Investigating the Effectiveness of Bayesian Spam Filters in Detecting LLM-modified Spam Mails

Malte Josten[0000−0003−2102−1575], Torben Weis[0000−0001−6594−326X]

University of Duisburg-Essen
Duisburg, Germany
{malte.josten,torben.weis}@uni-due.de

Abstract. Spam and phishing remain critical threats in cybersecurity, responsible for nearly 90% of security incidents. As these attacks grow in sophistication, the need for robust defensive mechanisms intensifies. Bayesian spam filters, like the widely adopted open-source SpamAssassin, are essential tools in this fight. However, the emergence of large language models (LLMs) such as ChatGPT presents new challenges. These models are not only powerful and accessible, but also inexpensive to use, raising concerns about their misuse in crafting sophisticated spam emails that evade traditional spam filters. This work aims to evaluate the robustness and effectiveness of SpamAssassin against LLM-modified email content. We developed a pipeline to test this vulnerability. Our pipeline modifies spam emails using GPT-3.5 Turbo and assesses SpamAssassin's ability to classify these modified emails correctly. The results show that SpamAssassin misclassified up to 73.7% of LLM-modified spam emails as legitimate. In contrast, a simpler dictionary-replacement attack showed a maximum success rate of only 0.4%. These findings highlight the significant threat posed by LLM-modified spam, especially given the cost-efficiency of such attacks (0.17 cents per email). This paper provides crucial insights into the vulnerabilities of current spam filters and the need for continuous improvement in cybersecurity measures.

Keywords: Spam Detection · Bayesian Spam Filter · Large Language Models

1 Introduction

Spam and phishing are persistent threats, accounting for nearly 90% of security incidents [5]. The prevalence and sophistication of these attacks necessitate continuous advancements in defensive mechanisms. Among the various tools employed to combat spam, Bayesian spam filters have been widely adopted [1][2][3][4][12]. In this work, we only evaluate *SpamAssassin*[1], as it is an open-source software that offers transparency and allows for community-driven enhancements, ensuring continuous improvement and security updates. However,

[1] https://spamassassin.apache.org

the advent of large language models (LLMs) introduces new challenges and opportunities in the cybersecurity landscape. LLMs are omnipresent, easily accessible, inexpensive to use, and exhibit remarkable power in generating and modifying text. This raises concerns about their potential misuse in reformulating (rephrasing) existing spam emails that can evade traditional spam filters. To address this emerging threat, we have developed a comprehensive pipeline to evaluate the robustness and effectiveness of Bayesian spam filters, specifically targeting SpamAssassin and its performance against LLM-modified email contents. Our research aims to illuminate the capabilities of LLMs in bypassing these filters and to identify potential vulnerabilities that need to be addressed. The findings underscore the importance of enhancing spam filters to counteract the sophisticated manipulations enabled by modern language models. In this paper, we present the results of our experiments, providing insights into our developed pipeline and demonstrate the (in)effectiveness of one of the most-used open-source spam filters, SpamAssassin, against LLM-modified spam emails.

2 Related Work

In their article, Gallagher *et al.* [10] discuss the evolving threats posed by LLMs in cybercrime, particularly in phishing and social engineering. They observe that the traditional high-volume, low-quality spamming techniques have increasingly been replaced by more sophisticated, targeted spear phishing methods that require greater effort. Our approach significantly reduces this effort; while it is fully automated for tasks other than the actual sending of spam emails and is also cost-effective, it is not specifically tailored for spear phishing attacks. Moreover, the authors highlight that despite existing guidelines, LLMs remain susceptible to manipulation through prompt engineering, a challenge detailed further in Section 5 of this paper.

Motlagh *et al.* also explore the adversarial and defensive use cases for LLMs [16]. The authors outline the various ways in which LLMs are already misused, such as gathering information to prepare an attack (reconnaissance), gaining unauthorized access to target systems, evading detection, and creating malicious code. They also include research by Karanjai [14], who already investigated the ability of various LLMs to generate convincing phishing emails. However, Karanjai's work did not specify a proper evaluation metric to measure the effectiveness of these emails. In contrast, we aim to refine and reformulate existing spam emails that are currently unable to pass the filter-under-test (FUT). We further provide two metrics in the form of the success rate and semantic similarity of the LLM-modified texts to analyze the resulting email contents.

