USER ORIENTED EMAIL THREAT ANALYSIS

Connor Best

Contents

Acknowledgements

I would like to thank Dr Phil Legg for the assistance throughout the project development, Lindsey Gillies and everyone else at the Project in Progress day for all of the suggestions that helped further the projects development and everyone who took part in the testing for their feedback.

Abstract

Email threats are ever present and all too unknown in this current day, with millions of people using email and it is making a constant backbone in social and professional communication the lack of understanding of the threats present to the lay user may be one of the biggest issues we face going forwards as technology continues to advance and progress. This project looks into ways to improve not only the average users understanding of the threats they face but also ways to ensure that their continued safety as these threats continue to grow and this is done by creating a sample email system that retrieves emails and informs users of the threats at hand, focusing on wording the explanations as simply and understandably as possible. Through this research it can be understood that users are often split between being frightened of the apparent threats of using email and not being aware they may encounter threats at all. This projects development hopes to rectify this for all users of any knowledge level.

Introduction

As a society humanity has entered what could be best described as the age of digitalization; as of 2016 roughly 2.1 billion people own a smart phone but with roughly one third of businesses having been successfully hit by malware[1] (Anon, 2018) it is obvious that as a society we do not understand the security risks and methods of protection that are crucial for keeping ourselves secure.

In terms of email security, it can be seen through sources like the media[2] (Zeng, 2017) that a large portion of the threat arrives from active and in some way malicious sources. Spam and phishing emails are incredibly common but statistically it is very difficult to come up with accurate numbers on how many people have been successfully hit, an estimated 40% of successful malware infections go completely unnoticed likely because its users do not understand what has happened to them.[3] (Gammons, 2017) This is an issue that must be rectified, not only to allow users a better understanding of the threat faced but to also reduce the amount of attacks that were successful.

As part of this project there are three key questions that we must first identify;

- What can we do to assist all users in understanding the threat posed?

In our current paradigm the individuals that understand the threat posed are the ones tasked with tackling the issues, this is obviously not enough and as such we must ensure all users can grasp the threats posed to them and thus we must take great care in ensuring that all users have at least a basic understanding of the threats posed.

- How do we go about displaying the threat posed?

Currently we do have multiple layers of protection in the form of spam filters and anti-virus, these systems try to automatically detect and remove any threat posed to the end user and while this system works well it must be asked if having a more transparent approach would be beneficial.

- Will informing the user reduce the success rate of these malicious emails?

Continuing on from above if we take a completely transparent approach we must still ask ourselves if this will help, displaying everything that an antivirus does would provide complete transparency, but would it actually be informative to the user enough that security be improved, and will the information provided actually improve a user's state of security.

By finding an answer to these questions it may be viable to improve the overall security of all users but to do so it is necessary to completely alter the current paradigm of security in favour of an improved one. This report hopes to find an

[1] Anon (2018) Number of smartphone users worldwide from 2014 to 2020 [online] Available from: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/ [Accessed 20 Feb 2018]

[2] Zeng, Y.G. 2017, Identifying email threats using predictive analysis, *IEEE* , pp. 1

[3] Gammons, B. (2017). *6 Must-Know Cybersecurity Statistics for 2017, Barkly Blog*. [online] Blog.barkly.com. Available at: https://blog.barkly.com/cyber-security-statistics-2017 [Accessed 27 Dec. 2017].

answer to these questions and find a method to improve user safety and knowledge in relation to email security.

Research

To better understand the current situation a basic question must first be asked; Why are email based threats so prevalent? To answer this question first it must be understood that email is everywhere, as stated in Detecting Targeted Malicious Email (Armin, 2012)[4] "the biggest issue is that all companies allow email into their networks", and why wouldn't they? Email is the most commonly used method for digital communication in many aspects with roughly 4.6 billion email addresses active worldwide so a company not utilizing email would be severely detrimental to themselves. From this it is easily understood why email threats are so numerous and threatening, with so many email addresses so easily accessible a single malicious user can send a few thousand emails with a malicious script and be sure at least one succeeds, in a lot of cases that's all that's needed. To fix our problem we cannot realistically reduce or alter the availability or integrity of any email system even if we wanted to so at this stage it is easy to understand any alterations must be client side.

At current our methods are effective, we utilize anti-virus programs to categorize, locate and deal with any threat that comes our way and often we see this as enough. Our current paradigm is well summarized by Kunal Pandove in the international journal of computer applications[5] (Pandove, 2010). In the article they discuss that the best practices for email security are "protecting users from unwanted email and inbound email vulnerabilities, preventing the dissemination of spam or infected email from the organization and training employees in email best practices" and while this method is effective, having a select group being the protectors that watch over the others, there is a notable distinction which is troublesome. Users should be protected from unwanted email while employees should be trained in the best practices. From a business perspective, training users in the same way employees are taught isn't financially viable but as a general idea this thought that "users must be protected" may be where our current paradigm is falling short.

Of course, considering users and employees different is only a matter of perspective, a user is most likely also an employee from the angle of a different company and it's because of this that the distinction should be further investigated. From a business perspective a user and an employee are differently based on their actions, a user that is professionally operating over their works remote network should be considered an "employee" while a professional browsing Facebook on their break should be considered a user (Brook, 2015)[6]. The obvious issue here is that

[4] Amin, R., Ryan, J. & van Dorp, J. (2012), "Detecting Targeted Malicious Email", *IEEE Security & Privacy,* vol. 10, no. 3, pp. 64-71.

[5] Pandove, K., Jindal, A. & Kumar, R. (2010), Email Security, *International Journal of Computer Applications,* vol. 5, no. 1, pp. 23-26.

[6] Brooke, P. & Paige, R. (2015), User-visible cryptography in email and web scenarios, *Information and Computer Security,* vol. 23, no. 1, pp. 58.

considering an employee and a user differently is a very business centric idea and in a realistic environment there isn't really a difference. As such we see that all users should be treated the same way. We can also gain from this a similar issue in looking from a business perspective is that user activity and employee activity will vary greatly, employees will be expected to operate in a very particular way with a rigid structure to email usage which means that any system implemented can rely on this as a backbone, but users do not have this some predictability. As such this project cannot be built with the perceptions of operating within a business but more as seeing all users as a whole.

The next step is to understand the capacity of what needs to be done to ensure the user understands any new system given to them. As discussed in 2001's Building the Ubiquitous Computing User Experience (Edwards, 2001)[7], having a new system be too complicated can cause a user to feel more uncomfortable using it to the point they simply may not use it at all, even if it is beneficial, and the fact that this remains an issue to this day means that it is of crucial importance to consider now, at the very least most elderly people are less familiar with modern technologies and at most they are often afraid to try and learn, This is best explained by the article Factors Influencing the Acceptance of Technology by Older People (Raymundo, 2014)[8], "Whereas the young are born into a world in which advanced technology is already a part of their daily lives, the elderly were adults at the time of their first contact with such technology". So, if we are to introduce new ideas to the user we must ensure the users can feel comfortable using it. Going further into Edwards article above it explains that developing new technologies is a balance between user experience and function as if a technology is too difficult or frustrating for a user then they will not use it, no matter how beneficial it would be. A good example of this would be Windows OS security updates, often a user will not install an update until they have to, sometimes going to the extent of not turning their machine off, because the update will take so long, this is being mediated by the use of update times when a user is less likely to be using their machine.

At this stage a basic outline of what's required can be built to proceed. A client-side email application that is used to locate and inform a user is something that we can use to answer our main questions however the application must be built to be as accommodating to user experience as possible. The host device chosen should that which is most commonly used and as of November 2016 (Titcomb, 2016)[9] that device is the smartphone. Usability between the two primary brands is also important so utilizing a framework that can operate on both devices smoothly is also crucial. So

[7] Edwards, W., Newman, M. & Sedivy, J. (2001), "Building the ubiquitous computing user experience", ACM, pp. 501.

