Content Page

Abstract

In this report, we will analyze 2 open-source spam filters: Apache SpamAssassin and Rspamd. We run a dataset of known phishing emails against these filters and analyze the emails that are misclassified as non-spam.

We run correlation analysis and feature importance (using gradient boosting regressors, where the spam scores are the target variables and email headers are the predictor variables) to identify email headers that the 2 open-source spam filters are vulnerable to.

Lastly, we craft an test email using these email headers and test it out on online email services such as Gmail and Hotmail.

Introduction

Spam filters are employed to filter all email traffic and determine if the contents are safe or malicious. If the filter has determined the content to be unsafe, the incoming email will either be blocked or filtered to the spam inbox.

The filters can work using a variety of ways. One such way is by whitelisting or blacklisting. Another way is rule-based scanning of email headers. The last way is by using machine-learning based techniques. Often, modern spam filters would employ all 3 methods to correctly classify spam emails.

In this report, we will be focusing on 2 email filters: Apache SpamAssassin [1] and Rspam [2]. These are 2 widely used, open-source spam filters. We will be using these spam filters with their default configurations, rules and without any extra plugins.

Our analysis will focus on the various email headers and resulting spam scores obtained from the 2 email filters. We will be using an online email dataset of phishing data [3] in our analysis.

From the phishing email dataset, we will identify the relevant headers of emails that are wrongly classified as non-spam by the filters. Subsequently, we will craft another email which makes use of these headers and test it out on online email services.

The relevant codes and processed datasets can be found in the accompanying folder.

Methods

Dataset Used

For the analysis of spam filters, we will be using Nazario Jose's phishing email dataset [3]. We will be using the 2018, 2019 and 2020 phishing email datasets. In total, we would have around 600 emails to test with. All of these emails are phishing / spam emails and have been manually classified by the author himself.

Each email dataset is available as a single text file. A sample email is as follows.

```
From jose@monkey.org Thu Jan  9 09:06:59 2020 +0000
Return-Path: admin@monkey.org
Delivered-To: jose@monkey.org
X-FDA: 76357516158.40.toes64_772c40c07f233
X-Spam-Summary:
10,1,0,e081f950fd44864a,d41d8cd98f00b204,admin@monkey.org,:,RULES_HIT:41:46:72:150:153:355:375:379:495:800:962:967:973:983:988:989:1021:11
33:1189:1208:1224:1260:1263:1311:1313:1314:1345:1381:1431:1433:1434:1436:1513:1515:1516:1517:1521:1534:1542:1559:1571:1588:1589:1593:1594:
1710:1711:1714:1719:1730:1747:1777:1792:2194:2198:2199:2200:2393:2525:2527:2560:2563:2610:2682:2685:2771:2828:2859:2933:2937:2939:2942:294
5:2947:2951:2954:3022:3038:3138:3139:3140:3141:3142:3585:3586:3876:3877:3934:3936:3938:3941:3944:3947:3950:3953:3956:3959:4321:5007:6114:6
117:6119:6238:6261:6300:6642:6650:6669:6671:6678:7809:8599:8603:8957:9025:9388:10004:10346:11473:11537:11638:11639:11658:11914:11984:12043
:12297:12438:12522:12555:12740:12895:12955:12986:13007:13139:13255:13439:14093:14096:14149:14181:14721:14877:18000:21080:21433:21436:21451
:21483:21524:21554:21627:21819,0,RBL:185.7.76.33:@monkey.org:.lbl8.mailshell.net-62.14.175.100
64.201.201.201,CacheIP:none,Bayesian:0.5,0.5,0.5,Netcheck:n
X-HE-Tag: toes64_772c40c07f233
X-Filterd-Recvd-Size: 3680
Received: from mail.otea.com (mail.otea.com [185.7.76.33])
          by imf20.b.hostedemail.com (Postfix) with ESMTP
          for <jose@monkey.org>; Thu,  9 Jan 2020 09:06:58 +0000 (UTC)
Received: (qmail 6732 invoked by uid 89); 9 Jan 2020 07:20:49 -0000
Received: from unknown (HELO freenex.com) (postmaster@otea.com@103.133.109.78)
  by mail.otea.com with ESMTPA; 9 Jan 2020 07:20:48 -0000
From: "monkey.org" <admin@monkey.org>
To: jose@monkey.org
Subject: jose@monkey.org =?UTF-8?B?MjRIUlMg7J207KCE7JeQIO2ZleyduO2VmOyngCDslYrsnLzrqbQg6rOE7KCV7J20IO2PkOyHhOuQqeuLiOuLpCA=?=
Date: 08 Jan 2020 22:59:26 -0800
Message-ID: <20200108225926.05D9BB7F2F15698A@monkey.org>
MIME-Version: 1.0
Content-Type: text/html;
          charset="iso-8859-1"
Content-Transfer-Encoding: quoted-printable
Status: RO
X-Status:
X-Keywords:
X-UID: 3

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.=
w3.org/TR/html4/loose.dtd">

<HTML><HEAD>
<META name=3DGENERATOR content=3D"MSHTML 11.00.9600.19377"></HEAD>
<body style=3D"MARGIN: 0.5em">
<P> </P>
<DIV>
<DIV style=3D"FONT-SIZE: 11px; MAX-WIDTH: 780px; BORDER-TOP: rgb(239,233,23=
3) 1px solid; FONT-FAMILY: Verdana,Arial,Helvetica,sans-serif; BORDER-RIGHT=
: rgb(239,233,233) 1px solid; BORDER-BOTTOM: rgb(239,233,233) 1px solid; CO=
LOR: rgb(51,51,51); PADDING-BOTTOM: 2px; PADDING-TOP: 2px; PADDING-LEFT: 2p=
x; BORDER-LEFT: rgb(239,233,233) 1px solid; PADDING-RIGHT: 2px">&#52828;&#5=
0528;&#54616;&#45716; &#49324;&#50857;&#51088;<BR><BR>
&#52572;&#44540; &#49436;&#48260; &#50629;&#44536;&#47112;&#51060;&#46300;&=
#47196; &#51064;&#54644; &#51060;&#47700;&#51068; &#44228;&#51221; (jose@mo=
nkey.org)&#51012; &#45796;&#49884; &#54869;&#51064;&#54644;&#50556;&#54633;=
&#45768;&#45796;. <BR>&#51060;&#47700;&#51068;&#51060; &#45803;&#55176=
;&#51648; &#50506;&#46020;&#47197; 48 &#49884;&#44036; &#51060;&#45236;&#50=
640; &#54869;&#51064;&#54616;&#49901;&#49884;&#50724;.  <BR> =
;=20
<table style=3D"FONT-FAMILY: New" width=3D"209">
```

```
<TBODY>
<TR>
<td style=3D"FONT-SIZE: 14px; MAX-WIDTH: 820px; BORDER-TOP: rgb(0,120,215) =
1px solid; FONT-FAMILY: WP Tahoma,sans-serif; BORDER-RIGHT: rgb(0,120,215) =
1px solid; WIDTH: 163px; VERTICAL-ALIGN: middle; BORDER-BOTTOM: rgb(0,120,2=
15) 1px solid; COLOR: rgb(255,255,255); TEXT-ALIGN: center; PADDING-LEFT: 2=
0px; MARGIN: 0px; MIN-HEIGHT: 30px; BORDER-LEFT: rgb(0,120,215) 1px solid; =
LINE-HEIGHT: 20px; PADDING-RIGHT: 20px; BACKGROUND-COLOR: rgb(0,120,215)">
<A style=3D"text-decoration-line: none" href=3D"https://staima.com/new/wp-c=
ontent//k/acc0unt/komail.php?email=3Djose@monkey.org" rel=3Dnofollow target=
=3D_blank data-saferedirecturl=3D"https://www.google.com/url?q=3Dhttps://st=
aima.com/new/wp-content//k/acc0unt/komail.php?email%3D%5B%5B-Email-%5D%5D&a=
mp;source=3Dgmail&amp;ust=3D1578616770977000&amp;usg=3DAFQjCNGxp-bNDqA8mraE=
nFioRmFD5X4Suw"><FONT size=3D+0><FONT size=3D+0>&#51648;&#44552; &#54869;&#=
51064;</FONT></FONT></A></TD></TR></TBODY></TABLE><BR>
<HR>
&#51060;&#44163;&#51008; jose@monkey.org &#50640; &#51473;&#50836;&#54633;&=
#45768;&#45796;.</DIV>
<DIV style=3D"FONT-SIZE: 11px; MAX-WIDTH: 780px; BORDER-TOP: rgb(239,233,23=
3) 1px solid; FONT-FAMILY: Verdana,Arial,Helvetica,sans-serif; BORDER-RIGHT=
: rgb(239,233,233) 1px solid; BORDER-BOTTOM: rgb(239,233,233) 1px solid; CO=
LOR: rgb(51,51,51); PADDING-BOTTOM: 2px; PADDING-TOP: 2px; PADDING-LEFT: 2p=
x; BORDER-LEFT: rgb(239,233,233) 1px solid; PADDING-RIGHT: 2px"><SPAN style=
=3D"COLOR: rgb(110,120,139)"><FONT size=3D+0><FONT size=3D+0>
&#48372;&#45240; &#49324;&#46988; : &#49436;&#48260; &#44288;&#47532;&#5108=
8;</FONT></FONT></SPAN></DIV></DIV></BODY></HTML>
```

