replytodatabase”,
              ”body”: ”Foo Bar”,
              ”mail”: 1}
```

Listing 1: Example of a JSON object received by the server.

Here, the type indicates that this is a reply to some scam mail with index = 1, and its body should be inserted into the database. Early in development, only a few types were used and using a couple of if-statements to call the correct functions was sufficient. However, as the number of types grew, this became cluttered. For this, an elegant solution was found. As seen in listing 2, instead of twelve if-statements, a single switcher object was created. This object is a dictionary containing all different types as keys, and functions as values. This way, the JSON processor can be called for every type, and the correct function will then be called. These functions extract the necessary values from the JSON object and then call the appropriate manager.

```
def process_json(message):
    """
    Process the incoming JSON objects and call
    the correct managers based on the field 'type' of the object.

    :param message: The incoming json string
    :return: Returns the requested data, or an empty string as json
    """
    obj = json.loads(message.decode('utf8'))
    if 'type' not in obj:
        return JSONEncoder().encode("")
    switcher = {
        "requestmailreplies": get_replies,
        "requestprofile": get_profile,
        "requestreply": get_reply,
        "requestnewname": generate_new_name,
        "scammailtodatabase": insert_scammail,
        "requestnewreplies": get_new_replies,
        "reportandfetchreply": report_reply,
        "ratingtoelo": rating_to_elo,
        "replytodatabase": insert_reply,
        "reporttodatabase": report_mail,
        "sentmailreply": process_chosen_reply,
        "updatesendtoaddress": update_address
    }

    func = switcher.get(obj['type'], lambda(x): {})
    return JSONEncoder().encode(func(obj))
```

Listing 2: JSON processor code snippet.

3.1.3. Managers and Rating

The managers handle all the logic before a call to the database can be made and before ratings can be done. There are three managers that can be called, one for all plugin-related calls, one for the crowdsourcing platform, and one meant for extracting keywords and the recommendation system. These managers then either call the recommendation system or one of the query builders.

3.1.4. Query Builders

The query builders receive necessary data from the managers and then build the query needed for retrieving or inserting the relevant data. Next, the database class is called, passing the filter and other parameters. The result is then passed back to the managers, which passes it to the JSON processor. The JSON processor then encodes the data as a UTF-8 string and sends it to the server. Finally, the server sends a message through the WebSocket, either to the plugin or

crowdsourcing platform.

3.2. Database

In this section, the reasons for choosing MongoDB will be explained, as well as our database design.

3.2.1. NoSQL and MongoDB

Even though the choice for NoSQL and MongoDB is explained in appendix E, it will still be shortly elaborated here.

NoSQL was a must because replies have a varying amount of keywords, and users have ever changing conversations. If regular SQL were to be used, every relation would require a join table, resulting in an inefficient database with many large join tables. There are multiple types of NoSQL database, however a key-document structure matches our interests the best. Since this type of NoSQL uses documents which are simply JSON objects, nested relations are easy to create, update and search through.

MongoDB was the best fit for this project for a number of reasons. It has excellent support for Python and runs on all platforms. Another positive aspect is that it was available for free and the client has experience with MongoDB as well.

3.2.2. Database Design

The database contains four collections: scammers, replies, scam mails and users. These collections and their properties can be found in listing 3. To elaborate on this design, each collection and why their properties are necessary will be explained.

Scammers
This collection is used for the identification of scammers, by keeping track of their (hashed) email addresses. The id of the scammer can then be used to keep track of the conversations users have. Section 5.1.1 will explain why multiple addresses are stored.

Replies
This collection contains all the replies. The only property that is not self-explanatory is the keywords property. This is a dictionary with keywords as keys, and their assigned scores as values. The index is used to be able to get a random reply, as MongoDB does not support the fetching of a random document. Index always ranges from 1 to the amount of documents in this collection.

Scammail
This collection contains all the scam mails. Similar to replies, the index is used to get a random document. Contrary to replies, keywords is not a dictionary but merely a list of keywords, as no values are assigned to keywords of a scam mail.

User
This collection contains all the users of the plugin. This is required to maintain conversations and gather metrics. The idhash of the user is their hashed mail address, this is used to identify them.

Each user has a randomly generated fake name and can specify a mail address to send their mails from. The most important property is convs, containing all the conversations of a user. It is a dictionary with the id of a scammer as a key, and as a value a list of ids of scam mails and replies. This way it is possible to keep track of which scam mails the user replied to and which reply they chose.

```
scammers: {
    "_id": ObjectId,
    "addresses": [256-bit hash]<1...3>
}

replies: {
    "_id": ObjectId,
    "body": String,
    "index": Integer,
    "used": Integer,
    "report": Integer,
    "keywords": {
        String: Integer
    }<0...*>
}

scammail: {
    "_id": ObjectId,
    "body": String,
    "index": Integer,
    "report": Integer
    "keywords": [String]<0...*>
}

user: {
    "_id": ObjectId,
    "fakename": String,
    "sendtoaddress": String,
    "idhash": [256-bit hash],
    "convs": {
        scammers._id: [replies._id/scammail._id]<0...*>
    }<0...*>
}
```

Listing 3: Database Design expressed as JSON.

4

Crowdsourcing Platform

To get the system started it needs example mails, replies and a way to train the recommendation system on this dataset. The crowdsourcing platform creates an environment for uploading example mails, creating replies to these mails and rating the quality of a reply when combined with one of the mails. Of course, this platform also provides the possibility to display some neat metrics.

4.1. Purpose

The reason a crowdsourcing platform is useful in this project is due to the fact that the feedback from a plugin on its own would be insufficient to create realistic replies. The feedback from the plugin is limited to replies being chosen to be sent or inserting a new reply manually. If, in the future, the plugin would be set to reply automatically, there would not even be any feedback from the plugin.

4.1.1. Training the System

Due to the lack of feedback from the plugin, an additional tool is required to train the system. The largest deficiency of the system without the crowdsourcing platform would be the fact that, especially when set to automatic, its feedback is not specific enough or even non-existent. The only feedback you get from the plugin is which reply has been chosen and perhaps a reported reply. In the use of the plugin there is always one winner and multiple losers. To create a broader opportunity for new replies with low scores to rise to their potential, the crowdsource platform offers the rating page. On this page users give feedback on a certain mail and reply combination, giving a more specific feedback on the quality of one reply.

Thus the main purpose of the crowdsourcing platform is to create specific feedback while also gathering new mails and replies.

4.1.2. Metrics

Since a website is an easily accessible platform, it creates the perfect environment for creating a central display of system statistics. At the end of the project when some results started rolling

in, the project team deemed that having a display of metrics would be a valuable addition to the crowdsourcing platform. The client agreed that this would indeed be the case, thus a metrics page was created.

As the python processing server incorporated some clean-up functions which had yet to find a place in the system to be called, the metrics page proved to be the perfect location. Buttons were added to the metrics page which would, when clicked, clean up the database or re-calculate the system metrics. Since some of these calculations would get rather heavy as the database grows, a couple of these calculations have been set to run exclusively if their last run was twenty-four hours ago. An example of a function like this is cleaning up the keywords of the replies, which walks through all the replies and checks all keywords of each reply and deletes all keywords between the scores of eleven-hundred and thirteen-hundred. This would get incredibly intensive with a large database because it walks through every single reply and for each reply it walks through every single keyword to check its score. Deleting the set of keywords within this range of scores is the best way to clean up the keyword set since a keyword is added with a score of twelve-hundred. This means that if a keyword is stuck with a score of around twelve-hundred, it is neither very good nor very bad and thus not significant for the reply.

Standard users of the crowdsourcing site are not supposed to find this page so there is no direct link to it. It seemed to be overkill to add login functionality to protect it but it is still preferred to keep it private, so it can only be accessed by knowing the URL.

4.2. Design

In the opinion of the team, a successful crowdsourcing page is supposed to be easy to use and straightforward. Thus, while designing the page the emphasis was on creating a user-friendly simplistic page. Though of lesser importance than smooth functionality, design was a key feature.

4.2.1. Interface

Bootstrap[1] was chosen because of its possibility to quickly develop a modern page, resulting in an extensible system which is also easy to modify for a developer who is new to the system since it is a generally well known framework. This is not the only reason to use Bootstrap. Bootstrap offers handy components like the alert boxes which are used to show a user his input has been received successfully. The possibility to collapse the text of the header jumbotron is also a Bootstrap component. All together this creates the perfect environment to develop the crowdsourcing platform.

4.2.2. User friendliness

After a few user tests, it became apparent that the page was going the right way but lacked some clarifications and small features which may greatly increase user friendliness.

The clarifications were added by making a collapsible text area. This proved to be a great way

[1] `http://getbootstrap.com/`

to add details to the page while still keeping it minimalistic. Additional extra features included a character counter for the text areas and alerts for successful actions.

4.3. Communication

Since the data collected by the crowdsourcing platform has to be processed and saved, it requires a couple of communication lines. The main elements of the communication of a website are of course the receiving and sending of data. While this seems trivial, there was a bit more to it than one would expect. The crowdsourcing platform collects full mail bodies which still needs extraction of keywords. This means the output of this module needs to be sent to the python processing server instead of directly sending it to the MongoDB.

4.3.1. Getting Data

Getting the data from the database is pretty straight-forward. It is possible to do this through the python processing server to eliminate the use of a PHP MongoDB driver, but this would put unnecessary stress on the python server since this server is already loaded with processing tasks. Simple tasks such as getting data from the database are therefore done through the PHP MongoDB driver. Also, since this communication can be written in PHP it allows the crowdsource platform to get data from the database locally when both services run on the same server.

4.3.2. Outputting Data

Delivering the data gathered by the crowdsourcing platform is all done through a connection with the python processing server which is set up through a WebSocket. All data from the crowdsourcing platform requires elaborate processing. For example, a simple report of an inappropriate mail or reply requires a check if the existing amount of reports on said instance exceeds a specific amount. If it does, the instance needs to be deleted, otherwise it has to be incremented. This is the simplest example of an output of the crowdsource platform needing processing. Also, since the plugin and crowdsource platform use the same location for processing its data, a lot of functions can be re-used which is very good practice by the means of code quality. The previously stated functionality of reporting a mail or reply is also used by the plugin, giving a relevant example of re-using the functions of the python processing server by multiple platforms.