[8] Raymundo, T.M. & da Silva Santana, C. (2014), "Factors Influencing the Acceptance of Technology by Older People: How the elderly in Brazil feel about using electronics", *IEEE Consumer Electronics Magazine,* vol. 3, no. 4, pp. 63-68.

[9] Titcomb, J (2016), Mobile web usage overtakes desktop for first time, *The Telegraph* [online], Available At: http://www.telegraph.co.uk/technology/2016/11/01/mobile-web-usage-overtakes-desktop-for-first-time/ [Accessed 17 December 2017]

now that a basic framework is outlined the methods used for email threats is the next step.

Utilizing research done in the past on user behavioural patterns in relation to emails (Sheng, 2015)[10] we can see that users are most likely to click on emails sent from sources they are familiar with and rarely click on ones that aren't. However still over 18% of users are still willing to click a link within a website from a sender they are unfamiliar with. So, during development due to application of resources it will not be possible to designate an entire list of legitimate business emails but beyond that the development of methods to find the specific portions of an email that are threatening should be the goal. Going back to *Detecting Targeted Malicious Email* it is discussed that one of the primary ways that the main objective could be solved is working with a company that already operates with identifying email threats and utilizing these newly created methods with their database structure to identify user threats. In this instance that is not viable and because of such what is achieved within this report must be held to the same standard as if we were working with a third party in case that is a viable alternative in the future.

Discovering the most common methods of spam detection is therefore the next best step, utilizing the most common methods is a reliable method to catch most sources of spam and phishing attempts. Analysing a collection of blogs and articles finds many methods of spam detection and their avoidance. As discussed in A Marketer's Guide to Email Deliverability: How to Avoid Email Spam Filters (Kolowich, 2014)[11] a large portion of what is looked for is text, spam word lists, subjects in all caps, embedded video and images that contain text are all commonly and very forcefully pushed into the bracket of spam emails, these are all things that many groups will use as they either know no better or are too lazy to try otherwise because if one person falls into their scheme then they are successful. The existence of these articles is something to be considered onto itself, the creator and distributor of spam and phishing emails can look at these articles too and find an almost complete what not to do if they wish to be successful. As such we must move beyond these simple methods and find other, better methods.

To further understand where email analysis is heading in the future a few different papers will be analysed in detail. In the journal article Improvement of Email Threat Detection by User Training (Cousin, 2017)[12] a similar idea to the one posed within this report is passed but in a different manner, it states "Mail credibility becomes difficult to discern because of growing useless "information" polluting inboxes." Which is an accurate statement of the impact of spam emails, to simplify the

[10] Sheng, Y., Rong, J. & Xiang, W. 2015, "Simulation of the Users' Email Behaviour Based on BP-BDI Model", *IEEE*, pp. 16.

[11] Kolowich, L (2014), A Marketer's Guide to Email Deliverability: How to Avoid Email Spam Filters, *Hubspot,* [blog] *Available online:* https://blog.hubspot.com/blog/tabid/6307/bid/30594/a-marketer-s-guide-to-getting-past-email-spam-filters.aspx [Accessed 3 Jan 2018]

[12] Cousin, P., Bernard, V., Lefaillet, A., Mugaruka, M. & Raibaud, C. 2017, "Improvement of Email Threats Detection by User Training".

argument the more spam a user gets the less they will trust the information provided by their emails, users feel threatened just using a simple service like emails as they fear that their personal information will be stolen and the fault lies heavily in the hands of the media as most lay users will only hear scaremongering about how dangerous doing anything online is. As such it can be understood the importance of teaching users enough to not be quite so afraid, this goes back to the first question and shows that any information displayed must be displayed honestly without sounding too intimidating. The journal continues to discuss how most security methods are defeated and how "most existing emails filtering approaches are static easy to defeat by modifying emails content and link strings." And with this the above questions can be modified even more, any system developed must have at least the innate capacity to be developed further, static values and security systems cannot be the focus (though they aren't to be ignored). The journal also mentions a browser addon called Thunderbird, this addons functionality operates similarly to how the email application being developed will operate but has an interesting function to consider. The functionality of Thunderbird doesn't directly interact with the emails but instead informs the users of possible threats, this is something of importance to note for the future as displaying tips to the user for items that cannot be reliably discovered might be an alternative option. The article goes on to discuss how use of Thunderbird may make a user complacent and while this may be somewhat true it is for that reason why interactions need to be clear and concise. It is known and mentioned that Thunderbird isn't completely accurate as no system like this can be, legitimate emails will be flagged. The journal in this regard isn't exactly accurate, users are often already complacent with their emails as this is what social engineering prays upon, creating a system that displays these issues should not do so with the premise of finding suspicious emails but instead informing the user what parts of an email are suspicious. Thunderbird had the right idea but was too focused on user protection and not user information.

This journal is still useful to us for a different reason however, it is also attempting to discover if user-oriented email threat detection is worthwhile. The journal used a large number of subjects and came out with mixed results that, while reasonably bias, do provide some useful information. The results of their study showed that there is not a large decrease in the number of phishing emails sent however for the largest category in this study the results would come under spear phishing as the email was targeted, this is not mentioned within the study but does show us that general phishing attempts can be reduced by a worthwhile amount but further research is needed to discover if evidence of targeted malicious content is viably teachable to a user.

This leads into the next journal to be analysed, Detecting of Targeted Malicious Email (Deshmukh, 2014)*[13]* discusses utilizing machine learning to discover suspicious items targeted at a single user. The journal takes a mathematical approach of calculating the occurrence of certain keywords gained from the rest of the user's inbox and while this method would be effective it is time consuming for a

[13] Deshmukh, P., Shelar, M. & Kulkarni, N. 2014, "Detecting of targeted malicious email", *IEEE*, pp. 199.

large number of emails would be needed to learn from. Though these methods would be time consuming this is an effective and reliable method for detection of targeted materials though with this more than anything reliability is important as the distinction between safe and unsafe would be crucial. The journal suggests multiple machine learning methods, but the crux of the situation is that there is a trade-off between reliability and time, either many thousand emails are needed for supervised learning and unsupervised learning has little chance of distinguishing between threats and non-threats.

With the above information a good understanding of what is needed to be developed can be built, the email application must be reliable but also fulfil the above specifications. The next step is understanding how it is to be built.

Requirements

To understand what needs to be done with the project a set of requirements was created to ensure that any goals could be met with certainty.

No.	**Functional (F)**	**Non-Functional (N)**
1	**Log into a standardised email account (Must)**	**Display clearly that an email may be suspicious (Must)**
2	**Analyse the contents of an email for evidence of phishing (Must)**	**allow the user to decide whether an e-mail is suspicious, based on the recommendations generated by the system (Must)**
3	**Analyse the contents of an email for evidence of malware (Must)**	**Ensure suspicious activity is shown immediately upon it being accessible to the user (Must)**
4	**Investigate attachments for obvious malicious activity (Must)**	**Display clearly the source of all emails (Must)**
5	**Investigate hyperlinks for obvious malicious activity (Must)**	**Inform the user in an understandable and non-demeaning way (Should)**
6	**Flag unusual emails and html elements. (Must)**	**Be set out in such a way that all operations of the app are understood and utilized (Should)**
7	**Investigate malware without allowing the device to be compromised. (Must)**	**Allow the user to mark down specific emails to warn them in the future (Should)**

8	**Operate as a standard android email application would send & receive emails, allow emails to be archived and deleted (Should)**	**Advise users on what actions to take to process specific threats (Should)**
9	**Load server data in regular intervals (Should)**	**Ensure that emails are available as soon as the user can access them. (Should)**
10	**Securely delete flagged emails to not appear within the "deleted" folder (Could)**	**Display in a variety of fonts and sizes dependent on the user's settings (Could)**
11	**Set rules that can be altered by the user to determine what is viewed as malicious or suspicious (Could)**	**Teach users the standard red flags of a suspicious email. (Could)**
12	**Update the malware database remotely (Could)**	
13	**Collect data about suspicious emails to detect usage information of malware. (Wont)**	
14	**Adapt to find specific spear phishing data. (Wont)**	