Each email has their own respective headers. The headers are usually located at the top of the email.

The Python email library [4] will be used to convert the text into email objects. Following which, the headers present in each email will be recorded [5] before classifying the email via the spam filters.

Spam Filters Used

Our analysis will be done on 2 open-source spam filters: Apache SpamAssassin [1] and Rspamd [2]. These 2 filters will be installed on an Ubuntu VM and can be executed as follows.

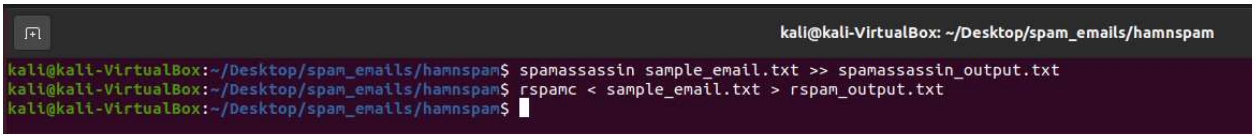

Figure 1. Using the Command Line Interface (CLI) for SpamAssassin and Rspamd

spamassassin_output.txt

```
From jose@monkey.org Thu Jan  2 18:11:18 2020 +0000
Return-Path: bana.massimo@siciabitare.com
X-Spam-Checker-Version: SpamAssassin 3.4.4 (2020-01-24) on kali-VirtualBox
X-Spam-Level: *
X-Spam-Status: No, score=1.4 required=5.0 tests=HTML_MESSAGE,MIME_HTML_ONLY,
        RDNS_NONE,SPF_HELO_NONE,SPF_NONE autolearn=no autolearn_force=no
        version=3.4.4
Delivered-To: jose@monkey.org
X-FDA: 76333486236.39.bean05_6298de60e9e36
X-HE-Tag: bean05_6298de60e9e36
X-Filterd-Recvd-Size: 8138
Received: from siciabitare.com (unknown [104.216.216.103])
        by imf18.b.hostedemail.com (Postfix) with ESMTP
        for <jose@monkey.org>; Thu,  2 Jan 2020 18:11:17 +0000 (UTC)
From: Wellsfargo Update <bana.massimo@siciabitare.com>
To: jose@monkey.org
Subject: Your Wellsfargo Account and Card are Temporarily Locked .
Date: 02 Jan 2020 19:11:16 +0100
Message-ID: <20200102191116.C8427573F77B8618@siciabitare.com>
MIME-Version: 1.0
Content-Type: text/html;
        charset="iso-8859-1"
Content-Transfer-Encoding: quoted-printable
Status: RO
X-Status:
X-Keywords:
X-UID: 1
```

Above is a sample output from SpamAssassin. SpamAssassin will add 3 new headers to the email file. They are:

1. X-Spam-Checker-Version: Version of SpamAssassin used.
2. X-Spam-Level: Spam level indicated by asterisk (*). The more asterisk, the more likely to be spam.
3. X-Spam-Status: Numerical score given based on the rules used.

In this particular email, the following SpamAssassin's rules were triggered. This had resulted in a score of 1.4 being assigned.

1. HTML_MESSAGE
2. MIME_HTML_ONLY
3. RDNS_NONE
4. SPF_HELLO_NONE
5. SPF_NONE

SpamAssassin's rule settings can be found in the '*/etc/spamassassin/local.cf*' file. The config file used for this instance of SpamAssassin can be found in Appendix C.

The current threshold score is set at 5.0. If an email has a score more than 5.0, it will be classified as spam by SpamAssassin.

<u>rspam_output.txt</u>

```
Results for file: stdin (5.687 seconds)
[Metric: default]
Action: greylist
Spam: false
Score: 5.79 / 15.00
Symbol: ARC_NA (0.00)
Symbol: ASN (0.00)[asn:40676, ipnet:104.216.216.0/24, country:US]
Symbol: AUTH_NA (1.00)
Symbol: DATE_IN_PAST (1.00)
Symbol: DMARC_NA (0.00)[siciabitare.com]
Symbol: FROM_EQ_ENVFROM (0.00)
Symbol: FROM_HAS_DN (0.00)
Symbol: HAS_DATA_URI (0.00)
Symbol: HFILTER_HOSTNAME_UNKNOWN (2.50)
Symbol: MID_RHS_MATCH_FROM (0.00)
Symbol: MIME_HTML_ONLY (0.20)
Symbol: MIME_TRACE (0.00)[0:~]
Symbol: ONCE_RECEIVED (0.10)
Symbol: PHISHING (0.89)[wellsfargo->usa.s3-website-us-east-1.amazonaws]
Symbol: PREVIOUSLY_DELIVERED (0.00)[jose@monkey.org]
Symbol: RBL_SARBL_BAD_FAIL (0.00)[query timed out]
Symbol: RCPT_COUNT_ONE (0.00)[1]
Symbol: RCVD_COUNT_ONE (0.00)[1]
Symbol: RCVD_NO_TLS_LAST (0.10)
Symbol: R_DKIM_NA (0.00)
Symbol: R_SPF_NA (0.00)
Symbol: TO_DN_NONE (0.00)
Message-ID: 20200102191116.C8427573F77B8618@siciabitare.com
Urls: ["wellsfargo.usa.s3-website-us-east-1.amazonaws.com","wellsfargo.com"]
```

Rspamd's output is slightly different. Instead of appending spam headers to the email, Rspamd will generate a report for the given email. In this example, the above symbols were found in the email. This had resulted in a score of 5.79 / 15.00

Rspamd's configuration settings can be found in '*/etc/rspamd/actions.conf*'. The config file used for this instance of Rspam can be found in Appendix D.

The current threshold score is set at 15.00. If an email has a score more than 15.00, it will be classified as spam by Rspamd.

Spam Results Collection

From the corpus of phishing emails [1], we can automate the process needed to get the spam filter results. We make use of Python's email library and the Pandas library (for dataframes).