5

Google Chrome Plugin

In order for users to reply to scam mails with the system, a plugin had to be developed. As seen in appendix E, Google Chrome and Gmail were chosen as development platforms for their market share and available development tools. A key point for the plugin is user friendliness, as it is supposed to make people do something they generally do not want to do, e.g. reply to scam mails. It should be easier and faster than replying by hand. In this section first the design of the system and design choices will be discussed. Then communication between the plugin and Google servers, and the plugin and the own built server will be explained, before ending with a section about external libraries that the plugin uses and listing possible improvements for the future.

5.1. Plugin Design

Design is a key point for the plugin, as it both needs to be easy to use for the user, and the code also needs to be maintainable. This section will thus discuss both system design and user interface, starting with system design.

5.1.1. System design

As mentioned in the introduction, the plugin is developed for Google Chrome and Gmail. This means that the plugin needs to be developed in HTML and JavaScript, and a plugin is basically a small web page rendered as a pop-up in Chrome. As it is a web page, the state of a system, including JavaScript variables among others, is lost when the plugin is closed. Another limitation set by Google is that a plugin is not allowed to swap web pages. This limitation greatly impacts the maintainability of the code as a way around this had to be found. This also means that the different parts in the JavaScript code are more tightly connected than otherwise needed.

Code structure

The code of the plugin is split in three parts; HTML, CSS and JavaScript. Chosen was to use one CSS file for the entire project, for the reason mentioned above that a plugin can only use one web page. To get around the limitation, several, incomplete, HTML pages were made, that could be

loaded into a *div* element in the main page. By toggling the display of the elements, it appears that the plugin shows different pages. Every page also has a JavaScript script associated with it, to control any necessary dynamic elements on the page.

Outside of the pages, there are a few more JavaScript files. One file is run only when the plugin is first installed, or whenever it is updated, and ensures that the plugin will only load when the user is on a Gmail page. The other files are for underlying structure, and are structured in an object oriented way.

Callback functions
Because the plugin makes many calls to both Google and the own server, a lot of code needs to be delayed in execution until the other side has replied. JavaScript solves this issue with *callback functions*. Callback functions are functions passed to a server call that are executed whenever there is a reply back. Often the reply from the server is used as argument for the callback function. The plugin makes extensive use of callbacks to ensure a smooth process. The downside of using callbacks is that the code is no longer linear, meaning it cannot be read from top to bottom while also understanding when what is executed.

Processing the content of an email
One of the parts where callback functions are extensively used is when the plugin retrieves and processes an email so it can be sent to the server for further processing. This process quickly proved to be more difficult than expected, as the data was rarely consistent with other results.

Scammers benefit from spoofing headers, as it can hide their identity, but they still need to make it possible for potential victims to email them relatively easy. So often the headers have up to three potential email addresses, of which usually one or two are not real addresses. To find the correct address, the plugin looks through the headers, and will either select the *Reply-to*, *Return-Path* or *From* header, in that order. Experience has shown that the addresses from the *Reply-to* header are almost always correct, but not always present. The *Return-Path* header is usually incorrect, unless the *Reply-To* header is missing, and the *From* header is rarely correct. In some cases, none of the headers work, which usually means that the scammer put an email address in the mail itself, the plugin does not currently account for this. As the actual email address of a scammer changes often during a conversation, all email addresses are hashed and sent to the server for identification and for finding what conversation the email is part of.

Finding the body of an email is also a very inconsistent process. The object returned by Google sometimes has a *body* field, from where the body of the email can be extracted. Usually however, this field is left empty, meaning the structure of the email has to be searched through. However, because an email can be built in a form of a tree structure, a recursive algorithm had to be made to search through this tree for the correct field. This field can either be plain text, or a HTML page. Wherever the body is found, it will always be encoded in base-64, but not always in the same standard that JavaScript uses for their encoding and decoding functions. To circumvent this, the library Base64.js, mentioned in section 5.3.3, was used. After decoding, the final step before sending the data to the server, is replacing all HTML tags in case the email body was delivered as HTML. Removing these tags is vital for the keyword extraction, as else false results will be

generated.

5.1.2. User Interface

The other part of design is the user interface. Replying to scam mails is usually not something people want to do, rather they are told that you should delete them straight away. The plugin thus needs to break this rule and make people want to reply to scam mails. A major part of this is user friendliness. If the plugin is not easy to use, people will not want to use it. Special care in this regard was taken for colour blind users.

User friendliness

There is some explanation needed for people to use the plugin correctly, especially when the user starts the plugin for the first time. However, there cannot be too much texts, because that leads to annoyance as well. To combat this, all text is a short as it can be, usually explaining what needs to be told in two to three sentences. In line with this, any button will also very shortly tell what they do, and are also colour coded in case there are buttons next to each other. To make the plugin more user friendly, a number of users tests were done, and feedback was processed. This is further explained in chapter 7.

Care for colour blindness

The plugin uses colours quite extensively, as the background is generated by Trianglify, a library that generates triangle art that can be used in various parts on a website. (More info on Trianglify in section 5.3.2). The plugin uses this triangle art for the background of the entire plugin, and for four buttons. As the plugin, of course, still needs to be usable for colour blind people, special care was taken to see if they could still use the plugin without any trouble.

To simulate colour blindness, the program Color Oracle[1] was used to simulate three extreme types of colour blindness, Deuteranopia, Protanopia and Tritanopia. The assumption made by Color Oracle is that if a program is usable with these extreme types, it will certainly work for milder types.

It turned out that most aspects of the program were completely usable with all three types. The only worrying aspects were several buttons that started to blend in with the background on different types of colour blindness. However, the buttons also have a short text explaining their function, and also have icons that attempt to explain what they do. The plugin is very likely to work for colour blind people, although it was not possible to test this with someone that is actually colour blind.

5.2. Communication

Communication to external sources is vital for the plugin, as it needs to request and send data to both Google and the developed server. On Google's side, mainly email data needs to be retrieved, and an email needs to be sent. The server, on the other hand, receives an email and gives replies in return. This section will discuss how both of these interactions are implemented, starting with Google and then going to the server.

[1] http://colororacle.org

5.2.1. Communication with Google

The plugin has a handful of connections with Google API, all related to Gmail. However, in order to access these features, the user first needs to authenticate with Google to allow the plugin to use these features.

Authentication

The plugin uses the Identity Toolkit[2] to handle authentication and will always try to silently authenticate the user. In case the silent authentication process fails, the plugin will explain to the user why authentication is necessary and offer a way to start authenticating. The only time this will happen by default is when the user starts the plugin for the first time.

Requesting data from Google

When authentication is done, the plugin will start to request information of the user. First the current tab's URL is requested. The URL is used to determine whether the user is looking at a mail or not, which allows the plugin to show the correct page. Next the profile of the user is requested to get the users email address. A hashed version of the email address is used as profile name for our own server. Finally, if the user is looking at an email, the email will be requested from Google's server as well. The email is requested by getting the email id from the URL, and then requesting the latest email in the conversation of which the id is part of. This approach ensures that always the latest retrieved email is selected.

Sending data to Google

If the user is viewing an email and has selected a reply, a final request is send to Google, this time by means of an AJAX POST request. This request asks Google to send an email to a given email address with the chosen reply as body.

5.2.2. Communication with the server

Unlike the communication with Google, the connection with the plugin and server was self-made, meaning there was more control over how communication would happen. Like the connection with Google, the plugin sends and retrieves data from the server.

Retrieving user data

When the plugin has retrieved the email address of the user from Google, it will send a hashed version, using a SHA-256 algorithm, to the server to request user data. This data contains the generated name of the user, all conversations they have had with the plugin and, if applicable, a email address the user would prefer to send the replies from, other than his standard email address.

Sending email and retrieving replies

After the user data has been retrieved, and the user has confirmed that they are looking at a scam mail, the plugin will request a number of replies that are suitable for the body. It will send the body and the subject of the email, up to three (hashed versions of) email addresses of the scammer, the id of the user for identification and finally a number to specify how many replies should be returned.

[2] https://developers.google.com/identity/toolkit/

While the user is looking at replies, they can decide to request new replies or report a reply, both will result in new replies being send back to the plugin by the server, and any fields will be updated in the database if applicable. Additionally, the user has the option to write their own reply, but the user is discouraged from writing their own replies with a tooltip. It turns out the self-written replies are usually very specific, based on the email that it was a reply to. This reply, if sent, counts as the winning reply, while all four replies given by the server are counted as losers.

Sending a chosen reply
If the user found a reply suitable to send back to the scammer, the plugin will send the winning reply and losing replies back to the server, so the ranking system can be updated with new points. At this point no reply back to the plugin or any further communication will happen until the user opens the plugin again.

5.3. Used libraries

The plugin makes use of several libraries. in this section an overview will be given of these libraries, and why they are included.

5.3.1. jQuery and Bootstrap

JQuery[3] is a JavaScript framework that builds on standard JavaScript by making, among others, HTML document traversal, manipulation and event handling easier. jQuery is one of the most commonly used frameworks for JavaScript and many libraries or frameworks require jQuery for their functionality.

One of these frameworks, that is also used in the plugin, is Bootstrap[4]. Bootstrap is a very popular HTML, CSS and JavaScript framework that makes making responsive web pages very easy. Because the plugin is a web page, Bootstrap can, and is used, to handle the styling.

5.3.2. Trianglify

Trianglify[5] is a library that generates triangle art that can be used in a web page. The library usually generates completely random backgrounds every time the function is called, but several options exists to make it more customized. Tweaking the options can also allow the library to generate an image slightly faster. In the case of the plugin, Trianglify is used to make a background of the whole page, and several buttons have their looks generated by Trianglify as well.

5.3.3. Base64.js

During development on communication with the servers from Google, noted was that the body of an email was encoded in base64, but not in the standard version of base64 that JavaScript expects. This resulted in either garbled output or in raised exceptions. Luckily, Base64.js[6] was

[3] https://jquery.com
[4] http://getbootstrap.com
[5] http://qrohlf.com/trianglify/
[6] https://github.com/dankogai/js-base64

found. The developer of this library noted the same problem and made a small library that will encode to and decode from base64 in a correct way in almost all cases.

