

Contents

1	Introduction	1
2	Methods	2
2.1	Parsing Data	3
2.2	Spamassassin	4
2.3	Rspamd	8
2.4	Avoid Detection and Classification	11
2.4.1	Increasing Required Score	11
2.4.2	Lowering Individual Scores	13
2.4.3	Bypassing Email Service Providers	14
3	Results	15
3.1	Spamassassin VS Rspamd	15
3.1.1	Outputs of Not Spam	15
3.1.2	Outputs of Spam	16
3.1.3	Default Classifications	17
3.2	Triggered Rules and Scoring System	19
4	Discussion	21
4.1	Strengths	21
4.1.1	Parsing Email Datasets	21
4.1.2	Avoiding Detections	21
4.1.3	Comparison Methodology	21
4.2	Limitations	21
5	Conclusion	22

Abstract

A phishing email is effective as it essentially targets the weakest link in the cybersecurity chain - people.¹ In defending against phishing emails, spam filters are used as protection, depending on the severity of the email classified by these filters. Such filters have scanning and analysis capabilities to more accurately detect spam emails. This paper will cover 2 such filters and readers can use this paper as a reference to set up such filters in their own personal networks.















1 Introduction

This paper starts of by exploring the methodologies in the implementation of 2 different spam filters, Spamassassin² and Rspamd,³ on a set of emails extracted from Jose's phishing emails data set⁴ and elaborates on the configurations and defaults being used for both the spam filters. We will also include methods on bypassing email service providers' spam filters. In the results section, a comparison will be made between these 2 spam filters and we will discuss on the strengths and weaknesses of our implementations.

2 Methods

Before we can proceed with installation of either of the spam filters, we need to download data from Jose's phishing email data set.

Index of /~jose/phishing

Name	Last modified	Size	Description
 Parent Directory		-	
 20051114.mbox	2014-02-10 17:00	3.9M	
 README.txt	2018-07-13 17:15	844	
 phishing-2015	2018-05-15 19:18	8.4M	
 phishing-2016	2018-05-15 19:18	12M	
 phishing-2017	2018-05-15 19:18	11M	
 phishing-2018	2019-01-11 19:41	8.4M	
 phishing-2019	2020-01-03 15:08	5.0M	
 phishing-2020	2021-03-08 15:59	5.6M	
 phishing0.mbox	2005-06-13 14:37	3.0M	
 phishing1.mbox	2005-11-15 16:01	4.0M	
 phishing2.mbox	2006-08-07 13:09	9.9M	
 phishing3.mbox	2007-08-07 01:20	19M	
 private-phishing4.mbox	2018-07-24 21:46	31M	

Apache/2.4.25 (Debian) Server at monkey.org Port 443

Figure 1: Screenshot of the phishing email data sets

For this paper, we explored using the 'phishing-2020' data set. To save the data, we simply save page as a text file which we will be using as our data set to test both the spam filters on. The majority of the computations in this paper was carried out on a Ubuntu Linux virtual machine version 18.04.2.

```
user1@groot-VirtualBox:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 18.04.2 LTS
Release:        18.04
Codename:       bionic
```

Figure 2: Ubuntu Version

The file is saved as 'phishing-2020' on our Ubuntu VM and has 157 emails, both email headers and email contents.

2.1 Parsing Data

Initially, scripts were made to parse the data to only extract the email headers as it was assumed that such spam filters would only be working with email headers, similar to Google Admin Toolbox's email header analyser.⁵ However, upon closer inspection of the configurations of Spamassassin and Rspamd, we realised that it could take the whole email, both headers and contents and classifies the email as spam or otherwise accordingly.

Therefore, to improve on 'phishing-2020' text file and make it more user-friendly to do more manipulations on and simply reading it, we created a script `parser.py` to insert a serial number to the beginning of each email. This serial number references the X-UID of each email.

Script: parser.py

```
xuid = 1
with open('phishing-2020', 'r', encoding="utf8") as f:

    # boolean flag whether we want to printing lines or not
    print_lines = False

    for line in f:
        with open('headers_phishing-test', 'a', encoding="utf8") as ff:
            xuid_str = str(xuid)

            # find an indicator
            if line.startswith("From jose@monkey.org"):

                # insert serial number using X-UID
                ff.write('S/N: %s\n' % xuid_str)
                print_lines = True
                xuid += 1

        if print_lines:
            ff.write(line)
            print(line)
```

So with this script, we output a text file that has serial numbers beginning each email. For example, at the sixth email of the text file:

```

S/N: 6
From jose@monkey.org Mon Jan 13 15:13:13 2020 +0000
Return-Path: support@transmed.com.vn
Delivered-To: jose@monkey.org
X-FDA: 76372954266.51.form23_38efa41443b1a
X-Spam-Summary: 75,0,0,17064968886a2a9d,d41d8cd98f00b204,support@transmed.com.vn,,:RULES_HIT:
41:72:355:375:379:495:800:960:967:973:983:988:989:1208:1224:1260:1261:1263:1311:1313:1314:1345
202.79.32.243:@transmed.com.vn:.lbl8.mailshell.net-62.14.177.100
X-HE-Tag: form23_38efa41443b1a
X-Filterd-Recvd-Size: 4418
Received: from titan-02.wlink.com.np (mx-02-243.wlink.com.np [202.79.32.243])
    by imf05.b.hostedemail.com (Postfix) with ESMTP
    for <jose@monkey.org>; Mon, 13 Jan 2020 15:13:11 +0000 (UTC)
Received: from titan-02.wlink.com.np (localhost [127.0.0.1])
    by titan-02.wlink.com.np (Postfix) with ESMTP id 5795111291A
    for <jose@monkey.org>; Mon, 13 Jan 2020 20:48:55 +0545 (+0545)
Received: from localhost (localhost [127.0.0.1])
    by titan-02.wlink.com.np (Postfix) with ESMTP id 54655112959
    for <jose@monkey.org>; Mon, 13 Jan 2020 20:48:55 +0545 (+0545)
X-Virus-Scanned: by SpamTitan at wlink.com.np
Received: from titan-02.wlink.com.np (localhost [127.0.0.1])
    by titan-02.wlink.com.np (Postfix) with ESMTP id E1471112810
    for <jose@monkey.org>; Mon, 13 Jan 2020 20:48:51 +0545 (+0545)
Authentication-Results: titan-02.wlink.com.np; none
Received: from ssmtp.wlink.com.np (ssmtp.wlink.com.np [202.79.32.114])
    by titan-02.wlink.com.np (Postfix) with ESMTP id AD07D1128EB
    for <jose@monkey.org>; Mon, 13 Jan 2020 20:48:50 +0545 (+0545)
Received: from transmed.com.vn (unknown [45.143.223.110])
    (Authenticated sender: shop@com.np)
    by ssmtp.wlink.com.np (Postfix) with ESMTPSA id 00FC723C49
    for <jose@monkey.org>; Mon, 13 Jan 2020 16:11:27 +0545 (NPT)
From: monkey.org <support@transmed.com.vn>
To: jose@monkey.org
Subject: Pending incoming emails(jose@monkey.org).
Date: 13 Jan 2020 02:26:27 -0800
Message-ID: <20200113022627.522C51475FA39644@transmed.com.vn>
MIME-Version: 1.0
Content-Type: text/html;
    charset="iso-8859-1"
Content-Transfer-Encoding: quoted-printable
Status: R0
X-Status:
X-Keywords:
X-UID: 6