As part of their research, Webb *et al.* [21] identified the potential risk of *camouflaged* spam emails, which blend elements of both spam and legitimate content, complicating detection by conventional spam filters. They show that filters that were originally only trained on solely ham and spam emails struggle to deal with camouflaged emails, with the Naive Bayes filter performing particularly poorly. Motivated by these findings, our study seeks to delve deeper

into the limitations of Bayesian filters. Unlike Webb *et al.*, we do not focus on emails explicitly crafted from mixed spam and ham content. Instead, we aim to carefully rephrase certain parts of emails to investigate potential vulnerabilities in Bayesian spam filters.

3 Experimental Setup

Our pipeline[2] is designed to target a mail server configuration that uses a spam filter and offers an API for accessing and modifying data on received emails. Specifically, our focus is on SpamAssassin, which serves to classify incoming emails as either legitimate (ham) or spam using a Bayesian classifier. Typically, SpamAssassin is set up as a gateway, filtering emails before they reach the inbox and blocking those identified as spam. However, in our setup, it functions differently by assigning a spam score to emails post-receipt.

Figure 1 depicts the system architecture that demonstrates both, the relations between the necessary components, and the data flow. For simplicity's sake, the system-under-test (SUT) comprises two docker images: one containing a mail server, realized through Mailpit [19] and the other image provides a SpamAssassin instance that gets integrated into the former [20]. Internally, the SpamAssassin image utilizes Linux's `spamassassin` service[3] with its default configuration.

To rephrase a spam email and to test its resulting classification and the overall robustness of the FUT, our pipeline communicates with various components. The core part, having access to the LLM's API, facilitates the possibility of reformulating and modifying an existing spam email, intending to make it sound

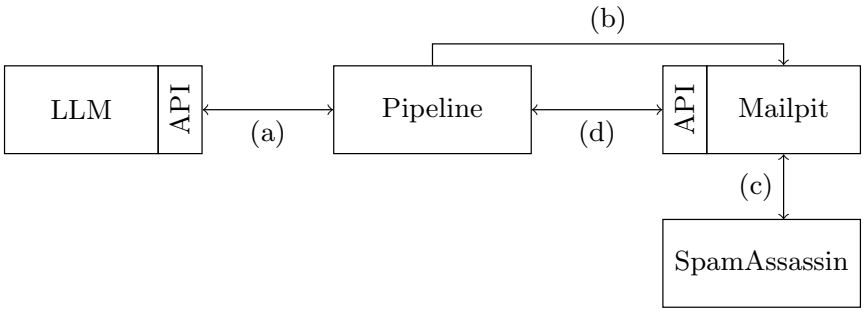

Fig. 1. To rephrase a spam email and test it's resulting classification, the pipeline communicates with various components to (a) modify the spam email body, (b) send the email via SMTP to the mail server, and (d) retrieve the classification label through the Mailpit API. With the help of SpamAssasin, (c) Mailpit labels the email as either ham or spam.

[2] https://github.com/MalteJosten/llm-spam-test
[3] https://linux.die.net/man/1/spamassassin

less aggressive and spam-like. Thus, the modified emails acquired through (a) are sent via SMTP to the mail server and subsequently classified by SpamAssassin (c). By utilizing Mailpit's API (d), we retrieve the spam classification for the email and use it to evaluate our method's Success Rate as well as the effectiveness of the FUT against LLM-modified spam emails.

4 Method

We designed and implemented a pipeline used to test the efficacy of LLM-modified spam email contents. This pipeline can be split up into three major parts: (1) pre-processing, (2) LLM modification, and (3) robustness evaluation. Figure 2 depicts these parts and puts them into context of the other system components.

4.1 Pre-processing

For this setup, we use the spam-labeled emails from the SpamAssassin Public Mail Corpus[4] dataset (2.399 spam emails in total). First, we convert all emails to `.eml` files to ensure they can be used more efficiently by the SMTP python library. Next, the STMP header fields `Bcc`, `Cc`, `From`, and `To` were anonymized (for example to *from@example.com*). This eliminates any speculations regarding the addresses' impact during the classification. We further chose to work with two datasets *original* and *minimal*, as SMTP headers are usually considered during the classification process and thus contribute to the decision of whether an email is considered legitimate or not [8][11][13][17]. This decision is also supported by the rise in misclassifications of spam mails after removing the headers (see Table 1) It is also important to note that we update the `Date` field to the current date and time to further minimize the SMTP header's impact on the classification.

