Research Paper - Using Artificial Intelligence to Mitigate File Injection Attacks in Emails

Denis Arva
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
d.arva@student.utwente.nl

ABSTRACT

Searchable Encryption is a cryptographic technique that allows a client to search for keywords across encrypted files on a server while ensuring privacy and protection of the user data. Such a technique is ideal for poorly secured servers because even if these servers are compromised in a cybersecurity attack, the files are encrypted, and the content cannot be read. However, it is possible to retrieve plaintext of an encrypted keyword using a file injection attack because the *Searchable Encryption* leaks to the server information about the query results.
As proposed by Zhang et al. at Usenix Security in 2016, the *file injection attack* consists of a server sending a set of files to its client to recover the plaintext of the searched encrypted keywords. The number of injected files depends on the attacks and countermeasures adopted. One of the possible countermeasures for such an attack is to limit the files returned in a query by setting a threshold of keywords that an indexed file should have.
In this work, two countermeasures are compared , one unsophisticated proposed by Zhang et al. at Usenix Security in 2016 - threshold countermeasure - and another based on artificial intelligence proposed by Lui et al. at International Workshop on Security and Privacy Analytics in 2020. Each of them comes with advantages and disadvantages. However, this research paper proved that, when mitigating malicious files, the countermeasure based on artificial intelligence performs better than the threshold; still, it underperforms when it comes to the effects on benign files.

Keywords

Machine learning, searchable encryption, injected files, threshold, binary search attack, artificial intelligence, natural language processing, enron, malicious, benign, emails

1. INTRODUCTION

This research is essential for various reasons, one of which is the growing demand for cloud services. *Searchable Encryption* is used as a technique in these services precisely because cloud service is generally used to store files. Therefore, the files must be encrypted for privacy principle before being stored to prevent them from being read if servers are compromised.

It is possible for an attacker to use a binary search attack for a longer time and map each recovered keyword to an email, after which to deduce the context of that email.

For instance, if a file is given that has the following content: "Mr Denis, you will receive the new bank card by mail at Roerstraat on Wednesday." Suppose the attacker injected a large set of keywords containing the addresses of the city where the person lives and more. In that case, he will finally have the following set mapped to the file $S = \{'Denis', 'card', 'Roerstraat', 'Wednesday'\}$, and he can already understand the context of the email. Therefore, the attacker could use the situation to take the card from the mailbox.

Also, during the covid pandemic, Lallie et al.[10] showed that the cyberattacks increased for several reasons. Since people are much more active online, they are also more exposed to cyber-attacks. In addition, in a pandemic context, the unemployment rate is rising rapidly, out of the need to have a job, some people get involved in these illegal attacks to support their financial situation disrupted by the effects of the pandemic.

The role of *Searchable Encryption* is to keep search functionality on an untrusted server while ensuring privacy and data protection such as encrypted files and search queries. Over time, efficient *Searchable Encryption* schemes [4, 6, 13, 15] have been developed that have reached an optimal search cost, but this is achieved with the cost of leaking list information with the files containing the searched keyword and pointers to these files. Such a technique is also used by email, for example, Pmail[8]. The purpose of this research is to use artificial intelligence to mitigate file injection attacks that can be performed precisely because of these leaks of information that *Searchable Encryption* has.

Zhang et al.[16] researched the file injection attack on different methods, and some use fewer injected files than others to recover the plaintext of the encrypted keywords. One of these methods, being the basis for the rest of the methods, is called *Binary Search Attack*.

One way to countermeasure this attack - as explained by Zhang et al.[16] - is to set a threshold of keywords that a file may have. If a file exceeds this threshold, it will not be indexed by the search functionality. It is essential to mention that the threshold must be carefully chosen not significantly to affect the user experience. However, the server can modify the *Binary Search Attack* to avoid this countermeasure by reducing the keywords placed in a single file from $\frac{|K|}{2}$ to $\frac{|K|}{2T}$, but this means that the server must inject more files to identify the plaintext of an encrypted keyword.

Alternatively, it is possible to use artificial intelligence to

35^{th} Twente Student Conference on IT July 2^{nd}, 2021, Enschede, The Netherlands.

mitigate file injection in email systems. An example would be an algorithm that runs on the client-side, analyzes the content of a file, and accepts or refuses its indexing. This research is focused on analyzing and creating a countermeasure using artificial intelligence and its performance to be compared with the performance of the threshold method. Also, to the best of our knowledge, no research has been done comparing these two countermeasures.

