DEPARTMENT OF COMPUTER SCIENCE

# Is content-based filtering still broken?

Yijing Zhu

---

A dissertation submitted to the University of Bristol in accordance with the requirements

of the degree of Master of Science in the Faculty of Engineering

September 2020 | CSMSC-20

# Declaration

This dissertation is submitted to the University of Bristol in accordance with the requirements of the degree of Master of Science in the Faculty of Engineering. It has not been submitted for any other degree or diploma of any examining body. Except where specifically acknowledged, it is all the work of the Author.

Yijing Zhu, September 2020

# Executive Summary

This thesis reviews the evolution of scam emails and replicates a series of experiments which reflects the weaknesses in widely used content-based filtering. In addition, it explores how Gmail, Yahoo Mail, Outlook, and ProtonMail respond to scam emails modified with confusable characters. Given the fact that confusable characters are similar to the Latin Alphabets, if Latin letters in sensitive words are altered, email filters will not recognize keywords used in spam filtering, yet the differences are not noticeable for human's eyes. It is also defined as homograph attacks. In terms of this thesis's findings, it could be concluded that some email providers, such as Gmail and ProtonMail, have taken steps to combat the attacks, while it remains to be improved.

To fulfill the research purpose, the author has carried out the explorations as follows.

- The evolution history of scam emails and its sending frequency were researched and learned.
- The high-fidelity confusable characters were selected from Unicode.
- An obfuscator and a de-obfuscator were built in Java.
- 90 selected scam emails were obfuscated and sent from 8 senders' mailboxes (two for each email provider) to 4 inboxes (one for each email provider).
- 30 obfuscated normal emails were also sent. The receiving results of these obfuscated normal emails and the previous 90 selected scam emails were both recorded and analyzed.

The author has drawn the following conclusions in this thesis.

- Gmail performs best at the inbound time among the four email providers.
- ProtonMail also performs well but it could not detect messages sent from ProtonMail account, which could be deemed as n hole when defending against homograph attacks.
- Outlook appears to be a good neighbor that blocks most of the obfuscated scam emails at the outbound time among the four email providers. However, it is strange that it did not apply this defense on the receiver's side.
- Yahoo Mail seems to have not taken any action against homograph attacks.
- The filters for all four mailboxes currently have a blocking mechanism that may block regular emails with homographs. Therefore, it is recommended to use de-obfuscator to increase the blocking accuracy.

# Acknowledgments

I would like to thank my supervisor Dr. Matthew Edwards for his continuous support and guidance throughout my MSc project. He is a responsible person who provided me with samples of scam emails and gave me useful advice on several occasions.

I would also like to thank my friends for helping me sending emails and record receiving results.

Special thanks also go to my family for their constant support and encouragement.

# Content

# 1. Introduction

People may have all experienced spam emails, as they have constituted up to 75–80% of email messages since 1998 [1]. Spam emails include advertisements, publications, or other useless materials that users did not subscribe to and may lead to several harmful effects. Scam emails are only one of them and are the most dangerous and deceptive.

For instance, spam leads to the misuse of storage space and computational power [2]. It results in a loss of productivity due to wastes users' time by making them browse and categorize additional email and violates their privacy rights. Finally, spam raises legal issues through advertising pornography, etc. [3]. However, scam email not only possesses the aforementioned harm that spam has, they also deceive people, cause financial losses, and in some cases psychological trauma from being deceived [15].

The good news is that scams have been on the decline nowadays due to several methods created to deal with it. However, scams still exist, and scammers have never stopped finding new ways to evade detection.

By the end of the nineties, scam emails had become mainstream for scammers to defraud people [4]. Anyone with an email can send scams, thus it is unrealistic to build a defense on the termination of online services directly. Instead, email providers decided to scan the email content to look for sensitive keywords they identified in the previous scam messages. For example, providers may choose to block "password", as some scammers ask recipients to give them passwords using various excuses. When filters detect an email with a sensitive word as such, it will send it to the spam box, which significantly reduces the spam. However, spammers quickly found a counter to this method by morphing sensitive words like

"password" into "p@s.sW?or.d" to avoid detection. This method could work for those who aim to advertise and sell products, but for those trying to commit financial fraud, it does not work, since recipients can identify them as scams immediately as these morphed keywords showed no authenticity.

Thus, in 2017, Dhiman et al. believed the next move of scammers would be to use homograph attacks [4]. Homographs are letters that are visually identical to Latin letters in the eyes of recipients, and with these homographs, scam emails would evade detection and blockage. For example, the Latin letter "w" could be replaced by the Cyrillic letter "w" because of their resemblance. They demonstrated that the existing content-based filtering for detecting scam emails had no means to detect such homograph attacks and built a simple obfuscator to prove this. For each letter of an original message, the obfuscator randomly decided whether it should replace the character with a glyph chosen from a list of manually identified homographs. The obfuscated messages are not detectable to human eyes; thus, they can appear authentic to recipients. Dhiman et al. also demonstrated that these messages were not detectable to scam email detectors, as homographs hid sensitive words. More specifically, homographs may alter some letters in sensitive words, and the content-based filtering systems would not regard these words as suspicious. In the experiment, they changed 90 scam messages and sent them respectively to Hotmail, Gmail, and Yahoo mail inboxes. The result was 96% of posted scam messages were successfully delivered. Therefore, Dhiman et al. demonstrated that the existing content-based filtering could not defend against homograph attacks in 2017.

Webmail providers should be able to counter these attacks now, as they have had at least three years since researchers raised this problem. To protect their users. they should update their identification system accordingly to accommodate the rapid development of scam methods. This paper aims to replicate the experiment presented by Dhiman et al. and hopes to demonstrate that there would be few

successes of homograph attacks. This study seeks to re-analyze the protection offered by Hotmail, Gmail, and Yahoo mail providers and extends to another email provider ProtonMail. This webmail provider positions itself as a secure email service provider, based in Switzerland, accessible among people who are particularly concerned about their privacy. Therefore, it makes sense to research the security of ProtonMail and compare it to the other three.

Moreover, this paper also extends the study to obfuscated regular messages. This step aims to test if email providers regard all obfuscated messages as scams or only block obfuscated scam messages. It is essential to include this extension because if email providers block all messages with homographs, they are not responsible for their users who write non-Latin text as their first language or non-Latin academic research communication, as it may prevent regular message flow. Ideally, webmail providers' content-based filtering would only block malicious messages.

With these additional contents based on the original study, the proposed work are sectioned mainly into four steps. Firstly, find confusable characters and use them to build an obfuscator as the original paper did. Secondly, choose 90 scam messages and resend them to ensure their validity as scam messages, by the measurement that they should be blocked by at least two email providers. The results are provided as a benchmark for comparison of the obfuscated contents. Thirdly, the chosen obfuscated scam messages will be sent respectively to the inboxes of four email providers. To mimic the frequency of which scammers send scam emails to get around the volume and reputation technical checks, I designed to send scam messages in small batches and sent 90 scams over five days. Finally, 30 innocent messages will be obfuscated and sent to all four inboxes.

The remainder of this thesis is structured as follows: Section 2 gives an overview of relevant work regarding homograph attacks and an in-depth look at the original

study by Dhiman et al. Section 3 describes the methodology of the experiment and discusses changes of the design in detail. Section 4 presents the results. Result analysis is performed in Section 5, and the discussion is listed in Section 6.