Using Python, we created a script which does the following:

1. Extract individual emails from Jose's text collection of emails
2. Records the headers present in the email being tested
3. Sends the email to the spam filter
4. Records the spam filter's results for the email being tested
5. Saves the results to a csv file

Below is the Python function used to split and obtain email objects from the phishing email corpus.

```python
# Helper function to load Jose's phishing email file (by year)
# Returns a list of email objects
# Eg filename = 'phishing-2020'
def load_jose_dataset(filename):

    # Read raw email lines
    email_list = []
    with open(filename, 'r', encoding="utf8") as f:
        current_email = []
        for line in f:

            # Start of a new email
            if line.startswith("From jose@monkey.org"):

                # Store previous email
                email_list.append(current_email)

                # Start a new email
                current_email = []
                current_email.append(line)
            else:

                # continue storing current email
                current_email.append(line)

    # Convert raw emails to Python email object
    # Parse email data into an email object
    email_obj_list = []
    for single_email in email_list:

        # Write content to disk
        with open('email.txt', 'w') as f:
            for item in single_email:
                f.write("%s" % item)

        # Read in content to email object
        with open('email.txt', 'rb') as f:
            email_obj = email.parser.BytesParser(policy = email.policy.default).parse(f)

        email_obj_list.append(email_obj)
```

```
    # Clean up
    os.remove('email.txt')

    return email_obj_list
```

Each email is identified by the line that starts with the "*From jose@monkey.org*" string. This is an appropriate way to identify an email as in the author's readme [6], he states that all of the phishing emails are sent to his personal inbox.

To convert the email text to a Python email object, the email text is written to disk and imported again using Python's *email.parser.BytesParser()* class.

Below is the code used to extract all possible headers from the list of emails.

```
# Pre-process and obtain all possible headers in email dataset
headers = set()
all_emails = jose_2018_emails + jose_2019_emails + jose_2020_emails
for single_email in all_emails:
    try:
        email_items = single_email.items()
        for tuple_obj in email_items:
            headers.add(tuple_obj[0])
    except Exception as e:
        pass
```

Below is the code used to extract header counts from a single email.

```
# Returns counts of headers as a dictionary
def return_header_counts_as_dict(email_obj, header_set):
    header_dict = dict.fromkeys(header_set, 0)

    # Extract email headers from single email
    email_headers = None
    try:
        email_headers = [x[0] for x in email_obj.items()]
    except Exception as e:
        return header_dict

    # Increment counts
    for email_header in email_headers:
        if email_header in header_dict:
            header_dict[email_header] += 1
        else:
            pass

    return header_dict
```

Below is the code used to obtain Apache SpamAssassin scores. The email object is written to file and the SpamAssassin CLI application is called to classify the email. The results are saved to an output file *email_spamassassin_score*. Subsequently, the Python script will record down the results and save it into a DataFrame. Finally, the results are saved to a csv file, along with each email's header counts.

```
# Processes the jose dataset, sends to spam classifier, saves the result to csv
def assasin_jose_dataset(email_list, csv_filename):
    results_df = pd.DataFrame()

    counter = 0
    for single_email in email_list:
        counter += 1

        try:

            # Save email to file
            with open('email.txt', 'w') as f:
                for item in str(single_email).splitlines():
                    f.write("%s\n" % item)


            # Send file to spamassasin
            os.system('rm email_spamassassin_score')
            os.system('spamassassin email.txt >> email_spamassassin_score')

            # Read in spam assassins scores
            star_score, spam_status, spam_score = None, None, None
            found_status, found_score = False, False
            with open('email_spamassassin_score', 'r', encoding="utf8", errors='ignore') as f:
                for line in f:
                    if 'X-Spam-Level' in line:
                        star_score = (line.split(':')[1]).strip()

                    if 'X-Spam-Status' in line:
                        spam_status = line.split(':')[1]
                        spam_status = (spam_status.split(',')[0]).strip()
                        found_status = True

                        # spam_score = line.split('score=')
                        result = re.search('score=(.*?) required=', line)
                        spam_score = result.group(1)
                        found_score = True

                    if found_score and found_status:
                        break

            # Get header counts
            header_dict = return_header_counts_as_dict(single_email, headers)

            # Store spam scores
            email_info = header_dict
            email_info['spam_status'] = spam_status
            email_info['spam_score'] = spam_score

            # Append in dataframe
            results_df = results_df.append(email_info, ignore_index=True)
```

```
            # Save to file
            if counter % 10 == 0:
                print(counter)
                results_df.to_csv(csv_filename, index = False)

        except Exception as e:
            pass
```

There is a similar function which does the same for the Rspamd spam filter.

The final scripts, which collect the counts and spam scores for Apache SpamAssassin and Rspamd, are in Appendix A and Appendix B respectively. They can also be found in the files *generate_rspam_scores.py* and *generate_spam_assassin_scores.py.*

Results

Data Analysis

After running the scripts to generate the data, we can analyze the subsequent results. The output csvs are in the files *rspamd_results.csv* and *spam_assassin_results.csv*. The file '*Correlation Analysis & Machine Learning.ipynb'* will contain the code and results obtained from the analysis.

Recall that all emails in the dataset are spam / phishing emails. Hence, we can generate an accuracy score for each spam filter.

For SpamAssassin, the accuracy achieved was 43.181%. For Rspamd, the accuracy achieved was 44.694%. Hence, at their default configurations, both filters are ineffective when tested against this dataset.

SpamAssassin

X-Authority-Analysis	x-envid	To	X-Note	X-GPG	X-Spam-Filter	Precedence	...	Iplanet-SMTP-Warning	X-Epoch	X-Lookup-Warning	X-Yahoo-Profile	X-Face-Viewer	DKIM-Signature	spam_status	spam_score	source
0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	No	1.7	jose_2020_emails
0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	No	2.6	jose_2020_emails
0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	Yes	7.1	jose_2020_emails
0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	Yes	8.1	jose_2020_emails
0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	No	4.1	jose_2020_emails

Table 1. SpamAssassin results csv obtained from Python script

From the Python script (Appendix A), we would obtain a csv file (*spam_assassin_results.csv*) with the above information. All of the column headers (except for the last 3) represent counts of headers present in the mail.

For example, for a single email record, the *DKIM-Signature* column would be 0 if no *DKIM-Signature* header was present or 1 if a *DKIM-Signature* header was present.

The last 2 columns: *spam_status* and *spam_score* are obtained from the SpamAssassin filter. The last column *source* indicates which dataset was the email obtained from.

With this data, we can do a correlation analysis and feature importance on the email headers. This will help us identify which headers are more likely to trigger a certain classification.

	Emails classified wrongly as non-spam
spam_score	1.000000
Sender	0.338195
X-OriginalArrivalTime	0.266934
X-Dropbox-Message-ID	0.236848
CC	0.236848
In-Reply-To	0.212929
References	0.212929
X-KSE-Attachment-Filter-Triggered-Rules	0.206574
X-KSE-Antivirus-Interceptor-Info	0.206574
X-KSE-ServerInfo	0.206574
X-KSE-BulkMessagesFiltering-Scan-Result	0.206574
X-KSE-Antivirus-Info	0.206574
X-KSE-Attachment-Filter-Triggered-Filters	0.206574
Message-ID	0.173592
DomainKey-Signature	0.159481
Return-Path	0.156342
Content-Transfer-Encoding	0.153145
X-imss-scan-details	0.152811
X-TMASE-SNAP-Result	0.152811
X-TM-AS-GCONF	0.152811

Table 2. Top 20 column headers correlation coefficients for emails classified as non-spam

	Emails classified correctly as spam
spam_score	1.000000
X-Sender	0.314313
Mail-Reply-To	0.314313
User-Agent	0.314313
To	0.222660
Content-Transfer-Encoding	0.213491
X-MimeOLE	0.204535
X-PHP-Script	0.197112
X-MSMail-Priority	0.174072
X-SG-EID	0.168697
X-PHP-Originating-Script	0.168338
X-Source	0.163495
X-Source-Args	0.163495
X-Source-Dir	0.163495
X-Mailer	0.134234
X-Priority	0.130713
MIME-Version	0.126158
X-AntiAbuse	0.124629
Reply-To	0.123729
Content-type	0.119316

Table 3. Top 20 column headers correlation coefficients for emails classified as spam

From the data, we can obtain the correlation coefficients of the headers for both sets of emails with respect to the spam scores: *Correctly classified as spam* and *Incorrectly classified as non-spam*.

