혁신성장 청년인재 집중양성 사업

(AI 피싱메일 악성 URL 탐지)

2021년 10월 29일

인공지능을 활용한 보안 전문가 양성 과정
(Detectives)
(정병현 연람희 권채연)

목 차

1. 프로젝트 개요

1.1 프로젝트 기획 배경 및 목표

우리는 인공지능을 활용한 보안 전문가 양성 과정을 통해 배운 피싱 메일 분석 자동화 프로세스를 구축하여 보다 정확한 피싱 URL 탐지율로 피싱 메일을 분류해내는 것을 목표로 함. 메일 서버를 직접 구축하여 사용하는 중소기업들을 대상으로 서비스를 개발하고, 동 프로젝트를 통해 중소기업의 비용 부담 절감과 피싱 메일에 대한 취약점이 감소되는 것을 기대 함.

피싱 메일이 일반 메일함에 접근하지 못하도록 스팸함으로 필터링 하여 사용자가 쉽게 피싱 URL 에 접근할 수 없도록 하고, 추가로 헤더 제목에 피싱 문구를 삽입하여 피싱 메일을 눈에 띄게 할 수 있도록 기획하였음

1.2 구성원 및 역할

이름	전공	역할	구현 부분
정병현	컴퓨터공학과	팀장 /개발	1. 프로젝트 관리 2. 데이터 가공 3. 머신러닝 모델 개발
연람희	영어학과	시스템 구축 /개발 보조	1. 메일/네임 서버 구축 2. 데이터 가공 보조 3. 머신러닝 모델 개발 보조
권채연	미생물학과	문서	1. 문서 정리 2. 데이터 가공 보조 3. 팀원 자원 관리

1.3 프로젝트 추진 일정

구분	기간	활동	비고
사전 기획	8/20 ~ 8/31	프로젝트 기획	
	9/1 ~ 9/4	Outline 구상 및 자료 조사	
프로젝트 수행	9/4 ~ 10/1	시스템 구축	메일/DNS 서버 구축 및 Pymilter 연동
	9/4 ~ 10/15	1. 데이터 가공 2. ML 모델 구현	Feature score 함수 구현
	10/15 ~ 10/30	1. 시스템 + ML 모델 연동 2. 모델 성능 향상 구현	
문서	10/10 ~ 11/3	문서 작성	보고서, PPT

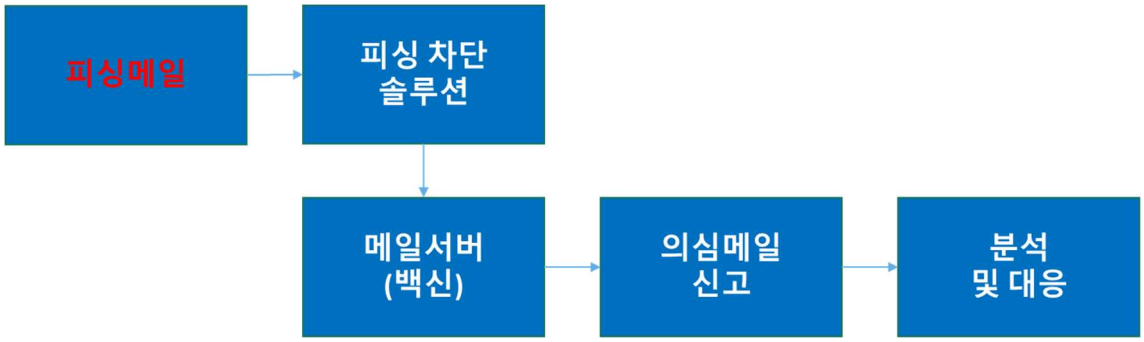
2. 프로젝트 현황

2.1 시장분석

① 대부분의 악성 메일 솔루션들은 스팸을 중점으로 형성됨

② 스팸 + 피싱 솔루션 비용 약 (1 천 ~ 4 천 만원. 기업마다 상이)

<그림 1. 기존 솔루션 동작 흐름도>



2.2 기존 서비스의 장단점 분석(As-is)