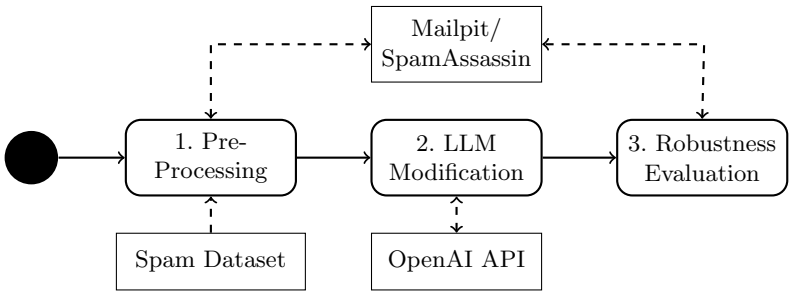

Fig. 2. Pipeline to (1) pre-process spam emails extracted from a spam dataset, (2) hand them over to the OpenAI API to be modified by the LLM, and (3) evaluate the spam filters robustness against the rephrased email bodies.

[4] https://spamassassin.apache.org/old/publiccorpus/

Original uses all the original SMTP headers "as they are" (except the anonymization and the updated `Date` field).

Minimal only contains the mandatory SMTP header fields `Date` and `From` (as defined by RFC 2822 [18]). In addition, we also include the syntactically optional `Bcc`, `Cc`, `Subject`, and `To` fields, to make the dataset more realistic.

Working with the *minimal* dataset ensures that the SMTP body's content is the main factor during the spam classification process. Hence, we can reason that any changes in the results can clearly be attributed to the SMTP body. Additionally, we can use *original* to compare our findings.

In the course of the dataset preparation (*Raw* → *Post-Mod*), some emails are filtered out due to, for example, encoding or formatting errors. After cleaning and updating the datasets, we define the ground truth, which consists of emails labeled as spam by both the dataset and the spam filter. Emails not detected by the spam filter are excluded. This ensures a more accurate and reliable evaluation of the success rate. We send each of these emails to the Mailpit instance (*Sent*) and retrieve their spam classification. Based on the gathered classifications, we build the ground truth, only using the correctly classified spam emails. Hence, for our work, we only consider 1.457 for the *original*, and 189 spam emails for the *minimal* dataset (*Spam*), as they constitute the ground truth.

Table 1. Number of emails during the first step: 1. Pre-processing.

Dataset	Raw	Post-Mod	Sent	Classification		Misclass. Rate
				Ham	Spam	
original	2.399	2.399	2.361	904	1.457	38.3%
minimal	2.399	2.111	2.091	1.902	189	91%

4.2 LLM Modification

To test the spam filter's robustness against LLM-modified content, we employed the OpenAI API with GPT-3.5 Turbo as the model to rephrase the email bodies. Prompts P1 and P2 were used in conjunction with the '`system`' role to prime the model by providing it some context in the form of the SMTP body, as well as instructions on how to answer and what (not) to include in the response [6]. Prompt P3 is the actual user prompt that triggers the LLM's response generation.

(P1) Keep in mind the following text I wrote: ≪ SMTP body ≫

(P2) Give your answer as a JSON object, only using the fields defined by the `print_result` function. I only need the response, no additional text. Don't generate a subject line. Instead of using placeholders, just leave out the placeholder brackets. You're allowed to use line breaks in your answer. Preserve all links.

(P3) Please rephrase the previous content to be less aggressive and replace spam-like words and formulations.

Since simple text instructions did not yield consistent responses, thus preventing automated processing, our system utilizes OpenAI's function calling capability to make and retrieve the results of the API request. Inspired by the work of Koide *et al.*, we established the function call `print_result` [15]. The `is_success` property indicates whether the LLM was able to comply with the request or not and implies, if, for example, the request violates the model's ethical guidelines. If the request failed, the user gets an explanation of why it failed (`failed-description`) and a keyword, summarizing the reason for failing (`failed-keyword`), respectively. In Section 5, the latter is utilized to show the shortcomings of this prompt engineering approach, which in turn may help to understand the model's restrictions and enable further refinement of the function call.

Importantly, only the email body was modified, with no changes made to the headers by the LLM. The modified SMTP body was then merged with the SMTP headers to create new spam emails with altered content but consistent metadata.