Both countermeasures are designed to work on the client-side so the attacker cannot influence the implementation of countermeasures in one way or another.

1.1 Contribution

This research paper will focus on using and comparing two countermeasures to mitigate the files injected by the *Binary Search Attack* in a context of an email dataset. After these countermeasures are analyzed, it will determine which of the two are recommended for use. The first countermeasure analyzed is the one presented by Zhang et al.[16], namely threshold, which set to a number N, it will filter out emails that have the number of keywords greater than N.

The second countermeasure analyzed is presented by Lui et al.[11] and is based on artificial intelligence that will read the content of emails. If an email has weak semantics content, it will be filtered out, and if the email's content is high in semantics, it will be left to be indexed to the user.

The contribution comes when the performance of these two countermeasures is measured. The role of a countermeasure is to binary classify an email as malicious or benign. For this research, the following statistical instruments are used to measure the binary classification performance: true positive, true negative. When a countermeasure qualifies an email as positive, it means that the email is malicious. The true-positive measure is given by the percentage of emails correctly classified malicious by a countermeasure in a given set of emails containing both malicious and benign. The true negative measure is given by the percentage of emails correctly classified benign by the countermeasure.

1.1.1 Research questions

- **RQ:** How efficient is the Artificial Intelligence in mitigating injected files in emails compared to the threshold countermeasure?
  - **SubRQ:** How well does artificial intelligence perform compared to threshold when true positive is considered?
  - **SubRQ:** How well does artificial intelligence perform compared to threshold when true negative is considered?

To determine which of these two countermeasures are recommended to be used, the term efficiency is defined. In the context of this research project, A countermeasure is all the more effective as the percentages of true positives and true negatives are high. For example, an 100% effective countermeasure would have a percentage of 100% true positive and 100% true negative. The lower these percentages, the inefficient is the countermeasure. However, which is the best option is directly dependent on someone's needs. Maybe a person wants privacy; then the true positive must be as high as possible. Conversely, if a person wants to have as many emails available as possible, then one will choose a true negative as high as possible.

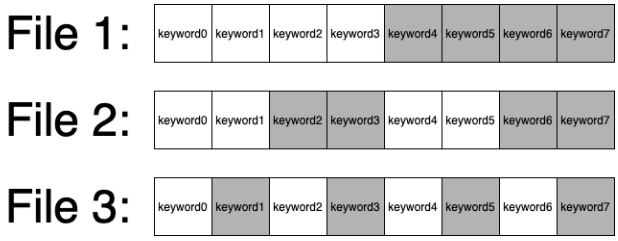

Figure 1. Three injected files and its content. This files are injected with Binary Search Attack, using a universe of keywords of 8 keywords.

The main research question is answered through the experiments made on a data set by Enron[5] public emails, but in order to answer the main question, the sub research question is answered privately in the experimental phase. Therefore, the following research questions are relevant for the research:

1.2 Organization of the Paper

This research paper is organized into several parts in which the following aspects are clarified: methodology, experiments and results.

In the methodology part, a systematic approach is given that answers the proposed research questions. It also contains implementation aspects of *Binary Search Attack*, for which pseudocode is attached to understand how it is implemented. The methodology also presents the implementation aspects of the threshold and artificial intelligence countermeasures. At the same time, this part mentions one of the most critical aspects of this research, namely how machine learning was trained to reach a prediction score of approximately 97% on the data used for training.

In the experiments, different plots are examined, that are relevant to answer the research questions assumed.

In the results and conclusion section, conclusions are drawn about the experiments performed, which of the countermeasure is more suitable to be used in this Enron email dataset [5].

2. BACKGROUND

Searchable Encryption is a technique by which the user can store their encrypted files on a server and use the search functionality simultaneously. The process is as follows: The user generates a token p by encrypting the keyword k, and the server responds with a list of file identifiers containing the token p. The user now knows which of the files contains the searched keyword and in this way its privacy is protected.