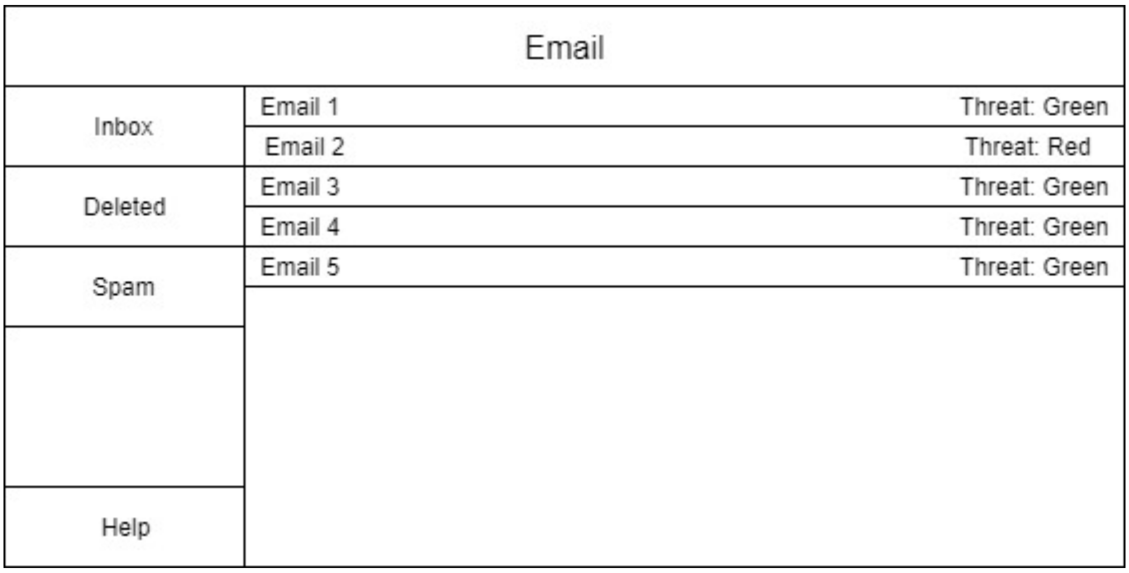
Methodology

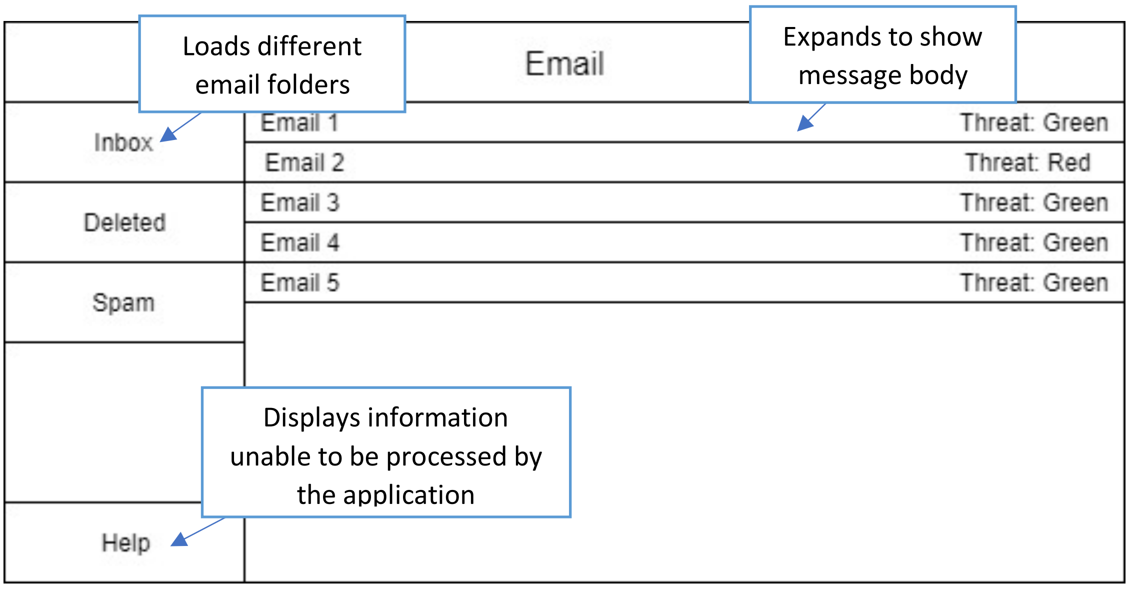
An Agile methodology was chosen for this project as this methodology allows continual and regular updates as the project continues development. This allows for constant bug fixes, regular stages of minor testing for both bugs and functions and the option to alter functionality to meet the demands of later testing and research.

The details of each sprint will be discussed in further detail in the Implementation and Testing section, but a basic rundown of each sprint shows that the first sprint was implementing a functioning text-based email retrieval system, the second sprint was implementing detection methods that integrated with sprint one and sprint three was the implementation of this into a GUI.