5.3.4. Sha256.js

The last used library is Sha256.js[7], which is used to hash the email addresses of the user, and emails found in scam mails. SHA-2, of which SHA-256 is part, is a set of one of the safest ways to generate a hash. By using a hash, almost no email addresses have to be stored in the database, and the email address associated with the user will never have to be stored.

[7] http://www.movable-type.co.uk/scripts/sha256.html

The Process

6

Unforeseen Circumstances

In the early days of the project, there were some expectations and predictions of how the product should be designed, which were documented in the research report and project plan. However, some of these expectations appeared incorrect, and resulted in slight changes in the process. This chapter will discuss unforeseen and exceptional situations that have occurred during the project and their respective solutions.

6.1. Incorrect Expectations

This section will start off with a description of all situations which turned out to be different than expected.

6.1.1. Package Use

During the research phase a package for crowdsource platforms was found, PyBossa. This platform showed great promises since it incorporates a machine learning algorithm which already processes the crowdsourcing input. This package turned out to be unusable in the project due to lack of extendibility. It turned out to be hard to host, difficult to develop and thus it was decided that it was more efficient to create a custom website for crowdsourcing purposes.

6.1.2. Database Design

The initial design of the structure of the database turned out to be incorrect. This is not uncommon, but is still interesting to point out. While developing the interface, features like reporting stored mails and replies for containing prohibited information were brought up. Another feature was fetching random mails and replies from the database, requiring an indexing. Due to the fact that the MongoDB driver for PHP did not support a function for getting a random element and all items in MongoDB have an ObjectId as identifier, a random number had to be generated and a mail could be retrieved from the database according to this random number, thus requiring a numeric indexing. These two features are the most obvious examples of incorrect expectations requiring a change in design.

6.2. Design Issues

In this section an explanation will be given of issues in the software design.

6.2.1. User Anonymity

While thinking of how to send the actual replies to the scammers, a problem came up concerning the anonymity of the user. Using the standard mail-address of a user may cause the scammer to harass the user as a form of retaliation. This meant a privacy protection feature would have to be developed but the problem in this was to choose which method to use. The two options were to have the user create a new Gmail address as an alias or to create a mail server from which all replies would be sent.

What made the choice between these options difficult was the fact that having users create an additional Gmail account would create unnecessary data-storage on the side of Google since a Gmail account is not only just a mail account but also stores info about every available Google service.

The second option was to create a mail-server through which all replies would be sent. The problem with this option was that this would create an obvious pattern for scammers to recognize a mail from our server hence they would all come from a single, unique domain. Also, this domain would directly lead to the host and creator of the anti-scam service, creating a breach of privacy.

Finding a middle ground was the way to solve this issue, the plugin now allows users to add an existing alias of Gmail to protect their identity. If they do not use this, their original Gmail address will be used to send replies but an extension will be added to their name to make it a bit more abstract and maybe even unrecognisable to the scammer.

6.2.2. Multi Threading

From the start the python processing server was seen as a crucial component which it has actually proven to be. An unforeseen aspect in this component is that after a couple of uses while monitoring the python server, it was noticed that if the server was busy processing a heavy command it did not process additional commands. Commands given while the server was busy appeared to be ignored, it was unforeseen that the server turned out to be single threaded. This is not ideal but in the scope of the project it was not a catastrophe, so the remaining time was put in issues with a higher priority.

7

User Testing

As mentioned in chapter 5, user friendliness is very important. Since replying to scam mails is not something people usually like to do, it should take very little extra effort. The same goes for the crowdsourcing website, as it can cost the user quite some time. To improve user friendliness, several user tests were done near the end of the project. The summary of these tests will be given for both the plugin and the crowdsourcing page. Next, changes that were made because of the user feedback will be discussed. For the full user feedback, see appendix G.

7.1. Plugin

The users generally liked the idea of the plugin, and were enthusiastic about using it in the future. However, all testers were confused by the texts that explain the various parts of the plugin. The texts were usually too long, and did not explain the context well enough. The same goes for several of the buttons in the plugin, as they usually did not explain their function either. The testers also mentioned that some buttons might need a tooltip to explain their function, as the explanation would be longer than what would fit in the button.

The colour scheme was well received, but some noted that the buttons and progress bar clashed with the overall colour scheme.

Another major piece of feedback is that people would like to have the option to write their own replies in the plugin. As a conversation continues, the available replies would not match the email well enough, and users would like to create their own reply. During the testing phase, in order for users to create a new reply to a specific scam mail, they would have to go to the crowdsourcing page and refresh it until they would find that specific email. They would then have to write a reply and submit it. Finally, they would have to open the plugin again and refresh the replies until they would find their own. This is a cumbersome process, and should of course be changed.

Finally, some users mentioned that they would like to see how well the server ranks the replies, and update the rating if they deem it incorrect. However, if the plugin would show a rating, users

would be more inclined to always pick the top reply, while the idea is that the system gets trained by people using it, so this idea would actually counter the effectiveness of the rating system.

7.2. Crowdsourcing web page

The crowdsourcing page was also received very well, though it got some minor feedback points. Most users found the explanations well written, albeit a bit on the long side.

What users generally found confusing was the fact that a page reloads soon after submitting or rating a reply. The reloading is done to fetch new scam mails or replies, but the users found this to be confusing as no confirmation was given once the user submitted their answer.

Another point of confusion was that on the *upload a new scam mail* page, it was not clear that users should not write their own scam, but rather copy paste any scam mails they received. In line with this, when writing a reply, it was not clear that no name should be written below the reply, as the plugin will generate this automatically.

7.3. Changes

The following changes were made according to the user feedback:

Several texts in the plugin were changed so the explanations for the pages were clearer. The colour scheme of the plugin received minor updates, with mainly the buttons and progress bar having a different colour that fits the overall style better.
The most interesting change is the option to write your own reply using the plugin. Almost all testers mentioned they were missing such a feature. Although this is a very interesting feature, users are urged to only use this feature if no suitable replies are found.

Finally, on the crowdsourcing page, the ”upload a new scam mail”, ”create a reply” and ”rate existing replies” buttons are now next to each other. The font of the menu on the website is now a bit bigger.

8

Final Remarks

Several things will be discussed in this chapter. First the evaluation about what we did and what we would do differently if we did the project again will be discussed. Second several recommendations on features that could be implemented later will be discussed. Third, a discussion about the results will be held. Last the conclusion of the entire project will be discussed.

8.1. Evaluation

In hindsight, it can be said that the project has been executed fairly smoothly. In the beginning of the project, all members focused on their own part of the solution. This meant that everyone had their own area of expertise, while knowing relatively little about the parts others were working on. This meant that one could not switch between tasks easily as they would have to learn about the others parts first. While this is generally a negative thing, there was one positive aspect about this as well. Because everyone had their own part to work on and knew relatively little about the other parts, this resulted in four parts which are quite independent of each other. Because of their independence, these parts are easy to edit and modify, without breaking the other components.

During the project, there were weekly meetings with the client and the coach. This meant that newly requested or developed features could be discussed often enough. We are highly satisfied by the amount of feedback we received every week.

While Trello was used to keep track of what was to be done, overview of the tasks was lost from time to time. This resulted in some less productive workdays and some longer than usual lunches. This lost time however was usually compensated by working harder on other days and generally being ahead of schedule. One thing we would try to do differently is keep better track of the tasks and planning in the future.

8.2. Recommendations

This section contains suggestions for improvements and extra features that were not yet implemented during the course of this project.

- Add the option to use a broken picture as attachment when writing new replies on the crowdsourcing website. This could ensure a greater conversation depth because a scammer has to determine why he can't open the attachment. It should also work well because if a scammer asks for certain information, like an id-card or a picture, you could send him this file. Otherwise you have to reply negatively to his question and that could lead to an earlier end of the conversation.
- Explore other means of keyword extraction. Not very much research was done before NLTK was chosen for this and it only extracts nouns at this moment. It could very well be that there are better solutions out there. However for this project NLTK works very well.
- Recommending replies could be more efficient when using for example Google's Pagerank algorithm. This is because at the moment in order to find the best replies in the database the total mean for every reply is calculated for all keywords that coincide with the keywords from the scam mail. If there are a lot of replies with many keywords in each reply then this algorithm will become pretty slow.
- At the moment if a scammer spoofs his email address, the replies will become undeliverable. The plugin looks at the headers in the email to try and find the correct email path to send an email to. Scammers usually have the correct address in one of the headers, as it is not to their benefit to spoof all of the headers. Still, a small amount has spoofed all return paths, so thus the email address needs to be found in the email itself, something that is currently not done yet.
- Add the option to activate a ”conversation bot”. When someone has received a scam mail but does not want to choose replies himself, they could activate this bot. It then sends the best replies according to the system and keeps a conversation going for as long as possible.
- The recommendation system does not take previous replies from the same conversation into account. It could happen that the same reply is recommended as one that you already sent earlier in the conversation.
- Develop the plugin not only for Google Chrome but also for Mozilla Firefox. These two have the biggest market share with a combined total of 87.7% (more can be read on this in section 4.2.1 of appendix E. WebExtensions is a common API for Firefox, Chrome and Opera. A Firefox Nightly version is available and when the system is completely ready, development will be completely the same for all browsers.
- Develop the plugin not only for Gmail but also for Microsoft Outlook Live Mail and Outlook.com (Outlook on phone will need a complete new application), Yahoo Mail and Thunderbird. These three have a combined market share of 7%. Together with Gmail the total covered market share would then be only 23%. However it should be relative easy to develop as most of the already existing code could be reused. Apple iPhone, iPad and Mail

have a combined market share of 51%. It would require one application for iPhone and iPad and a plugin for Mail. So it would be really profitable to develop that. However it would be like developing two complete new systems and that would take a lot of time and effort.

- Make delete a reply and scam mail not only depend on the report function but also on how much it is used and in case of replies also on how ”good” it is.
- Special care was taken to make the plugin usable for people who are colour blind, as readable in 5.1.2, however, not all cases can be taken care of. Therefore a colour blind mode could be added, which would remove all colours of the plugin, leaving behind a white background and the text, so any confusion would be removed.