In this attack, the server generates an universe of keywords $K = \{keyword_0, keyword_1, \dots, keyword_n\}$, with the length of $|K| = n+1$; it generates several files $\log_2 |K|$, and each file contains precisely half of the keywords in the K universe. All these files are sent to the client, who, in turn, it will encrypt and store them. The server learns the plaintext of a keyword searched by the client if the injected files are in the query result.

Using Figure 1 as an example where the three files are generated, the server injects the files and learns that the plaintext of an encrypted keyword is equal to $keyword1$ if, after a search, only file three returned. Alternatively, if $File_3$ and $File_2$ returned after a search, the plaintext of the encrypted keyword used in the search is $keyword3$. Moreover, the Figure shows a set of files $F = \{File_1, File_2, File_3\}$

Algorithm 1 Binary Search Attack - Inject Files

Input: K (universe of keywords)
Output: $Files$ (array of malicious files)

```
1: function InjectFiles(K)
2:     Files ← empty array of files
3:     N ← length(K)
4:     for i = 1, ..., log2 N do
5:         Generate a file F_i containing the words in K
6:         whose ith bit is 1
7:         Append F_i to F
8:     return Files
```

Figure 2. Function of the *Binary Search Attack* that generates a set of files according to the given keyword universe. At the end, an array of files is returned that are ready to be sent to the client.

and if any $File_M$ is given, the keyword that has the 1st bit equal to 1, of the bit sequence generated from the position number, that keyword is stored. For example, $keyword_4$ is stored in $File_1$ because position 4 has the first bit 1.

2.1 Related work

Zhang et al.[16] researched in their paper how the information leaks of *Searchable Encryption* can be exploited to obtain the plaintext of a keyword. They looked at different file injection methods, some of which even used a few injected files to get the plaintext. However, this paper focuses on attacks and not defence methods.

Other countermeasures for the attacks mentioned above were studied by Bost et al.[2]. In *Searchable Encryption*, forward privacy is a property that mitigates injected files by ensuring that the server cannot link the result of the current query and past queries. Also, backward privacy is, in turn, a property that mitigates injected files: search query should not return deleted matching entries.

Furthermore, Liu et al.[12] came up with a countermeasure called Vaccine. The idea is that every time a client receives a file, whether it is legitimate or injected, the client injects a file created by him, and this file contains a set of random keywords that are not intersected with the set of keywords in the received file. As a result, the file injected by the client obscures the file access pointer of some files injected by the server.

Finally, Liu et al.[11] propose a new defence method based on natural language processing. Because they have confirmed that injected files have low semantics, they concluded that it is feasible to automatically check the semantics of a file to mitigate file injection by accepting to index if the file has high semantics or refuse it otherwise.

3. METHODOLOGY

Before going into the aspects of implementing prototypes for Searchable Encryption, *Binary Search Attack*, Threshold and AI countermeasure, it is presented systematically how this research is done:

1. A set of email data available for research is used, after which a prototype is made in python to simulate the core functionality of *Searchable Encryption* and a prototype to simulate a *Binary Search Attack*.

2. A threshold countermeasure is performed, and its efficiency against this attack is analyzed.

Algorithm 2 Binary Search Attack - Recover keyword

Input: $R, Files$ (query results, injected files)
Output: $keyword$ (plaintext of the encrypted keyword)

```
1: function RecoverKeyword(R, Files)
2:     Analyse the query results R and return the
3:     keyword associated with the files returned.
4:     return keyword
```

Figure 3. Function of the *Binary Search Attack* that takes as input query results to a search that the client did and returns the plaintext of the searched keyword.

3. A prototype based on artificial intelligence is implemented to distinguish between an injected email and a legitimate one. Such a prototype is using natural language processing to determine the semantic level of an email; if the level is low, this email will not be indexed, but it is indexed if the semantic level is high. This countermeasure is also be analysed.

4. Finally, both prototypes are analyzed to determine the differences between them, which has a more significant impact on legitimate emails and is more effective in mitigating file injection.

The Enron email dataset[5] contains the emails of a former American corporation, with a size of 1.7Gb and it is organized in 150 folders. Each folder contains the emails of a former corporate employee, and the folder name is *lastname-initial*.

3.1 Searchable Encryption, Binary Search Attack and Threshold

As proof of concept, a prototype is made that mimics the functionality of a *Searchable Encryption*, but the encryption of files and keywords is avoided because it is out of the research goal and does not influence the success or fail rate of the attack. This is because the proof-of-concept of this research is to demonstrate how the attack works and can be mitigated. As explained in the Background section, the user sends a p token of the keyword he wants to search, the server returns a list of file identifiers that contain the word and *Binary Search Attack* uses only that list.