```

Figure 3: 'S/N: 6' beginning the email for X-UID: 6

2.2 Spamassassin

To install and set up Spamassassin on our VM, we run the following commands:

```

# optional install
$ apt-get install spamassassin spamc

# if output resource temporarily unavailable
$ sudo killall apt apt-get
$ sudo rm /var/lib/apt/lists/lock
$ sudo rm /var/cache/apt/archives/lock
$ sudo rm /var/lib/dpkg/lock*

# Install SpamAssassin and its client
$ sudo dpkg --configure -a
$ sudo apt update
$ sudo apt-get install spamassassin spamc

# adding user for spamd daemon
$ adduser spamd --disabled-login

```

We then edit configuration settings at `/etc/default/spamassassin`:

```

GNU nano 2.9.3 /etc/default/spamassassin
# /etc/default/spamassassin
# Duncan Findlay
# WARNING: please read README.spamd before using.
# There may be security risks.
# If you're using systemd (default for jessie), the ENABLED setting is
# not used. Instead, enable spamd by issuing:
# systemctl enable spamassassin.service
# Change to "1" to enable spamd on systems using sysvinit:
# ENABLED=0
# Options
# See man spamd for possible options. The -d option is automatically added.
# SpamAssassin uses a preforking model, so be careful! You need to
# make sure --max-children is not set to anything higher than 5,
# unless you know what you're doing.
OPTIONS="--create-prefs --max-children 5 --username spamd --helper-home-dir /home/spamd/ -s /var/log/spamassassin/spamd.log"
# Pid file
# Where should spamd write its PID to file? If you use the -u or
# --username option above, this needs to be writable by that user.
# Otherwise, the init script will not be able to shut spamd down.
# PIDFILE="/var/run/spamd.pid"
# Set nice level of spamd
# NICE="--nicelevel 15"
# Cronjob
# Set to anything but 0 to enable the cron job to automatically update
# spamassassin's rules on a nightly basis
CRON=0

```

Figure 4: Edit configurations

and also edit `/etc/spamassassin/local.cf` to set up some anti-spam rules:

```

GNU nano 2.9.3 /etc/spamassassin/local.cf
# 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_SCORE_ *****
# Save spam messages as a message/rfc822 MIME attachment instead of
# modifying the original message (0: off, 2: use text/plain instead)
report_safe 0
# 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

```

Figure 5: Set up rules

After that, we need to set Postfix to pass incoming messages to our anti-spam system for checking at `/etc/postfix/master.cf`:

```

GNU nano 2.9.3 /etc/postfix/master.cf
# Postfix master process configuration file. For details on the format
# of the file, see the master(5) manual page (command: "man 5 master" or
# on-line: http://www.postfix.org/master.5.html).
#
# Do not forget to execute "postfix reload" after editing this file.
#
# =====
# service type private unpriv chroot wakeup maxproc command + args
# (yes) (yes) (no) (never) (100)
# =====
smtp inet n - y - - smtpd
-o content_filter=spamassassin
#smtp inet n - y - 1 postscreen
#smtpd pass - - y - - smtpd
#dnsblog unix - - y - 0 dnsblog
#tlsproxy unix - - y - 0 tlsproxy
#submission inet n - y - - smtpd

```

Figure 6: Add a content filter to SMTP server

```

spamassassin unix - n n - - pipe
user=spamd argv=/usr/bin/spamc -f -e
/usr/sbin/sendmail -oi -f ${sender} ${recipient}

```

Figure 7: Add to end of file

To complete installation, we restart all related services:

```

$ systemctl restart postfix.service
$ systemctl enable spamassassin.service
$ systemctl start spamassassin.service

```

We have installed and started spamassassin. To check whether its running, we run command:

```

$ sudo systemctl start spamassassin.service

```

and we have confirmed that it is indeed loaded, active and running:

setvtrgb.service	loaded	active	exited	Set console scheme
snappyd.service	loaded	active	exited	Load AppArmor profiles managed internally by snapd
snappyd.import.service	loaded	inactive	dead	Auto import assertions from block devices
snappyd.core-fixup.service	loaded	inactive	dead	Automatically repair incorrect owner/permissions on core devices
snappyd.failure.service	loaded	inactive	dead	Failure handling of the snapd snap
snappyd.recovery-chooser-trigger.service	loaded	inactive	dead	Wait for the Ubuntu Core chooser trigger
snappyd.seeded.service	loaded	active	exited	Wait until snapd is fully seeded
snappyd.service	loaded	active	running	Snap Daemon
snappyd.repair.service	loaded	inactive	dead	Automatically fetch and run repair assertions
spamassassin.service	loaded	active	running	Perl-based spam filter using text analysis
speech-dispatcher.service	loaded	active	exited	LSB: Speech Dispatcher
spice-vdagentd.service	loaded	active	running	Agent daemon for Spice guests
sssd.service	not-found	inactive	dead	sssd.service
systemd-ask-password-console.service	loaded	inactive	dead	Dispatch Password Requests to Console
systemd-ask-password-plymouth.service	loaded	inactive	dead	Forward Password Requests to Plymouth
systemd-ask-password-wall.service	loaded	inactive	dead	Forward Password Requests to Wall
systemd-binfmt.service	loaded	inactive	dead	Set Up Additional Binary Formats

Figure 8: Spamassassin is up and running

To utilize Spamassassin, we need to run a command as follows to send it a text file input and it will output a text file with the classification and results of the degree of spam.

```

$ spamassassin input.txt > output.txt

```

Sadly, Spamassassin as far as we know can only do its check on the first email entry in the text file, ignoring the rest of the emails i.e. only check and give output for S/N: 1. Therefore, we created a script `spamassassin_scorer.py` to create a text file, email by

email, from the phishing-2020 text file, and send each text file containing each email to Spamassassin for its output, and compile all the output to `spamassassin_scores`. Now, we will be able to run Spamassassin on all the emails automatically without having to extract single individual emails for analysis to get a single output.

```
# Script: spamassassin_scorer.py

import os

# iterate 157 times as we know there are 157 emails
for serial_number in range(1, 158):

    # boolean flag
    print_lines = False

    # setting indicators per iteration
    serial_number_string = str(serial_number)
    full_serial_number = 'S/N: ' + serial_number_string + '\n'
    next_serial_number = serial_number + 1
    next_serial_number_string = str(next_serial_number)
    full_next_serial_number = 'S/N: ' + next_serial_number_string +
    ↪ '\n'

    # open and read the parsed phishing-2020 text file
    with open('headers_phishing-test', 'r', encoding = 'utf8') as f:

        # open to append on an empty text file. Its temporary
        ↪ because we only want 1 email per text file to be sent to
        ↪ spamassassin since it can only do on one email at a time
        with open('temp_spamassassin', 'a', encoding = 'utf8') as ff:
            for line in f:
                if full_serial_number in line:

                    # add serial number to output:
                    ↪ spamassassin_scores
                    with open('spamassassin_scores', 'a', encoding =
                    ↪ 'utf8') as fff:
                        fff.write(full_serial_number)
                        fff.close()
                    next(f)

                    # set boolean flag to write out this email
                    ↪ content only
                    print_lines = True

                elif line.startswith(full_next_serial_number):
                    print_lines = False

            if print_lines:
                ff.write(line)
```



```

        print(line)
    ff.close()

    # run the bash command from within python to execute
    → Spamassassin and append to the serialized
    → spamassassin_scores file
    os.system('spamassassin temp_spamassassin >>
    → spamassassin_scores')

    # clears content of temp file to append again in next iteration
    with open('temp_spamassassin', 'w', encoding = 'utf8') as fff:
        fff.close()

    serial_number += 1

```

With this script, we now have `spamassassin_scores` text file with the output results of all the 157 emails in the original phishing-2020 text file. We have successfully used Spamassassin on all the emails with a script and did not have to pick and choose manually which emails we want to use Spamassassin on.