4.3 Robustness Evaluation

To assess the effectiveness of the modified spam emails in evading the spam filter, the newly constructed spam emails were sent to the mail server for classification by SpamAssassin. Using the mail server API, we extracted the spam ratings and documented the classification results. Consequently, we were able to then calculate the success rate by determining how many of the modified spam emails were misclassified as legitimate. Additionally, we analyzed the extent to which the content's semantics had changed to understand the impact of LLM modifications on the emails' effectiveness. The following section presents the results of both, the success rate and the semantic similarities, respectively.

Table 2. Properties of functional call `print_result`

Property	Type	Description
`is_success`	boolean	A boolean value indicating whether the text could successfully be reformulated (true) or whether reformulating the text was not possible due to your rules (false).
`failed-description`	string	Detailed description on why the text could not be reformulated.
`failed-keyword`	string	One keyword summarizing the main reason why the text could not be reformulated.
`body`	string	If `is_success` is true, this field is filled with the reformulated text. Otherwise, this field is empty.

5 Evaluation

In the course of this evaluation, we want to look at the success rate of our approach (also in comparison to a rather simple dictionary-replacement attack approach), what might be a factor in hindering the success rate to be even higher, and whether our approach is able to preserve some form of semantic similarity between the original and the LLM-modified texts.

The effectiveness of the proposed approach is primarily assessed by measuring the rate at which modified spam emails are incorrectly labeled as legitimate emails. This metric of success is quantified using the success rate, which is defined as the ratio of emails classified as ham to the number of emails sent, as shown in Table 4. Employing our pipeline and applying our evaluation metric, we can see that 9.4% (*original*) and 73.7% (*minimal*) of the modified spam emails that were previously labeled as spam (*Spam* in Table 1) are now misclassified as ham. One can observe an even more remarkable success rate of 95.8% when considering the initial number of spam emails (*Sent* in Table 1) in relation to emails falsely labeled as ham (*Ham* in Table 1 and Table 4). However, it remains unclear whether these results can be entirely attributed to our pipeline or if the obsolescence of the dataset (almost 20 years old) and the choice of FUT are the driving factors. Based on the noticeable improvement from removing the non-required SMTP header fields (*original* → *minimal*), we can conclude that rephrasing the email's body actually has a great impact on its (mis)classification.

The number of rejected emails displayed in Table 3 shows that 20-25% of the requests made were rejected by GPT-3.5 Turbo. Since the rejections affect nearly a quarter of all emails, we utilize the `failed-keyword` property of the function call to get an understanding of the most frequent reasons, displayed in Figure 3.

Table 3. Number of emails during the second step: 2. LLM-modification.

Dataset	Initial	Pre-LLM	Rejected	Post-LLM
original	1.457	1.401	262	1.139
minimal	189	181	44	137

Table 4. Number of emails during the third step: 3. Robustness Evaluation.

Dataset	Initial	Sent	Classification Ham	Classification Spam	Success Rate
Dictionary					
original	1.457	1.457	6	1.451	0.4%
minimal	189	189	0	189	0%
LLM-modified					
original	1.139	1.135	107	1.028	9.4%
minimal	137	137	101	36	73.7%

After sending the same requests to GPT-4o, we observed nearly 95% of them being blocked. This high blocking rate was to be expected since GPT-4o operates under stricter guidelines as compared to its predecessors [7]. Therefore, GPT-3.5 is the easier-to-use solution for now.

As part of our pipeline's evaluation, we compare the semantics within the email bodies before and after the modification by the LLM. A spam email that was rephrased and misclassified by the filter as ham is not very useful to the email's author if the original intent or message is no longer conveyed correctly. This is also why Prompt P2 instructs the model to preserve all links. To enable accurate comparison of text bodies, it is essential to first standardize their format. This involves removing any line breaks and HTML tags, and reducing multiple spaces to a single space between words. Additionally, all non-letter and non-space characters are eliminated. Once the texts are uniformly formatted, the next step is to analyze their similarity. This can be accomplished using GPT4All.Embed4All()[5] to obtain vector embeddings of the pre- and post-modified email bodies. With that, we calculate the cosine similarity, quantifying the degree of semantic similarity between both texts - where 1 indicates equal semantics and -1 opposite meaning.