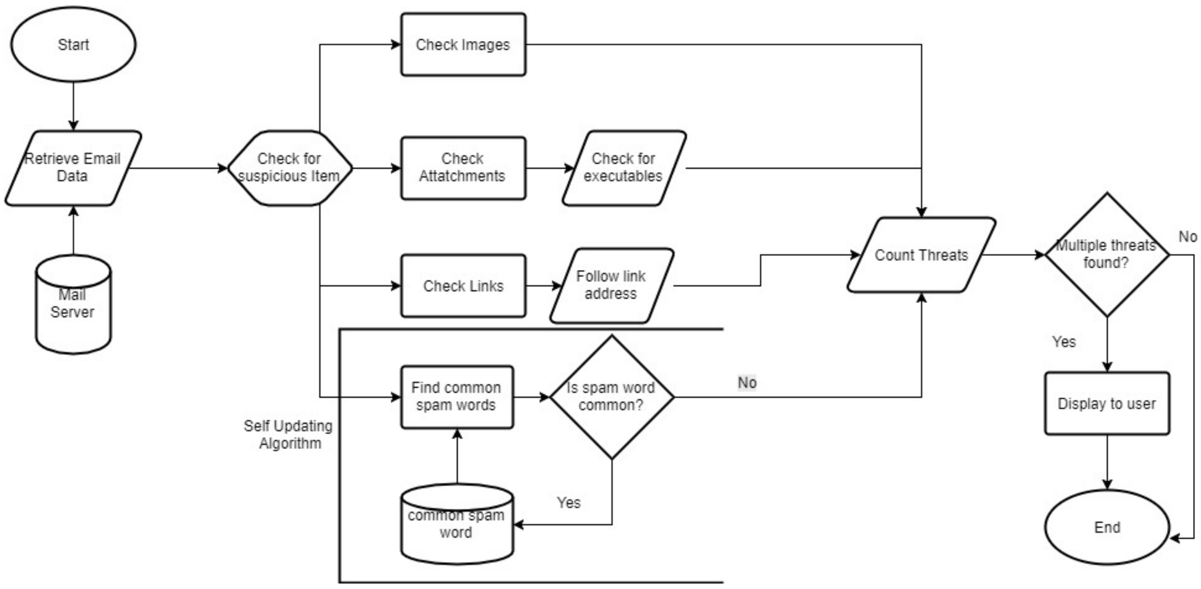
Designs

UI design





Activity Diagram



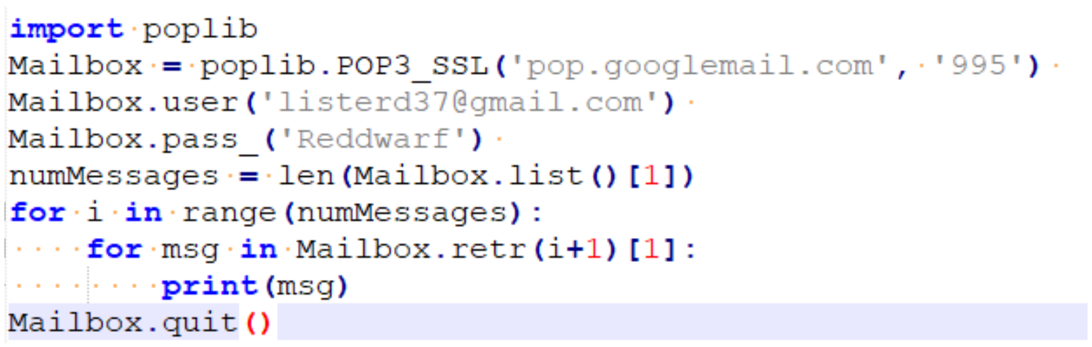
Implementation & Testing

Planning

Once the outline of the project had been decided the first goal was to decide the specifics of its development. The first step taken was the decision on the environment and languages used; Out of all options two programming languages were seen as most viable due to a preconceived grasp of the language; Java or Python. Java is known to be versatile at object-oriented development and its usage with programming android apps, but Python is more versatile and has the option of utilizing environments like Flask or Bottle to create a web application and after the discovery of Framework 7 and its capacity to operate for either mobile environment it was decided that utilizing Python Flask and Framework 7 would be the most beneficial.

Sprint 1: POP3 Email retrieval

The next step taken was development of a method to retrieve emails. The original script was built utilizing the POPlib library to retrieve and read emails, while the code utilized was functional the library used was found to be very limited, the most particular issue was the interaction between the POPlib library and the mail server used (in this case Gmail), having the two interact even with Gmail's POP3 system enabled would often fail on certain emails. This was likely because POP3 is now outdated, replaced by IMAP, and while most emails are still identifiable some emails will be lost. Due to this it was decided that an IMAPlib library or something similar would be utilized instead.

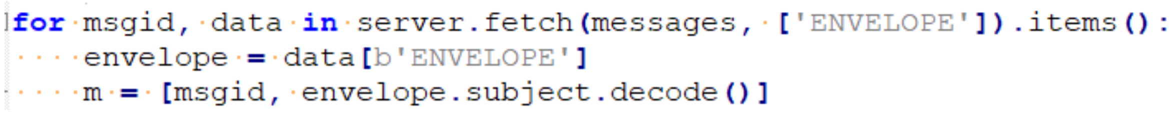


```
import poplib
Mailbox = poplib.POP3_SSL('pop.googlemail.com', '995')
Mailbox.user('listerd37@gmail.com')
Mailbox.pass_ ('Reddwarf')
numMessages = len(Mailbox.list()[1])
for i in range(numMessages):
    for msg in Mailbox.retr(i+1)[1]:
        print(msg)
Mailbox.quit()
```

Sprint 1: IMAP email retrieval

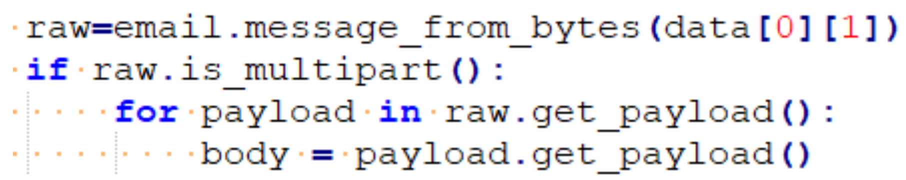
IMAPlib operated far more simply and allowed for a larger variety of interaction with the email server, IMAPlib retrieves the information about subject and header simply but required alteration to allow the body of the email to be read. Without alteration the raw body of the email appeared as a random string of text and numbers and did not display any relevant or usable information.

At this stage two different options were investigated, utilizing IMAPclient allowed for a simpler method of retrieving the sender address, subject and header of the email but had no method to retrieve the body at all.



```
for msgid, data in server.fetch(messages, ['ENVELOPE']).items():
    envelope = data[b'ENVELOPE']
    m = [msgid, envelope.subject.decode()]
```

After which the python email library was tested, this allowed for reliable processing of email body, header and attachment through IMAPlib and as such was chosen as the best option. The code used to parse the body via text is shown below.

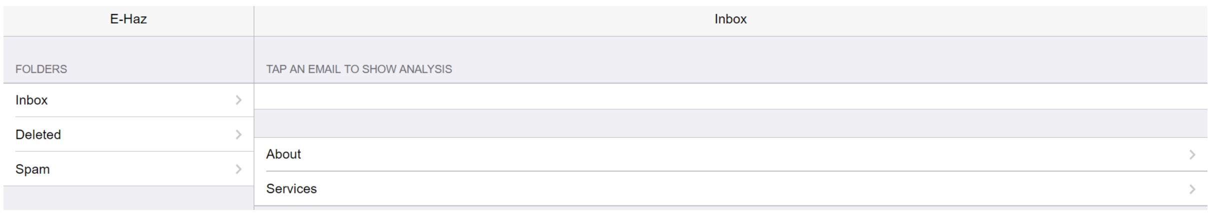


```
 raw=email.message_from_bytes(data[0][1])
 if raw.is_multipart():
     for payload in raw.get_payload():
         body = payload.get_payload()
```

Utilizing IMAPlib and the email library allowed for two requirements to be met in this early stage. IMAPlib retrieved emails from the server instantaneously with no threat of major client-side delay allowing for the confirmation of F1 and N9 as emails will be available the moment they are on the server instead of having to parse through and find specific ID’s as is often necessary when searching through POPlib.

Sprint 1: GUI design

Alongside development of a method to retrieve the contents of an email was the creation of the GUI using Framework 7. A few examples were tested mostly involving altering the standard templates provided, in the end utilizing a view panel with infinite scrolling in the body was decided for its simplicity and ease of use on both android and IOS. With the templates available further alterations down the line were easily done if necessary. The functionality of this GUI would allow for continued alterations and testing as the project progresses as well as ensuring that the GUI functions on all standard devices, including computers.



Sprint 2: Image analysis

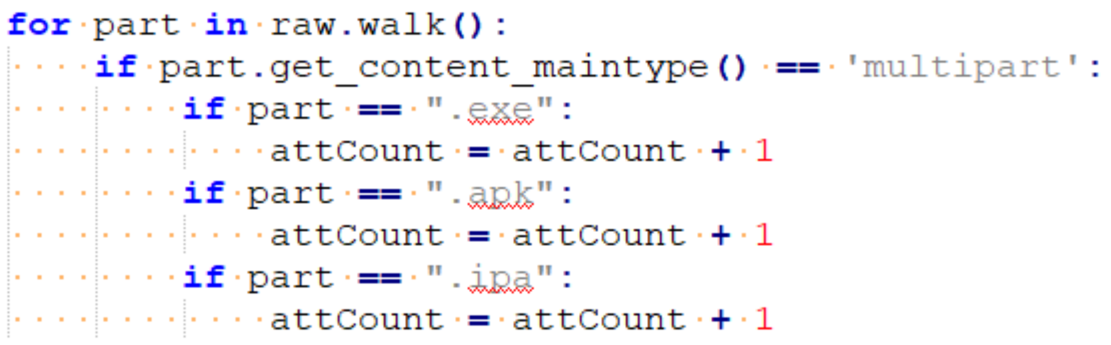
Once the body of the email could be read the analysis of the emails began, this started with a simple procedure of checking if a picture within an html email is legitimate. A common item in many spam emails are images that appear to not load in. Originally this was detected by simply finding the image and checking if the source equalled "//0", this represents a null image and is something reliably suspicious as it is difficult to do accidentally, images that fail to load legitimately will have a source. There are also spam emails that utilize the height and width of an image to be 0, therefore having an image but making it useable. This, as well as many of the other detection systems, was done with python's beautifulsoup library, a library used for scraping portions of data from a given source. Down the line these suspicious items were given numeric values which were planned for use with an identification system to allow a brief look at how much of a threat a single email was. The values used were given as such depending on how reliably they could be done accidentally, if an item was something that a lay user could accidentally do then the number was smaller but if something was legitimately more difficult to do accidentally then the number was higher. For instance, the above examples had the null image as a 2 while the height and width were a 1. The values chosen were not originally specific and were more a representation of what could be.

```
for img in soup.findAll('img'):
    images.append(img.get('src'))
    if '//0' in images:
        imgCount = imgCount + 2
        imgCounter.append(imgCount)
    if 'width = 0' in images:
        imgCount = imgCount + 1
        imgCounter.append(imgCount)
    if 'height = 0' in images:
        imgCount = imgCount + 1
        imgCounter.append(imgCount)
```