From Table 2, some of the headers that are significant for emails to be classified as non-spam are (ignoring X headers): CC, In-Reply-To, Message-ID, Return-Path, DomainKey-Signature.

Intuitively, this makes sense as if an email has Return-Path and DomainKey-Signature headers, there is a high chance it came from a reputable source somewhere along the chain.

Additionally, as we have the spam scores from SpamAssassin, we can fit a gradient boosting regression model [7] and obtain the feature importances. This will allow us to see features that are relevant in discriminating between the SpamAssassin's classification.

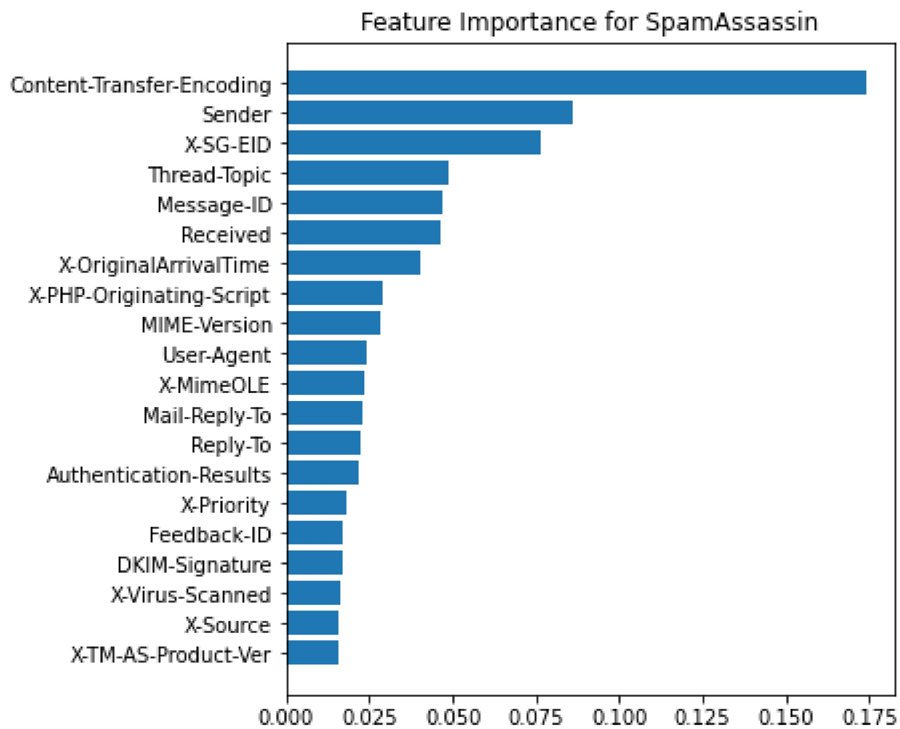

Figure 2. Feature importance from Gradient Boosting Regressor

Fitting the machine-learning model, we are able to obtain the top 20 headers. Notable headers are: MIME-Version, User-Agent, Authentication-Results, DKIM-Signature, X-Virus-Scanned.

With the above 20 headers, an attacker can manually insert those headers and get past SpamAssassin's default filters.

Rspam

X-REPORT-ABUSE-TO	X-Originating-IP	X-CanItPRO-Stream	...	X-Authentication-Warning	X-Cloudmilter-Processed	X-Antispam-Training-Forget	List-Unsubscribe	X-Mailer-RecptId	X-Lookup-Warning	X-Outgoing-Spam-Report	DKIM-Signature	spam_status	spam_score
0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	True	8.40
0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	False	5.79
0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	True	8.30
0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	True	6.30
0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	True	10.05

Table 4. Rspam results csv obtained from Python script

From the Python script (Appendix B), we would obtain a csv file (*rspamd_results.csv*) with the above data. All of the column headers (except for the last 2) represent counts of headers present in the email. For example, for a single email record, the *DKIM-Signature* column would be 0 if no *DKIM-Signature* header was present or 1 if a *DKIM-Signature* header was present.

The last 2 columns: *spam_status* and *spam_score* are obtained from the Rspamd filter.

Similar to the SpamAssassin results dataset, we can do a correlation analysis and feature importance on the headers to identify which headers are more likely to trigger a certain classification for Rspamd.

	Emails classified correctly as spam
spam_score	1.000000
X-MimeOLE	0.363561
X-MSMail-Priority	0.332385
X-Priority	0.235952
Priority	0.234193
X-PHP-Originating-Script	0.220233
Message-ID	0.205854
Importance	0.197362
Reply-To	0.194167
List-Unsubscribe	0.171176
X-Mailer	0.170235
Content-Type	0.164213
Precedence	0.159283
Content-type	0.154885
List-id	0.141841
Received	0.138407
To	0.133948
X-mailer	0.133917
X-CSA-Complaints	0.133917
Content-Language	0.117276

Table 5. Top 20 column headers correlation coefficients for emails classified as non-spam

	Emails classified wrongly as not spam
spam_score	1.000000
X-Api-Host	0.509068
Recipient-Id	0.509068
X-BounceEmailVersion	0.509068
X-Email-Rejection-Mode	0.509068
Site-Id	0.509068
X-Debug	0.509068
X-OriginalArrivalTime	0.364793
Sender	0.293727
In-Reply-To	0.198988
References	0.198988
Received	0.170741
X-AntiAbuse	0.141568
X-Get-Message-Sender-Via	0.141475
X-Dropbox-Message-ID	0.139290
CC	0.139290
X-Virus-Scanned	0.135344
X-Authenticated-Sender	0.133264
X-Source	0.123056
X-Source-Args	0.123056

Table 6. Top 20 column headers correlation coefficients for emails classified as spam

From Table 5, we can see that for emails that are correctly classified as spam by Rspamd, the notable non-X headers are Priority, Message-ID and Importance.

Whereas for emails that are incorrectly classified as not-spam (Table 6), the notable headers are X-OriginalArrivalTime, X-Virus-Scanned and so on.

Similar to the SpamAssassin dataset, we can fit a Gradient Boosting Regressor to the Rspamd spam scores and view what features are the most important.