8.3. Discussion

During and after development, there have been some experiments with the product. At the beginning of the project a few conversations were held with scammers to determine the do’s and don’ts. One can basically say anything to them as long as you feign interest. One of the biggest *don’ts* is telling the scammer that you are homeless or have no money. The maximum conversation depth without the plugin was thirty. At the end of the project a couple of conversations were held with the use of the plugin. The maximum conversation depth was now eight, which is a lot lower than by hand. This is because the replies are much more adapted to the mail when writing them by hand. However it also takes a lot more time and effort. Regardless of using the plugin or not, the scammers seem to react in the same way, however the scammers do seem to get annoyed much faster after a few replies from the plugin than by replying without the plugin. This is probably because these replies from the plugin are not specific enough.

All in all, the plugin greatly increases the speed of replying to scam mails and scammers do react to the plugin mostly the same way as when writing replies without using the plugin.

Conclusion

The TED talk ”This is what happens when you reply to spam email” by James Veitch inspired our client, David Beijer. His idea was to automate conversations with scammers and he contacted the TU Delft to find out if it was feasible for students to make a prototype as their bachelor project. The project team seized this opportunity to develop a completely new system from scratch.

The key selling points of the resulting product can be listed as the following headlines:

- (section 1.3) Revolutionary new theory on scam fighting
- (section 2.3) A unique take on and use of the Elo rating system
- (section 3.1) Python server for cross-platform communication.
- (subsection 3.1.2) Python processing server call handling
- (subsection 5.1.2) User friendliness
- (subsection 5.1.1) Email body processing

In consultation with the bachelor project coordinators, it was decided to change the goal slightly. Instead of fully automating conversations with scammers, the user would have to choose replies from a list of recommendations. This is because scammers are criminals and retaliation could happen if the project is a success.

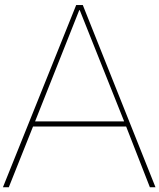
When the team started to work on this project, two weeks of research were conducted to investigate the problem and potential solutions. It became clear that knowledge on many different systems and principles was required including on python, plugins and machine learning algorithms. A research document was made with all the requirements and goals in mind.

During seven weeks of development and testing, the team has stumbled upon multiple unexpected challenges. The fact that PyBossa was not usable after all and a crowdsourcing website had to be made from scratch was definitely a challenge. However it turned into a great learning opportunity. The choice to use aliases instead of a separate mail server meant that the system would be sturdier against attacks from scammers. In the end our python server turned out to be single threaded. However none of the requirements were compromised and it has even improved the solution in some cases.

At the end of the project a fully functional software system has been delivered. This system exists of four parts: the recommender system, the storage and processing servers, the crowdsourcing website and the Google Chrome plugin. While there are still a few things that could be improved upon, the overall system could immediately be used. As the user tests in chapter 7 point out,

lots of people are very enthusiastic. All goals mentioned in section 1.1.4 were met and thus the project can be called a great success.

Appendices



Project Description

This is the project description as found on BEPSys[1].

Humanity has been approaching the problem of scam and spam e-mail from the wrong angle. Filtering out the scam mails and deleting them does not hurt the sender in any way. The costs of sending out such a mass mailing are virtually zero, so if only a small fraction responds or results in a ”sale” they can still run a profitable business case. Better filtering will not hurt them in a way that increases their costs to prohibitive levels. Instead, we start responding to them. As the amount of responses grows, the amount of time that has to be spent on responding to e-mails increases.

With modern Machine Learning algorithms, it should be possible to build software that automatically responds to scam e-mails with legitimate questions, in order to string them along for as long as possible. From a database of 1000s of sentences or even complete e-mails (which could be crowdsourced in a matter of days) the algorithm picks the right response. With improving realism from the software, the scammers should not be able to tell which ones are automatic responses, and which are genuine potential victims. This way, they can not afford to ignore the responses, as the response might be a genuine potential victim. Every time a conversation ends because the scammer gives up or gets suspicious, this is an opportunity to train the algorithm. Either train the algorithm by optimizing for longer conversations, or ask a human for a better suggestion the next time this occurs.

The assignment is to develop an AI e-mail responder that wastes as much of the time of the spammers/scammers as possible. An important part of the research will be to find out how to create the largest costs on the scammers’ side. The success of this approach depends largely on scale, so the software package should be easily distributable, and should easily be deployable as an add-on to existing mail boxes or domains. Open sourcing is not preferred as this might give away ciritical knowledge on how to circumvent or recognize our strategy. Important challenges are recognizing content of mail to give relevant responses, protecting the people who deploy our

[1] `https://bepsys.herokuapp.com/projects/view/156`

software against retaliation, adapting and learning, and distributing the learned strategies to the other clients.

As spam and scam operate based on different mechanisms, it is probably best to select one of the two to focus on before trying the other category.

B

SIG Evaluation 1

SIG evaluated our code and concluded the following (translated to English):

> The system's code scores 3,5 stars on our maintainability model, which means that it is maintainable above average. You didn't reach the highest score because of a lower score on Unit Size.
>
> For Unit Size we looked at the percentage of code that was above average long. Splitting these methods in smaller parts makes sure that every part is easier to understand, test and therefore easier to maintain.
>
> In your project are a few longer methods in the Python code. For example calc_points in elo.py is very long. You could move the part, where the data structures winnerkeys/loserkeys are build, to a separate method. Another example is handle_halfnbestreplies in mlqueries.py. Inside that method are a few blocks of code that are separated from each other with comments. The content of these comments, like "calculate the mean score" show that these are actually different responsibilities implemented in one method. Instead of showing this with comments, it is better to move this part to a method like calculate_mean_structure. This will also make the code more self-explanatory.
>
> Finally, it is nice to see that you wrote unit tests. However the test coverage could still be a bit better, hopefully this will improve during the course of the project.

C

SIG Evaluation 2

SIG evaluated our code for a second time and concluded the following (translated to English):

> In the second upload we can see that code volume has grown, while the maintainability score has remained about the same.
>
> Considering Unit Size, which was mentioned in the first upload, we see a slight improvement. You have adapted said examples, but outside of these examples we see no structural refactoring of similar constructions. That's why the slight improvement does not lead to a significant rise of the total score.
>
> It is positive that you have also written new tests for the new code, but this is also a slight improvement.
>
> From these observations we can conclude that the recommendations from the last evaluation have largely been included in the development process.

D

Project Plan

Introduction

This is the first document that is written for the TI3806 Bachelor Project. It will contain all the agreements made with the client. In the following sections we will discuss the project and what we will deliver. After this our planning will be discussed and summarized. The last two sections are about the organization of this project and the quality control.

Project Description

In this section we will discuss the project. In the first section we will give background information on our client, in the consequent sections we will discuss the problem itself and the goals of the project. The deliverables and the prerequisites will be discussed and finally we will illustrate our view on how to protect the user's identity.

Client

Our client is David Beijer, CEO of Peeeks, a tech startup at YES!Delft focusing on predicting energy prices and buying and/or selling accordingly. This project is however not related to Peeeks, and is considered more of a hobby project of David.

Problem Description

Scammers make money by baiting people into sending them money by sending out emails. While not many people fall for these scams, easily millions of emails per day are sent by them. If a scammer gets a response, he will try to convince the victim to send money, which leads to an email conversation between the scammer and the victim. Because the amount of responses is generally very low, the scammer can focus more easily on a single victim.

Prerequisites

While there are no hard requirements, our client would prefer to see the project built using Python and running on the Amazon Web Services as he has prior experience with it, and his company

Peeeks uses Python as well.

Goals
The goal of this project is to reduce the damages done by scammers by building an extension that automatically responds to scam mails with the intent to flood their inbox and making it difficult to assess which response is real and which is sent by our tool. If scammers can not discern whether a response is sent by a victim or our tool, they will waste their time by responding to the wrong mails, and thus their business model is less effective.

Success Criteria
Defining the success of the project is crucial to make sure we consistently work towards the correct goal. As the project is mostly meant to make an implemented proof of concept, the project will be a success when people are able to have a conversation of around four to five emails between the scammer and the person using the plugin. Another criteria is that the system functions autonomously. This means that the crowdsourcing system will gather replies, remove inferior ones and keep improving itself based on the results of its users.

Deliverables
At the end of the project we will have three deliverables. The extension itself, a server running the machine learning algorithm and database consisting of crowd-sourced emails and reactions. Aside from this, a number of reports will be written during this project. These reports are the Research Report, Design Report and the Final Report. Finally there is a presentation at the end of the project period.

Privacy
The user of the program may become prone to retaliation of the spammers. To prevent this, it is best to make sure the identity of the user never becomes known. This can be done through various methods which will be explored in the research report. These methods include responding through a pseudo-mail address or pretending to have forwarded the mail to someone else who is then leading the spammer on. In the research report a broader understanding of this issue will be developed to determine the best method for protecting our users. Another issue is the risk of harassing people when crowd-sourcing replies to e-mails. A malicious user of the crowd-source system may create replies containing the contact information of innocent people. This is to be prevented by filtering out these kinds of data. Finally our own privacy can be in danger. Spammers might get very annoyed if they notice that there is an automated process that replies to their mails, and might start some kind of retaliation. We will also explore how to prevent this in the research report.

Legality

Project Plan
This section contains the planning of the following weeks of the project. There are four phases: the research phase, the design phase, the implementation phase and the testing and release phase. The project will take place over the course of 10 weeks. At the end of week 9 the product

and report have to be finished. At the end of week 10 there is the final presentation in front of the Customer, the TU coach, other teachers and fellow students.

Research Phase
The research phase will last two weeks. We will look at the solutions of similar products. This way we can decide what the best software and technologies are to use for our project.

Design Phase
The design phase lasts one week and starts after the research phase. In this week the decisions for the software plugin, the crowd-source website and the database design will be made. There will also be a test plan created.

Implementation Phase
In the implementation and Testing phase the actual implementation of the plugin, website and database will be done. We will use continuous integration for testing. In the last week of this phase This phase lasts 5 weeks and is thus the biggest phase of the project.

Testing and Release Phase
The last phase will be the User testing and Release phase which will take 1 week. In this week we will release the product and gather information on its use. We will also perform a user test to find the strength and flaws of our product. At the end of this week the Final Report will be submitted.