# 2. Background

## 2.1 Unicode

Most homograph attacks are implemented under Unicode's help; therefore, homograph attacks can also be called Unicode attacks. Unicode is a worldwide character coding standard developed by the Unicode consortium. The standard was established to solve the limitation of the traditional character coding scheme that does not support different languages. It sets a unified and unique binary code for each character in each language to meet the requirements of text conversion and processing across languages and platforms [5].

The Unicode standard provides a uniform, fixed-width, 16-bit character recognition and encoding scheme that covers most language systems used in the world today. The Standard Encoding for version 1.0 has about 28,000 characters, of which about 3,000 are letters and phonetic symbols of European, Indian and Asian languages, and 1,000 are symbols and graphic elements, 24,000 characters are ideographic, syllables and phonetic symbols used in Chinese, Japanese and Korean [7].

However, as national characters are added to Unicode and used worldwide, it also brings potential risks. Many visually similar characters coexist in Unicode, such as Latin 'a' (0041), Cyrillic 'a' (0410), and Greek Alpha (0391) are distinct Unicode characters that could all be represented by one glyph. Certain Latin letters could be replaced by a similar character, and the change would not be detectable by human eyes, but the word would then slip past content-based filtering. Scammers quickly exploited these potential risks for scam email attacks, phishing attacks, and web identity attacks [6].

## 2.2 Various applications of homograph attack

Helfrich et al. introduced various cases of homograph attacks using Unicode in 2012 [9]. The first type is commonly used in URL spoofing and phishing attacks. In these cases, attackers provide a URL that appears to be a trusted website while in fact a malicious site that may advertise to the user or, worse, steal the user's information. The second type is commonly used in scams, where scammers attempt to pass a scam message through the content-based filtering of email providers to trick recipients to cheat their money. These two types of attacks both rely on the multiple similar visual representations of Latin characters in Unicode. Some researches on these aspects of homograph attack applications are described in more details below.

There was a study of homographs application on websites in 2006. Holgers et al. demonstrated that attackers might use homographs to register domain names, such as paypaI.com, with a capital 'I' instead of the lower case 'l', to trick users into revealing sensitive information [8]. Thao et al. also suggested that some attackers used homographs to register brand domains. They will use homographs in the Unicode to substitute one or two letters in a brand name to confuse consumers and lead them to fake websites [11]. Many prominent companies have been targeted in this way, such as Apple (apple.com), Lloyds Bank (lloydsbank.co.uk), and Adobe Systems Incorporated (adobe.com). Scammers can send emails with a link to a non-authoritative and fake PayPal site to lure victims to the website. Moreover, with the adoption of International Domain Names (IDNs) support by browsers and DNS registrars, a non-trivial number of IDN homographs were found as Cyrillic letters could be applied as a new kind of homograph of Latin letters. The Cyrillic letters are the closest to the glyph of Latin letters in Unicode.

This issue has drawn a lot of attention, but Holgers et al. had not found any data to quantify the extent of the Web homograph attacks at that time. They used a

combination of passive network tracing and active DNS detection to measure several aspects of the Web homographs, and they observed and tracked 828 Web clients for nine days, eventually reached the following four conclusions. First of all, many authoritative websites that users visit have multiple registered domain names that can easily confuse human eyes. Especially some popular sites are more likely to have these registered confusing domain names. Second, although they have found examples of multiple character substitutions (up to five), the registered obfuscated domain often consists of single-character replacement from its authoritative domain. Obfuscated domains mostly tend to have Latin letter homographs, but there were a significant number of IDN homographs, accounting for about 12-15% of the total. Third, websites associated with non-authoritative obfuscation domains often have relatively benign intentions, such as displaying advertisements to users. However, a few sites link users to competitors' sites and deceive the content of authoritative sites. Fourth, they concluded that homograph attacks were rare and not severe at the moment, but this seemed to be an attractive way for future attackers to trick users.

In the case of homograph attack applied in spam, as described in the introduction, spam accounts for 75 to 85 percent of all email in the past few years, and spammers started to use obfuscation to bypass email content-based filtering in recent years. For scammers, Unicode is a convenient tool, and an email box is a suitable approach to fraud money. It allows scammers to generate large amounts of different scam messages obfuscated by different homomorphic characters but look similar; since emails can be sent by anyone who knows how to use a computer. To defend against these Unicode obfuscated emails and domain names, many researchers have come up with several methods.

## 2.3 Counters to confusable characters

Since homograph attacks were first identified in the literature, some mitigation methods have been studied by researchers to defend against them. However, these counters still have various disadvantages. The simplest one would be to prohibit homographs directly, which would prevent confusion between the Cyrillic, Greek, and Latin "o" but would also prohibit some innocent websites that use Cyrillic or Greek character, such as, "CNNenEspañol.com" [12]. The second method is to identify text that resides in non-browser native scripts and convert the spoofed text to a distinct different text with Punycode, such as convert "www.homograph.com" to "www.xn--hmgraph-8ofb.xn--cm-jbc" [13]. This approach still has the disadvantage of making users vulnerable to Greek-Latin homograph attacks when they set their browser to read Greek and Latin text. Another approach is to color different scripts in a distinct color, mostly for browser-based add-ins [14], but there are two downsides to this approach. The first is the same as the first approach, and it makes some sites containing Greek characters appear suspicious to users. The second is that the tool only warns users but does not provide an action to it. Moreover, this method does not work for color-blind users.

In 2012 Helfrich et al. proposed a more efficient method called dual canonicalization. They defined homographs as "a set of encodings that, when displayed to the user, are perceived as being the same." [9] They set several components to this definition:

- Encoding: a representation of some presentation such as Unicode for text.
- Rendition: a way to make a given encoding appear to the observer.
- Rendering function: a way to render or display a given encoding into the rendition format.
- Observer function: the probability a given observer will consider two renditions same.

With these components, they can define homographs, and the method for detecting whether two encodings are homographs is dual canonicalization. Thus, homographs can be identified as long as rendering function and observer function is derived from a large amount of data analysis, regardless of the effects of encoding and presentation formats. From the results of their experiment, the dual canonicalization could detect a wide variety of homograph attacks, including Unicode homographs in a URL phishing attack, near Unicode homographs used in body-text phishing attacks, and HTML homographs in a spam filter-avoidance attack. Therefore, email providers can use this counter to detect and prevent homograph attacks.

Thao et al. also proposed the first classification model that uses the feasibility and novelty of machine learning to detect homographs on brand domains registered by attackers in 2019 [11]. They used state-of-the-art measures of visual similarity, the Structural Similarity Index (SSIM) for each character, combined with some information provided by Whois (a tool for detecting phishing). Their implementation results show that their method can bring up to 95.90% of accuracy.

Specific to the area of defending against spams with homograph attacks, Liu et al. developed a prototype tool that can be used with SpamAssassin, a spam filter, to catch obfuscated messages and reverse their obfuscation process [10]. They used the UC-Simlist to detect "Unicode attacks" and chose SpamAssassin to integrate their scripts to improve its function. Their experiment included more than 100 spam emails, and the result showed that after being de-obfuscated by their scripts, the spam could be caught more easily. However, they also mentioned shortcomings of their tool: taking 5 seconds to check each email, challenging to measure its accuracy, and spammers regularly change techniques.

## 2.4 The original study

In 2017, to demonstrate a weakness of the current content-based filtering system on defending against homograph attacks, Dhiman et al. [4] built an obfuscator modifying scam messages to a homograph version. They showed the success of this potential attack against Hotmail, Gmail, and Yahoo mail.