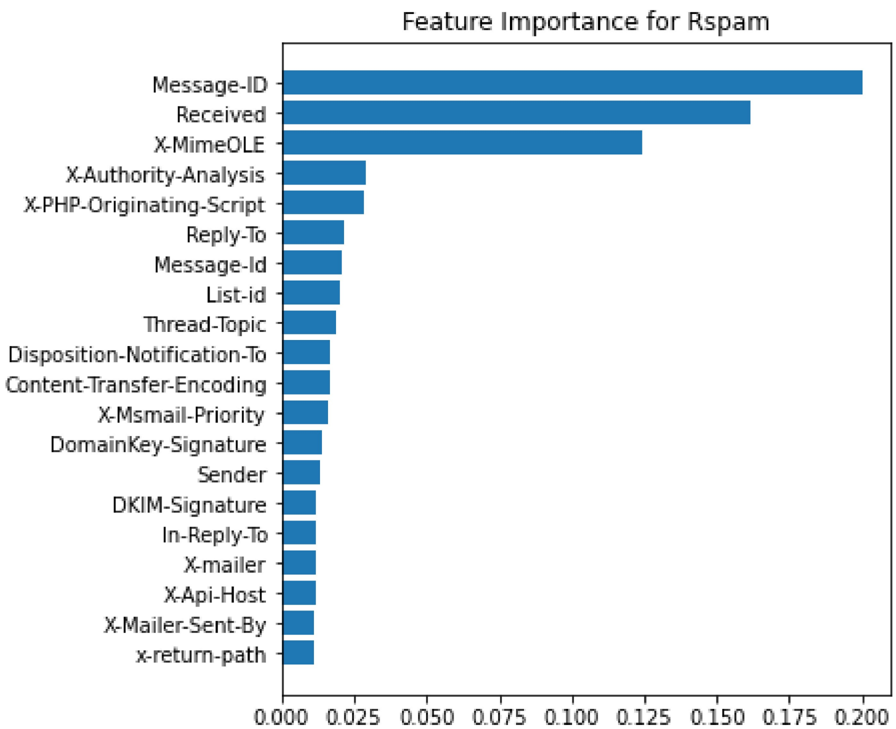

Figure 3. Feature importance for Rspamd model

From the machine learning model, the top headers are Received, Message-ID, X-MimeOLE, DKIM-Signature and so on.

Bypassing the Filters

From the top headers identified (from correlation analysis and feature importances), we were able to craft an email which is able to avoid both SpamAssassin and Rspam filters.

```
Return-Path: alseda@mat.uab.cat
Delivered-To: jose@monkey.org
X-FDA: 76362067320.58.sort78_1a3b4e716255f
X-Spam-Summary: =?utf-8?q?1=2C0=2C0=2C=2Cd41d8cd98f00b204=2Calseda=40mat=2Eu?=
 =?utf-8?b?YWIuY2F0LDosUlVMRVNfSElUOjMwMDAxOjMwMDIxOjMwMDI2OjMwMDU0OjMw?=
 =?utf-8?b?MDY5OjMwMDcwLDAsUkJMOjE1OC4xMDkuMTY4LjEzMjpAbWF0LnVhYi5jYXQ6Lmxi?=
 =?utf-8?q?l8=2Emailshell=2Enet-62=2E8=2E11=2E2_64=2E201=2E201=2E201=2CCache?=
 =?utf-8?b?SVA6bm9uZSxCYXllc2lhbjowLjUsMC41LDAuNSxOZXRjaGVjazpub25lLERvbWFp?=
 =?utf-8?q?nCache=3A0=2CMSF=3Anot_bulk=2CSPF=3Afn=2CMSBL=3A0=2CDNSBL=3Asearc?=
 =?utf-8?q?hbourne=2Ecom-sh=2Edbl=2Eurbl=2Ehostedemail=2Ecom-127=2E0=2E0=2E1?=
 =?utf-8?b?NzUsQ3VzdG9tX3J1bGVzOjA6MDowLExGdGltZTo3OCxMVUFfU1VNTUFSWTpu?=
 =?utf-8?q?one?=
X-HE-Tag: sort78_1a3b4e716255f
X-Filterd-Recvd-Size: 4790
Received: from venezia.uab.es (venezia.uab.es [158.109.168.132])
           by imf30.b.hostedemail.com (Postfix) with ESMTP
           for <jose@monkey.org>; Fri, 10 Jan 2020 15:13:00 +0000 (UTC)
Received: from venezia.uab.es ([127.0.0.1]) by venezia.uab.es (Sun Java System
 Messaging Server 6.1 HotFix 0.10 (built Jan  6 2005)) with ESMTP id
 <0Q3W000ZDATUUY00@venezia.uab.es> for jose@monkey.org; Fri, 10 Jan 2020
 15:27:30 +0100 (CET)
Received: from [144.217.117.210] by venezia.uab.es
 (Sun Java System Messaging Server 6.1 HotFix 0.10 (built Jan  6 2005))
 with ESMTP id <0Q3W00HI1AT8WSG0@venezia.uab.es> for jose@monkey.org; Fri,
 10 Jan 2020 15:27:30 +0100 (CET)
Date: Fri, 10 Jan 2020 06:27:29 -0800
From: Administrator <alseda@mat.uab.cat>
Subject: Verify ownership of your account now
To: jose@monkey.org
Message-id: <0Q3W00HJSATTWSG0@venezia.uab.es>
MIME-version: 1.0
Content-type: multipart/alternative;
 boundary="Boundary_(ID_g/KLvqtF0lbzM1IxaXcXqA)"
Status: RO
X-Status:
X-Keywords:
X-UID: 5
X-MimeOLE: Produced By Microsoft MimeOLE V6.3.9600.19203
X-SES-Outgoing: 2019.08.20-54.240.6.53
X-TM-AS-Product-Ver: IMSVA-8.2.0.1391-7.5.0.1017-20482.004
X-Authenticated-Sender: ml002.dnshigh.com: mailing@ml.ussnews.net
X-Virus-Scanned: by amavisd-new using ClamAV (18)
X-Priority: 3
X-MSMail-Priority: Normal
X-PHP-Originating-Script: 1002:class-phpmailer.php
User-Agent: Horde Application Framework 5
X-OriginalArrivalTime: 02 Jun 2017 20:23:25.0759 (UTC) FILETIME=[16B8ACF0:01D2DBDE]
X-Msmail-Priority: Low
Disposition-Notification-To: oinfo483@gmail.com
X-mailer: Cabestan DMS
DKIM-Signature: v=1; a=rsa-sha256; q=dns/txt; c=relaxed/relaxed;  d=ml.ussnews.net; s=default; h=Content-Type:MIME-Version:Message-ID:Date:
           Subject:To:From:Sender:Reply-To:Cc:Content-Transfer-Encoding:Content-ID:
Content-Description:Resent-Date:Resent-From:Resent-Sender:Resent-To:Resent-Cc
:Resent-Message-ID:In-Reply-To:References:List-Id:List-Help:List-Unsubscribe:          List-Subscribe:List-Post:List-Owner:List-Archive;
           bh=rsKw8amTFCxR3yhSjuHJ/o7QsWPzx6bY4psPWR7J3qk=; b=HFl30BSGe1F4uIC7KcZ8O6jGo9
XSu/9YTvINQgN+bimy5ck7yaDtixE0cWu1lGBP5775Cghn0a7cx/PT8ewuC884jAeCwDq+Ssag68Q
O+jePBaG5ICIgpD1S2pjUrf2UdkReur5WmneObFemOdnSxVegdb1J4e+iAG/HQ88K0vlbbQzjfsOC
O2P4sZgT4pMoghTv6slOOC55NcKwDsQRE75pps6AHLu6/6RQbdzcHBCcW4Koxn6l0DswoCMddmaBZ
h9s/z1XCQgmFMUzIL39g/HoKyciP1UPMAGXHuftqK0OqQihNj4I5XkidTKK7i1N983XmkgXvucHnb          /uGD9jMg==;
DomainKey-Signature: a=rsa-sha1; c=nofws; q=dns; s=dkim; d=ecoterracostarica.com;
b=Q+4p6nYeIX2XzJwCtey3hEkKXVDqaQztCyNnJ7YHLLAqzRval8TofCwqnu/TqaKXZ5pJ3SeFvJdh
NyKPQyj9BXFMLbHZTvInTUiixU3XLksZkCPT3tOnJgHMW2FsCX1ubmgKYO/pN6F+qFdLpasi/Yoi   f6sBUoMG//KzTRFIkNY=;

You will not see this in a MIME-aware mail reader.

--Boundary_(ID_g/KLvqtF0lbzM1IxaXcXqA)
MIME-version: 1.0
Content-type: text/plain; charset=iso-8859-1
Content-transfer-encoding: 7BIT
Content-description: Mail message body
```

```
   Update account activity

Hello jose@monkey.org,

Kindly note that all unverified and outdated E-mail account will loose their account if not verified and updated within 24hours. Kindly
follow the link below to verify and update your E-mail


   Review activity
  To opt out or change where you receive security notifications, click here.
  Thanks,
  Mailbox Administrator




--Boundary_(ID_g/KLvqtF0lbzM1IxaXcXqA)
MIME-version: 1.0
Content-type: text/html; charset=iso-8859-1
Content-transfer-encoding: 7BIT
Content-description: Mail message body

<HTML><head><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"/></head><BODY><P></P>
<P></P>
<P>
<TABLE dir=ltr>
<TBODY>
<TR>
<TD id=gmail-m_7970722707667627508gmail-m_4702478177916742601gmail-m_-5275758674431469700i2 style='FONT-FAMILY: "segoe ui light", "segoe
ui", "helvetica neue medium", arial, sans-serif; PADDING-BOTTOM: 0px; PADDING-TOP: 0px; PADDING-LEFT: 0px; PADDING-RIGHT: 0px'><SPAN
style="FONT-SIZE: 41px; COLOR: rgb(38,114,236)">Update account activity<BR></SPAN><BR><B><FONT size=4>Hello
jose@monkey.org,</FONT></B><BR><FONT color=#444444><BR>Kindly note that all unverified and outdated E-mail account will loose their
account if not verified and updated within 24hours. Kindly follow the link below to verify and update your E-mail</FONT><BR></TD></TR>
<TR>
<TD style='FONT-SIZE: 14px; FONT-FAMILY: "segoe ui", tahoma, verdana, arial, sans-serif; COLOR: rgb(42,42,42); PADDING-BOTTOM: 0px;
PADDING-TOP: 25px; PADDING-LEFT: 0px; PADDING-RIGHT: 0px'>
<TABLE cellSpacing=0 border=0>
<TBODY>
<TR>
<TD style="MIN-WIDTH: 50px; PADDING-BOTTOM: 5px; PADDING-TOP: 5px; PADDING-LEFT: 20px; PADDING-RIGHT: 20px; BACKGROUND-COLOR:
rgb(38,114,236)" bgColor=#2672ec><A id=gmail-m_7970722707667627508gmail-m_4702478177916742601gmail-m_-5275758674431469700i11
style='FONT-FAMILY: "segoe ui semibold", "segoe ui bold", "segoe ui", "helvetica neue medium", arial, sans-serif; FONT-WEIGHT: 600; COLOR:
rgb(255,255,255); TEXT-ALIGN: center; LETTER-SPACING: 0.02em'
href="https://www.searchbourne.com/wp-includes/css/bb/report/_rmi/session/?i=i&amp;0=jose@monkey.org" target=_blank>Review
activity</A></TD></TR></TBODY></TABLE></TD></TR>
<TR>
<TD id=gmail-m_7970722707667627508gmail-m_4702478177916742601gmail-m_-5275758674431469700i12 style='FONT-SIZE: 14px; FONT-FAMILY: "segoe
ui", tahoma, verdana, arial, sans-serif; COLOR: rgb(42,42,42); PADDING-BOTTOM: 0px; PADDING-TOP: 25px; PADDING-LEFT: 0px; PADDING-RIGHT:
0px'>To opt out or change where you receive security notifications, <A
id=gmail-m_7970722707667627508gmail-m_4702478177916742601gmail-m_-5275758674431469700iLink5 style="COLOR: rgb(38,114,236)"
href="https://www.searchbourne.com/wp-includes/css/bb/report/_rmi/session/?i=i&amp;0=jose@monkey.org" target=_blank>click
here</A>.</TD></TR>
<TR>
<TD id=gmail-m_7970722707667627508gmail-m_4702478177916742601gmail-m_-5275758674431469700i13 style='FONT-SIZE: 14px; FONT-FAMILY: "segoe
ui", tahoma, verdana, arial, sans-serif; COLOR: rgb(42,42,42); PADDING-BOTTOM: 0px; PADDING-TOP: 25px; PADDING-LEFT: 0px; PADDING-RIGHT:
0px'>Thanks,</TD></TR>
<TR>
<TD id=gmail-m_7970722707667627508gmail-m_4702478177916742601gmail-m_-5275758674431469700i14 style='FONT-SIZE: 14px; FONT-FAMILY: "segoe
ui", tahoma, verdana, arial, sans-serif; COLOR: rgb(42,42,42); PADDING-BOTTOM: 0px; PADDING-TOP: 0px; PADDING-LEFT: 0px; PADDING-RIGHT:
0px'>Mailbox Administrator</TD></TR></TBODY></TABLE></P>
<P><BR></P></BODY></HTML>

--Boundary_(ID_g/KLvqtF0lbzM1IxaXcXqA)--
```