① 표적 피싱 공격이 늘어나면서 이메일 보안 솔루션을 통과하여 침투

② 보안 팀이 해당 피싱 공격을 탐지하는데 걸리는 시간은 평균 83 시간으로 보안 위험 증가

③ 피싱 메일 현황

피싱 침투 건수(1천명 기업 기준)	피싱 메일 1 건당 보안 취약 인원 수
한달 평균: 15 건	약 10 명

④ 솔루션을 구매하여 사용하더라도 중소기업의 비용 부담 증가(1 천 ~ 4 천 만원)

2.3 차별화 핵심 전략 기술(To-be)

① 피싱 URL feature 를 분석하여 ML 로 피싱 메일 탐지율 증가

② 피싱 메일 분석 자동화

③ 중소기업의 비용 부담 감소

3. 프로젝트 개발 결과

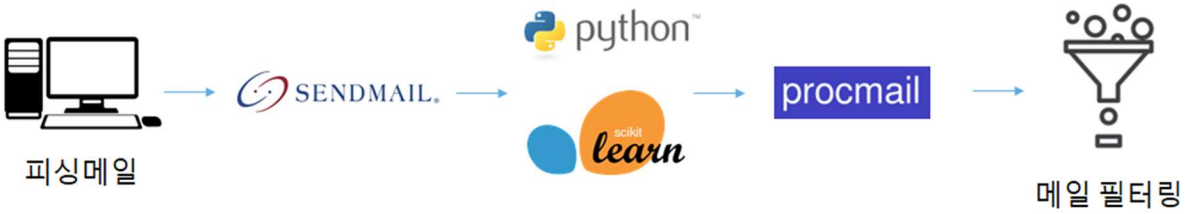
3.1 주요 기능

① 메일 내 피싱 URL 탐지

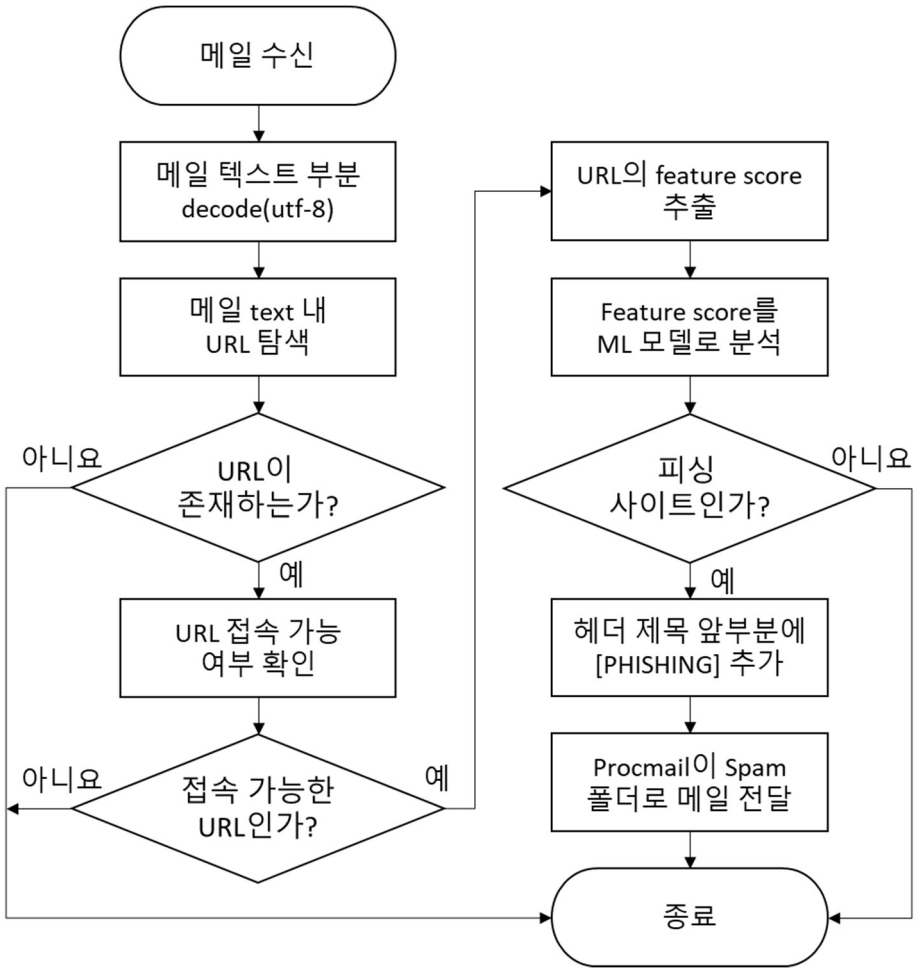
② 피싱 URL 탐지 시 제목에 [PHISHING] 문구 추가 후 Spam 함으로 메일 포워딩