Their experiment was divided into two steps. The first step was to select confusable characters for each Latin letter from Unicode to build an obfuscator. The next was to target spam filters of different email providers -- Hotmail, Gmail, and Yahoo mail, to test whether they were capable of blocking obfuscated spam.

In the first step, Dhiman et al. found a table of confusable characters in Unicode on the Unicode official website. They manually identified a list of visually similar confusable characters. In this list, each uppercase and a lowercase letter had dozens of confusable characters. The characters were inspected for visual similarity in three stages: when viewed by themselves, viewed within the context of words, and viewed within the text.

| | | | | | | |
|---|---|---|---|---|---|---|
| A | 𝔄 | 𝔸 | 𝕬 | 𝒜 | Ａ | 𝓐 |
| 𝖠 | 𝘼 | 𝐴 | 𝐴 | ᴀ | 𝑨 | 𝗔 |
| 𝙰 | 𝗔 | 𝖠 | 𝗔 | 𝗔 | 𝙰 | 𝐴 |
| 𝖠 | 𝘼 | 𝖠 | 𝖠 | 𝖠 | 𝘼 | |

**Figure 1. Confusable characters for the Latin upper-case A.**

In the first stage of selecting confusable characters, Dhiman et al. first removed characters that are distinct from the Latin letter. For example, as shown in figure 1, the second and the fourth characters in the first line, etc. are separate from Upper A, those are the characters that can be taken directly in this step. In the second part, they put confusable characters into words, replacing one of the letters with a

confusable character in Latin words. They observed whether there was a significant difference in subtle spacing or stroke that cannot be detected when confusable characters are displayed separately. For example, the Latin letter "i" and the Mathematical Sans-serif Bold Small letter "𝗶" in "affirm" and "aff𝗶rm" make the two words distinct. The third stage in the process is to examine confusable characters in the text. As we know, each mailbox reader and device's different font would make the characters look slightly different. At this point, the impact on confusable characters is even more significant. They may look very different from the Latin letters in some fonts and may not also be displayed if a device or software does not support that character. Researchers sent sentences that all had been replaced with confusable characters to friends or family and asked them if anything was off. In this way, more confusable characters were discarded.

Following these comparisons, 67 high-fidelity confusable characters were left, and were visually indistinguishable to their Latin counterparts, as Figure 2 shows below. They built an obfuscator with these confusable characters. The obfuscator could take a Latin text of any length as input, randomly decided whether it should replace the character with a glyph chosen from a list of manually identified homographs and output a converted message. For example, following the confusable filtering process described, two high-fidelity homographs for the Latin uppercase "P" were identified. Thus, each time a "P" is detected by obfuscator, there are three candidates: the original character, plus the two homographs.

| A | B | C | E | F | G | H | K | L | M | N | O | P | R | S | T | U | V | X | Z |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | E | F | G | H | K | L | M | N | O | P | R | S | T | U | V | X | Z |
| | | | | | | | | | | | O | P | | | T | | | X | |
| | | | | | | | | | | | | | | | T | | | X | |

| a | c | e | i | j | o | p | s | v | w | x | z |
|---|---|---|---|---|---|---|---|---|---|---|---|
| a | c | e | i | j | o | p | s | v | w | x | z |
| | c | | | j | | | | v | | | |
| | c | | | | | | | | | | |

**Figure 2. Dhiman et al.'s final list of high-fidelity confusable characters for the upper- and lower-case letters in the English alphabet [4]. On the top row of each column is the Latin character, while the rows below are its confusable Unicode characters.**

In the second step, Dhiman et al. obfuscated 90 scam messages with their obfuscator. They sent them to three email inboxes: Hotmail, Gmail, and Yahoo Mail, and their aim is to test the effectiveness of email filters at detecting obfuscated scam messages. They choose three email providers to test because these providers dominate the email market and are more likely providers to have deployed state of the art spam filtering systems than other service providers. This choice allowed the researchers to measure the degree to which obfuscated scams were filtered under "optimal conditions". Researchers sent the scam messages from six newly registered accounts (two from each email provider) to three recipients (one from each email provider). All six senders sent each obfuscated scam message to the same three inboxes over the five days. They simulated the low-and-slow behavior of scammers to avoid triggering rate-limits enforced by email providers.

In their result, 96% of 90 obfuscated scam messages were delivered successfully to three inboxes. However, the success rate of receiving the same message varied from sender to sender, as shown in Table 1. Yahoo sender accounts had a much lower success rate to the Gmail recipient than to Hotmail or Yahoo recipients. This

outcome showed that the obfuscated scam messages were filtered on the recipient side. Overall, the block rate of obfuscated scam messages was 7.8% for the Gmail recipient, 0.4% for the Hotmail recipient, and 0.2% for the Yahoo recipient. Dhiman et al. concluded that there was little protection against scam emails with homograph attacks. They also discussed a countermeasure named "reverse mapping" suggested by Liu et al. [10]. It can identify confusable characters and de-obfuscate them, and Dhiman also suggested some improvements to it. While English-language spam and hoax messages dominate email on the Internet, there are other languages (and character sets) on the market. Thus, the number of character sets used for messages can be scanned directly to identify obfuscated scam messages. For most legitimate messages, there is only one character set. For a small number of valid messages that contain multiple character sets, the characters are mostly not confusable characters. In these two situations, checking can be stopped early to save time. Besides, quantifying the degree of likely obfuscation in a message can be simplified as identifying and counting the homographs to detect spam and scams.

**Table 1**
**The portion of 90 obfuscated messages sent from each sender account to each recipient fails to reach the recipient's inbox [4].**

| recipient→ | Gmail | Hotmail | Yahoo |
|---|---|---|---|
| Gmail(sender1) | 1/90 | 0/89 | 0/90 |
| Gmail(sender2) | 1/90 | 1/89 | 2/90 |
| Hotmail(sender3) | 1/90 | 0/89 | 0/90 |
| Hotmail(sender4) | 0/90 | 0/89 | 0/90 |
| Yahoo(sender5) | 20/90 | 0/89 | 0/90 |
| Yahoo(sender6) | 19/90 | 0/89 | 0/90 |

Their research is worrying, as 96% of these scam emails were delivered successfully using this simple obfuscator, which most attackers can easily use and transmit. The tested email providers were world-renowned and most likely to have the highest filtering technology. It also showed that the filters applied by

most email providers were not capable of defending against homograph attacks in 2017. It has been three years since these results were published, and email providers should have made improvements on content-based filters to block scam emails obfuscated with homographs. Therefore, I aim to replicate this paper with an extended and updated methodology to examine whether the major email service providers have updated their users' protection. Moreover, as for the counter "reverse mapping" they discussed, I also built a simple de-obfuscator using the experimental character sets I get from my experiment to verify its feasibility.

# 3. Methodology

This experiment has demonstrated how to use confusable characters in Unicode to obfuscate scam messages while preserving the readability of the messages, and investigated the effectiveness of current email filters on detecting and blocking these obfuscated scam messages. To clarify the contrast between scam messages before and after being obfuscated, and to ensure email providers only prevent malicious messages, I also expanded and updated some research steps based on the original research.

The first three steps of the experimental process I followed were necessarily the same as those of the original study, namely identifying confusable characters manually, building an obfuscator, and sending emails. More specifically, I manually classified the most confusable characters in Unicode, used them to build an obfuscator, and then sent the obfuscated scam messages. I mainly extended and updated the third step of the process by adding two types of emails to send. They were unobfuscated scam emails and obfuscated regular emails, to ensure that all original emails could be blocked by email providers and analyze the blocking principle of email filters. Moreover, I additionally made a simple de-obfuscator as the fourth step to demonstrate email providers' feasibility to reverse-transforming and identifying scams in filtering.