2.3 Rspamd

To install Rspamd, we run the following commands:

```

# begin installation
$ sudo apt-get install -y lsb-release wget

# if dpkg was interrupted
$ sudo dpkg --configure -a
$ sudo apt-get install -y lsb-release wget

# continue installation
$ CODENAME=`lsb_release -c -s`
$ sudo mkdir -p /etc/apt/keyrings
$ wget -O- https://rspamd.com/apt-stable/gpg.key | gpg --dearmor |
    → sudo tee /etc/apt/keyrings/rspamd.gpg > /dev/null
$ echo "deb [arch=amd64 signed-by=/etc/apt/keyrings/rspamd.gpg] http
    → ://rspamd.com/apt-stable/ $CODENAME main" | sudo tee /etc/apt/
    → sources.list.d/rspamd.list
$ echo "deb-src [arch=amd64 signed-by=/etc/apt/keyrings/rspamd.gpg]
    → http://rspamd.com/apt-stable/ $CODENAME main" | sudo tee -a /
    → etc/apt/sources.list.d/rspamd.list
$ sudo apt-get update
$ sudo apt-get --no-install-recommends install rspamd

```

The final command will output as such indicating a successful installation:

```

user1@groot-VirtualBox:~$ sudo apt-get --no-install-recommends install rspamd
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  libc++1 libc++abi1 libhyperscan4 libpcre2-8-0
Suggested packages:
  clang
The following NEW packages will be installed:
  libc++1 libc++abi1 libhyperscan4 libpcre2-8-0 rspamd
0 upgraded, 5 newly installed, 0 to remove and 247 not upgraded.
Need to get 6,242 kB of archives.
After this operation, 29.8 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://sg.archive.ubuntu.com/ubuntu bionic/universe amd64 libhyperscan4 amd64 4.7.0-1 [2,208 kB]
Get:3 http://sg.archive.ubuntu.com/ubuntu bionic/universe amd64 libpcre2-8-0 amd64 10.31-2 [179 kB]
Get:4 http://sg.archive.ubuntu.com/ubuntu bionic/universe amd64 libc++abi1 amd64 6.0-2 [56.7 kB]
Get:5 http://sg.archive.ubuntu.com/ubuntu bionic/universe amd64 libc++1 amd64 6.0-2 [183 kB]
Get:2 https://rspamd.com/apt-stable bionic/main amd64 rspamd amd64 3.0-2-bionic [3,614 kB]
Fetched 6,242 kB in 13s (465 kB/s)
Preconfiguring packages ...
Selecting previously unselected package libhyperscan4.
(Reading database ... 186011 files and directories currently installed.)
Preparing to unpack .../libhyperscan4_4.7.0-1_amd64.deb ...
Unpacking libhyperscan4 (4.7.0-1) ...
Selecting previously unselected package libpcre2-8-0:amd64.
Preparing to unpack .../libpcre2-8-0_10.31-2_amd64.deb ...
Unpacking libpcre2-8-0:amd64 (10.31-2) ...
Selecting previously unselected package libc++abi1:amd64.
Preparing to unpack .../libc++abi1_6.0-2_amd64.deb ...
Unpacking libc++abi1:amd64 (6.0-2) ...
Selecting previously unselected package libc++1:amd64.
Preparing to unpack .../libc++1_6.0-2_amd64.deb ...
Unpacking libc++1:amd64 (6.0-2) ...
Selecting previously unselected package rspamd.
Preparing to unpack .../rspamd_3.0-2-bionic_amd64.deb ...
Unpacking rspamd (3.0-2-bionic) ...
Setting up libc++abi1:amd64 (6.0-2) ...
Processing triggers for ureadahead (0.100.0-20) ...
Processing triggers for libc-bin (2.27-3ubuntu1.2) ...
Processing triggers for systemd (237-3ubuntu10.50) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
Setting up libpcre2-8-0:amd64 (10.31-2) ...
Setting up libhyperscan4 (4.7.0-1) ...
Setting up libc++1:amd64 (6.0-2) ...
Setting up rspamd (3.0-2-bionic) ...
Created symlink /etc/systemd/system/multi-user.target.wants/rspamd.service → /lib/systemd/system/rspamd.service.
Processing triggers for libc-bin (2.27-3ubuntu1.2) ...
Processing triggers for systemd (237-3ubuntu10.50) ...
Processing triggers for ureadahead (0.100.0-20) ...
user1@groot-VirtualBox:~$

```

Figure 9: Rspamd installed

And we want to configure some basic rules. In this paper, the default actions were kept as originally configured with the installation. Internally, Rspamd operates with rules. Each rule can add positive or negative score to the result. Therefore it is required to have some thresholds for actions that are applied to a message. These thresholds are defined in actions:

```

user1@groot-VirtualBox:/etc/rspamd$ cat /etc/rspamd/actions.conf
# Actions settings
# Please don't modify this file as your changes might be overwritten with
# the next update.
#
# You can modify 'local.d/actions.conf' to add and merge
# parameters defined inside this section
#
# You can modify 'override.d/actions.conf' to strictly override all
# parameters defined inside this section
#
# See https://rspamd.com/doc/faq.html#what-are-the-locald-and-overrided-directories
# for details
#
# See also https://rspamd.com/doc/faq.html#what-are-rspamd-actions for actions definition

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"
}
user1@groot-VirtualBox:/etc/rspamd$

```

Figure 10: Actions configuration

Now, to utilize Rspamd, we need to run the following command:

```
$ rspamc < input.txt > output.txt
```

We also created a similar `rspamd_scorer.py` script to create a text file, email by email, from the phishing-2020 text file, and send each text file containing each email to Rspamd for its output, and compile all the output to `rspamd_scores`.

```
# Script: rspamd_scorer.py
```

```
import os
```

```
# iterate 157 times as we know there are 157 emails
```

```
for serial_number in range(1, 158):
```

```
    # boolean flag
```

```
    print_lines = False
```

```
    # setting indicators per iteration
```

```
    serial_number_string = str(serial_number)
```

```
    full_serial_number = 'S/N: ' + serial_number_string + '\n'
```

```
    next_serial_number = serial_number + 1
```

```
    next_serial_number_string = str(next_serial_number)
```

```
    full_next_serial_number = 'S/N: ' + next_serial_number_string +  
    ↪ '\n'
```

```
    # open and read the parsed phishing-2020 text file
```

```
    with open('headers_phishing-test', 'r', encoding = 'utf8') as f:
```

```
        # open to append on an empty text file. Its temporary
```

```
        ↪ because we only want 1 email per text file to be sent to
```

```
        ↪ rspamd since it can only do on one email at a time
```

```
        with open('temp_rspamd', 'a', encoding = 'utf8') as ff:
```

```
            for line in f:
```

```
                if full_serial_number in line:
```

```
                    # add serial number to output: rspamd_scores
```

```
                    with open('rspamd_scores', 'a', encoding = 'utf8')
```

```
                        ↪ as fff:
```

```
                            fff.write(full_serial_number)
```

```
                            fff.close()
```

```
                    next(f)
```

```
                    # set boolean flag to write out this email
```

```
                    ↪ content only
```

```
                    print_lines = True
```

```
                elif line.startswith(full_next_serial_number):
```

```
                    print_lines = False
```

```
            if print_lines:
```

```

        ff.write(line)
        print(line)
    ff.close()

    # run the bash command from within python to execute
    → rspamd and append to the serialized rspamd_scores
    → file
    os.system('rspamc < temp_rspamd >> rspamd_scores')

    # clears content of temp file to append again in next iteration
    with open('temp_rspamd', 'w', encoding = 'utf8') as fff:
        fff.close()

    serial_number += 1

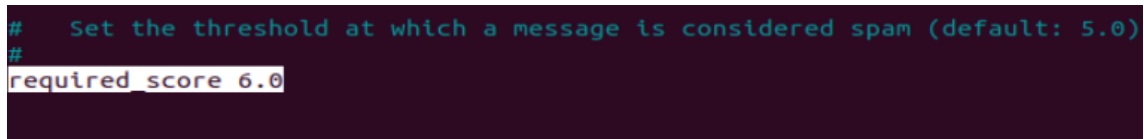
```

With this script, we now have `rspamd_scores` text file with the output results of all the 157 emails in the original phishing-2020 text file. We have successfully used Rspamd on all the emails with a script and did not have to pick and choose manually which emails we want to use Rspamd on.