This attack as presented by Zhang et al.[16] is divided into generating malicious files and recovering the keyword plaintext. First of all, this algorithm generates malicious files based on the pseudocode shown in Figure 2, which takes a universe of keyword or, more simply, a set of keywords that the attacker is interested in finding out. Then the files are sent to the client who stores them, encrypts them and sends them to the server to be stored. Second, after the client has searched for a keyword among its files, the server takes the query results and runs the recover function of the keyword plaintext presented in Figure 3 using the query results together with the universe of keywords.

A prototype is made after this *Binary Search Attack* and combined with the prototype specified above that mimics the functionality of *Searchable Encryption*; it can show how an attack can be performed using the Enron email dataset [5].

These countermeasures are meant to be executed on the client-side, so the threshold countermeasure is implemented inside the *Searchable Encryption* prototype. When the

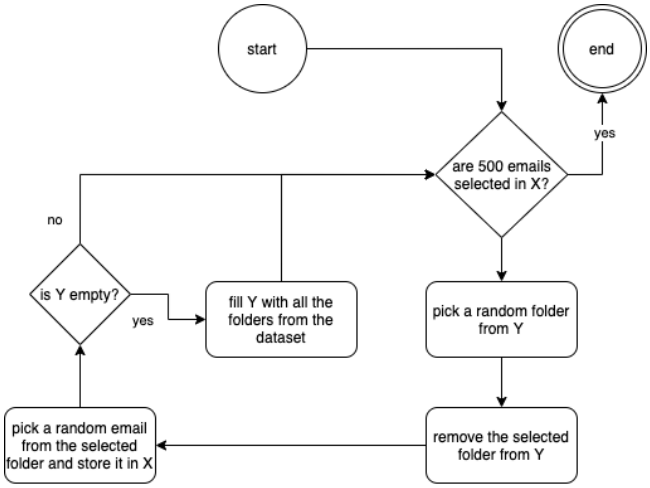

Figure 4. The process of randomly picking an email from the Enron email dataset

prototype opens and reads an email to search for the keyword requested by the client, the email is ignored if it has more keywords than it is set the threshold.

3.2 Countermeasure based on Artificial Intelligence

In their paper, Liu et al.[11] used machine learning with binary classification to distinguish between malicious and benign emails. However, it is not quite specified what kind of model it uses for training and prediction. At the same time, they mention that they used 500 benign emails and another 500 malicious emails for training. The features they have proposed and consider relevant to train machine learning in the proper way to determine the semantics level of an email are divided into syntactic and structural features.

The syntactic features are:

- the number of nouns
- the number of verbs
- the number of adjectives
- the number of adverbs
- the number of personal pronouns

The structural features are:

- the number of words
- the number of sentences

Furthermore, they used natural language processing toolkit[1] from Python Package Index(PyPI)[14] to be able to obtain these features from emails, which was also done in this research paper.

3.2.1 Prepare the data

Before training machine learning, the emails used for training must be prepared to avoid biasing the algorithm. Therefore, we implemented an algorithm that randomly selects emails from different folders until a certain number of random emails were selected. For example, in Figure 4 the algorithm picks random emails until 500 emails have been chosen. We chose to implement the algorithm in this way in order to avoid biases in the machine learning training.

Table 1. Frequency of Special Characters

TAG	Meaning
NN	Nouns
VB	Verb
JJ	Adjective
RB	Adverbs
PRP	Personal pronoun

After which a universe of keywords of 20000 keywords is generated, through a process in which an email is taken at random (as shown in Figure 4), its content is read, and all unique keywords are stored in an array, and the algorithm stops when 20,000 keywords have already been stored.

Using this universe of keywords generated exclusively from the keywords used in emails, an algorithm at each iteration generates a new subset of the universe of keywords, that set contains a randomly chosen number of keywords between 0 and 2000. This means that the length of a single malicious file cannot have more than 1000 characters. The algorithm counts how many malicious emails were injected into the dataset at each iteration and stops when more or 500 malicious emails were injected.