## 3.1 Select confusable character manually

This step is similar to the original study. I found confusable characters for upper- and lowercase Latin characters on the official Unicode website, and selected the most visually identical and confusable characters based on their glyphs to obtain homographs.

There are 1,351 confusable characters of upper- and lowercase Latin letters in the data on the official website. There are 26 cases for the Latin character "A", the same as the original study displayed. Thus, the data set is the same as the data set in the original research.

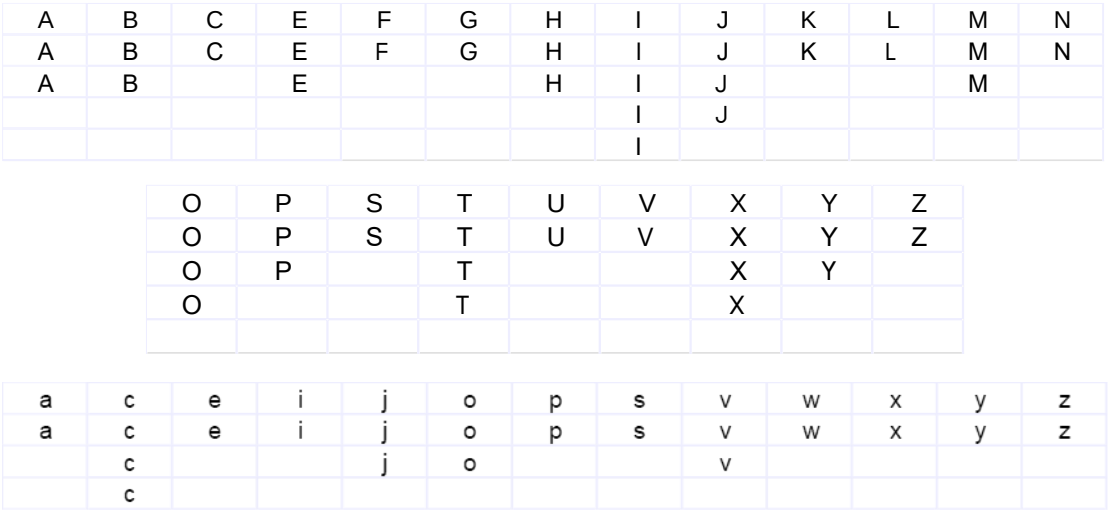


**Figure 3. The final list of high-similarity confusable characters for the upper- and lower-case letters in the English alphabet. On the top row of each column is the Latin character, while the rows below are its confusable Unicode characters.**

The experiment used multiple steps to select homographs. Firstly, I eliminated characters that are visually distinct from Latin characters. It is clear from figure 1, shown earlier, that many of the confusable characters are printed significantly differently, so this step removed most of the characters, leaving only about two hundred. Secondly, I put the remaining characters into words. Some of the characters are similar to Latin characters on their own. Still, the spacing between them and the other letters are significantly different when added in words, and the subtle differences in letter size are more pronounced in a word. For example, ‘i’ looks similar to Latin letter ‘i’, but when placed into the word “affirm”, the different spacing is apparent; ‘d’ looks identical to Latin letter ‘d’ while in the word ‘divide’, the difference is distinct. Finally, I put the selected confusable characters into the text. Different devices display characters in different fonts, and email applications on some devices may even not support some of the confusable

characters. Thus, figures may show differences. I also showed the text with all characters replaced with their corresponding confusable characters to different people and asked them to find out if something was wrong.

A list of 58 relatively high-similarity confusable characters can be obtained after these steps, as Figure 3 shows. They are also the homographs required for building an obfuscator. On the top row of each column is the Latin character, while the rows below are its confusable Unicode characters. The result of selected confusable characters is a little different from the original study. Some Latin letters have more corresponding confusable characters than before. In contrast, some Latin letters have less, or even no confusables left. However, considering that email applications might have been updated over the three years, and the fonts used by default in the webmail applications should be changed. Therefore, an individual result of this stage is allowed to vary, which do not have much impact on the experiment.

## 3.2 Build obfuscator

With the 58 homographs shown in Figure 3, an obfuscator can be built to modify scam messages. I chose Java to create it for Java already supports Unicode and could quickly establish the required obfuscator tool. When a message is being obfuscated, for each letter of the original message, the obfuscator randomly decides whether it should replace the character with a glyph chosen from a list of manually identified homographs. For instance, the Latin letter ‘a’ has one homograph, therefore when an ‘a’ is detected, the obfuscator has two choices, while the Latin letter ‘O’ has three homographs. Thus, there are four choices for the obfuscator. The obfuscated messages generally have an average of 69% of letters replaced. There is little difference in appearance between them and those made up of Latin letters as the next two messages show. The first paragraph below

is the original message, and the second is the obfuscated message.

“From the entire members of the Federal Ministry Of Works and Housing, on behalf of the Federal Republic of Nigeria Government, Under the auspices of the civilian Head of State, President umaru musa yar'adua and the Governor of Central Bank Of Nigeria, Prof Charles Soludo held a meeting last week concerning payment Of foreign contractors and some inheritance funds.”

“From the entire members of the Federal Ministry Of Works and Housing, on behalf of the Federal Republic of Nigeria Government, Under the auspices of the civilian Head of State, President umaru musa yar'adua and the Governor of Central Bank Of Nigeria, Prof Charles Soludo held a meeting last week concerning payment Of foreign contractors and some inheritance funds.”

## 3.3 Target scam email filters

Commercial spam filters have multiple methods to identify unwanted emails, from checking the email content to senders' reputation, or links, attachments in emails, etc. This experiment aims to measure the ability of filters of different email providers when faced with messages that are obfuscated with confusable characters.

This step included four email providers, which are Hotmail, Gmail, Yahoo Mail, and ProtonMail. The first three are from the original study, and they all dominate the email market, with a high probability of having the latest filtering technology. I included ProtonMail as an extension, which positions itself as a privacy-conscious email and is well received by those concerned about its privacy. Compared with the other three mailboxes, this extension can test whether it has unique advantages. It can also provide some references for security-conscious users who prefer to use

ProtonMail. There were two sending boxes and one inbox from each email provider during the experiment, which means each tested email was sent from a total of eight mailboxes to every four inboxes.

Three types of emails were sent in this experiment. The first was unobfuscated scam emails. Since the scam emails obtained were from the scam email extractor in my supervisor's previous research, it should be tested if the filter can still detect them. In the experiment, I want to ensure that at least two email providers could block all selected scam emails. The second category was scam emails that have been obfuscated. This is the subject of the experiment. The purpose is to measure whether the current existing email filters could successfully defend against scam messages obfuscated with confusable characters. The number of samples tested for scam emails was 90. The third sending type was obfuscated regular messages, taken at random from regular emails on the Internet. This step aims at measuring whether the email filters only block all messages containing homographs or accurately prevent the obfuscated scam messages. If they only take the former approach without additional filtering conditions, it will be inconvenient for non-Latin character users. The original study only contained sending the second type of email. I added the other two types to make a bright contrast between scam messages before and after being obfuscated to ensure email providers only block malicious messages. Below, I described an ethics application this step requires and outline the process for sending each of the three types of email.