Figure 4 displays the distribution of emails and the corresponding cosine similarities of their bodies, for both, our approach (*LLM-modified*) and the low-effort approach of a dictionary-replacement attack (*Dictionary Attack*). Overall, the mean cosine similarity for our approach lies at approximately 0.8, indicating a high semantic similarity to the original text. The variance of similarities can be explained by the fact that the LLM does not only exchange single words but entire sentences or paragraphs. Thus, some degree of divergence from the original email bodies is expected.

To justify the need for our pipeline, we implemented a simple dictionary replacement attack, where each identified spam-like word in the email bodies was replaced with a less spam-like alternative. To generate this dictionary, more than 750 spam-like words were extracted from [9] and we generated alternative, less spam-like formulations with GPT-4-Turbo. Our findings indicate that

Fig. 3. Word cloud with reasons given by GPT-3.5-turbo-0125 on why it did not process the API request.

[5] https://docs.gpt4all.io/gpt4all_python/ref.html#gpt4all.gpt4all.Embed4All

the success rates of the dictionary replacement attack were comparatively low, with only 0.4% for the *original* and 0% for the *minimal* data set. Furthermore, our evaluation, as shown in Figure 4, revealed that the cosine similarities were mostly 1, indicating that the replacements, while linguistically similar, did not significantly alter the underlying message. A result, that was anticipated, since only one word is replaced with a (semantically) similar one, not changing the underlying message too much. These results, depicted in Table 4, underscore the limitations of straightforward replacement strategies.

With a total expense of $2.45 (approximately 0.17 cents per email) for modifying 1,443 spam email bodies, the proposed method is highly cost-effective. Additionally, it is important to consider that each spam email is typically sent to thousands of email addresses. This widespread distribution means the returns are significantly amplified, making the method economically advantageous. Therefore, as LLMs are eminently accessible, relatively inexpensive, and easy to use, only requiring some prompt engineering, using LLMs was a reasonable choice. They provide efficient and cost-effective means to process and generate large volumes of text, leveraging their accessibility and ease of use without needing extensive (local) computational resources or specialized expertise.

6 Conclusion

In this work, we addressed the emerging threat of LLMs, with their remarkable power in generating and modifying text, to Bayesian spam filters, specifically targeting SpamAssassin and its performance against LLM-modified email contents. We developed a pipeline that utilizes ChatGPT-3.5 Turbo to rephrase spam emails and test the spam filter's efficacy against the modified contents. Our findings show that in our setup, SpamAssassin misclassifies 73.7% of the

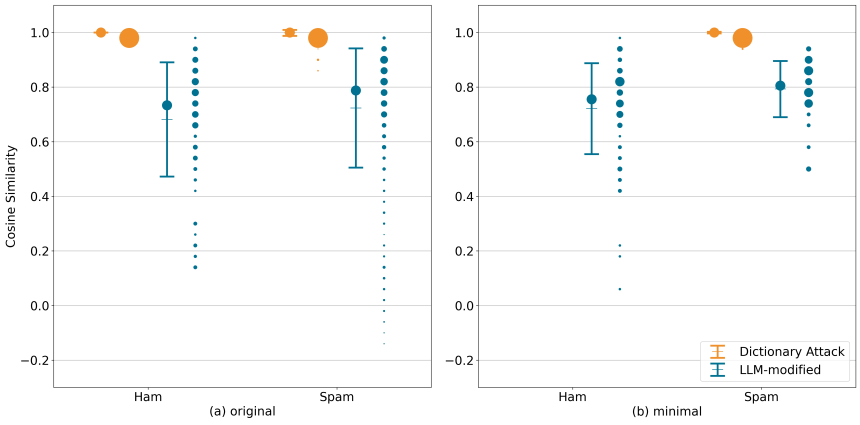

Fig. 4. Cosine similarity for the datasets (a) original and (b) minimal.