On average, after several tests, a malicious email has more keywords than a benign email. This situation is desirable because machine learning must learn that the more keywords in an email, the higher the chances of that email being malicious.

3.2.2 Training the machine learning

With the emails ready, for each of 1000 randomly chosen emails, natural language processing is applied to compose the features proposed above, and the respective email is labelled with one if it is malicious and with zero if it is benign. The toolkit available in Python reads an email and tags each word with its part of speech. Table 1 is presented on the first column TAG POS - part of speech - that natural language processing returns, and on the second column the whole meaning of that tag. For example, if a certain word is tagged as being JJ then it means that it is an adjective. Because the process is relatively slow, each time the feature set for an email is calculated, this set is stored in a row in a CSV file, and it can be used continuously, without being generated again.

With the CSV file completed with each feature and label of the 1000 emails, it is sent to training. The machine learning used is based on neural networks model from the sklearn[3] package, and the following configuration gives the best results: shuffle the CSV data, set the number of maximum iteration to 9000, set alpha to $1E-5$, and the hidden layers to $(150, 50)$. If the accuracy of this trained model is tested on the data used for training, the score calculated after several attempts is approximately 97%.

3.3 Comparing the results

In order to answer the research question, these implementations were needed in order to perform experiments on them finally. For the threshold, experiments were performed to determine what number should be assigned to this countermeasure to reduce false positives and what effects this discovered value has on *Binary Search Attack*. Additionally, for machine learning countermeasure, the accuracy of the prediction in different scenarios was analyzed. Throughout this process, the Jupyter Notebook[9] was used to draw the plots as a result of the experiments. Also, the implementations can be found on the SNT UT GitLab[7].

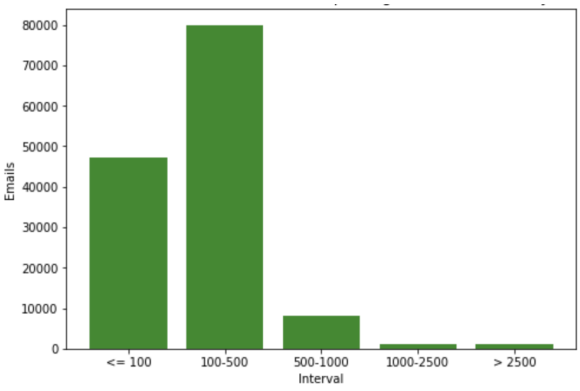

Figure 5. Distribution of emails on fixed size intervals in number of keywords

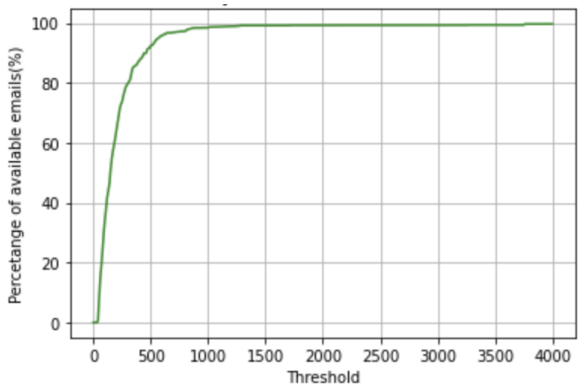

Figure 6. Plot with threshold effects on email availability.

4. EXPERIMENTS

For the beginning, Enron dataset[5] was analyzed, without malicious emails, to learn what value should be set to the threshold so that true positive and true negative are high. Malicious emails are not needed to analyze this countermeasure, because threshold filters all emails that are greater than its value, so true positive is not a percentage, but rather a binary aspect. True positive is 1 for emails larger than the threshold and 0 for emails lower than the threshold.

So Figure 5 shows the bar chart with the distribution of emails by ranges of number of keyword in email, and in addition to this bar, the following information was calculated: in total, there are about 137,000 emails, the largest email has 41,450 keywords. Along with this information, we see from Figure 5 that most emails have a keyword number between 100 and 500, about 58% of the dataset used.

The analysis of the threshold was concluded with Figure 6, where the plot shows the percentage of true positive that increases proportionally with the threshold. It can be seen that if the threshold is set to 1000 keywords, then almost 100% of the emails are available to the customer. It should be mentioned that no files injected into the dataset were introduced during this experiment because threshold filters any email that has the number of keywords higher than the set threshold.