3.2 피싱 메일 URL 탐지 분석서

<그림 2. Tool 동작 흐름도>



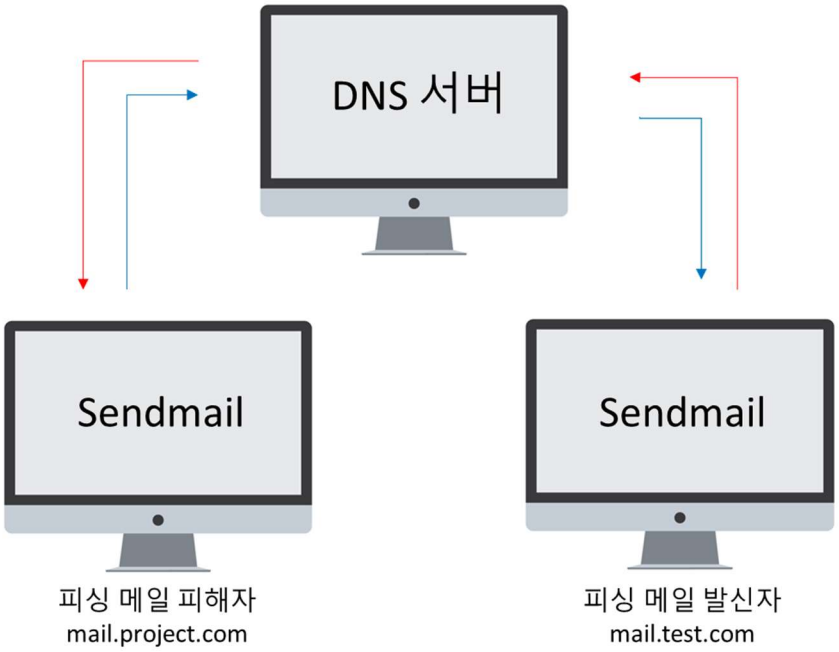
<그림 3. 알고리즘 순서도>



3.3 피싱 메일 URL 탐지에 대한 상세 설명

① DNS 서버 구축

<그림 4. 메일 동작 구성>



- DNS 서버를 통해 메일을 주고받을 수 있는 환경 구성

- mail.project.com(스팸 피해자): 192.168.100.146
- mail.test.com(스팸 발신자): 192.168.100.140
- DNS 서버: 192.168.100.143

- Hosts 파일에 각각의 도메인을 매뉴얼로 설정

```
yum -y install bind bind-chroot

vim /etc/hosts

# project.com 서버
192.168.100.146 mail.project.com

# test.com 서버
192.168.100.140 mail.test.com
```

```
vim /etc/sysconfig/network

# project.com 서버
HOSTNAME=192.168.100.146

# test.com 서버
HOSTNAME=192.168.100.140
```

- DNS 역할을 할 zone 설정

```
# DNS 서버
vim /etc/named.conf

options {

# 나의 어떤 IP 와 포트로 청취할 것인가의 옵션
listen-on port 53 { any; };

# 위 옵션과 같이 IPv6 의 IP 와 포트를 청취할 것인지 설정
listen-on-v6 port 53 { none; };

# 어떤 IP 한테서의 쿼리를 허용할 것인지 설정
allow-query { any; };
};

dnssec-validation no;

# zone 설정
zone "project.com" IN {
        type master;
        file "project.com.db";
        allow-update { none; };
};

zone "test.com" IN {
        type master;
        file "test.com.db";
        allow-update { none; };
};
```