### 3.3.1 Ethics application

To send these emails from 8 sender accounts to 4 recipient accounts in batches, some volunteers' help was required. I invited some participants and asked them to send and receive emails. They were required to inform the author whether the emails were successfully received or not. Participants were informed of the risks

of the experiment in advance – the major risk was that their email accounts might be blocked due to sending scam emails, so it was better to create some new email accounts. The experiment would not record their personal information. This study design was reviewed and approved by the Faculty of Engineering Research Ethics Committee.

### 3.3.2 Send unobfuscated scam emails

Some unobfuscated scam emails were selected to be sent from one of the eight sending email accounts to four email inboxes (one from each email provider) within five days. These messages were selected from the scam email data set from the previous project of my supervisor. Messages were manually filtered to ensure that the possibility of regular emails being filtered by the machine was excluded. This step aims to provide 90 scam emails that already have some features to be detected and blocked. Therefore, one sending account is enough. The original plan was to find 90 scam messages, which would be blocked by all four mailbox providers. Nevertheless, during the experiment, it turned out to be impractical. Almost all the emails would escape at least one email filter. Therefore, the plan was modified to find 90 scam emails blocked by at least two mailbox providers. Any emails that did not meet this condition would not be selected for the experiment, and new scam messages from the same data set would be added to the test.

### 3.3.3 Send obfuscated scam messages

This part is the core of the experiment. The ability of an email filter to deal with emails with easily confused characters is mainly reflected in the data obtained in this section. The 90 selected scam messages were obfuscated by the obfuscator and sent from eight sending email accounts (two from each email provider) to four email inboxes (one from each email provider) within five days. I recorded the

number of successfully received emails for each inbox from each sender to analyze the ability to filter different email providers to resist homograph attacks.

### 3.3.4 Send obfuscated normal messages

Thirty normal messages were chosen and obfuscated to be sent from one sending email account to four email inboxes (one from each email provider) over five days. The total number of sending emails was reduced because 30 obfuscated normal emails are enough to check whether an email filter only blocks malicious messages or all messages containing homographs.

Every type of message was sent over five days, which simulated the low-and-slow behavior of scammers to avoid triggering rate-limits imposed by the email providers. The reception results of obfuscated scam messages would be recorded to analyze the filters' ability to resist homograph attacks. In comparison, the reception result of obfuscated normal messages would be used to analyze the filtering principle.

## 3.4 Build de-obfuscator

I made a simple de-obfuscator based on the obfuscator I built. It can prove the feasibility of filters in detecting and blocking emails with confusable characters.

The principle of making an obfuscator is precisely the opposite of making an obfuscator. It detected the confusable characters in messages and then converted them into their corresponding Latin letters. For example, supposing the ‘a’ detected by de-obfuscator in “password” is a homograph of the Latin letter ‘a’, it will be de-obfuscated, and the filters can recognize the word, and then check it as a standard Latin fraudulent email. For simplicity, the confusable character data set was still

the one used to make obfuscator. In practice, it should be necessary to expand the scope of homographs for detection.

The de-obfuscator was tested, and the result showed that it could completely alter the previously obfuscated message back to its original form. Although the principle and the selected data set were relatively simple in this experiment, it still turned out that email providers could apply the de-obfuscator on content-based filtering. It is feasible to detect and filter messages that use the homograph attack if with a complete confusable character data set.

# 4. Result

This section focuses on the receiving result of three types of email and presents a horizontal comparison of each mailbox provider.

## 4.1 Email receiving result

There were three types of email sent in this experiment. The first was unobfuscated scam email, which aimed to select emails that have the feature of being blocked. I mainly presented and analyzed the second and third types of data: obfuscated scam email and obfuscated normal emails, to research individual email provider's filter effectivity when facing homograph attacks.

### 4.1.1 Unobfuscated scam messages

I sent 90 selected unobfuscated scam emails from one of the eight sending email accounts to four email inboxes (one from each email provider) over five days. One sending account is sufficient because different sending accounts should not impact the existing email filtering technology. I should determine whether most filters usually block these messages using Latin characters. At first, I tried to pick out the messages that were blocked by all four. However, during the experiment, many of the emails reached at least one inbox, so the selection criteria were changed to finding 90 scam messages that could be blocked by at least two mailbox providers.

More than a dozen fraudulent emails were excluded because they did not meet the requirements. I selected new ones from the original scam email data set to replace them and selected the final ninety for the experiment. Most of their content is impersonating celebrities or pretending to have an inheritance that requires someone to inherit, then deceiving recipients to transfer money to their bank cards.

The final failure rates for the unobfuscated scam emails are shown in Table 2. It can be seen that all four of these email filters did work for identifying the common types of fraudulent emails. Outlook almost blocked them all, followed by Gmail and Yahoo Mail, which blocked about two thirds, while ProtonMail blocked only a third. Interestingly, this is quite different from the result of the subsequent obfuscated fraudulent email. This shows that each mailbox filter has a different focus, and I have made a comparison in the next section 4.1.2.

**Table 2**
**The portion of 90 unobfuscated scam messages sent from each sender account to each recipient which failed to reach the recipient's inbox**

| | Yahoo | Proton | Outlook | Gmail |
|---|---|---|---|---|
| Yahoo(sender1) | 20/27 | 13/27 | 27/27 | 26/27 |
| Gmail(sender2) | 9/18 | 15/18 | 18/18 | 10/18 |
| Outlook(sender3) | 26/27 | 8/27 | 27/27 | 18/27 |
| ProtonMail(sender4) | 4/18 | 5/18 | 18/18 | 17/18 |
| Fail/Total | 59/90 | 31/90 | 89/90 | 61/90 |

### 4.1.2 Obfuscated scam message

The sending and receiving status of the 90 messages selected in the previous step is shown in Table 3. The emails were sent from eight sending email accounts (two from each email provider) to four email inboxes (one from each email provider) over five days. All obfuscated scam emails were successfully delivered to all four recipients, and 48.33% of emails made it into the recipients' inbox, which is much lower than the 96% in the original study.

As can be seen from Table 3, from the recipient's perspective, Gmail can protect users from homograph attacks in most cases, with the lowest rate of receiving success, followed by ProtonMail. At the same time, ProtonMail seems unable to

defend against scam emails sent by other ProtonMail accounts. Yahoo and Outlook look defenseless compared to Gmail and ProtonMail. From the sender side, Outlook seems to have a high success rate in blocking obfuscated scams at the outbound time, while other email providers seem to have no such measures. However, in the same way, Outlook does not appear to defend against scam emails sent to other Outlook accounts.

**Table 3**
**The portion of 90 obfuscated scam messages sent from each sender account to each recipient which failed to reach the recipient's inbox**

| | Proton | Gmail | Yahoo | Outlook |
|---|---|---|---|---|
| Proton(sender1) | 0/90 | 90/90 | 0/90 | 9/90 |
| Proton(sender2) | 0/90 | 90/90 | 0/90 | 12/90 |
| Gmail(sender1) | 85/90 | 90/90 | 9/90 | 6/90 |
| Gmail(sender2) | 86/90 | 90/90 | 0/90 | 34/90 |
| Yahoo(sender1) | 86/90 | 90/90 | 0/90 | 8/90 |
| Yahoo(sender2) | 86/90 | 80/90 | 0/90 | 5/90 |
| Outlook(sender1) | 83/90 | 86/90 | 85/90 | 5/90 |
| Outlook(sender2) | 86/90 | 89/90 | 85/90 | 22/90 |

This result is different from that in 4.1.1 unobfuscated scam messages. Outlook, the top performer in the previous section, and Yahoo Mail, which was still doing well in 4.1.1, performed poorly here, with little resistance to homograph attacks. ProtonMail, which blocked the fewest unobfuscated scam emails, did an excellent job in this section. Only Gmail did better in both experiments.