Images themselves are rarely an object used maliciously within emails, but they are usually a good flag of suspicious activity. With current functionality the images metadata is shown and while this is something that cannot be expressly checked by the application this is something that will be brought up to the user, if the Metadata seems odd then there is a reasonable chance that the containing email may be malicious or at the very least spam. Further development could utilize image analysis to detect if any images would be considered inappropriate or obscene as well as using the metadata and image link to calculate if the image is from the legitimate source. This image detection system allows for the confirmation of requirement F6 and N2 as images flagged are shown not as a threat but as something to investigate as well as ensuring that a common oddity shown in many emails is clarified to users.

Sprint 2: Attachment analysis

The next step was checking for suspicious attachments, this method was done using a similar method used to decode the email body, as shown below;



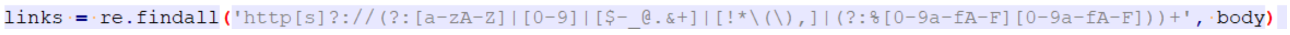
```
for part in raw.walk():
    if part.get_content_maintype() == 'multipart':
        if part == ".exe":
            attCount = attCount + 1
        if part == ".apk":
            attCount = attCount + 1
        if part == ".ipa":
            attCount = attCount + 1
```

The above code was used to check if the email was multipart and if an attachment was detected, if it met these criteria then the attachment was analysed to see if it was some form of executable by searching for an .exe, .ipa or .apk extension. The benefit of utilizing the email library functionality means that all the metadata surrounding the attachment will be analysed, meaning that a poorly hidden attachment will be more likely to be discovered.

In future development the idea of having the app send the flagged extensions back to a parent server that can be used to analyse the contents of an email in depth may be an option as this would potentially allow new waves of malware to be discovered early on. If this system were to be developed further still a quarantine zone could be set up similarly to how certain antivirus handles their analysis which would utilize an in system virtual machine to see what happens when the .exe is ran. This system would analyse file and permission changes as well as port access to see what changed when the executable was ran. A difficulty arises in the fact that most user mobile devices will not have the capacity to do this, not to mention that different executables will not run on mobile devices. The fact that these items can be investigated safely and without threat to the user's system means that requirement F7 and F4 can be confirmed.

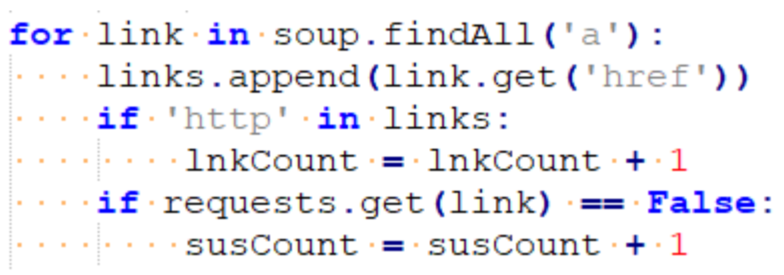
Sprint 2: Link Analysis

Due to their varied use and potential threat to the user links were an important point of analysis also. Originally links were analysed a regular expression library that allowed a findall to be ran to search for all links. This function (see below) found URL's but did not detect secure link URL's nor URL's with unconventional identifiers like those of other countries.



```
links = re.findall('http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\(\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+', body)
```

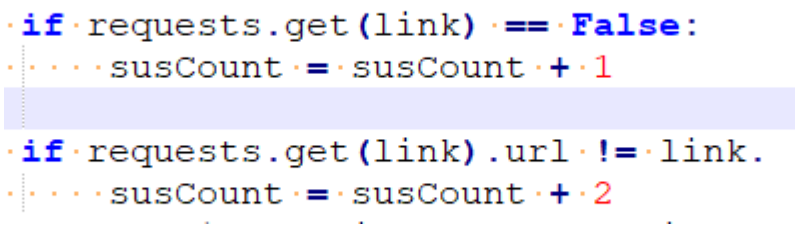
Instead beautifulsoup was used, this allowed all links to be defined and by using the "lxml" parser the entire link could be discovered and analysed.



```
for link in soup.findAll('a'):
    links.append(link.get('href'))
    if 'http' in links:
        lnkCount = lnkCount + 1
    if requests.get(link) == False:
        susCount = susCount + 1
```

Using this method, we can detect the basic security of a link, detecting if it is an http or https link can say a lot as while http is the more common a user can be more reassured if the link provided is https. Similarly, any links provided from legitimate websites that are not simply advertisements will likely be in a secure link. Utilizing the requests library, it is possible to check the general status of an SSL or TSL certificate on a website, the simple line of code above is enough to ensure that the link contains a valid SSL and is not an instantaneous redirect which can be flagged by the same library.

Shortened links were a more difficult resource to process as they may still appear shortened when displayed, as such utilizing requests to scrape the URL inside the link and displaying both allowed the bypass of shortened URLs but also to confirm if an address was correct, if the two URLs displayed were vastly different then something may be wrong and this was treated as a severe incident, utilizing a shortened link is a common occurrence but having the client side link be different than what it is reflecting is a dangerous and quite obviously malicious sign that can only be done by actively altering the URL destination. Hidden downloads could be detected using this method as requests flags links that lead directly to downloads as such, though most devices do not allow an application to automatically run upon start up some dangerous services still try and it can be especially effective on mobile devices where users rarely check the permissions they provide an application nor does the device particularly “announce” the start-up of an unknown script.



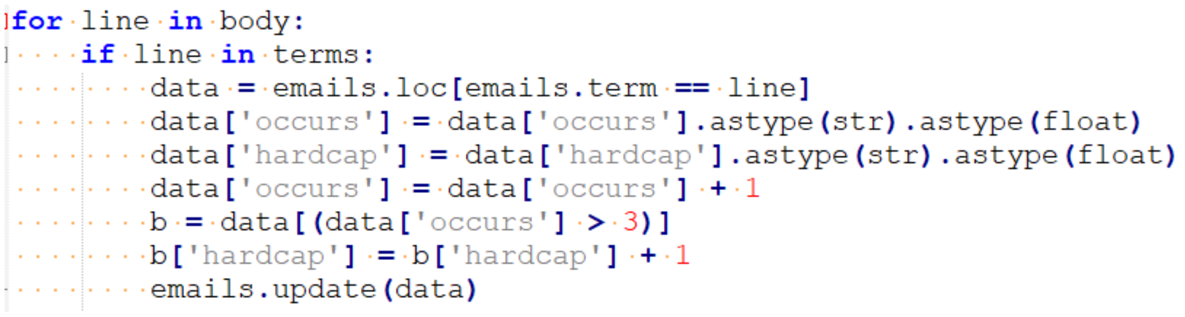
```
if requests.get(link) == False:
    susCount = susCount + 1

if requests.get(link).url != link.
    susCount = susCount + 2
```

Continuing to utilize links further in the future it may be viable to check the URL provided for TSL and SSL certificate information to allow detection of man in the middle attacks. A masquerading web page will not have the valid certificates. It may also be a viable strategy to scrape the URL for items of interest, this can also be done by requests and would require a basic setup looking for specific HTML items, likely forms that submit to other URLS, buttons that redirect or cookies that the site uses. All of which would allow for a more reliable understanding of if a link is secure. This goes towards the confirmation of requirements F7 and F5.