2.4 Avoid Detection and Classification

2.4.1 Increasing Required Score

As elaborated in the setup sections above, we can lower the likelihood emails are detected as spams by changing the default configuration of the required scores to be classified as spam. For Spamassassin, we edit the `/etc/spamassassin/local.cf` and increase the required score by 1 so now it is 6.0.



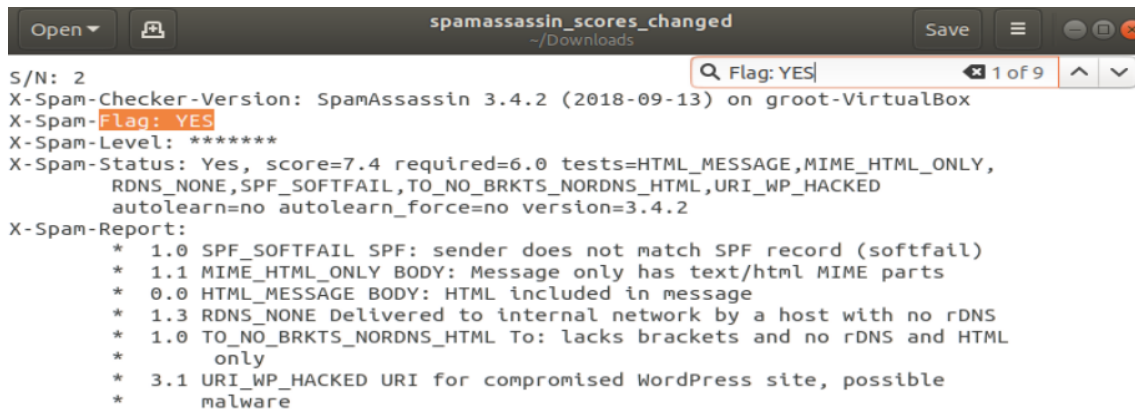
```

# Set the threshold at which a message is considered spam (default: 5.0)
#
required_score 6.0

```

Figure 11: Editing local.cf

We run the `spamassassin_scorer.py` and output `spamassassin_scores_changed` and then we count the number of emails from the same phishing-2020 data set now classified as spam by searching for 'Flag: YES'.

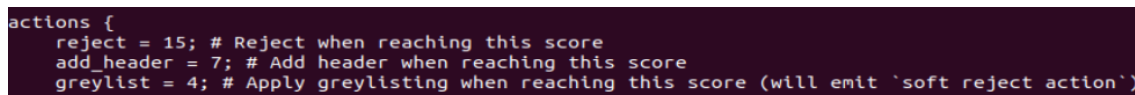


```
S/N: 2
X-Spam-Checker-Version: SpamAssassin 3.4.2 (2018-09-13) on groot-VirtualBox
X-Spam-Flag: YES
X-Spam-Level: *****
X-Spam-Status: Yes, score=7.4 required=6.0 tests=HTML_MESSAGE,MIME_HTML_ONLY,
RDNS_NONE,SPF_SOFTFAIL,TO_NO_BRKTS_NORDNS_HTML,URI_WP_HACKED
autolearn=no autolearn_force=no version=3.4.2
X-Spam-Report:
* 1.0 SPF_SOFTFAIL SPF: sender does not match SPF record (softfail)
* 1.1 MIME_HTML_ONLY BODY: Message only has text/html MIME parts
* 0.0 HTML_MESSAGE BODY: HTML included in message
* 1.3 RDNS_NONE Delivered to internal network by a host with no rDNS
* 1.0 TO_NO_BRKTS_NORDNS_HTML To: lacks brackets and no rDNS and HTML
* only
* 3.1 URI_WP_HACKED URI for compromised WordPress site, possible
* malware
```

Figure 12: New count = 9

We see now that only 9 emails are classified as spam compared to its default settings when 25 emails are classified as spam.

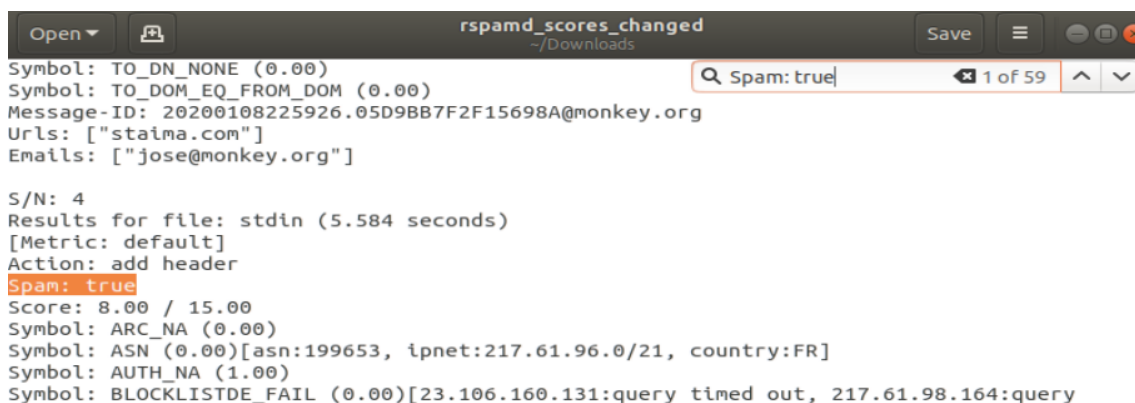
For Rspamd, we edit the `/etc/rspamd/actions.conf` and increase the required score on by 1 so `add_header` is now equals to 7.



```
actions {
    reject = 15; # Reject when reaching this score
    add_header = 7; # Add header when reaching this score
    greylist = 4; # Apply greylisting when reaching this score (will emit 'soft reject action')
```

Figure 13: Editing `/etc/rspamd/actions.conf`

We run the `rspamd_scorer.py` and output `rspamd_scores_changed` and then we count the number of emails from the same phishing-2020 data set now classified as spam by searching for 'Spam: true'.



```
Symbol: TO_DN_NONE (0.00)
Symbol: TO_DOM_EQ_FROM_DOM (0.00)
Message-ID: 20200108225926.05D9BB7F2F15698A@monkey.org
Urls: ["staima.com"]
Emails: ["jose@monkey.org"]

S/N: 4
Results for file: stdin (5.584 seconds)
[Metric: default]
Action: add header
Spam: true
Score: 8.00 / 15.00
Symbol: ARC_NA (0.00)
Symbol: ASN (0.00)[asn:199653, ipnet:217.61.96.0/21, country:FR]
Symbol: AUTH_NA (1.00)
Symbol: BLOCKLISTDE_FAIL (0.00)[23.106.160.131:query timed out, 217.61.98.164:query
```

Figure 14: New count = 59

We see now that only 59 emails are classified as spam compared to its default settings when 61 emails are classified as spam. Therefore, we have demonstrated how to avoid detection and classification by increasing the scores required by the email input by the spam filters. This is a blanket rule on all rules and actions and does not need to take into account which rules is triggering the filters specifically.