The SpamAssassin results are as follows:

```
X-Spam-Checker-Version: SpamAssassin 3.4.4 (2020-01-24) on kali-VirtualBox
X-Spam-Level: ****
X-Spam-Status: No, score=4.8 required=5.0 tests=DKIM_INVALID,DKIM_SIGNED,
        GB_FREEMAIL_DISPTO,GB_FREEMAIL_DISPTO_NOTFREEM,HTML_MESSAGE,
        RCVD_IN_DNSWL_MED,RCVD_IN_MSPIKE_H3,RCVD_IN_MSPIKE_WL,SPF_HELO_PASS,
        SPF_NONE,TVD_PH_BODY_ACCOUNTS_PRE,URI_PHISH,URI_WP_HACKED_2
        autolearn=no autolearn_force=no version=3.4.4
```

The Rspam results are as follows:

```
Results for file: stdin (1.436 seconds)
[Metric: default]
Action: no action
Spam: false
Score: 3.41 / 15.00
Symbol: ARC_NA (0.00)
Symbol: ASN (0.00)[asn:13041, ipnet:158.109.0.0/16, country:ES]
Symbol: DATE_IN_PAST (1.00)
Symbol: DKIM_TRACE (0.00)[ml.ussnews.net:~]
Symbol: DMARC_POLICY_SOFTFAIL (0.10)[uab.cat : No valid SPF, none]
Symbol: FROM_EQ_ENVFROM (0.00)
Symbol: FROM_HAS_DN (0.00)
Symbol: HAS_WP_URI (0.00)
Symbol: HAS_X_AS (0.00)[mailing@ml.ussnews.net]
Symbol: HAS_X_POS (0.00)
Symbol: HAS_X_PRIO_THREE (0.00)[3]
Symbol: HEADER_FORGED_MDN (2.00)
Symbol: MIME_GOOD (-0.10)[multipart/alternative, text/plain]
Symbol: MIME_TRACE (0.00)[0:+, 1:+, 2:~]
Symbol: PREVIOUSLY_DELIVERED (0.00)[jose@monkey.org]
Symbol: RCPT_COUNT_ONE (0.00)[1]
Symbol: RCVD_COUNT_THREE (0.00)[3]
Symbol: RCVD_IN_DNSWL_MED (-0.20)[132.168.109.158.list.dnswl.org : 127.0.11.2]
Symbol: RCVD_NO_TLS_LAST (0.10)
Symbol: RWL_MAILSPIKE_GOOD (0.00)[132.168.109.158.rep.mailspike.net : 127.0.0.18]
Symbol: R_DKIM_PERMFAIL (0.00)[ml.ussnews.net:s=default]
Symbol: R_SPF_NA (0.00)
Symbol: TO_DN_NONE (0.00)
Symbol: XM_CASE (0.50)
Symbol: XM_UA_NO_VERSION (0.01)
Message-ID: 0Q3W00HJSATTWSG0@venezia.uab.es
Urls: ["www.searchbourne.com"]
Emails: ["jose@monkey.org"]
```

By adding the appropriate headers, we are able to reduce the spam filter scores such that it is below the threshold. Hence, this email is not classified as spam.