Sprint 2: Spam Word Analysis

Once the commonly known and simple steps were met a more adaptive solution had to be met, something that could keep itself up to date with the tide of traffic within spam mail. Most companies accomplish this with a spam filter that searches for keywords and designates spam based off this and while that is a functional system it could be taken a step further. Utilizing a large database of spam emails gathered via the Enron Email Analysis Project[14]. Utilizing this dataset and a list of common spam words[15] allowed an algorithm to be set up that sweeps for each occurrence of a phrase and adds to a counter every time it appears, however this is a process that would case a lot of common terms that appear as standard to get flagged as the most dangerous and this would take away from the projects reliability, this was fixed by having two statistics, "counter" and "occurs". Occurs counted the number of occurrences in a file or email and once that number hit an ever-increasing cap value then it would increment the counter. This severity could naturally increase as trends in spam mail changed, though the dataset would require updating regularly.



```
for line in body:
    if line in terms:
        data = emails.loc[emails.term == line]
        data['occurs'] = data['occurs'].astype(str).astype(float)
        data['hardcap'] = data['hardcap'].astype(str).astype(float)
        data['occurs'] = data['occurs'] + 1
        b = data[(data['occurs'] > 3)]
        b['hardcap'] = b['hardcap'] + 1
        emails.update(data)
```

Once implementation is complete if the project is in regular use this algorithm will need to either reset after a period of time or be altered to make up for long term use as otherwise certain terms may become so severe that they will automatically flag an email by its appearance alone. This could be done with a hard cap on the severity but that itself would still lead to the same issue, so a quarterly examination of the dataset may be the best option. Utilizing this further it may be possible to create some kind of learning algorithm that detects occurrences of words, this system would need to be supervised as training data would need to be used to specify what words to flag, which are safe and even which to ignore entirely. This machine learning technique could be utilized to assist keeping knowledge of the most used spam words regularly up to date for other services too.

Upon completion of this project this aspect of the program could not be completely integrated, while emails are read there is an issue with interactions between the list of words to detect and the list of words within the email. This may be due to the body of the email being represented within the message object type, which interacts as a byte but does not have all the capacity of a byte instance in python 3.

Sprint 2: Spelling Analysis

Detection of misspelled words within a supposedly legitimate email is also a common method of suspicious item detection. This was originally done utilizing the python

[14] http://www.cs.cmu.edu/~enron/

[15] https://blog.hubspot.com/blog/tabid/6307/bid/30684/the-ultimate-list-of-email-spam-trigger-words.aspx

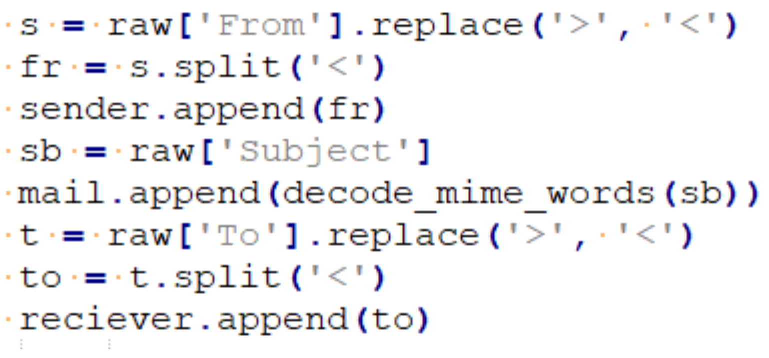
spell autocorrect library[16], this functioned and did discover any misspelled words found in basic testing however due to a reoccurring issue with this and similar libraries this was removed from the program.

The first issue found was that this library did not highlight words found, instead automatically correcting them, this meant that a simple spell checker was built instead[17], this flagged messages correctly and allowed them to be displayed however when utilized on email text the words flagged were often company titles, products and links included within the email. This meant that without further development of this spell-checking application any emails would retrieve many false flags and may scare the user into believing a legitimate email may be false. This could be worked on to improve its functionality but due to a separate underlying issue it was deemed to unreliable to function. Any word that contained any capital letters or unknown characters were not flagged, this is a very common and simple method of spam detection and as the spell checker used had no capacity to detect this the idea was scrapped.

If this idea were followed up creating a specifically designed spell checker that could have words added to its dictionary by the user and detected words that contained breaks or capital letters, then this would resolve this issue. Spell checkers like this exist for most major browsers and as such instead it has been suggested within the app that users utilize these in conjunction.

Sprint 3: Further Email Data Retrieval

Once the detection of common malicious and spam methods was complete and tested to a basic extent the completion of the GUI and email retrieval was continued. This began with finding sender information, an earlier suggestion was suggested in so that code used to analyse sender email would also check sender name against the sender email and analysed if the name fell under certain known company names, the issue with this method was that most companies, especially banks, are unwilling to openly provide their customer email making the list of legitimate emails difficult to create, hard coding software like this also isn't efficient as the list may change regularly meaning the system would require too much upkeep then was necessarily viable. Instead a simpler option was taken, the email and sender name are shown next to each other to allow users to see any oddities themselves.



```
 s = raw['From'].replace('>', '<')
 fr = s.split('<')
 sender.append(fr)
 sb = raw['Subject']
 mail.append(decode_mime_words(sb))
 t = raw['To'].replace('>', '<')
 to = t.split('<')
 reciever.append(to)
```

16 https://pypi.python.org/pypi/autocorrect/0.1.0
17 http://www.openbookproject.net/courses/python4fun/spellcheck.html

This same system was used for the receiver as while this may not always be relevant if an email is sent to either an incredibly large group of people or a group of people that share no association then this may be a good flag to the user that the email sent was not directed at them and may be a good method of detecting obvious spam email. The system implemented utilized the before mentioned email library and because of its in-built properties the sender (and receiver) name and email address can be seen at the same time. This grants users the ability to see for themselves if an email stating it is from a specific organization or company is utilizing a viable email address.

Three tips to get the most out of Gmail

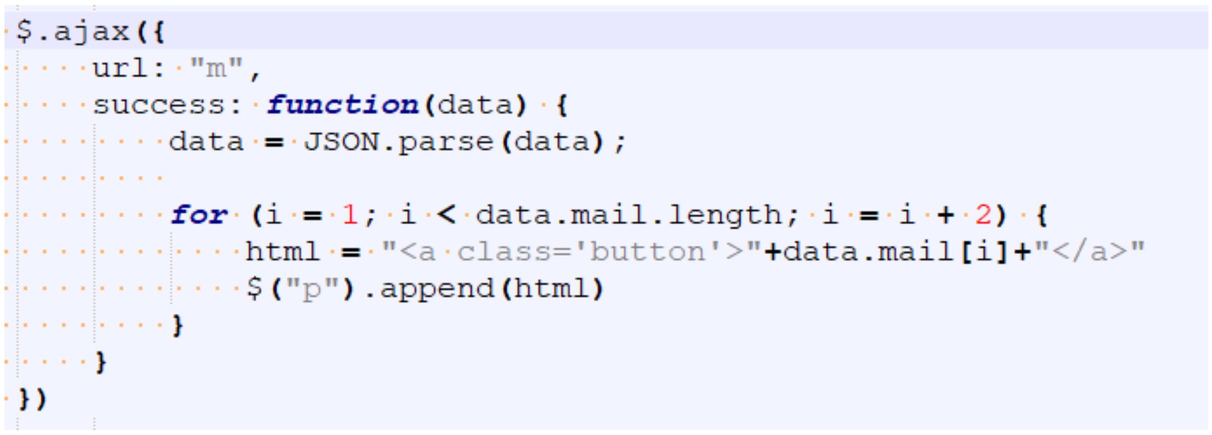
From: Gmail Team ,mail-noreply@google.com,

To: David Lister ,listerd37@gmail.com,

Sprint 3: GUI implementation

Once text-based analysis was completed implementing the methods alongside a GUI was the next step. Originally the library IMAPclient was utilized for its simplicity and capacity to bypass IMAPlibs inability to process certain data types. So, while IMAPclient functioned it had no innate capacity to display the body of an email so reverting back to the standard IMAPlib library and the python email library the code was instead developed allowing all details to be processed.