### 4.1.3 Obfuscated normal messages

From the previous step, we can see Outlook has a high success rate in filtering obfuscated scams at the outbound time on the sender side. In contrast, ProtonMail

and Gmail have a high success rate in filtering obfuscated scams at the recipient side's inbound time. However, Outlook and ProtonMail do not filter emails from or to other email accounts of the same provider, and Yahoo Mail has not put in place defenses against homograph attacks. Thus, this step focuses on what happens when these email providers are confronted with obfuscated regular emails. Whether they only block scam emails or block all emails with confusable characters. The receiving result is shown in Table 4.

**Table 4**

**The portion of 30 obfuscated normal messages sent from each sender account to each recipient fails to reach the recipient's inbox.**

| | Proton | Gmail | Yahoo | Outlook |
|---|---|---|---|---|
| Proton(sender1) | 0/30 | 29/30 | 0/30 | 1/30 |
| Proton(sender2) | 0/30 | 30/30 | 0/30 | 1/30 |
| Gmail(sender1) | 29/30 | 30/30 | 0/30 | 0/30 |
| Gmail(sender2) | 30/30 | 30/30 | 0/30 | 28/30 |
| Yahoo(sender1) | 29/30 | 30/30 | 0/30 | 0/30 |
| Yahoo(sender2) | 29/30 | 30/30 | 0/30 | 0/30 |
| Outlook(sender1) | 24/30 | 30/30 | 25/30 | 0/30 |
| Outlook(sender2) | 30/30 | 30/30 | 27/30 | 4/30 |

All obfuscated normal messages were successfully delivered to all four recipients, and 48.33% of emails made it into the recipients' inbox, almost the same as that of obfuscated scam messages. After analyzing each email provider individually, it was found that the results are also similar to those of the previous step. Both ProtonMail and Gmail blocked the most emails only on the recipient side, and Outlook still blocked emails only on the sender side, but neither ProtonMail nor Outlook filtered emails sent from their email accounts. Other mailboxes did not filter well either, which was not surprising as they did not filter obfuscated scam emails at the previous step. Gmail (sender2) -> Outlook is an exception, perhaps because the Outlook inbox trusted the domain name of Gmail (sender2) by accident and it no

longer blocked messages sent from that Gmail account, so this data can be treated as an accidental error. As a result, all email providers either did not take defensive measures against homographs or blocked all emails with confusable characters.

## 4.2 Comparison between email providers

According to the previous section's data, email providers respond to homograph attacks with different measures and levels. Some providers filtered messages on the recipient side at the inbound time, some on the sender side at the outbound time, and some seemed to have no measures.

Gmail was the best performing on the recipient side among the four email providers, with the highest obfuscated scam emails blocking rate of 97.92%. It hardly delivered obfuscated messages from various mailboxes, and it did not relax its guard against messages from other Gmail accounts. However, on the sender's side, there was no visible blocking performance. Gmail also blocked almost all obfuscated regular emails, as if its blocking mechanism is based on identifying suspicious email accounts or blocking all messages containing obfuscated characters.

ProtonMail also did a good job on the recipient's side, with a blocking rate of 71.11% of obfuscated scam emails, which almost prevented all obfuscated emails not sent from ProtonMail. The reason is that ProtonMail provides end-to-end encryption for its accounts. Therefore, even the ProtonMail provider cannot read the content. The end-to-end encryption was discussed more below in Chapter 5. Similarly, it blocked most obfuscated normal messages and showed no sign of resistance to homograph attacks on the sender's side.

Outlook can be seen as “strict” when doing outbound homograph scam filtering, blocking 75.14% of obfuscated scam emails on the sender side. Almost every obfuscated email was blocked except for the emails sent to other Outlook email accounts. It is like being responsible for other mailboxes, but it implies a loophole in the receiving end. Besides, many obfuscated ordinary mails were also blocked, which proved that it only checked homographs to identify untrusted emails.

Yahoo Mail performed poorly on both the receiving and sending ends. Yahoo Mail seemed to put no defense against homograph attacks. Even for the most blocked cases, it is because Outlook has a more effective outbound filtering.

Overall, Gmail mostly protects its users against homograph attacks. ProtonMail is good, but it does not appear to defend against homograph scam emails sent from other ProtonMail accounts. Moreover, they all block too many ordinary emails and do not consider users with non-Latin characters. Interestingly, Outlook appears to have outbound filtering while its inbox receives homograph scam emails at a high rate. On the other hand, Yahoo Mail does not appear to defend against any homograph scam emails.

# 5. Discussion

## 5.1 Result analysis

Based on the data obtained from the result part, I analyzed the basic situation of current email filtering against homograph attacks. As the overall blocking rate is 51.67%, better than the original study, it is reasonable to infer that homograph checking has been applied to email filtering by some email providers. Among the four, Gmail and ProtonMail performed better, blocking almost all messages with confusable characters at an inbound time on the recipient side. Some special cases deserve analysis and discussion.

As mentioned above, ProtonMail did not filter the scam messages obfuscated with homographs from other ProtonMail email accounts, but it almost completely blocked obfuscated emails from other email accounts. It is because ProtonMail uses end-to-end encryption. As shown on its official website, “We use end-to-end encryption and zero access encryption to secure emails. This means even we cannot decrypt and read your emails. As a result, your encrypted emails cannot be shared with third parties.” [16] This sentence could explain why ProtonMail does not filter messages from other ProtonMail boxes since homograph filtering is based on content checking. If it is end-to-end encryption from one ProtonMail end to another ProtonMail end, then even the email provider cannot see the content of the message, not to mention filtering. This feature is advantageous for protecting user privacy, but it also suggests a bug in defending against homograph attacks.

Outlook’s results are also impressive. It appears to be a “good neighbor” to other email inboxes, as homograph scam emails sent from Outlook accounts are not arriving, even at webmail providers like Yahoo that do not seem to protect against this from other domains. However, it would be odd not to set it up at the inbound

time if it has applied homographs filtering at the outbound time. This outcome could be attribute to Outlook's design, or it could be that other mailboxes prefer to consider Outlook's domain name as suspicious. This assumption is based on the fact that a Gmail was trusted by the Outlook recipient account for some unknown reasons during the experiment process, resulting in the subsequent emails being delivered smoothly. However, since this was a fortuitous event, and the number of receiving emails increased as the experiment progressed only between a Gmail Account and an Outlook inbox, the odds of that happening are slim.

As for the result obfuscated regular emails, to a large extent, it has shown that email providers that have defending action against homograph attacks simply detect the content of confusable characters in email content and screen out those that do not meet the condition. They do not consider users who use Latin homograph as their first language, or who study homograph academically and communicate by email. In conclusion, there is room for improvement in the availability and effectiveness of email filtering against homograph attacks.

### 5.1.1 Comparison with the original study's findings

Compared with the case Dhiman et al. displayed, things are overall improved. Some email providers such as Gmail and ProtonMail are catching homograph attacks. However, Yahoo Mail and Outlook appear not to be able to resist homograph attacks, and even for Gmail and ProtonMail, there are still some bugs and weaknesses that need to be improved.