Testing with other online services

Using the SMTP library, we sent the above email out to a test email on Gmail and Hotmail. Fortunately, the email was never received in the inbox or the spam inbox.

The main reason could be that despite the DKIM headers being present, the domains specified were invalid. Furthermore, Google and Hotmail services would be using much more complex email filters compared to the simple ones implemented via SpamAssassin and Rsap defaultoptions.

Discussion & Conclusion

In this report, we tested a dataset of approximately 600 phishing emails against the spam filters SpamAssassin and Rspamd. Subsequently, we analyzed the column headers that are vital in resulting in different classifications for both email filters.

Using the column headers, we crafted an email that could fool both SpamAssassin and Rspamd email filters. However, when the email was sent to online services (Gmail and Hotmail), the online filters were able to detect the phishing email.

Hence, it is vital for email administrators to not use default configurations of spam filters on their servers. This is as the default configurations are inadequate in protecting against spam emails. An alternative would be to use enterprise email filters (such as Google's filters) which are better equipped to detect spam emails.

Ideally, email administrators should implement stronger detection mechanisms and test it against the same dataset that we used. Their test accuracies should be more better than what was obtained with the default settings (43.181% for SpamAssassin, 44.694% for Rspamd).

References

[1] Apache SpamAssassin. (2021). The #1 Enterprise Open-Source Spam Filter. Available from http://spamassassin.apache.org/

[2] Rpsamd. (Aug 19 2021). Fast, free and open-source spam filtering system. Available from https://rspamd.com/

[3] Nazario, J. (2005). The online phishing corpus, Available from http://monkey.org/~jose/wiki/doku.php

[4] Python email library. (2021). Available from https://docs.python.org/3/library/email.html

[5] Mike. (Dec 7 2011). Extract email headers in Python. Available from https://stackoverflow.com/questions/8424317/extract-just-email-headers-in-python

[6] Nazario, Jose. (2020). README.txt. Available from https://monkey.org/~jose/phishing/README.txt

[7] sklearn. Gradient boosting regressor. (2021). Available from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html

Appendix

A - Python Script to collect data for SpamAssassin

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Oct 20 15:24:22 2021
"""

import os
import pandas as pd
import numpy as np
import email
import email.policy
import re
from email.parser import HeaderParser

# Change to email directory
email_folder = r"/home/kali/Desktop/spam_emails/hamnspam"
os.chdir(email_folder)

# Returns counts of headers as a dictionary
def return_header_counts_as_dict(email_obj, header_set):
    header_dict = dict.fromkeys(header_set, 0)

    # Extract email headers from single email
    email_headers = None
    try:
        email_headers = [x[0] for x in email_obj.items()]
    except Exception as e:
        return header_dict

    # Increment counts
    for email_header in email_headers:
        if email_header in header_dict:
            header_dict[email_header] += 1
        else:
            pass

    return header_dict

# Helper function to load Jose's phishing email file (by year)
# Returns a list of email objects
# Eg filename = 'phishing-2020'
def load_jose_dataset(filename):

    # Read raw email lines
    email_list = []
    with open(filename, 'r', encoding="utf8") as f:
        current_email = []
        for line in f:

            # Start of a new email
            if line.startswith("From jose@monkey.org"):
```

```
                # Store previous email
                email_list.append(current_email)

                # Start a new email
                current_email = []
                current_email.append(line)
            else:

                # continue storing current email
                current_email.append(line)

    # Convert raw emails to Python email object
    # Parse email data into an email object
    email_obj_list = []
    for single_email in email_list:

        # Write content to disk
        with open('email.txt', 'w') as f:
            for item in single_email:
                f.write("%s" % item)

        # Read in content to email object
        with open('email.txt', 'rb') as f:
            email_obj = email.parser.BytesParser(policy = email.policy.default).parse(f)

        email_obj_list.append(email_obj)

    # Clean up
    os.remove('email.txt')

    return email_obj_list

# Processes the jose dataset, sends to spam filter, saves the result to csv
def assasin_jose_dataset(email_list, csv_filename):
    results_df = pd.DataFrame()

    counter = 0
    for single_email in email_list:
        counter += 1

        try:

            # Save email to file
            with open('email.txt', 'w') as f:
                for item in str(single_email).splitlines():
                    f.write("%s\n" % item)

            # Get header counts
            header_dict = return_header_counts_as_dict(single_email, headers)

            # Send file to spamassassin
            os.system('rm email_spamassassin_score')
            os.system('spamassassin email.txt >> email_spamassassin_score')

            # Read in spam assassins scores
            star_score, spam_status, spam_score = None, None, None
            found_status, found_score = False, False
            with open('email_spamassassin_score', 'r', encoding="utf8", errors='ignore') as f:
                for line in f:
```

```
                    if 'X-Spam-Level' in line:
                        star_score = (line.split(':')[1]).strip()

                    if 'X-Spam-Status' in line:
                        spam_status = line.split(':')[1]
                        spam_status = (spam_status.split(',')[0]).strip()
                        found_status = True

                        # spam_score = line.split('score=')
                        result = re.search('score=(.*?) required=', line)
                        spam_score = result.group(1)
                        found_score = True

                    if found_score and found_status:
                        break

            # Store spam scores
            email_info = header_dict
            email_info['spam_status'] = spam_status
            email_info['spam_score'] = spam_score

            # Append in dataframe
            results_df = results_df.append(email_info, ignore_index=True)

            # Save to file
            if counter % 10 == 0:
                print(counter)
                results_df.to_csv(csv_filename, index = False)

        except Exception as e:
            pass

# Read in the 3 Jose's datasets
jose_2020_emails = load_jose_dataset('phishing-2020')
jose_2019_emails = load_jose_dataset('phishing-2019')
jose_2018_emails = load_jose_dataset('phishing-2018')

# Pre-process and obtain all possible headers in email dataset
headers = set()
all_emails = jose_2018_emails + jose_2019_emails + jose_2020_emails
for single_email in all_emails:
    try:
        email_items = single_email.items()
        for tuple_obj in email_items:
            headers.add(tuple_obj[0])
    except Exception as e:
        pass