Overview

Week	To Do	Deliverables
Week 1 (18 - 24 apr)	Work on Project Plan Start Research	Project Plan
Week 2 (25 apr - 1 may)	Work on Research Report	Research Report
Week 3 (2 - 8 may)	Work on Design Report	Design Report
Week 4 (9 - 15 may)	Implementation	
Week 5 (16 - 22 may)	Implementation	
Week 6 (23 - 29 may)	Implementation	SIG Evaluation 1
Week 7 (30 may - 5 jun)	Implementation Process Feedback	
Week 8 (6 - 12 jun)	Implementation Start User Testing and Release	
Week 9 (13 - 19 jun)	Continue User Testing Finish Final Report	SIG Evaluation 2 Final Report
Week 10 (20 - 26 jun)	Prepare Presentation	Presentation

Project Organisation
In this section we will discuss the personnel of the project, the administration and how the progress of the project will be reported. This project is worth 15 ECTS, meaning all four of us have to work on it for 420 hours in total. This roughly equates to working full time on this project. The project requires experience on databases, web-development and machine learning. While we all have

experience with databases and web-development, we have not yet used machine learning. Also, not all students have experience with Python so during the research phase time will be allotted to study this.

Administration

We are using Trello[1] for an overview of the progress of the project. Trello allows you to make a board with lists, similar to using a scrum board. These lists have cards representing tasks, and notes can be attached for further details. These cards can then be tagged, labeled and a due date can be set. This way we can keep track of who is doing what, and when the deadlines are.

Reporting

During the project we will have weekly meetings with the coach and the client. We will use the meetings with the client to show the progress and discuss the current developments, and the meetings with the coach to show progress and get feedback on the report.

Risks

In this section we will address risks in our project, to ensure that we know how to properly prevent these risks from happening and how to handle them in case they do happen.

The first risk is in the research phase. In order to get a good grasp of how the system should respond to scam mails, we need to find such mails ourselves. It might be though to get this spam intentionally. A reason for this is that modern mail clients already have strict spam filters that silently delete most spam emails sent to an email address, so we should look for mail clients that do not have such a strict filter. If these are not found, we could always write a crawler to gather emails from sites like 419eater[2].

Once we have collected enough scam mails, we need to crowd-source replies to these emails. The system will only work well if we manage to find enough replies, which might take some time to get. Another problem is that not all the answers will be usable, and manually filtering all the answers will also take some time.

With the complexity of this problem, there might not be enough time to run the system once we have built it. We need to make sure that we are done with the development of the system on time so we can actually test the system and get results before the report has to be handed in.

Finally we might run into issues with deadlines. There are a couple of strict deadlines that we *must* make. If we notice we run into issues with these deadlines, we will work longer hours to ensure we make the deadlines. Additionally there are weekly deadlines that we set ourselves, if we do not make these deadlines, we need to make sure that we work harder the week after. When we miss too many deadlines, we need to look if we are not setting too strict deadlines.

[1] https://trello.com
[2] http://www.419eater.com/

Quality Assurance

Quality is an important factor, to create and maintain quality some preventive actions like version and risk control, code evaluation and testing are essential. These actions are illustrated in the sections below.

Version Control

We will use Github for version control because it is widely used and we all have experience in using Git. Git also allows for easily setting up continuous integration and keeping track of issues and tasks.

Code Evaluation

Throughout the project our code will be evaluated in multiple ways. The first will be by Code Reviews done by ourselves and SIG. The second will be by Unit Testing and User testing done by ourselves.

Code Review

During the project we will use checkstyle to check our code quality. Furthermore we will use Git to make branches and pull requests to check each others code. Additionally, we will hand in our code twice to SIG, once in the sixth week, and once in the ninth. The feedback from the first review will be used to improve the code in the seventh week.

Testing

To ensure our code is working properly we must test our code. For this we will make use of a unit testing framework for Python. To make sure the code keeps functioning when new features are being added, we will use a continuous integration tool. In the research phase we will determine which frameworks to use.

E

Research Report

The spam filter that fights back

Research Report

by

Aurél Bánsági
Ruben Bes
Zilla Garama
Louis Gosschalk

Project Duration:	April 18, 2016 – June 17, 2016
Project Owner:	David Beijer, Peeeks
Project Coach:	Geert-Jan Houben, TU Delft

TU Delft

Contents

Introduction

This document is a research report to investigate aspects of a system that can function as a spam filter that fights back and to make choices based on the information gained from this research.

The problem of scam emails

Scammers try to make a living by baiting people into sending them money. They reach out to these people by sending unsolicited emails. While only about 1% of people reply to these emails, about 1 out of 20 people who reply wire an average of $7.500 USD. [15]

This analysis of the scammers' methods shows a vulnerability. Their system depends on the fact that only a small portion of the people reply and the people who do reply, have a fair chance of sending money. With such a success rate, scamming will remain lucrative and will most likely not simply cease.

Stopping this way of scamming is difficult and a perfect solution is yet to be found. Our take on this is that the scammer can be forced to stop by diminishing his success rate. The plan is to semi-automatically send human responses to scam mails.

Method of research

Throughout this document every chapter will present a question. These questions all lead up to answering the key question. Each chapter works towards answering the subquestion as seen in figure 1 and the answers to all of these questions will provide as a structure to answer the key research question ”How can a system successfully and automatically lead a scammer on without illegal functionality or privacy breaches?”.

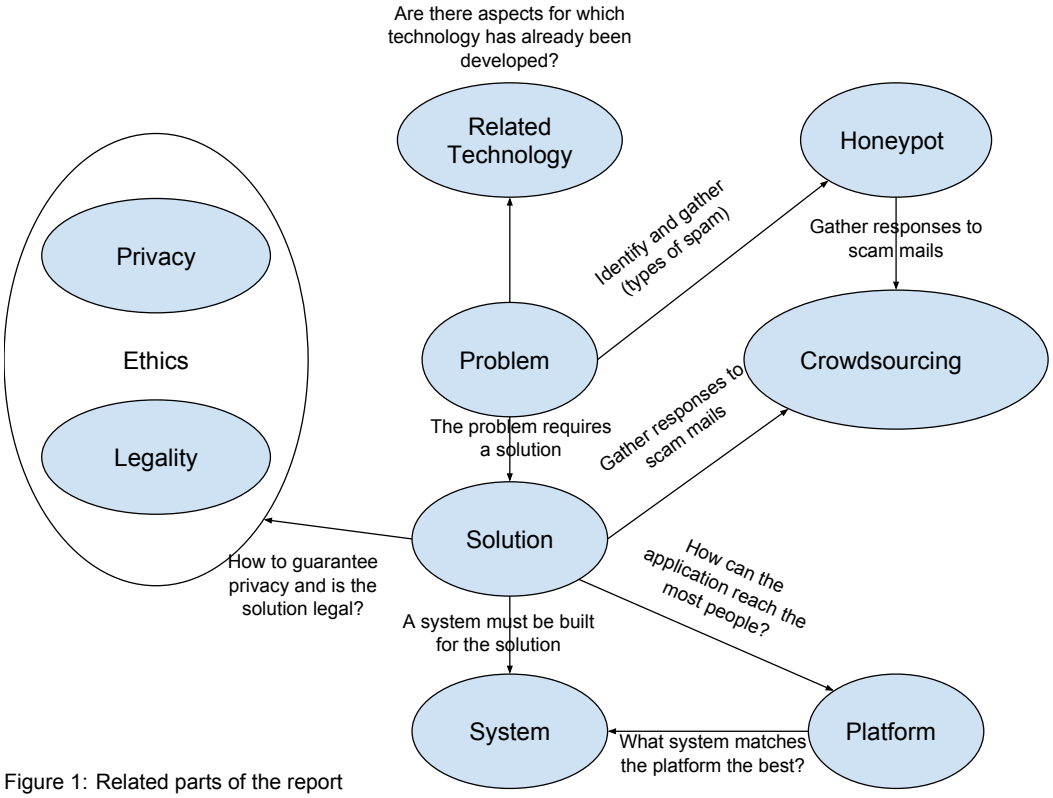


Figure 1: Related parts of the report

The problem of simplified scambaiting

First a view will be presented on technologies related to scambaiting. Scambaiting is the act of leading a scammer on in order to waste his time. An issue in creating this type of system is that you will need a lot of reference scam emails. It will be described how such a collection is going to be set up, which is called the honeypot. However, automated replies need to make sense in context. A quality automated response system needs a way to get sensible responses. In this project the take on this issue is to crowdsource responses to use in the solution.

A solution to simplified scambaiting

Creating such a system has multiple issues, requiring a set of choices. Choice of platform will determine dependencies and possibilities as well as the target audience, so this requires a fair elaboration. When a platform is chosen, a structure can be set up. It is important to carefully research all possibilities of existing libraries in languages to choose the optimal set of components. Lastly there is an issue of ethics on our solution such as privacy and legality. The project runs into privacy issues concerning mail addresses and liability of the project team. Furthermore, there is an international act against scam emails [3] which is of concern to the solution, so it will have to be proven that the solution will not offend this.

1

Related Scambaiting Technologies

At the moment, scambaiters reply to each scam email by hand. Our goal is to simplify this process, but in order to do this it is important to know what is already out there. In this chapter the question of which technologies already exist that can help with scambaiting will be answered. First will be discussed what scambaiting is exactly and what technologies already exist that help people with this. Then email autorepliers that are used to make emailing easier and faster will be discussed.

1.1. Scambaiting

Definition. Scambaiting is a form of internet vigilantism, where the vigilante poses as a potential victim to the scammer in order to waste their time and resources, gather information that will be of use to authorities, and publicly expose the scammer [12].

There are two websites that are the main source of information on scam baiting, namely 419eater [1] and thescambaiter [2]. Here scambaiters can find information, ask questions and post entire conversations with scammers. However everything is done by hand as there are no tools on these websites that can help you with this. To the best of our knowledge there only exists one tool that can do that. This tool is called Autobait [3] and it is a bot that automatically replies to scam emails trying to keep them talking. This tool is however still private and can not be used or researched. This also implies that the source code is not available. Thus it is impossible to use it for further reference.

1.2. Email autorepliers

There are two different ways an email autoreplier could work. The first way would be to fully automate the replies and requires zero effort from the user after the initial setup. This way was chosen by SlightlyCyborg [4], a system in which the user first has to label an email as ”annoying”.

[1] www.419eater.com
[2] www.thescambaiter.com
[3] www.autobait.com
[4] www.github.com/SlightlyCyborg/Email-Autoresponder

This application will then pick up this email and make an automatic response for it. An advantage of SlightlyCyborg is the fact that the source code is available and could thus be used as a reference for the proposed solution. The second way is Google's SmartReply from Inbox by Gmail [7]. It provides three different replies to an email which the user can then choose from. The advantage of this is that the user has more choice but it also requires a bit more effort. Another disadvantage is that Google has not shed a light on how they do this. This means it can't really be used as a reference apart from the interface.