2.4.2 Lowering Individual Scores

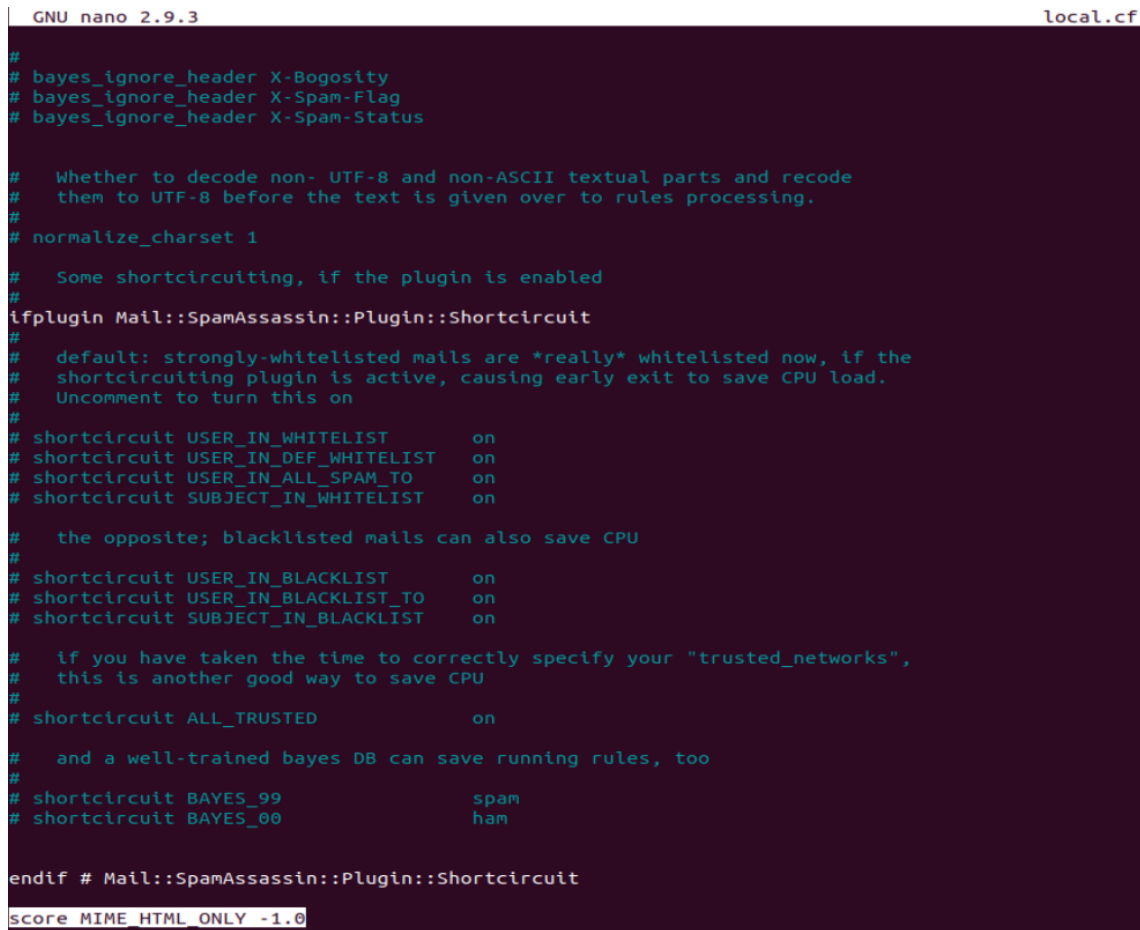
For Spamassassin, 22 of the 25 classified spam emails have this rule triggered:

MIME_HTML_ONLY

always giving a score of 1.1, and can be seen in screenshots under spamassassin's outputs of spam emails subsection. So to lower this rule that is majority of the times being triggered, we go to edit the local.cf file by running

```
sudo nano /etc/spamassassin/local.cf
```

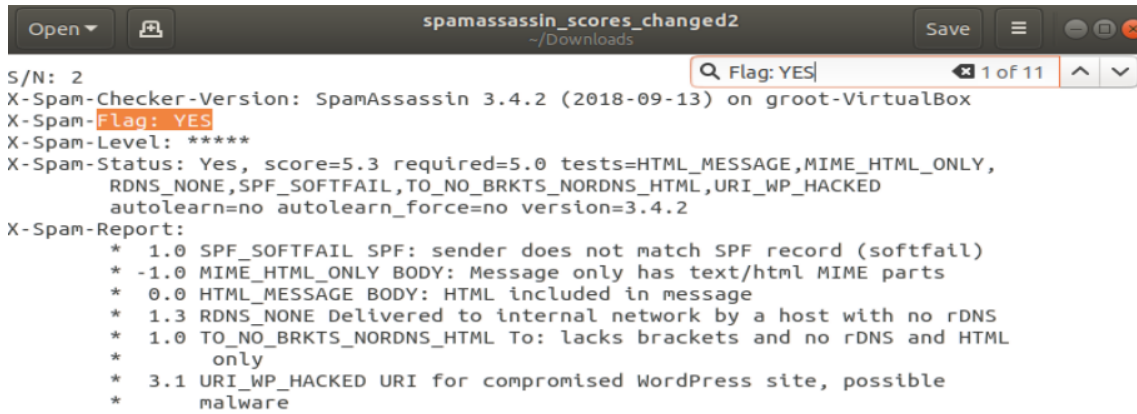
and add in a custom score of -1.0 instead to reduce the score:



```
GNU nano 2.9.3 local.cf
#
# 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
#
# 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
#
# 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
score MIME_HTML_ONLY -1.0
```

Figure 15: Add in score MIME_HTML_ONLY -1.0

We see now that after running our `spamassassin_scorer.py`, we only now have 11 emails classified as spam. We also see that our customized score of -1.0 is also observed in the report screenshot below:



```
S/N: 2
X-Spam-Checker-Version: SpamAssassin 3.4.2 (2018-09-13) on groot-VirtualBox
X-Spam-Flag: YES
X-Spam-Level: *****
X-Spam-Status: Yes, score=5.3 required=5.0 tests=HTML_MESSAGE,MIME_HTML_ONLY,
RDNS_NONE,SPF_SOFTFAIL,TO_NO_BRKTS_NORDNS_HTML,URI_WP_HACKED
autolearn=no autolearn_force=no version=3.4.2
X-Spam-Report:
* 1.0 SPF_SOFTFAIL SPF: sender does not match SPF record (softfail)
* -1.0 MIME_HTML_ONLY BODY: Message only has text/html MIME parts
* 0.0 HTML_MESSAGE BODY: HTML included in message
* 1.3 RDNS_NONE Delivered to internal network by a host with no rDNS
* 1.0 TO_NO_BRKTS_NORDNS_HTML To: lacks brackets and no rDNS and HTML
* only
* 3.1 URI_WP_HACKED URI for compromised WordPress site, possible
* malware
```

Figure 16: Edited score MIME_HTML_ONLY -1.0

So since we have set that score to -1.0 instead of its defaulted +1.1 and since it is in 22 of the 25 spam classified emails, we have successfully targetted a rule that majority of the emails are triggered for and consequently reduced the number of emails classified as spam from 25 to 11, without changing the required score of 5.