After machine learning was trained with the methods explained above, two plots and a bar chart were obtained. Figure 7 analyzed only injected emails and followed the accuracy of this machine learning. The percentage of correct

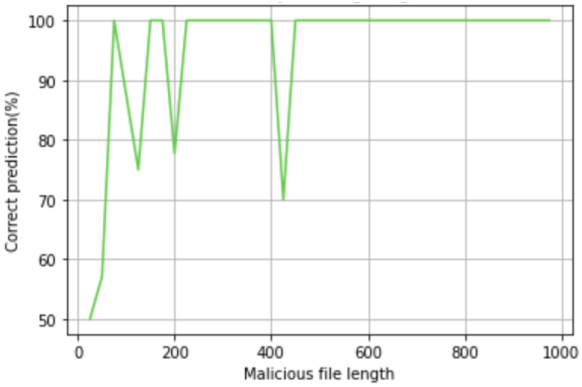

Figure 7. Machine learning prediction score over the malicious file containing from 10 to 1000 keyword.

prediction depends on the email file length. The higher the file length, the higher the percentage of correct prediction. Machine learning has a low prediction score that the injected files have less than 500 keywords. Approximately 92% of the dataset contains emails that have less than 500 keywords, and AI also considers how many keywords are in an email.

In Figure 8, it is a plot similar to Figure 7 except that it plots machine learning accuracy for injected files that have between 0 to 500 keywords. The reason for these sudden fluctuations from 100% accuracy to 10% is that the smaller the universe of keywords used for injection, the lower the number of emails injected. Recall from the introduction that *Binary Search Attack* injects log 2 from | K | for any K. For example, in Figure 8, if the injected files have 100 keywords, the universe of keywords used is 200 keywords and seven emails were injected. With an accuracy of about 10%, it means that an email out of seven malicious emails is filtered out.

In order to draw the bar chart in Figure 9, 100 benign emails were randomly extracted, and 100 malicious emails were injected in the number of keywords range on the x-axis. It is observed that the percentage of true positive decreases and the percentage of true negative decreases as the keywords increases. For emails that contain less than 100 keywords, machine learning has a true positive of almost 70% and a true negative of almost 95%. This figure is important for this research because it reflects the performance of machine learning in mitigating malicious emails.

5. RESULTS

With Figure 5 and 6 in mind, the threshold should be set to 1000 keywords because most 98% of emails have less than 1000 keywords, and by setting the keyword to this size, only 1.54% of all benign emails are filtered. In this situation, the threshold has a true negative percentage of about 98%. The true positive in this case is not measured in percentage, but in two numbers, 1 or 0, because it totally depends on how big or small the injected files are. If the injected files have fewer keywords than threshold value, then the true positive is 0, otherwise is 1. The disadvantage of this countermeasure is that it cannot mitigate malicious emails that have less than 1000 keywords in the current context. However, the attacker is placed in a difficult situation because one can not use a universe of keywords larger than 1999 keywords.

In the case of artificial intelligence, if we consider Figure 9,

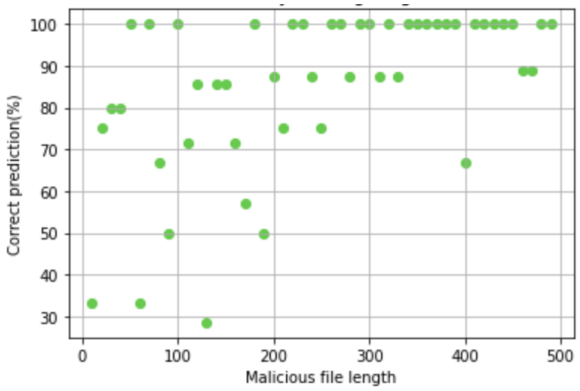

Figure 8. Machine learning prediction score over the malicious file containing from 10 to 500 keyword.