Conclusion

In this chapter, technologies that help with scambaiting have been discussed. Sadly, a lot is still done by hand and only Autobait has been found as an automatic tool. When it comes to email autorepliers that can be used as a way to answer a lot of emails a lot faster and easier, only SmartReply and SlightlyCyborg have been found as working examples.

2

Honeypot

For scambaiting it is important to actually receive scam emails and have conversations with an actual human. To the best of our knowledge there has not been much research in the area of scam mails, especially not around the question if scam mails are sent by bots or humans. Reports often include the fact that most scam mails are sent by humans as trivial [4], but this may not be as trivial as it seems. To test this, mail accounts need to be set up and tested. This chapter will thus discuss the ways scam emails can be gathered by answering the following question: What is an effective method to collect scam emails? This question will be answered by first showing a method of collecting scam emails and then showing how effective this method is.

2.1. Setting up a honeypot

Definition. A honeypot is a computer system that is set up to act as a decoy to lure cyberattackers, and to detect, deflect or study attempts to gain unauthorized access to information systems.

In this case a honeypot is an email address that will act as a gathering point for scam emails. In order to attract enough scammers to the honeypot the scammers need to be able to find it. To accomplish this, one can leave behind email addresses on different guestbooks. A guestbook is a section of a website where visitors can leave comments, thereby exposing their email address. Scammers have bots that browse the internet searching for email addresses and because everything you post on these questbooks is public the bots can easily find this. In order to find the right kind of guestbooks a tip from 419eater[1] was used. This tip stated that one should search for ”guestbook +guymen +mugu”. This resulted in questbooks like Muguguestbook [2].

2.2. Results

After setting up a honeypot, its effectiveness has to be determined by trying to set up a conversation with scammers.

[1] `http://www.419eater.com/html/baiting.htm`
[2] `http://muguguestbook.com/`

2.2.1. Setting up email accounts

By applying the tips from 419eater, three email accounts were set up on Gmail, Outlook and Yahoo. They were then set up as honeypots by applying the tips from 419eater and leaving behind the email addresses on a few guestbooks. After a few hours this method had proven to work very well as several scam emails were received on the baiting email addresses.

2.2.2. Replying to the scams

After setting up a honeypot, research had to be done to determine whether or not it was possible to hold a conversation with a scammer, and what answers would prolong the conversation. After sending a couple dozen emails it was noted that a good reply to the first email in a conversation with the scammers would be something like ”I am very interested. Please tell me more.” or ”I am very bad with long texts. Could you please summarize?”. A good reply later in the conversations would be asking for information that a scammer already gave before, acting like you, the victim, are dumb. All of these replies are as general as possible to make it easier to respond to almost every email with the same response. This way several conversations were held with scammers, who seemed none the wiser after sending very generic replies every time. By sending generic replies, it also makes automation easier and takes the least time for the user but the most time for the scammer.

Conclusion

A very effective way to collect scam emails is to make an email account and set it up as a honeypot by posting the email address on various guest books and other sites. This method is proven to be very effective as several conversations were held with scammers sending generic replies.

3

Crowdsourcing

Crowdsourcing is a vital part of the project, as it will gather replies that will be send back to scammers. A separate platform will need to be set up to get these replies. Additionally, in order to know how well the replies will perform, they will need to be ranked as well. As there is little time available for the project, there is no time to make a crowdsource platform from scratch, meaning a crowdsourcing platform will need to be chosen as well.
This chapter will answer the question, ”What data needs to be crowdsourced and what is the best platform to do so?”, by first exploring what data needs to crowdsourced, then describing possible crowdsourcing platforms and finally choosing a platform.

3.1. Gathering replies to scam mails

Definition. Crowdsourcing is the practice of obtaining needed services, ideas, or content by soliciting contributions from a large group of people and especially from the online community rather than from traditional employees or suppliers [6].

An automated system is not capable of interpreting text, something which humans usually will not have the same trouble with. [10] Thus, it is better to let humans write replies to the emails, instead of letting it be generated by a system. There are several different kinds of topics in scam emails, and it is not always viable to send the same reply to the initial email. As mentioned in section 2, scammers will usually reply to a very generic email as first reply, but not so much for possible later replies. To prevent the scammers from recognizing emails sent by our system, there will need to be a lot of diverse replies.

As a solution, a crowdsource service will be made. Here, users will see an email somewhere in the conversation and be asked to write a serious reply to the email. By having enough users the replies might be similar to each other, but they will be diverse and humanlike enough for the scammers to not notice [13].

3.2. Processing replies

After a large set of replies has been generated, they need to be processed to templates manually. By processing is meant that phrases like names need to be removed and labeled so that another name can be inserted later. Additionally it is vital to remove replies that expose personal (or fake) information that can be used to identify anyone.

3.3. Ranking replies

After getting replies, they need to be ranked to know about their performance. This way the system will know what good replies and bad replies are, and it will be able to assign a rating to new entries too. Users will rate replies by ordering a number of replies to an email in order of how well the user expects the conversation to proceed by selecting that reply. As this is a fairly complex task, it might be useful to make a short tutorial or show a few examples to make it more clear for the user.

3.4. Possible platforms

As mentioned in the introduction, time constraints means that there is no time to build a crowdsourcing website from scratch, and an existing platform has to be chosen. The choice is then to either crowdsource on an existing platform, or by making a platform with available tools. The most popular existing platforms are Amazon Mechanical Turk[1], CrowdFlower[2] and Microworkers[3]. An example of a tool to make a crowdsource platform is PyBossa[4].

3.4.1. Amazon Mechanical Turk

Amazon Mechanic Turk (MTurk) is a platform set up by Amazon, and is one of the most commonly used crowdsource platforms. However, to request tasks a U.S. billing address is required [1], so this is not a viable option.

3.4.2. CrowdFlower

CrowdFlower is also a popular crowdsource platform that does allow requesters from the entire world. They also have all the tools available that are needed for the project. However, they do not allow the creation of free tasks. As it is preferred that the tasks are free to minimize the money needed for the project, CrowdFlower is a possible alternative, but it will not be the optimal choice.

3.4.3. Microworkers

Much like CrowdFlower, Microworkers does not not allow free tasks, and additionally members need to either have a credit card or set up a Paypal account. Furthermore, the process to sign up generally takes a long time. In order to make sure there will be enough replies, the plan was to ask people themselves to fill in some tasks. Having no free tasks and a lengthy application process thus makes Microworkers unsuited.

[1] https://www.mturk.com/mturk/welcome
[2] http://www.crowdflower.com
[3] https://microworkers.com
[4] http://pybossa.com

3.4.4. PyBossa

PyBossa is a toolkit to make a custom crowdsource website or platform written in Python. Additionally there is functionality to generate a complete site with the necessary features and control. This means that the tasks can be made free, and the data can be accessed quickly as well. It does however mean that there is some extra development time.

Conclusion

In this chapter crowdsourcing was discussed by answering the question: What data needs to be crowdsourced and what is the best platform to do so? First the needed data to be crowdsourced was researched. It is necessary to gather many replies to emails, because else the scammers might detect that the replies are not genuine. The replies then also need to be processed into templates which can then be ranked by the crowd.

The chapter ended by showing a few crowdsourcing platforms. There were four systems total, these being: Amazon Mechanical Turk, CrowdFlower, Microworkers and PyBossa. The first three are existing crowdsourcing platforms, but were shown to have disadvantages that make it impossible or unfavorable to work with them. PyBossa remained as the only option, that while causing more development time, means that there will be full control of the crowdsourcing aspect.

4

Platform

In this chapter possible platforms on which the system can be built on will be decided. All browser-based email platforms have their own tools suitable for development, and these are often not directly compatible with each other. Additionally, the system needs to work with a browser wide plug-in, and the different popular web browsers are also not directly compatible with the others. As the project has limited time and is fairly complex, it is vital to not focus on too many platforms at the same time, and only develop for one specific platform. The following question will thus be answered: What are the optimal email platform and web browser to develop a plug-in that can meet the requirements?

4.1. Email Platform

An email platform needs to have a few characteristics to be suitable for the project. It needs to have an open library with functions like sending an email and sending and retrieving data from a server. Additionally the platform needs to be widely used and scam needs to actually be somewhat of an issue on the platform, meaning that it cannot have a strict spam filter.

4.1.1. Available platforms

First a list of possible candidates need to be defined. Table 4.1 provides a list of the most used platforms and their market shares [5].

Table 4.1: A list of the 10 most popular email platforms.

No.	Email Platform	Market Share
1	Apple iPhone	33%
2	Gmail	16%
3	Apple iPad	11%
4	Google Android	10%
5	Apple Mail	7%
6	Outlook	7%
7	Yahoo! Mail	3%
8	Outlook.com	2%
9	Windows Live Mail	1%
10	Thunderbird	1%

A number of mail platforms can be excluded from this list, for instance because they are not browser based, or do not allow the development of plug-ins for their system. This leaves the following platforms available: Gmail, Yahoo! Mail, Outlook.com and Thunderbird.

4.1.2. Development tools

The next requirement is that the platform needs a well documented development library. As mentioned before, the functions we need focus on being able to send an email, reading an email and processing the results, and sending and retrieving data from a server. These four platforms will now be discussed.

Yahoo! Mail

According to Yahoo's own development website[1], they have abandoned their own mail API in favor of access to their IMAP service starting from February 16, 2015. [14] This means that they no longer have documentation available, and developers need to "guess" to what functions are available. It is still possible to connect to their servers and get information [9], but with no available documentation, it would be a guess work on how to make it work like required for the project.

Outlook.com

There are a number of tools available for Microsoft's products, including Outlook. They expose the functions needed for the project, e.g. sending a message. However, the website that contains the documentation for their development tools[2] does not function properly. This might make development frustrating or difficult as no official documentation will be readily available when needed.