# Get assassin spam scores
assasin_jose_dataset(jose_2020_emails, 'jose_2020.csv')
assasin_jose_dataset(jose_2019_emails, 'jose_2019.csv')
assasin_jose_dataset(jose_2019_emails, 'jose_2019.csv')
```

B - Python Script to collect data for Rspamd

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Oct 20 23:25:35 2021
"""

import os
import pandas as pd
import numpy as np
import email
import email.policy
import re
from email.parser import HeaderParser

# Change to email directory with Jose's email files
email_folder = r"/home/kali/Desktop/spam_emails/hamnspam"
os.chdir(email_folder)

# Returns counts of headers as a dictionary
def return_header_counts_as_dict(email_obj, header_set):
    header_dict = dict.fromkeys(header_set, 0)

    # Extract email headers from single email
    email_headers = None
    try:
        email_headers = [x[0] for x in email_obj.items()]
    except Exception as e:
        return header_dict

    # Increment counts
    for email_header in email_headers:
        if email_header in header_dict:
            header_dict[email_header] += 1
        else:
            pass

    return header_dict

# Helper function to load Jose's phishing email file (by year)
# Returns a list of email objects
# Eg filename = 'phishing-2020'
def load_jose_dataset(filename):

    # Read raw email lines
    email_list = []
    with open(filename, 'r', encoding="utf8") as f:
        current_email = []
        for line in f:

            # Start of a new email
            if line.startswith("From jose@monkey.org"):

                # Store previous email
                email_list.append(current_email)

                # Start a new email
```

```
                current_email = []
                current_email.append(line)
            else:

                # continue storing current email
                current_email.append(line)

    # Convert raw emails to Python email object
    # Parse email data into an email object
    email_obj_list = []
    for single_email in email_list:

        # Write content to disk
        with open('email.txt', 'w') as f:
            for item in single_email:
                f.write("%s" % item)

        # Read in content to email object
        with open('email.txt', 'rb') as f:
            email_obj = email.parser.BytesParser(policy = email.policy.default).parse(f)

        email_obj_list.append(email_obj)

    # Clean up
    os.remove('email.txt')

    return email_obj_list

# Processes the jose dataset, sends to rspam, saves the result to csv
def rspamc_jose_dataset(email_list, csv_filename):
    results_df = pd.DataFrame()

    counter = 0
    for single_email in email_list:
        counter += 1

        try:

            # Save email to file
            with open('email.txt', 'w') as f:
                for item in str(single_email).splitlines():
                    f.write("%s\n" % item)

            # Get header counts
            header_dict = return_header_counts_as_dict(single_email, headers)

            # Send file to rspam
            os.system('rm output.txt')
            os.system('rspamc < email.txt > output.txt')

            # Extract spam scores
            spam_status = None
            spam_score = None
            with open('output.txt', 'r', encoding="utf8", errors='ignore') as f:
                for line in f:
                    if 'Spam:' in line:
                        spam_status = (line.split('Spam: ')[1]).strip()
                    if 'Score:' in line:
                        spam_score = line.split('Score: ')[1]
```

```
                        spam_score = spam_score.split('/')[0]
                        spam_score = spam_score.strip()

            # Store spam scores
            email_info = header_dict
            email_info['spam_status'] = spam_status
            email_info['spam_score'] = spam_score

            # Append to dataframe
            results_df = results_df.append(email_info, ignore_index=True)

            # Save to file
            if counter % 10 == 0:
                print(counter)
                results_df.to_csv(csv_filename, index = False)

        except Exception as e:
            pass


# Read in the 3 Jose's datasets
jose_2020_emails = load_jose_dataset('phishing-2020')
jose_2019_emails = load_jose_dataset('phishing-2019')
jose_2018_emails = load_jose_dataset('phishing-2018')

# Pre-process and obtain all possible headers in email dataset
headers = set()
all_emails = jose_2018_emails + jose_2019_emails + jose_2020_emails
for single_email in all_emails:
    try:
        email_items = single_email.items()
        for tuple_obj in email_items:
            headers.add(tuple_obj[0])
    except Exception as e:
        pass


# Process jose 2020 dataset
rspamc_jose_dataset(jose_2020_emails, 'jose_2020_rspam.csv')

# Process jose 2019 dataset
rspamc_jose_dataset(jose_2019_emails, 'jose_2019_rspam.csv')

# Process jose 2020 dataset
rspamc_jose_dataset(jose_2018_emails, 'jose_2018_rspam.csv')
```

C - SpamAssassin’s configuration file

```
# This is the right place to customize your installation of SpamAssassin.
#
# See 'perldoc Mail::SpamAssassin::Conf' for details of what can be
# tweaked.
#
# Only a small subset of options are listed below
#
###########################################################################

#   Add *****SPAM***** to the Subject header of spam e-mails
#
# rewrite_header Subject *****SPAM*****


#   Save spam messages as a message/rfc822 MIME attachment instead of
#   modifying the original message (0: off, 2: use text/plain instead)
#
# report_safe 1


#   Set which networks or hosts are considered 'trusted' by your mail
#   server (i.e. not spammers)
#
# trusted_networks 212.17.35.


#   Set file-locking method (flock is not safe over NFS, but is faster)
#
# lock_method flock


#   Set the threshold at which a message is considered spam (default: 5.0)
#
# required_score 5.0


#   Use Bayesian classifier (default: 1)
#
# use_bayes 1


#   Bayesian classifier auto-learning (default: 1)
#
# bayes_auto_learn 1


#   Set headers which may provide inappropriate cues to the Bayesian
#   classifier
#
# bayes_ignore_header X-Bogosity
# bayes_ignore_header X-Spam-Flag
# bayes_ignore_header X-Spam-Status


#   Whether to decode non- UTF-8 and non-ASCII textual parts and recode
#   them to UTF-8 before the text is given over to rules processing.
```

```
#
# normalize_charset 1

#   Textual body scan limit    (default: 50000)
#
#   Amount of data per email text/* mimepart, that will be run through body
#   rules.  This enables safer and faster scanning of large messages,
#   perhaps having very large textual attachments.  There should be no need
#   to change this well tested default.
#
# body_part_scan_size 50000

#   Textual rawbody data scan limit    (default: 500000)
#
#   Amount of data per email text/* mimepart, that will be run through
#   rawbody rules.
#
# rawbody_part_scan_size 500000

#   Some shortcircuiting, if the plugin is enabled
#
ifplugin Mail::SpamAssassin::Plugin::Shortcircuit
#
#   default: strongly-whitelisted mails are *really* whitelisted now, if the
#   shortcircuiting plugin is active, causing early exit to save CPU load.
#   Uncomment to turn this on
#
#   SpamAssassin tries hard not to launch DNS queries before priority -100.
#   If you want to shortcircuit without launching unneeded queries, make
#   sure such rule priority is below -100. These examples are already:
#
# shortcircuit USER_IN_WHITELIST       on
# shortcircuit USER_IN_DEF_WHITELIST   on
# shortcircuit USER_IN_ALL_SPAM_TO     on
# shortcircuit SUBJECT_IN_WHITELIST    on

#   the opposite; blacklisted mails can also save CPU
#
# shortcircuit USER_IN_BLACKLIST       on
# shortcircuit USER_IN_BLACKLIST_TO    on
# shortcircuit SUBJECT_IN_BLACKLIST    on

#   if you have taken the time to correctly specify your "trusted_networks",
#   this is another good way to save CPU
#
# shortcircuit ALL_TRUSTED             on

#   and a well-trained bayes DB can save running rules, too
#
# shortcircuit BAYES_99                spam
# shortcircuit BAYES_00                ham

endif # Mail::SpamAssassin::Plugin::Shortcircuit
```

D - Rspam’s configuration file

```
# Actions settings
# Please don't modify this file as your changes might be overwritten with
# the next update.
#
# You can modify '$LOCAL_CONFDIR/rspamd.conf.local.override' to redefine
# parameters defined on the top level
#
# You can modify '$LOCAL_CONFDIR/rspamd.conf.local' to add
# parameters defined on the top level
#
# For specific modules or configuration you can also modify
# '$LOCAL_CONFDIR/local.d/file.conf' - to add your options or rewrite defaults
# '$LOCAL_CONFDIR/override.d/file.conf' - to override the defaults
#
# See https://rspamd.com/doc/tutorials/writing_rules.html for details

actions {
    reject = 15; # Reject when reaching this score
    add_header = 6; # Add header when reaching this score
    greylist = 4; # Apply greylisting when reaching this score (will emit `soft reject action`)

    #unknown_weight = 1.0; # Enable if need to set score for all symbols implicitly
    # Each new symbol is added multiplied by gf^N, where N is the number of spammy symbols
    #grow_factor = 1.1;
    # Set rewrite subject to this value (%s is replaced by the original subject)
    #subject = "***SPAM*** %s"

    .include(try=true; priority=1; duplicate=merge) "$LOCAL_CONFDIR/local.d/actions.conf"
    .include(try=true; priority=10) "$LOCAL_CONFDIR/override.d/actions.conf"
}
```