Therefore, we have demonstrated 2 ways in which emails can be classified as spam; first by increasing the overall required score to be classified as spam and second by reducing the specific triggered rules score.

2.4.3 Bypassing Email Service Providers

Bypassing email service providers is trivial as it can be done from the user's side. Similar to how the user can avoid detection and classification by increasing the required score to be classified a spam email or lowering individual triggered rules score to reduce the overall spam score of the email, users can also do similar such actions on email service providers.

For example, for gmail, users can follow this method to set up SPF and DKIM and also DMARC properly to allow incoming emails to bypass gmails filtering as these will be seen by gmail as passes.⁶ It bypasses as an SPF record identifies the mail servers and domains that are allowed to send email on behalf of the user's domain. Receiving servers check the corresponding SPF record to verify that incoming messages that appear to be from the user's organization are sent from servers allowed by the user.

As for Office 365, users can configure mail flow rules to set the spam confidence level (SCL) in messages in Exchange Online to be able to bypass Office 365's spam filtering.⁷ For example, if the user sets the SCL to Bypass spam filtering, then the message should be sent to the specified recipient's inbox.

We consider such methods trivial as all these methods can be set up from the user's side i.e. not the attacker bypassing the spam filters inherently without changing any of the configurations of the spam filters. We could have simply added SPF into our domain and sent it from our Ubuntu VM to allow it to bypass gmail's spam filtering.

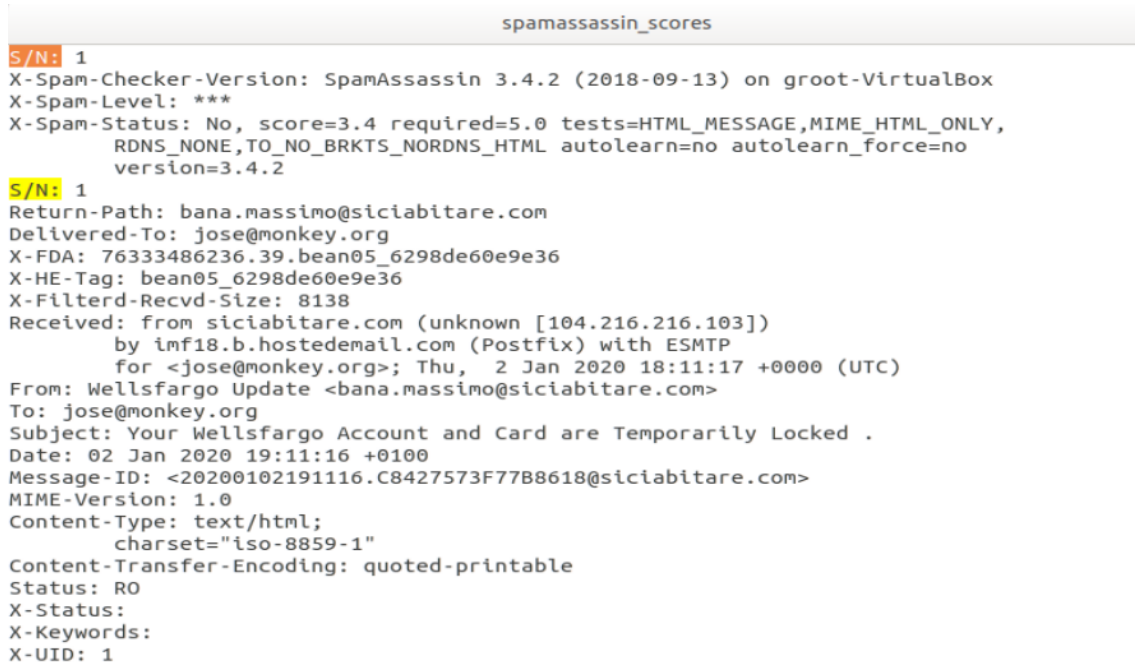
3 Results

3.1 Spamassassin VS Rspamd

For spamassassin, it is considered spam if the score is larger or equals to the default setting of 5. For rspamd, it is equals to the 'add header' action which is 6. To compare the outputs of each of the spam filter, we take a look at the first entry X-UID: 1 or S/N: 1 which are both classified to be not spam.

3.1.1 Outputs of Not Spam

spamassassin_scores compiled output is as follows:



```
spamassassin_scores
S/N: 1
X-Spam-Checker-Version: SpamAssassin 3.4.2 (2018-09-13) on groot-VirtualBox
X-Spam-Level: ***
X-Spam-Status: No, score=3.4 required=5.0 tests=HTML_MESSAGE,MIME_HTML_ONLY,
RDNS_NONE,TO_NO_BRKTS_NORDNS_HTML autolearn=no autolearn_force=no
version=3.4.2
S/N: 1
Return-Path: bana.massimo@siciabitare.com
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
```

Figure 17: Spamassassin output under the first 'S/N: 1'. The screenshot does not capture the whole input we gave Spamassassin which is under the second 'S/N: 1'

rspamd_scores compiled output is as follows:


```

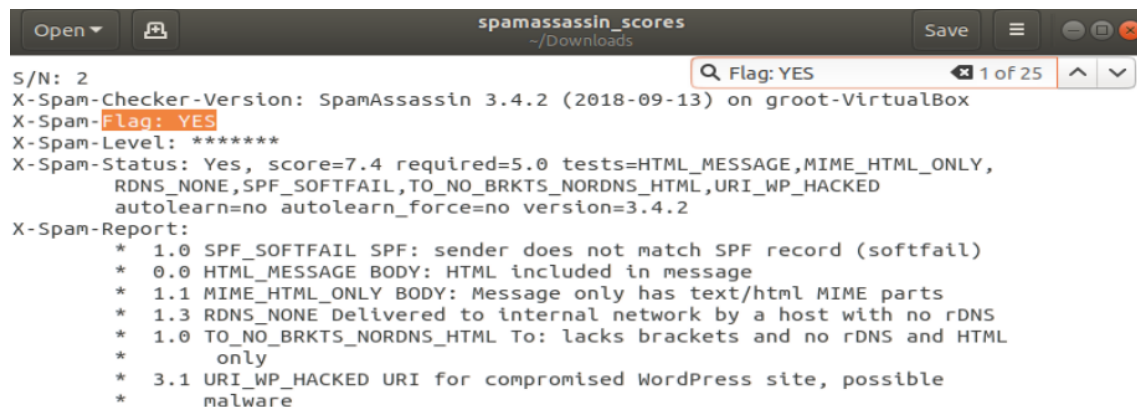
*rspamd_scores
S/N: 1
Results for file: stdin (2.124 seconds)
[Metric: default]
Action: greylist
Spam: false
Score: 5.79 / 15.00
Symbol: ARC_NA (0.00)
Symbol: ASN (0.00)[asn:40676, ipnet:104.216.216.0/24, country:US]
Symbol: AUTH_NA (1.00)
Symbol: DATE_IN_PAST (1.00)[15692]
Symbol: DMARC_NA (0.00)[siciabitare.com]
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: 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)[no SPF record]
Symbol: TO_DN_NONE (0.00)
Message-ID: 20200102191116.C8427573F77B8618@siciabitare.com
Urls: ["wellsfargo.com", "wellsfargo.usa.s3-website-us-east-1.amazonaws.com"]
Emails: []