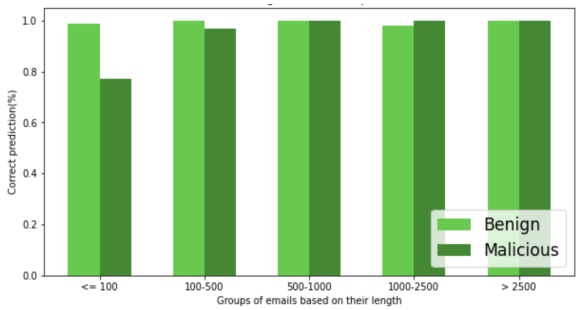

Figure 9. A bar chart with the machine learning prediction score for every group of emails having the number of keywords in that interval. 100 are benign and 100 are malicious for every group.

the percentage of true negative for each interval is decreasing, but all are over 95%. Moreover, the percentage of true positive is 70% in the range of 0-100 keywords but is already almost 100% in the range of 500-1000, representing 58% of the data set.

6. CONCLUSION

Threshold and artificial intelligence countermeasure have their advantages and disadvantages. The threshold can mitigate to a small extent malicious emails but comes with the advantage that it has minimal effects on the user experience with the email service by filtering very few benign emails.

On the other hand, countermeasure based on artificial intelligence mitigates malicious emails very well but filters more benign emails compared to the threshold. Following the experiments, it was observed that for malicious emails with few keywords, machine learning had a true positive percentage that could reach even 30%. However, it is essential to mention that if, for example, out of 7 emails injected by *Binary Search Attack*, one was filtered, the the attacker fails to recover the true plaintext of the searched keyword.

All in all, it is clear that the countermeasure based on artificial intelligence is more efficient than the threshold, but it is also a decision that the client wants to make. One is put in the position to choose between privacy or availability. Privacy is better ensured by artificial intelligence and availability is better ensured by the threshold countermeasure.

7. REFERENCES

[1] E. L. Bird, Steven and E. Klein. Natural language processing with python. *O'Reilly Media Inc*, 2009.

[2] R. Bost, B. Minaud, and O. Ohrimenko. Forward and backward private searchable encryption from constrained cryptographic primitives. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 1465–1482, New York, NY, USA, 2017. Association for Computing Machinery.

[3] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[4] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Roşu, and M. Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In R. Canetti and J. A. Garay, editors, *Advances in Cryptology – CRYPTO 2013*, pages 353–373, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[5] W. W. Cohen. Enron Email Dataset. https://www.cs.cmu.edu/~enron/, 2015. [Online; accessed 1-June-2021].

[6] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, page 79–88, New York, NY, USA, 2006. Association for Computing Machinery.

[7] A. Denis. Research - Using Artificial Intelligence to Mitigate File Injection Attacks in Emails. https://git.snt.utwente.nl/s2188511/research-mitigate-file-injection-ai-in-se, 2021. [Online; accessed 20-June-2021].

[8] D. Harris. Pmail. http://www.pmail.com/, 2000. [Online; accessed 1-June-2021].

[9] IPythonProject. Project Jupyter. https://jupyter.org/, 2014. [Online; accessed 1-June-2021].

[10] H. S. Lallie, L. A. Shepherd, J. R. Nurse, A. Erola, G. Epiphaniou, C. Maple, and X. Bellekens. Cyber security in the age of covid-19: A timeline and analysis of cyber-crime and cyber-attacks during the pandemic. *Computers Security*, 105:102248, 2021.

[11] H. Liu and B. Wang. Mitigating file-injection attacks with natural language processing. In *Proceedings of the Sixth International Workshop on Security and Privacy Analytics*, IWSPA '20, page 3–13, New York, NY, USA, 2020. Association for Computing Machinery.

[12] H. Liu, B. Wang, N. Niu, S. Wilson, and X. Wei. Vaccine:: Obfuscating access pattern against file-injection attacks. In *2019 IEEE Conference on Communications and Network Security (CNS)*, pages 1–9, 2019.

[13] M. Naveed, M. Prabhakaran, and C. A. Gunter. Dynamic searchable encryption via blind storage. In *2014 IEEE Symposium on Security and Privacy*, pages 639–654, 2014.

[14] PythonCommunity. Python Package Index . https://pypi.org/, 2021. [Online; accessed

1-June-2021].

[15] E. Stefanov, C. Papamanthou, and E. Shi. Practical dynamic searchable encryption with small leakage. *IACR Cryptol. ePrint Arch.*, 2013:832, 2014.

[16] Y. Zhang, J. Katz, and C. Papamanthou. All your queries are belong to us: The power of file-injection attacks on searchable encryption. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 707–720, Austin, TX, Aug. 2016. USENIX Association.