In order to join the python back end and the html front end together a json library was used, the script began as a basic statement to send all of the outputs to the html document but was later refined to allow specific items to be selected which was used in defining the later body page. The json script counted how many emails were found on the server and printed them all, for each email found the script created a button for each. After the basic emails could be displayed the threat count was shown as a base number.



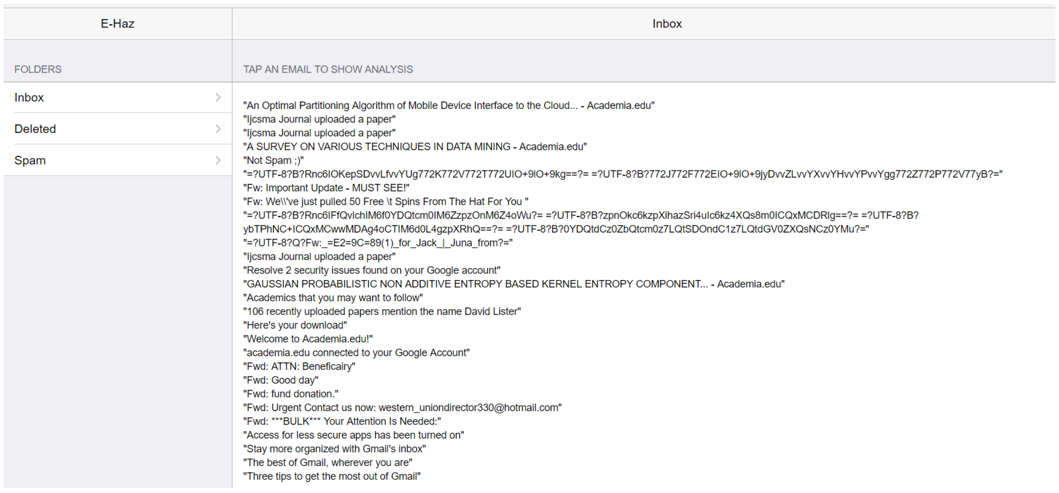
```
$.ajax({
    url: "m",
    success: function(data) {
        data = JSON.parse(data);

        for (i = 1; i < data.mail.length; i = i + 2) {
            html = "<a class='button'>"+data.mail[i]+"</a>"
            $("p").append(html)
        }
    }
})
```

Once this was proven to function a more complicated method was used that produced framework 7 items as shown below;

```
for (i = 0; i < data.mail.length ; i = i + 1) {
    html = ''
    html += '<li class="accordion-item">'
    html += '<a href="" class="item-link item-content">'
    html += '<div class="item-inner">'
    html += '<button>'+data.mail[i]+'</button>'
    html += '</div>'
    html += '</a>'
    html += '<div class="accordion-item-content">'
    html += '<div class="block">'
    html += '<div class="row">'
    html += '<div class="col-40" onclick="sender()"> From: '+data.from[i]+'</div>'
    html += '<div class="col-40" onclick="reciever()"> To: '+data.to[i]+'</div>'
```

This displayed, though some apparent issues were found as can be seen from this screenshot;



```
"An Optimal Partitioning Algorithm of Mobile Device Interface to the Cloud... - Academia.edu"
"Ijcsma Journal uploaded a paper"
"Ijcsma Journal uploaded a paper"
"A SURVEY ON VARIOUS TECHNIQUES IN DATA MINING - Academia.edu"
"Not Spam ;)"
"=?UTF-8?B?Rnc6IOKepSDvvLfvvYUg772K772V772T772UIO+9lO+9kg==?= =?UTF-8?B?772J772F772EIO+9lO+9jyDvvZLvvYXvvYHvvYPvvYgg772Z772P772V77yB?="
"Fw: Important Update - MUST SEE!"
"Fw: We\\'ve just pulled 50 Free \t Spins From The Hat For You "
"=?UTF-8?B?Rnc6IFfQvlchIM6f0YDQtcm0IM6ZzpzOnM6Z4oWu?= =?UTF-8?B?zpnOkc6kzpXihazSri4uIc6kz4XQs8m0ICQxMCDRlg==?= =?UTF-8?B?
ybTPhNC+ICQxMCwwMDAg4oCTIM6d0L4gzpXRhQ==?= =?UTF-8?B?0YDQtdCz0ZbQtcm0z7LQtSDOndC1z7LQtdGV0ZXQsNCz0YMu?="
"=?UTF-8?Q?Fw:_=E2=9C=89(1)_for_Jack_|_Juna_from?="
"Ijcsma Journal uploaded a paper"
"Resolve 2 security issues found on your Google account"
"GAUSSIAN PROBABILISTIC NON ADDITIVE ENTROPY BASED KERNEL ENTROPY COMPONENT... - Academia.edu"
"Academics that you may want to follow"
"106 recently uploaded papers mention the name David Lister"
"Here's your download"
"Welcome to Academia.edu!"
"academia.edu connected to your Google Account"
"Fwd: ATTN: Beneficairy"
"Fwd: Good day"
"Fwd: fund donation."
"Fwd: Urgent Contact us now: western_uniondirector330@hotmail.com"
"Fwd: ***BULK*** Your Attention Is Needed:"
"Access for less secure apps has been turned on"
"Stay more organized with Gmail's inbox"
"The best of Gmail, wherever you are"
"Three tips to get the most out of Gmail"
```

Certain email subjects were appearing erroneously, the original assumption was that their emails were a common method used to trip up spam filters where unknown ascii characters would be utilized but it was later found that they simply required a different form of decoding then standard, which was fixed via the below code;

```
def decode_mime_words(s):
    return u''.join(
        word.decode(encoding or 'utf8') if isinstance(word, bytes) else word
        for word, encoding in email.header.decode_header(s))
    return decode_mime_words(u'=?utf-8?Q?Subject=c3=a4?=X=?utf-8?Q?=c3=bc?=')
```

Once this was utilized the subjects displayed, though issues still continued with the body of the email. The original plan was to have a button that would switch between plain text and html views of the emails. Plain text worked as expected however the html could not be displayed. This issue continues to be prevalent and as such a permanent decision was made to change the displays to only text. This is something that will continue to be worked on in the future, but the issue stands that due to the python 3 email library utilizing an email payload as a byte's type object converting it into json was found to be extremely difficult as certain emails broke the UTF-8 codec, meaning they could not be displayed at all. This does cause an issue in completing a function and means that the tool will not operate as a standard email system, though this does not take away from its overall functionality.

In order to display the body of the email over text however the below line was used to filter the output.

```
if payload.get_content_type() == 'text/plain':
```

This altered to outputs as such;

```
Three tips to get the most out of Gmail [image: Google] Hi David Tips to get the most out of Gmail [image: Contacts] Bring your contacts and mail into Gmail On your computer, you
can copy your contacts and emails from your old email account to make the transition to Gmail even better. Learn how . [image: Search] Find what you need fast With the power of
Google Search right in your inbox, it's easy to sort your email. Find what you're looking for with predictions based on email content, past searches and contacts. [image: Search] Much
more than email You can send text messages and make video calls with Hangouts right from Gmail. To use this feature on mobile, download the Hangouts app for Android and Apple
devices. [image: Gmail icon] Happy emailing, The Gmail Team © 2017 Google Inc. 1600 Amphitheatre Parkway, Mountain View, CA 94043
```

Sprint 3: Threat counters

Due to the reoccurring UTF-8 codec issue displaying the threats detected visually was instead replaced with counting their instances and displaying that to the user, creating "counters" for every instance of a threat was something already done for images and words and as such continuing to do so was simple.