Thunderbird

The Thunderbird system used to be part of Mozilla, but as of late 2015 it has been separated into a separate entity. Because of this, Thunderbird uses the same architecture for plug-ins as Firefox, with their own functions built on top of it. All necessary functions are available, but it is

[1] `https://developer.yahoo.com`

[2] `https://msdn.microsoft.com/en-us/office/aa905340`, checked on 21-04-2016

only possible to compose emails, and not send them without user interaction[3]. Another advantage about Thunderbird and Gmail is that they both support WebExtensions, which means that plug-ins are very easy to port [2].

Gmail
Finally the development tools for Gmail will be discussed. Google does not support plug-ins on Gmail directly, but they do allow plug-ins to interact with Gmail and mails directly. Google allows all the functions needed, and they have proper documentation available in a number of programming languages[4]. As mentioned above, plug-ins for Thunderbird and Gmail will be compatible with each other in the near future.

4.1.3. Spam filters
None of the above platforms publicly talk about their spam filter. As mentioned in section 2 the email accounts on Yahoo, Outlook and Gmail did retrieve spam, meaning they do not have a too aggressive spam filter. This means that spam is an actual issue on these platforms, and would be a viable choice in this aspects. Additionally this also means that spam can be retrieved to do research on while developing the project. Thunderbird is only an email client and does not allow the creation of accounts and thus was not tested.

4.2. Web browser
Having made the choice of an email platform, a choice for web browser has to be made. The requirements for a suitable platform are the same as for the email platform except that obviously the web browser does not have a spam filter. Thus the requirements are that it needs to be widely used, and has development tools available with a reasonably good documentation.

4.2.1. Available web browsers
First off a list of possible web browsers will be given in the following table, again sorted by market share [11].

Table 4.2: A list of the 5 most popular web browsers.

No.	Web browser	Market Share
1	Chrome	69.9%
2	Firefox	17.8%
3	Internet Explorer	6.1%
4	Safari	3.7%
5	Opera	1.3%

As seen in the table, Google Chrome has a big lead on its competitors. Internet Explorer and Safari can be excluded as they are mostly only used on their respective operating systems Windows and OS X, with barely used versions on the other operating system. For now Opera

[3] https://developer.mozilla.org/en-US/Add-ons/Thunderbird
[4] https://developers.google.com/gmail/api/

will also be left out, as their market share compared to Chrome and Firefox is not big enough to warrant development for this web browser.

4.2.2. Development tools

Both Chrome and Firefox allow developers to build plug-ins for their web browsers. For Firefox the programming language that plug-ins are developed in is JavaScript, for Chrome there are a couple of languages are available, among these are Java, JavaScript and Python. Both browsers are open enough and have well enough documentation that development should not be an issue for either of those. Additionally, as mentioned before, Mozilla and Google are developing a new system that allows plug-ins to work on Firefox, Chrome and Opera without having to port the program, this makes it very easy to reach a wide audience.

Conclusion

This chapter looked at options for a mail platform and web browser while answering the question: What are the optimal email platform and web browser to develop a plug-in that can meet the requirements?

Mail platforms were first looked at by taking the following things into consideration: The market share, the development tools and the aggressiveness of the spam filter. After looking at possible platforms, a list of four platforms was made, these being: Gmail, Outlook, Thunderbird and Yahoo! Mail. Shown was that Gmail has a substantial lead in market share, a decent set of development tools and their spam filter is not too aggressive either. Therefore Gmail will be used as mail platform over the other options.

After picking a mail platform, web browsers were explored. Initially five web browsers were looked at, but after looking at the market share, only two remained: Google Chrome and Mozilla Firefox. As Chrome has by far the biggest market share and Gmail was already chosen as mail platform, it was chosen to be the platform to develop on.

As a final note, Chrome, Firefox and Opera, and thus also Thunderbird and Gmail will use the same system for plug-ins in the near future. This will make porting the plug-in to Firefox and Opera very easy.

5 System

As mentioned in the introduction, aspects of our solution must be researched. There are many different possible programming languages, database and servers, each with their own advantages and disadvantages. In order to make well informed choices, these must be assessed. This chapter will answer two questions. First, what languages, databases and servers could be used in this project and secondly, which of these fit the requirements of this project the best.

5.1. Programming Language

The choice of programming languages is dependent on a number of factors, the most influential of which being the platform and our client. The client is used to and prefers Python code, while the developers have relatively little experience with this and are more experienced in writing Java code. This section will discuss the advantages and disadvantages of both languages when it comes to machine learning, testing and dependency management.

5.1.1. Machine Learning Libraries

Both Python and Java have a couple of available machine learning libraries. For general purpose machine learning, Python has scikit-learn with over 20.000 commits and nearly 600 contributors [1]. Other, less popular packages, such as PyLearn2[2] and NuPIC[3], are either no longer in development or are focused more on neural networks. Java's most popular machine learning package is Waikato Environment for Knowledge Analysis (Weka)[4]. When it comes to natural language processing, for example for finding keywords in mails, Python can make use of the Natural Language Toolkit (NLTK)[5]. A widely used Java package is MALLET[6]. This package can be used for statistical natural language processing, document classification, clustering, topic modeling, information

[1] https://github.com/scikit-learn/scikit-learn
[2] https://github.com/lisa-lab/pylearn2
[3] https://github.com/numenta/nupic
[4] http://www.cs.waikato.ac.nz/ml/weka/
[5] www.nltk.org/
[6] mallet.cs.umass.edu/

extraction, and other machine learning applications to text.

5.1.2. Testing Suites

When it comes to unit testing, Python has two built-in testing tools, these are unittest and doctest. The unittest tool provides standard unit testing while doctest tests using comments. All developers have experience using standard unit testing in Java, so it would be logical to use this in Python as well. Besides unit testing, several packages have been developed for mocking, fuzzing and web testing. There are also tools for code coverage and continuous integration. The same functionality exists for Java, and the group has experience with those as well because we have all followed a course on testing using Java.

5.1.3. Dependency Management

Both Python and Java have tools for dependency management, but the tools for Python seem a lot less streamlined. For Python, dependency management is done through PyPI, the Python Package Index, also known as Pip. Pip is used to search for packages and install them, but can also output which packages are currently installed. This can be saved in a file, which pip can use to install the needed packages if they have not been installed yet. This will be run in a virtual environment provided by the virtualenv package, to ensure other projects do not interfere with specific versions needed by this project, and vica versa. Switching between branches however does not automatically change the installed packages. Several tools exist for dependency management in Java. The most used are Maven, Gradle and Ant + Ivy. The developers have used Maven in previous projects and courses, so if we were to chose Java Maven would also be used. In general, the dependency management tools for Python are less mature than those developed for Java.

5.2. Database

The database will have to store a number of things. These are the scam mails, the response mails, keywords per email, scores per keyword and minimal user data. As not all emails have the same keywords, and keywords might be added later, our database should support this behavior. This means that a NoSQL database with a document structure would fit best as it allows updating or extending objects without having to change other objects as well. A database that works well with the chosen language and platforms is a must as well. Possible database systems are MongoDB[7], RethinkDB[8] and DocumentDB[9]. These all support both Java and Python and use JSON as a language for the database. MongoDB and RethinkDB have a GNU AGPL license, whereas DocumentDB has a proprietary license and can only be used when using Microsoft Azure. Additionally, MongoDB and DocumentDB provide a RESTful API.

5.3. Server

A server will be needed for this project for three purposes. It is used for storage and must run our algorithm and crowdsource page. As it is difficult to assess beforehand what its specifications

[7] www.mongodb.org/
[8] www.rethinkdb.com/
[9] azure.microsoft.com/services/documentdb/

must be, a service that allows upgrading must be chosen. It is also preferred to use a service that is free for students, or offers a steep discount, as there is no funding. Our client would prefer us to use the Amazon Web Services (AWS)[10], as his company uses this as well. Other options are the Google Cloud Platform (GCP) and Microsoft Azure. AWS provides students $35 dollars in credit, to be used for buying processing time and power, as well as storage. GCP has a similar service, providing a user with $300 dollar in credits which is usable for 2 months. Microsoft offers DreamSpark for free for students, allowing you to host your web apps for free. This however does not give complete access to everything Microsoft Azure would otherwise have to offer.

Conclusion

To come back to the questions posed in the beginning of this chapter, each aspect offers several different solutions. Python is the programming language that will be used for this project. Both Java and Python have plenty of machine learning libraries, testing suites and other tools. Only the dependency management is not quite optimal for Python, though it will be sufficient for the goals of the project. The main reason to choose Python over Java however is because the client prefers Python.

The database that will be used is MongoDB, this is for a number of reasons. Besides using the document model, it works well with the rest of the project. MongoDB stores its documents using JSON, the extension we will write uses JavaScript and Python has a built-in JSON encoder and decoder. Another advantage is that the client is used to MongoDB as well.

The project will be run on AWS. The client is currently using this for Peeeks as well. Because this is a project by students, Amazon provides $35 dollar in credits for Amazon Web Services. This will very likely be sufficient for the duration of the project, after which the client can choose to continue on this platform or choose another. The benefit of AWS is that it scales well because it is easy to upgrade or downgrade the server and storage depending on the needs.

[10] `aws.amazon.com`

6

Privacy

Privacy is the key to making this project work. If the user of the software is easily identifiable and uses his own address to converse with the scammer, the user may be prone to retaliation from the scammer. Preventing this is thus very important. This chapter will answer the following question: How can the system ensure the privacy of all people involved? This chapter will first describe the privacy of users, then the developers and finally privacy risks involving the crowdsourcing aspect will be discussed. Noted should be that the privacy of the scammer is ignored in this chapter since their information is likely already faked.

6.1. User Privacy

The users need to be protected from any revenge a scammer may try to take. This is risky because the mail address of the user is already known to the scammer, since they received the scam already. After conversing with the scammers, it was noted that they do not pay any attention to who sent them the reply. This means that if the users email address is edited, or sent from a different email address, the scammer will very likely not notice.

Modification of the mail address can be done through various methods but the easiest way is to forward the mail to another address and reply through that address.

6.2. Project Team Privacy

As a project team developing a tool to ruin an illegal business, the developers may be a target of the retaliation of a scamming community. Therefore it is of high priority that the identity of team members can not be linked to the product. The developers are linked to the project through the university, but any place other than the university will not be connected. This means that the GitHub repository will be made private, the crowdsource platform will be hosted on a server which is not traceable to the developers personally and the software will be distributed on the Chrome Web Store through an anonymous account. Additionally fake mail accounts and aliases will be used in case any situation comes up in which identity is in danger of being revealed.