```

Figure 18: Full screenshot of Rspamd output showing S/N of 1 i.e. first entry

3.1.2 Outputs of Spam

The following 2 screenshots shows the output when email is classified as spam by the 2 spam filters:



```

Open  spamassassin_scores  Save  1 of 25
S/N: 2
X-Spam-Checker-Version: SpamAssassin 3.4.2 (2018-09-13) on groot-VirtualBox
X-Spam-Flag: YES
X-Spam-Level: *****
X-Spam-Status: Yes, score=7.4 required=5.0 tests=HTML_MESSAGE,MIME_HTML_ONLY,
RDNS_NONE,SPF_SOFTFAIL,TO_NO_BRKTS_NORDNS_HTML,URI_WP_HACKED
autolearn=no autolearn_force=no version=3.4.2
X-Spam-Report:
* 1.0 SPF_SOFTFAIL SPF: sender does not match SPF record (softfail)
* 0.0 HTML_MESSAGE BODY: HTML included in message
* 1.1 MIME_HTML_ONLY BODY: Message only has text/html MIME parts
* 1.3 RDNS_NONE Delivered to internal network by a host with no rDNS
* 1.0 TO_NO_BRKTS_NORDNS_HTML To: lacks brackets and no rDNS and HTML
* only
* 3.1 URI_WP_HACKED URI for compromised WordPress site, possible
* malware

```

Figure 19: Spamassassin has 25 matches classified as spam

```

S/N: 2
Results for file: stdin (2.548 seconds)
[Metric: default]
Action: add header
Spam: true
Score: 7.30 / 15.00
Symbol: ARC_NA (0.00)
Symbol: ASN (0.00)[asn:4788, ipnet:219.93.0.0/18, country:MY]
Symbol: DATE_IN_PAST (1.00)[15540]
Symbol: DMARC_NA (0.00)[monkey.org]
Symbol: FROM_EQ_ENVFROM (0.00)
Symbol: FROM_HAS_DN (0.00)
Symbol: HAS_WP_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: PREVIOUSLY_DELIVERED (0.00)[jose@monkey.org]
Symbol: RCPT_COUNT_ONE (0.00)[1]
Symbol: RCVD_COUNT_SEVEN (0.00)[7]
Symbol: RCVD_NO_TLS_LAST (0.10)
Symbol: RCVD_VIA_SMTP_AUTH (0.00)
Symbol: RECEIVED_SPAMHAUS_PBL (0.00)[103.133.109.78:received]
Symbol: R_DKIM_NA (0.00)
Symbol: R_SPF_SOFTFAIL (0.00)[~all:c]
Symbol: TO_DN_NONE (0.00)
Symbol: TO_DOM_EQ_FROM_DOM (0.00)
Symbol: VIOLATED_DIRECT_SPF (3.50)
Message-ID: 20200108184416.60559ADF7F381160@monkey.org
Urls: ["staima.com"]
Emails: ["jose@monkey.org"]

```

Figure 20: Rspamd has 61 matches classified as spam

We can see that the outputs for Rspamd is the same for both spam and not spam classifications. However, spamassassin will have additional 'X-Spam-Report:' if the email is classified as spam.

3.1.3 Default Classifications

We first count the total number of emails classified as spam by each of the 2 email filters. For `spamassassin_scores`, we search by 'Flag: Yes' and the text editor will show how many matches there are. For `rspamd_scores`, we search by 'Spam: true'. Referring to the 2 screenshots in the previous subsection 'Outputs of Spam', we can see that Spamassassin has 25 matches classified as spam while Rspamd has 61 matches classified as spam. In percentages, Spamassassin marks 15.92% as spam while Rspamd more strictly marks 38.85% as spam, more than twice that of Spamassassin. Therefore, Rspamd's default configuration is more than 2 times more strict in classifying email as spam.

To further compare, we wrote a script for each score text file to parse the scores into compiled scores arranged by serial number into a text file.

First script: spamassassin_compiler.py

```

serial_number = 1
while serial_number < 158:
    with open('spamassassin_scores', 'r', encoding = 'utf8') as f:
        with open('spamassassin_compiled_scores', 'w', encoding =
            'utf8') as ff:
            for line in f:

                # find a match from spamassassin_scores file

```

```

if 'X-Spam-Status' in line:

    # if found, take only the score
    aa = line[line.index('='):line.index('required')]
    aa = aa[1:]
    ff.write('%s: %s\n' %(serial_number, aa))
    serial_number += 1

# Second script: rspamd_compiler.py

serial_number = 1
while serial_number < 158:
    with open('rspamd_scores', 'r', encoding = 'utf8') as f:
        with open('rspamd_compiled_scores', 'w', encoding = 'utf8') as
        ↪ ff:
            for line in f:

                # find a match from rspamd_scores file
                if 'Score:' in line:

                    # if found, take only the score
                    aa = line[line.index(' '):11]
                    ff.write('%s:%s\n' %(serial_number, aa))
                    serial_number += 1

```

We then transferred that compiled scores into an excel to do data manipulation on it. By doing this, we hope to see if there is a trend in how both classify according to the scores given. For example, a high score given by Spamassassin on a certain email should also have been given a high score by Rspamd. The converse should also be true. For Spamassassin, the score given was divided by 5 (which is the score needed to be classified as spam) and multiplied by 100 to give a spam-percentage score. For Rspamd, the score given was divided by 6 (which is the score needed to be classified as spam) and multiplied by 100 to give a spam-percentage score.



Figure 21: Trend comparison

From this graph, both spam filters somewhat classify to the same magnitude. This

can be seen where most of the Spamassassin's line fits into that of Rspamd's and vice versa. While majority of emails classified where both give high spam-percentage or both gives low spam-percentage, it is also good to note that there are also emails where the 2 spam filters disagree.

3.2 Triggered Rules and Scoring System

Here, we will explore the triggered rules and the scoring system. As mentioned previously, for Spamassassin, an email is classified as spam if score is 5 or more while for Rspamd, an email is classified as spam if score is 6 or more. To explore most elaborately, we choose the highest scored email to be able to see all the triggered rules. For both Spamassassin and Rspamd, this happens to be email number 18.

For Spamassassin, it has X-Spam-Flag which indicates if an email is a spam. This header will not be generated if email is classified to be not-spam. X-Spam-Level indicates the overall score severity with more asterisks corresponding to the higher likelihood email is a spam. The X-Spam-Status will indicate whether it has been classified as a spam. Score values show the numerical spam score of the email, and will be classified spam if higher than the required value. Output of Spamassassin on S/N: 18 with score 16.2 / 5.00 as follows:

```
S/N: 18
X-Spam-Checker-Version: SpamAssassin 3.4.2 (2018-09-13) on groot-VirtualBox
X-Spam-Flag: YES
X-Spam-Level: *****
X-Spam-Status: Yes, score=16.2 required=5.0 tests=AXB_XMAILER_MIMEOLE_OL_1ECD5,
DKIM_ADSP_NXDOMAIN,DKIM_SIGNED,FORGED_MUA_OUTLOOK,FORGED_OUTLOOK_HTML,
FORGED_OUTLOOK_TAGS,FROM_MISSP_MSFT,FROM_MISSP_PHISH,FROM_MISSP_XPRIO,
HTML_MESSAGE,HTML_MIME_NO_HTML_TAG,MIME_HTML_ONLY,MISSING_HEADERS,
MISSING_MID,NO_DNS_FOR_FROM,RCVD_IN_SORBS_WEB,SUBJ_ALL_CAPS,
TO_NO_BRKTS_FROM_MSSP,TO_NO_BRKTS_MSFT,T_DKIM_INVALID autolearn=no
autolearn_force=no version=3.4.2
X-Spam-Report:
* 0.4 NO_DNS_FOR_FROM DNS: Envelope sender has no MX or A DNS records
* 1.5 RCVD_IN_SORBS_WEB RBL: SORBS: sender is an abusable web server
* [154.0.26.181 listed in dnsbl.sorbs.net]
* 0.8 DKIM_ADSP_NXDOMAIN No valid author signature and domain not in
* DNS
* 1.6 SUBJ_ALL_CAPS Subject is all capitals
* 1.2 MISSING_HEADERS Missing To: header
* 0.0 HTML_MESSAGE BODY: HTML included in message
* 1.1 MIME_HTML_ONLY BODY: Message only has text/html MIME parts
* 0.1 DKIM_SIGNED Message has a DKIM or DK signature, not necessarily|
* valid
* 0.6 HTML_MIME_NO_HTML_TAG HTML-only message, but there is no HTML
* tag
* 0.1 MISSING_MID Missing Message-Id: header
* 0.0 FORGED_OUTLOOK_HTML Outlook can't send HTML message only
* 0.0 FROM_MISSP_MSFT From misspaced + supposed Microsoft tool
* 0.0 AXB_XMAILER_MIMEOLE_OL_1ECD5 Yet another X header trait
* 0.6 FORGED_OUTLOOK_TAGS Outlook can't send HTML in this format
* 0.0 FROM_MISSP_XPRIO Misspaced FROM + X-Priority
* 0.0 T_DKIM_INVALID DKIM-Signature header exists but is not valid
* 3.5 FROM_MISSP_PHISH Malformed, claims to be from financial
* organization - possible phish
* 0.0 TO_NO_BRKTS_FROM_MSSP Multiple header formatting problems
* 2.8 FORGED_MUA_OUTLOOK Forged mail pretending to be from MS Outlook
* 1.9 TO_NO_BRKTS_MSFT To: lacks brackets and supposed Microsoft tool
```