LLM-modified spam emails, and after going through the entire pipeline, 95.8% of the initial spam emails as ham. Compared to the low-effort approach of utilizing a dictionary-replacement attack that only showed a maximum success rate of 0.4%, our work shows much more promising results. Combined with the GPT-3.5 Turbo's cost-efficiency (0.17ct/email), LLM-modified spam emails pose a substantial threat to Bayesian spam filters, for they successfully bypass existing filtering rules without majorly distorting the embodied message and its objective. In our future work, we intend to evaluate the performance of our pipeline in comparison to highly configured instances of SpamAssassin and other filters. Additionally, we plan to use various datasets and other LLMs to validate our findings.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. ASSP - Anti-Spam-SMTP-Proxy, http://www.thockar.com/assp-home/, Accessed: 2024-07-03
2. Bogofilter Home Page, https://bogofilter.sourceforge.io/, Accessed: 2024-07-03
3. DSPAM Project Homepage, https://dspam.sourceforge.net/index.php, Accessed: 2024-07-03
4. SpamBayes: Bayesian anti-spam classifier written in Python, https://spambayes.sourceforge.io/, Accessed: 2024-07-03
5. Stop Ransomware - General Information, https://www.cisa.gov/stopransomware/general-information, Accessed: 2024-06-24
6. Text generation - OpenAI API, https://platform.openai.com/docs/guides/text-generation, Accessed: 2024-06-27
7. Hello GPT-4o — OpenAI (May 2024), https://openai.com/index/hello-gpt-4o/, Accessed: 2024-07-04
8. Al-Jarrah, O., Khater, I., Al-Duwairi, B.: Identifying potentially useful email header features for email spam filtering (2012)
9. Dubrulle, J.: Taking on the Spam Filters: 750 Spam Words to avoid in 2023 (Jan 2022), https://mailmeteor.com/blog/spam-words, Accessed: 2024-06-21
10. Gallagher, S., Gelman, B., Taoufiq, S., Vörös, T., Lee, Y., Kyadige, A., Bergeron, S.: Phishing and Social Engineering in the Age of LLMs. In: Kucharavy, A., Plancherel, O., Mulder, V., Mermoud, A., Lenders, V. (eds.) Large Language Models in Cybersecurity: Threats, Exposure and Mitigation, pp. 81–86. Springer Nature Switzerland, Cham (2024). https://doi.org/10.1007/978-3-031-54827-7_8
11. Gascon, H., Ullrich, S., Stritter, B., Rieck, K.: Reading between the lines: Content-agnostic detection of spear-phishing emails. In: Bailey, M., Holz, T., Stamatogiannakis, M., Ioannidis, S. (eds.) Research in Attacks, Intrusions, and Defenses. p. 69–91. Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-030-00470-5_4
12. Infrabot, A.: WhyUseRules - SPAMASSASSIN - Apache Software Foundation (Sep 2009), https://cwiki.apache.org/confluence/display/SPAMASSASSIN/WhyUseRules, Accessed: 2024-07-03

13. Islam, R., Abawajy, J.: A multi-tier phishing detection and filtering approach. Journal of Network and Computer Applications **36**(1), 324–335 (Jan 2013). https://doi.org/10.1016/j.jnca.2012.05.009
14. Karanjai, R.: Targeted phishing campaigns using large scale language models (Dec 2022), http://arxiv.org/abs/2301.00665, arXiv:2301.00665 [cs]
15. Koide, T., Fukushi, N., Nakano, H., Chiba, D.: Chatspamdetector: Leveraging large language models for effective phishing email detection (2024). https://doi.org/10.48550/ARXIV.2402.18093, https://arxiv.org/abs/2402.18093
16. Motlagh, F.N., Hajizadeh, M., Majd, M., Najafi, P., Cheng, F., Meinel, C.: Large Language Models in Cybersecurity: State-of-the-Art (Jan 2024). https://doi.org/10.48550/arXiv.2402.00891
17. Nabeel, M., Altinisik, E., Sun, H., Khalil, I., Wang, H.W., Yu, T.: Cadue: Content-agnostic detection of unwanted emails for enterprise security. In: Proceedings of the 24th International Symposium on Research in Attacks, Intrusions and Defenses. p. 205–219. RAID '21, Association for Computing Machinery, New York, NY, USA (Oct 2021). https://doi.org/10.1145/3471621.3471862
18. Resnick, P.: Internet Message Format. RFC 2822 (Apr 2001). https://doi.org/10.17487/RFC2822, https://www.rfc-editor.org/info/rfc2822
19. Slooten, R.: Mailpit. https://hub.docker.com/r/axllent/mailpit (Jun 2024), Version v1.19.0
20. Slooten, R.: Spamassassin. https://hub.docker.com/r/axllent/spamassassin (Jan 2024), Latest build
21. Webb, S., Chitti, S., Pu, C.: An experimental evaluation of spam filter performance and robustness against attack. In: 2005 International Conference on Collaborative Computing: Networking, Applications and Worksharing. pp. 8 pp.– (Dec 2005). https://doi.org/10.1109/COLCOM.2005.1651219