For the Gmail recipient, the block rate of obfuscated scam messages was 7.8% three years ago. It is 97.9% in this experiment, which is a great improvement from the past. Therefore, Gmail is the most improved and best performing email. For the Outlook recipient, the block rate was 0.2%, which was the worst three years

ago. It is 14.0% in this experiment. It blocked about 10 messages from each sender, making it hard to see the obvious response to homograph attacks. However, the sender of Outlook's block rate of obfuscated scam messages is 75.1% at the outbound time, in contrast to the almost no blocking in the original study. The strange thing is not knowing why this technology does not work for recipients. For the Yahoo Mail recipient, the block rate was 0.4% in the original study. It is now 24.9%, and even this figure is due to Outlook's tight control at the outbound time, so there has been essentially no improvement.

## 5.2 Advice

### 5.2.1 Advice for email providers

Compared to the original study, some email providers, such as Gmail and ProtonMail, have improved their ability to filter content against the homograph attacks. However, as a widely used and trusted product, there is still shortcomings and needs improvement.

One of ProtonMail's better points is its focus on privacy, but it also allows scammers to send out fraudulent emails. Messages sent from ProtonMail to ProtonMail are end-end encrypted, so even the email provider cannot see the content and, therefore, cannot detect deceptive content. Hence, it is possible to block scam emails from other mailboxes but miss out on ProtonMail. Then scammers may take advantage of this loophole and apply a ProtonMail account to send fraudulent emails to ProtonMail users. Sometimes people think it is a good idea for all emails to be end-to-end encrypted. [17] However, this experiment's findings suggested that it will probably make themselves vulnerable to attack because there is no third-party guardianship. Therefore, if people do concern about their privacy, they should practice the ability of identification. As email

providers cannot read the content of the messages, they have to find solutions elsewhere, such as limiting the number of times that users can send emails and blocking accounts that have been reported by users multiple times.

Another thing that needs to be improved is the experience of users who write in non-Latin scripts should be considered. In the case of sending obfuscated normal messages, most email providers that have defending actions to homograph attacks appear to simply detect scam emails by the percentage of confusable characters in the email content. This method is quite easy to hit ordinary mail accidentally. Every user uses confusable characters for different reasons. Therefore, it is inappropriate to block emails that all letters are confusable characters or set a percentage range. Consequently, it is recommended to use de-obfuscator to prevent emails that use confusable characters.

De-obfuscator can convert obfuscated characters directly to their corresponding Latin letters, as long as there is a corresponding Latin confusable character list. When a filter encounters a message that contains homographs, it does not have to analyze how much homographs occupy or how homographs occupy a message. The obfuscated message can be converted to Latin words and filtered directly by the original method. This way will significantly reduce the number of messages that are blocked by mistake and provide convenience for users. For Yahoo Mail and Outlook, they need to do more. The first step is to add some counters to homograph attacks in filters, and then consider the improvement advices mentioned above.

### 5.2.2 Advice for users

There are a few things that users can do. First of all, users should choose an email account based on their own needs. Private users can choose to use ProtonMail;

those who do not want to be disturbed by spam can use Gmail; users whose first language that happens to contain many Latin characters' homographs, can avoid using Outlook and notice recipients look emails in junk mailboxes. Besides, supposing the sending regular emails are often blocked, users can consider whether or not relate to homographs and seek for mailbox customer service's help to improve the efficiency of use.

Also, since some mailboxes do not take corresponding measures against the homograph attacks, users of those mailboxes should pay more attention to the recognition and identification of fraudulent emails. Here is a better way to identify confusable characters. Users can convert the contents of a scam message into an infrequently used font when they are unsure whether it is a fraudulent message. Most homographs look utterly different from the Latin alphabet in less commonly used fonts, leading to the different sizes of letters in a sentence, making it very easy to spot the fraud information.

Taking the previously showed obfuscated messages as an example, the first paragraph below is the obfuscated messages in Times New Roman, and the second is the obfuscated messages in CASTELLAR, a less commonly used font chosen randomly.

"From the entire members of the Federal Ministry Of Works and Housing, on behalf of the Federal Republic of Nigeria Government, Under the auspices of the civilian Head of State, President umaru musa yar'adua and the Governor of Central Bank Of Nigeria, Prof Charles Soludo held a meeting last week concerning payment Of foreign contractors and some inheritance funds."

"FRoM THE ENTIRe MeMBeRS oF THE FEDeRaL MiNiSTRY OF WoRKs aND HoUsiNG, ON BEHaLF OF THe FEDeRAL RepUBLiC oF NiGERia

GovERNMENT, UNDeR THE aUspicEs oF THe ciViLIaN HeaD oF STATE, PReSIDENT UMaRU MUSA YaR'aDUA aND THE GoVERNoR oF CeNTRaL BANK OF NiGeRiA, PRoF CHaRLeS SOLUDo HeLD a MEETING LaST WEEK CONCERNiNG PaYMENT OF FOREiGN CoNTRacTORS aND soME iNHERITANce FUNDs."

It is obvious that the letters in the paragraph above are in different size. The Latin letters become the CASTELLAR font, while some letters look smaller are confusable characters, because they are not supported by this font. Therefore, it is not hard to identify whether a suspicious email is a scam email if users are careful enough to try it out a few times in different fonts.

# 6. Conclusion

This work replicated their experiment and expanded upon it. The author built an obfuscator that works on the same principle to show how Gmail, Yahoo Mail, Outlook, and ProtonMail resisted emails containing confusable characters after three years.

Using the same method as in the original study, the author selected 58 characters with high fidelity to the Latin letters from the confusable character table, and used them to make obfuscator to simulate the homograph attacks in email sending. Obfuscator was written in Java, and the entire program is shown in Appendix A. On this basis, the author also made de-obfuscator, which can change the obfuscated messages back to the Latin alphabet. This step has proved that email providers can use de-obfuscator to resist scam messages that use confusable characters if provided the appropriate table of confusable characters. The entire program of de-obfuscator is shown in Appendix B.

The author expanded on the original experiment with one more test email provider ProtonMail and added a type of email in the experiment -- obfuscated regular messages. The study concluded that Gmail performed best against the homograph attacks, blocking almost all obfuscated scam emails. The second was ProtonMail, which blocked virtually all fraudulent emails from mailboxes except ProtonMail because of its mailboxes' end-end encryption. The outlook appeared to have tight control over the sender's side, but there was no clear indication that it was applied over the receiver's side. Yahoo Mail did not appear to have taken any action against homograph attacks. In addition, Gmail and ProtonMail's blocking effect on obfuscated normal emails was the same as that for fraudulent emails. It has been turned out that they only rely on the percentage of confusable characters to identify homograph attacks. Overall, there is a lot of room for improvement.

The author has also made corresponding recommendations to users and mailbox providers regarding the homograph attack and the range of issues it raises. Mailbox providers should improve the method of blocking emails with confusable characters by using de-obfuscator to convert text into Latin letters that can be detected by the original filter, increasing the accuracy of obfuscated fraudulent messages. For users, if normal emails are often blocked, they should consider the use of homographs of the Latin character, and actively seek solutions from mailbox providers.

# Reference


[1] MAAWG. Messaging anti-abuse working group. Email metrics report. Third & fourth quarter 2006. Available at http://www.maawg.org/about/ MAAWG Metric 2006 3 4 report.pdf Accessed: 04.06.07, 2006.

[2] M. Siponen and C. Stucke, *Effective Anti-Spam Strategies in Companies: An International Study.* 2006, pp. 127c-127c.

[3] E. Moustakas, C. Ranganathan, and P. Duquenoy, "Combating spam through legislation: a comparative analysis of US and European approaches," in *Conference on Email & Anti-spam*, 2005.