Figure 22: Output showing all the triggered rules and corresponding scores

For Rspamd, it has these headers; Action suggests action to be taken, whether accept, reject or greylist. This suggestion is given based on computed spam score. Spam indicates if email is categorised as spam. Score indicates the score severity, with

higher scores meaning email has higher likelihood of being a spam. Symbol indicates list of test used to check the email if it is a spam. URLs contains all the URLs in the email input and emails contains the email of the recipient. Output of Rspamd on S/N: 18 with score 21.8 / 6.00 as follows:

```
S/N: 18
Results for file: stdin (2.096 seconds)
[Metric: default]
Action: reject
Spam: true
Score: 21.82 / 15.00
Symbol: ARC_NA (0.00)
Symbol: ASN (0.00)[asn:57497|, ipnet:158.58.187.0/24, country:IR]
Symbol: DATE_IN_PAST (1.00)[14994]
Symbol: DKIM_TRACE (0.00)[sarawveg.com:~]
Symbol: DMARC_NA (0.00)[sarawveg.com]
Symbol: FORGED_MUA_OUTLOOK (3.00)
Symbol: FORGED_OUTLOOK_HTML (5.00)
Symbol: FORGED_OUTLOOK_TAGS (2.10)
Symbol: FROM_EQ_ENVFROM (0.00)
Symbol: FROM_HAS_DN (0.00)
Symbol: HAS_X_ANTIABUSE (0.00)
Symbol: HAS_X_AS (0.00)[bdo-onlinebanking@sarawveg.com]
Symbol: HAS_X_GMSV (0.00)[sarawveg/from_h]
Symbol: HAS_X_PRIO_THREE (0.00)[3]
Symbol: HAS_X_SOURCE (0.00)
Symbol: HFILTER_FROMHOST_NORES_A_OR_MX (1.50)[sarawveg.com]
Symbol: MIME_HTML_ONLY (0.20)
Symbol: MIME_TRACE (0.00)[0:~]
Symbol: MISSING_MID (2.50)
Symbol: MISSING_TO (2.00)
Symbol: OLD_X_MAILER (2.00)
Symbol: RCVD_COUNT_TWO (0.00)[2]
Symbol: RCVD_NO_TLS_LAST (0.10)
Symbol: RCVD_VIA_SMTP_AUTH (0.00)
Symbol: RECEIVED_SPAMHAUS_PBL (0.00)[154.0.26.181:received]
Symbol: R_DKIM_PERMFAIL (0.00)[sarawveg.com:s=default]
Symbol: R_NO_SPACE_IN_FROM (1.00)
Symbol: R_SPF_NA (0.00)[no SPF record]
Symbol: SUBJ_ALL_CAPS (1.42)[19]
Message-ID: undef
Urls: ["sarawveg.com"]
Emails: []
```

Figure 23: Output showing all the triggered rules and corresponding scores

4 Discussion

4.1 Strengths

4.1.1 Parsing Email Datasets

We have demonstrated a clever implementation of using multiple scripts to parse email data sets in such a way that spamassassin and rspamd is able to be used within this scripts to give a compiled output of all the spam scores and the initial input. This is a clever implementation as the user does not need to handpick a few emails or even worse, manually input one by one the different emails and run spamassassin or rspamd's commands to get a spam score output.

4.1.2 Avoiding Detections

We have also demonstrated that there are multiple methods to avoid detections by the 2 spam filters that we have been using. We showed that for both spamassassin and rspamd, we can increase the required scores for an email to be classified as spam. This is a great implementation as we do not need to dig deep into all the email spam scores and analyse which of the triggered rules should be edited to avoid detections. We also showed a second method on spamassassin where we analysed to find the most frequent triggered rule and we edited the configurations and lowered the score given by that specific rule to influence the final spam score of the emails.

4.1.3 Comparison Methodology

Our comparison methodology not only compares the rates of the spam classification between the 2 spam filters, but it also compares the degree of classification. We showed that majority of the classifications even if outcome of whether spam or not might be different, the degree of the scores are similar i.e. a high spam score given by spamassassin usually has a high spam score given by rspamd as well and the converse is true.

4.2 Limitations

We believe that a limitation in this paper is the inability to be able to actually send spam emails and bypassing spam filters without needing to change the configurations on the receiver's side. Essentially, we have proven that if we were able to change the configurations by accessing the receiver's machines and spam filter softwares, we will be able to change certain settings to allow future spam emails to bypass these filters. However, if we were not able to change these configurations, the spam filters would be able to prevent our spam email from reaching the receiver's inbox.

5 Conclusion

This paper has explored and proven successfully through the use of scripts and configuration changes that spam email can easily bypass spam filters. However, more study can be done to be able to bypass spam filters without the need of any change in configurations from the receiver's side.

References

- ¹ People – the weakest link in cybersecurity. <https://www.infoguardsecurity.com/people-the-weakest-link-in-cybersecurity/>. Last Accessed: 2021-09-20.
- ² Spamassassin 3 on ubuntu 16. <http://www.geoffstratton.com/spamassassin-3-ubuntu-16>. Last Accessed: 2021-09-20.
- ³ Rspamd packages. <https://rspamd.com/downloads.html>. Last Accessed: 2021-09-20.
- ⁴ Index of / jose/phishing. <https://monkey.org/~jose/phishing/>. Last Accessed: 2021-09-20.
- ⁵ Google admin toolbox messageheader. <https://toolbox.googleapps.com/apps/messageheader/>. Last Accessed: 2021-09-20.
- ⁶ Define your spf record—basic setup. <https://support.google.com/a/answer/10685031#workspace-record>. Last Accessed: 2021-09-20.
- ⁷ Use mail flow rules to set the spam confidence level (scl) in messages in exchange online. <https://docs.microsoft.com/en-us/exchange/security-and-compliance/mail-flow-rules/use-rules-to-set-scl>. Last Accessed: 2021-09-20.