6.3. Crowdsource Risks

When crowdsourcing for answers and scam mails, malicious users may create answers in the system that share contact information of people they dislike. To prevent this from happening a filter on all crowdsourced responses to detect contact information. It is possible to do this filtering by checking the responses for preset formats like ”+00 0 00000000” or ”x@x.x”. These measures are deemed sufficient for preventing the distribution of the data of innocent people.

A debatable topic on this is whether or not to scan for names. This would require a bit more intensive filtering since it would probably have to distinguish between names and grammar. It is possible to do this by using the NLTK algorithm [8] which uses grammar recognition to distinguish names and tag them. The main problem with doing this is that it compromises the ability to invent fake names or corporations to bait the scammer. The use of this will be determined along the project’s progression, since it is not possible to know how the results of the crowdsource will turn out.

It is hard to foresee whether the results will contain a lot of fake names or harassment through existing names. To prevent blocking the creative minds that will create replies through the crowdsource, the system will allow names in the beginning so later on it can be decided whether or not to delete the replies containing (fake) names.

Abuse of the crowd source system will also be balanced out by itself since the quality of responses can be rated in the rating module. This also has the possibility to report malicious, invalid or harassing responses.

Conclusion

In this chapter the question ”how can the system ensure the privacy of all people involved?” was answered by talking about user and developer privacy and exploring risks by crowdsourcing. User privacy will be protected by replying with a fake name and a different email address. Developer privacy will be ensured by making all resources private or anonymized. Finally there are risks involving crowdsourcing but it was decided to trust the users for now and only act if there are indeed privacy issues.

7

Legality

Scammers are criminals trying trick people into giving them money. The scambaiter community is trying to solve this problem but often enter a grey area in which one could say they are going too far. The solution will be very different but must stay legal. Thus this chapter will answer the following question: What can be done to ensure that the system does not break any laws? The chapter will first discuss global laws and then look at local laws.

7.1. Global laws

To prevent scam an anti-spam act [3] has been agreed upon internationally. This act states that it is illegal to send unsolicited mail without providing an option to unsubscribe and it puts ties to the content of such mails.

This act is violated by the scammers since they send unsolicited mails to people with misleading information and no way to unsubscribe, if that would even have a meaning in this situation. However, the reason the system lives up to the standards of the CAN-SPAM Act lies in the same part as the reason the scammers violate it. In the sense that no unsolicited mails will be send. The mails of the system are solicited since the system is designed to respond to scams instead of randomly contacting listed scammers, for instance.

7.2. Local laws

The CAN-SPAM Act is international but local law enforcement is able to enforce additional legislation. It is very time consuming to check the law for every single country and since the system will be distributed through the Chrome extensions store, the system will simply be excluded from distribution in countries where it is reported to run into legal issues. This way there does not have to be research for legal issues for the system in every single country.

Conclusion

This chapter discussed the legality of the project. Only local laws might be an issue. If a local law is a problem, a solution is to block the plug-in from that country in the Chrome extension store.

Conclusion

This research report was made to answer the following question, ”How can a system successfully and automatically lead a scammer on without illegal functionality or privacy breaches?”, and did so by answering a couple subquestions.

It first showed that there has not been a lot of research to related technologies, but did show some examples in scambaiting and machine learning, and that research to Nigerian prince scamming in general is very lackluster. It then went to explain how a honeypot can be made, and that most of the interactions were probably with a human. However, further and deeper research should be done in this field.

Following this, crowdsourcing was discussed. First the kinds of data needed were explored, before a choice for a crowdsourcing platform was chosen. The choice was PyBossa, a tool to quickly make crowdsourcing websites.

Next a choice for the platform to be developed on and the system itself had to be made. For the platform multiple options for the email platform and webbrowser were explored and reasoned with before settling on Gmail and Google Chrome. For the system several programming languages, databases and server setups were explored, before choosing Python, MongoDB and Amazon Web Services.

Finally the privacy and legal aspect of the project were discussed, showing that privacy will need to be very carefully looked at, but that there will probably be no legal issues.

In the end, these subquestions combined answer the research question of the report.

Bibliography

[1] Amazon. Faqs. https://requester.mturk.com/help/faq#can_international_requesters_use_mturk, April 2016. Accessed 26-04-2016.

[2] chrisdavidmills. Webextensions. https://developer.mozilla.org/nl/Add-ons/WebExtensions, April 2016. Accessed 21-04-2016.

[3] FTV. Can-spam act: A compliance guide for business. https://www.ftc.gov/tips-advice/business-center/guidance/can-spam-act-compliance-guide-business, September 2009. Accessed 25-04-2016.

[4] Jelena Isacenkova, Olivier Thonnard, Andrei Costin, Aurélien Francillon, and David Balzarotti. Inside the scam jungle: A closer look at 419 scam email operations. *EURASIP Journal on Information Security*, 2014(1):1–18, 2014.

[5] Litmus. Email client market share. https://www.campaignmonitor.com/blog/email-marketing/2014/05/gmail-focus-on-engagement/, march 2016. Accessed 21-04-2016.

[6] Merriam-Webster. Crowdsourcing. http://www.merriam-webster.com/dictionary/crowdsourcing, April 2016. Accessed 26-04-2016.

[7] Bálint Miklós. Computer, respond to this email: Introducing smart reply in inbox by gmail. https://gmail.googleblog.com/2015/11/computer-respond-to-this-email.html, November 2015. Accessed 21-04-2016.

[8] NLTK Project. Natural language toolkit. http://www.nltk.org/. Accessed 25-04-2016.

[9] skunkwerk. No way to search yahoo mail programmatically with java? http://stackoverflow.com/questions/29066466/no-way-to-search-yahoo-mail-programmatically-with-java, April 2015. Accessed 21-04-2016.

[10] Shimon Ullman, Daniel Harari, and Nimrod Dorfman. From simple innate biases to complex visual concepts. *Proceedings of the National Academy of Sciences*, 109(44):18215–18220, 2012.

[11] W3Schools. Browser statistics and trends. http://www.w3schools.com/browsers/browsers_stats.asp, March 2016. Accessed 21-04-2016.

[12] Wikipedia. Scambait definition. https://en.wikipedia.org/wiki/Scam_baiting, February 2016. Accessed 21-04-2016.

[13] Ting Wu, Lei Chen, Pan Hui, Chen Jason Zhang, and Weikai Li. Hear the whole story: towards the diversity of opinion in crowdsourcing markets. *Proceedings of the VLDB Endowment*, 8(5):485–496, 2015.

[14] Yahoo. Yahoo mail api. `https://developer.yahoo.com/mail/`, December 2014. Accessed 21-04-2016.

[15] Andreas Zingerle and Linda Kronman. Transmedia storytelling and online representations–issues of trust on the internet. In *Cyberworlds (CW), 2011 International Conference on*, pages 144–151. IEEE, 2011.

F

Infosheet

The next page will contain the infosheet that is handed in together with this report. The rest of this page is intentionally left blank.

The Spam/Scam Filter That Fights Back 24-06-2016

BackFire

Filtering out scam mails and deleting them does not hurt a scammer in any way, but wasting their time by sending fake replies does. As the amount of replies grows, the amount of time a scammer has to spend responding to e-mails increases. This bachelor project has created a revolutionary system that recommends replies that can be sent to answer a scam mail. This makes it very easy for users to waste a scammer's time and destroy their business model. The project has a unique adaption of the Elo rating system, which is a very popular algorithm used by systems ranging from chess to League of Legends. A Google Chrome plugin was developed with user friendliness in mind, to let people reply to scam mails quickly. To gather replies, a crowdsourcing website, which also trains the Elo system, was set up. Finally, a Python server connects all the parts, allowing for cross platform communication.

Contact details

- Aurél Bánsági (aurelbansagi@gmail.com) Plugin development
- Ruben Bes (rubenbes94@gmail.com) Server/database development
- Zilla Garama (zillaamarag@gmail.com) Rating system development
- Louis Gosschalk (lgosschalk@gmail.com) Crowdsource development

Client
David Beijer
TU Coach:
Geert-Jan Houben,
Web Information Systems

The final report for this project can be found at: http://repository.tudelft.nl

G

User feedback

In this appendix, the actual quotes of users giving feedback from user tests is given. The names are anonymized.

Plugin

Person 1

” The conversation statistics are nice. The text above the ”Get Replies” button is not coming across. It is not concise and telling enough. Call the suggested replies ”options” and the one that’s opened ”Your Reply”. The text on the button to close the plugin after sending a reply could be shortened to ”close plugin”. It would be nice if there was an option to write your own reply inside the plugin. Add a star rating to show how good the reply is for this email according to the system and also let you submit this star rating. A bit like the IMDB star rating for movies. With some changes and small improvements I would certainly like to use it.”

Person 2

”The settings button is a bit unclear, maybe you could use a tooltip for this? The text above the ”Get Replies” button is unclear and too long. It is also unclear what is meant with the ”Report Reply” button. When would someone have to use this button? A text such as ”Inappropriate Reply” like the crowdsource website has is much clearer.”

Person 3

”The background is really nice and the footer works well. It is a bit unclear that the user name is generated on the settings page. However all the other things on the settings page are clearly explained. The colour of the ”Get Replies” button and loading bar is wrong. Though it is very nice that you get a bar when loading replies. The ”Report reply” button should be explained. The text just before authenticating should be longer.”

Person 4
”The begin screen is pretty clear. The generated names are very realistic. I don’t like to press a button to get replies. It is strange that authenticating opens a second screen. The plugin could close after a few seconds on it’s own after you have sent a reply. The overall design is really nice and works well.”

Crowdsourcing page
Person 1
”The ”see more”/”see less” buttons don’t look like it hides a paragraph. However on the home page the ”upload a new scam mail”, ”create a reply” and ”rate existing replies” buttons do look like they hide paragraphs instead of linking to another page. Maybe move these next to instead of on top of each other? The navigation bar doesn’t look like a menu. I think it would help if the font size was bigger and the buttons broader. It would be nice if there was a get new email button (when writing new replies) instead of having to refresh the page.”

Person 2
”I expected radio buttons instead of normal buttons for ”applicable”, ”inapplicable” and ”perfect” (when rating replies). It is unclear that a name is automatically generated when writing new replies. The texts on the home page are nice and catchy.”