[4] M. Dhiman, M. Jakobsson and T. Yen, "Breaking and fixing content-based filtering," *2017 APWG Symposium on Electronic Crime Research (eCrime)*, Scottsdale, AZ, 2017, pp. 52-56, doi: 10.1109/ECRIME.2017.7945054.

[5] C. Bigelow and K. Holmes, "The Design of a Unicode Font," vol. 6, 12/01 1997.

[6] A. Fu, X. Deng, L. Wenyin, and G. Little, *The methodology and an application to fight against Unicode attacks.* 2006, pp. 91-101.

[7] P. Daniels, "The Unicode Standard: Worldwide Character Encoding, Version 1.0," *Language*, vol. 69, p. 225, 03/01 1993.

[8] T. Holgers, D. Watson, and S. Gribble, *Cutting through the Confusion: A Measurement Study of Homograph Attacks.* 2006, pp. 261-266.

[9] J. Helfrich and R. Neff, *Dual canonicalization: An answer to the homograph attack.* 2012, pp. 1-10.

[10] C. Liu and S. Stamm, *Fighting unicode-obfuscated spam.* 2007, pp. 45-59.

[11] T. Phuong Thao, Y. Sawaya, Q. Nguyen, A. Yamada, K. Omote, and A. Kubota, "Hunting Brand Domain Forgery: A Scalable Classification for Homograph Attack," 2019, pp. 3-18.

[12] E. Gabrilovich and A. Gontmakher, "The Homograph Attack," *Commun. ACM*, vol. 45, p. 128, 02/01 2002.

[13] V. Gupta, "International domain names in IE7," 19 December 2005. [Online].

Available: http://blogs.msdn.com/b/ie/archive/2005/12/19/505564.aspx. [Accessed 26 April 2012].

[14] V. Krammer, "Phishing defense against IDN address spoofing attacks," in *Proceedings of the 2006 International Conference on Privacy, Security*, 2006, p. 32.

[15] M. R. J. D. Freiermuth and Communication, "Text, lies and electronic bait: An analysis of email fraud and the decisions of the unsuspecting," vol. 5, no. 2, pp. 123-145, 2011.

[16] ProtonMail. 2020. *Secure Email: Protonmail Is Free Encrypted Email.* [online] Available at: <https://protonmail.com/> [Accessed 3 September 2020].

[17] M. Ryan, "Enhanced Certificate Transparency and End-to-End Encrypted Mail." *NDSS*. 2014.

# Appendix A

Obfuscator:

```
import java.io.*;
import java.nio.charset.Charset;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.Random;
import java.util.Scanner;
public class obfuscator {
   public static void main(String args[]) throws IOException {
      ArrayList<Integer[]> confusablesDataList = new ArrayList<>();
      String lineTxt = null;
      Scanner scanner = new Scanner(new FileInputStream("src/confusablesFinal.txt"));
      while (scanner.hasNext()) {
         lineTxt = scanner.nextLine();
         int i=0;
         String[] datas = lineTxt.split(",");
         int n=datas.length;
         Integer[] confusablesData = new Integer[n];
         for (String data : datas) {
            data=data.substring(2);
            confusablesData[i]=Integer.parseInt(data,16);
            i++;
         }
         confusablesDataList.add(confusablesData);
      }//read confusables
      BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
      int cnt = 0;
      while(true) {
         String message = in.readLine();
         for (int i = 0; i < message.length(); i++) {
            for (int j = 0; j < 52; j++) {
               if (message.charAt(i) == confusablesDataList.get(j)[0]) {
                  Random ran = new Random();
                  if (confusablesDataList.get(j).length > 1) {
                     int num = ran.nextInt(confusablesDataList.get(j).length);
                     message = unicodeToUTF8(confusablesDataList.get(j)[num], i, message);
                  }
               }
            }
         }
```

```
            System.out.println(message);
        }
    }
    public static String unicodeToUTF8(int unicode, int num, String word) throws
UnsupportedEncodingException {
        Charset def = Charset.defaultCharset();
        String word1 = "";
        String word2 = "";
        char[] chars = Character.toChars(unicode);
        String charToPrint = new String(chars);
        byte[] bytes = charToPrint.getBytes(StandardCharsets.UTF_8);
        String message = new String(bytes, def.name());
        if(num>0) {
            word1 = word.substring(0, num);
        }
        if(num<word.length()-1) {
            word2 = word.substring(num+1);
        }
        word= word1+message+word2;
        return word;
    }
    public static void UTF8toUnicode(String message)  {
        for(int i=0;i<message.length();i++) {
            System.out.printf("%x, ",(int)message.charAt(i));
        }
    }
    public static int countConfusables(String message, ArrayList<Integer[]> confusables)
{
        int cnt = 0;
        for(int i = 0; i < message.length(); i++) {
            for(int j = 0; j < confusables.size(); j++){
                for(int k = 1; k < confusables.get(j).length; k++){
                    if((int)message.charAt(i) == confusables.get(j)[k]){
                        cnt++;
                    }
                }
            }
        }
        return cnt; }
}
```

# Appendix B

De-obfuscator:

```
import java.io.*;
import java.nio.charset.Charset;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.Random;
import java.util.Scanner;
public class obfuscator {
    public static void main(String args[]) throws IOException {
        ArrayList<Integer[]> confusablesDataList = new ArrayList<>();
        String lineTxt = null;
        Scanner scanner = new Scanner(new FileInputStream("src/confusablesFinal.txt"));
        while (scanner.hasNext()) {
            lineTxt = scanner.nextLine();
            int i=0;
            String[] datas = lineTxt.split(",");
            int n=datas.length;
            Integer[] confusablesData = new Integer[n];
            for (String data : datas) {
                data=data.substring(2);
                confusablesData[i]=Integer.parseInt(data,16);
                i++;
            }
            confusablesDataList.add(confusablesData);
        }//read confusables
        BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
        int cnt = 0;
        while(true) {
            String message = in.readLine();
            for (int i = 0; i < message.length(); i++) {
                for (int j = 0; j < 52; j++) {
                    if (message.charAt(i) == confusablesDataList.get(j)[0]) {
                        Random ran = new Random();
                        if (confusablesDataList.get(j).length > 1) {
                            int num = ran.nextInt(confusablesDataList.get(j).length);
                            message = unicodeToUTF8(confusablesDataList.get(j)[num], i, message);
                        }
                    }
                }
            }
```

```
            System.out.println(message);
        }
    }
    public static String unicodeToUTF8(int unicode, int num, String word) throws
UnsupportedEncodingException {
        Charset def = Charset.defaultCharset();
        String word1 = "";
        String word2 = "";
        char[] chars = Character.toChars(unicode);
        String charToPrint = new String(chars);
        byte[] bytes = charToPrint.getBytes(StandardCharsets.UTF_8);
        String message = new String(bytes, def.name());
        if(num>0) {
            word1 = word.substring(0, num);
        }
        if(num<word.length()-1) {
            word2 = word.substring(num+1);
        }
        word= word1+message+word2;
        return word;
    }
    public static void UTF8toUnicode(String message)  {
        for(int i=0;i<message.length();i++) {
            System.out.printf("%x, ",(int)message.charAt(i));
        }
    }
    public static int countConfusables(String message, ArrayList<Integer[]> confusables)
{
        int cnt = 0;
        for(int i = 0; i < message.length(); i++) {
            for(int j = 0; j < confusables.size(); j++){
                for(int k = 1; k < confusables.get(j).length; k++){
                    if((int)message.charAt(i) == confusables.get(j)[k]){
                        cnt++;
                    }
                }
            }
        }
        return cnt;
    }
}
```