These counts were directly interpreted into the html and displayed under the body of the email, once this was done a new counter was created that added together all those designed to track "malicious" activity, add them together and colour code the message via this, this is shown below.

```
if(data.all < 5) {
    html += '<div style="background-color: lightgreen" class="col-30"> Overall Threat:'+data.all+'</div>'
} else if(data.all > 4 && data.all < 9) {
    html += '<div style="background-color: orange" class="col-30"> Overall Threat:'+data.all+'</div>'
}
```

This method allowed a simple representation of all the stats of an email to be shown, in further development this system could be improved upon, allowing for emails to be sorted via their threat level too.

Testing: Lay User Experience

Testing was taken with a group of 11 individuals who all ran the application over their own email address and the sample email address to see what they could find and what conclusions they may draw from that.

These users were.

- 2 users between 8 and 15
- 4 users between 15 and 25
- 4 users between 25 and 40
- 1 users between 40 and 55

These tests were done in person and followed the basic conversational guidelines of the below questions.

Q1. Did you find any suspicious emails on your own account?

Yes / No

Q2. Did you find any suspicious emails on the test account?

Q2. Do you think you would have noticed these emails without using the app?

Yes / No

Q3. Do you feel any more confident about finding suspicious emails on your own after using the app?

Yes / No

Q4. What was discovered?

____

____

As these conversations were primarily informal and with a relatively small focus group no overall comparisons can be made on the information gained. By analysing their own email addresses only 3 of the 11 discovered a suspicious email though 6 of the 11 discovered emails on the test account. Through discussion and testing of other user accounts a few things were noted. Firstly, the email system was refined so that instead of displaying the folders in the standard way instead by having all emails displayed together meant that users felt more comfortable that they could spot the suspicious ones. Though this does lead to the threat of possibly losing better hidden emails this design choice was favoured as it directs people as to what they should be looking for instead of being critical across all emails.

Something of important note was a discussion with three of the individuals, it was stated that they felt comfortable spotting "spam mail" but when prompted to point out the spam emails the obvious phishing emails were selected and the subtler spam advertisements were missed. Due to this the distinction between non-threatening spam emails and potentially dangerous phishing emails should be designated within the application.

Sprint 4: updating the GUI

Once testing and discussion was complete the next step was to use the information and understanding of the tests to describe the objects within the app. Every counter gained a description of its functionality and other things in relation to it.

Two buttons were also added, functions that would describe to the user how things operated as well as suggest improvements they could make themselves to their own security. These functions were inserted primarily to discuss things that the application could not do in its current state of development but would be useful to a user, it includes links to step by step guides of how to achieve what is suggested as well as descriptive breakdowns of what to look for exactly.

As this application is designed for people of all ages the GUI has also been set up to enlarge the text at the press of a button, this resets upon reloading the page but allows all contents including headers, email subject, email body and descriptions to be enlarged.

Sprint 4: Login access

To ensure the program can operate efficiently it would require a login screen. However, in its current operational state due to time constraints and the necessity to ensure that passwords would need to be hashed and stored server side, bringing its own issues of user safety. It has currently been left in a state where the application opens to the testing account. If this were to be continued implementation of either a server-side database to contain user logins or some method to ensure that user login data is passed without any threat of leftover information being kept on the user's system. The system would also need to be registered as an official Gmail application and retrieve certification as would also be the case with Yahoo.

Similarly, by continuing the project including a simple method that changes the server accessed would also be an option, allowing accounts from not only Gmail but also Outlook, Yahoo and potentially any other email server. Alternatively, as discussed in Further Development it may be viable to set the application up as its own email service, though this would require more resources and secure authentication to do.

Conclusions

Justification

In order to understand if this project has been successful we must go back to the previous questions asked. These being;

- What can we do to assist all users in understanding the threat posed?
- How do we go about displaying the threat posed?
- Will informing the user reduce the success rate of these malicious emails?

The first question is easy to answer, from the research done it can be understood that in order to ensure all users understand the threats posed to them there must be something actively informing them of what is going on. The project developed has the capacity to display emails in a way that few lay users get to see them in as well as display statistics and information that a user rarely gets to see as well as providing simple descriptions of what they do. Not everything is displayed to the user, but this brings up the second question. From the testing done no user had any particular difficulty understanding the information told them, once it had been explained or annotated to them they can understand its functionality and gain benefit from it.

The third question is still a difficult one to answer, as discussed in the testing section some users overlooked a subtler threat in favour of the bigger, more obvious ones but here in lies the issue with any system of this nature. No system will ever have the capacity to flag 100% of spam, phishing and malicious content, just as these applications continue to evolve so will its target, if the application can allow a user to flag even a few emails that they wouldn't have otherwise then it is a worthwhile venture.

This brings up another question however, is this projects application successful in its endeavour? The testing shows that some users did not find use out of the program, which is to be expected on an account that has been pre-screened for spam email, but even if during testing a user hadn't found any emails the fact that users may feel more confident in looking themselves by proxy means that the operation of the project is worthwhile.

Further development

It should also be discussed the things that could have been done if more resources or time were available. These things are important though at least some form of counterpart has been attained to ensure that as little as possible is missed. The layout of phishing emails, especially spear phishing, are often designed in such a way to look enough like a company's design to go unnoticed by the lay user. This would have been a difficult thing to detect as not only are the layouts of emails easily able to be slightly different if one aspect is changed but also companies themselves change their layout designs regularly and accounting for this for large numbers of companies would not be worth the time put in. The workaround here is that the email layout will be shown to the user and if an email is a phishing email it will most likely hit at least a few other checkpoints before getting to the user. Embedded audio and video is also a concern, something that can easily link back to a site through the email without the user noticing. This issue is covered via the link flags as any video that links back to the original site will be picked up and shown to the user.

Due to its complexity the detection of spear phishing would also be an option in the future, this could be done theoretically similarly to the spam word algorithm but the items within the database would be pulled from common words found in the user's emails. The difficulty faced here would be clearing out all neutral words as well as distinguishing between a word that occurs a lot because of lifestyle and a word that occurs a lot threateningly. In this example a large amount of both spam and safe emails would be needed to allow any distinguishing between the two. Though it would require further research to consider its validity in the face of the lay user there is also an option to allow users to customize their own filters for spam and malware detection, the obvious risk though is that users could use this to allow threats through if they were unaware.

Continual development would allow certain functions to improve the quality of the application. If the application went live due to its server-side operability its list of spam detection methods could be regularly updated when new systems are discovered. The dataset used for spam words could also be gathered and analysed to allow a better information stream in the future as well as allowing a system to detect new spam words naturally, though this would face the exact issues as the above spear phishing scenario.

If this project had a large amount of resources a way to fix many of the present issues would be to create the system as its own email server complete with full android, IOS and computer support. This method would allow a far more detailed, in depth analysis as well as a more global filtering system. For instance, having the spam word system be globally updated by every user that uses the server, this would provide benefits of easier updating also but would need to be kept running constantly on a server with particular focus on security as if such a service became well known the chance of attacks from malicious sources to insert exceptions would be a considerable threat to consider. Utilizing this method would also allow the confirmation of requirement F12 and F13.

Reflection

As the developer of this application a few things come to mind after its creation, firstly due to the issues between Python 3, the email library and its interaction with json library if this project were to be taken on professionally either rebuilding it in a different programming language or processing it through its own application instead of through html. The program itself runs as I had wished and expected and due to its current state of development continuation and development of the current detection paths would be continued. The best position that this application would go into would be direct integration with an already present email GUI such as Gmail or Outlook as this would allow for a larger test database and more resources to ensure the projects reliability.

Going back to the requirements stated before and listing through which have and have not been completed it can be stated that what the project had been set out to do has been achieved. F1 to F8 were the focal points of the project and though F8 was not met this does not retract from its current functionality. Of the non-functional requirements from N1 to N9 were the main focus and while N7 and N8 are not completed all others have been. Thus, out of the 17 requirements attempted at least 14 are at least reasonably successful.

Closing

After completion of development and research of this project it can be best surmised that the projects development could be an important step in educating users of the threats they face in a way that will not intimidate them. With further resources and time, it may be possible that an application like this could become its own unique field as it works off the back of other email servers, it does not compete with them.