- 레코드 설정

```
# project.com 서버
vi /var/named/project.com.db

$TTL    3H
@       SOA     @       root.   ( 2   1D   1H   1W   1H )
        IN      NS      @
        IN      A       192.168.100.146
        IN      MX      10      mail.project.com.

mail    IN      A       192.168.100.146

# test.com 서버
vi /var/named/test.com.db

$TTL    3H
@       SOA     @       root.   ( 2   1D   1H   1W   1H )
        IN      NS      @
        IN      A       192.168.100.140
        IN      MX      10      mail.test.com.

mail    IN      A       192.168.100.140

# 적용이 잘 되었는지 확인
named-checkconf
named-checkzone project.com project.com.db
named-checkzone test.com test.com.db
```

- 접속을 원활하게 하기 위해 방화벽 정지

```
# DNS 서버 서비스 시작
systemctl restart named
systemctl enable named

# 방화벽 내리기
systemctl stop firewalld
systemctl disable firewalld
```

```
# project.com, test.com 서버
nmcli con mod ens33 ipv4.dns 192.168.100.143
systemctl restart NetworkManager
reboot
```

② Sendmail 구축

```
yum -y install sendmail-cf dovecot
```

- Sendmail config 파일 설정

```
vim /etc/mail/sendmail.cf

아래와 같이 작성
# project.com 서버
85 Cwproject.com
264 O DaemonPortOptions=Port=smtp, Name=MTA

# test.com 서버
85 Cwtest.com
264 O DaemonPortOptions=Port=smtp, Name=MTA
```

- RELAY 메일 전달하는 기능 추가

```
vim /etc/mail/access

# project.com, test.com 서버
project.com        RELAY
test.com           RELAY
192.168.100        RELAY

# 두 서버 모두에서 설정한 내용 적용
makemap hash /etc/mail/access < /etc/mail/access
```

- SSL 기능 추가

```
# 두 서버 모두
vim /etc/dovecot/conf.d/10-ssl.conf
8: ssl = yes
```

- Mail 위치 및 access group 설정

```
vim /etc/dovecot/conf.d/10-mail.conf

# 주석 해제
25 mail_location = mbox:~/mail:INBOX=/var/mail/%u
121 mail_access_groups = mail
166 lock_method = fcntl
```

- Sendmail 서비스 시작

```
systemctl restart sendmail
systemctl enable sendmail
```

③ 모질라 썬더버드 설정

- Incoming 메일은 POP3, outgoing 메일은 SMTP 로 설정하여 계정 설정

④ Pymilter 설정

- Pymilter 는 milter 를 python 으로 설정 가능하도록 만들어진 라이브러리. Pymilter 를 사용하여 메일 필터링 설정

```
# pymilter 다운로드

wget
https://files.pythonhosted.org/packages/ca/b0/0e314563fc802cd7f8f98c858acf5def0ba85acc5fb2
cef6db83d1b70431/pymilter-1.0.4.tar.gz
tar xvf pymilter-1.0.4.tar.gz
cd pymilter-1.0.4/

# setup.py 컴파일을 위해 아래 프로그램 설치
yum install gcc
yum install python-devel
yum install sendmail-devel

python setup.py install
```

<그림 5. Pymilter 파일들>

```
[project@mail pymilter-1.0.4]$ ls
build      MANIFEST.in         mime.pyc       README     testmime.py
ChangeLog  Milter              mysample.py    sample.py  test.py
COPYING    miltermodule.c      NEWS           setup.cfg  testsample.py
CREDITS    milter-template.py  PKG-INFO       setup.py   testutils.py
hey.txt    mime.py             pymilter.spec  test       TODO
```

- sendmail 과 연동하여 메일이 pymilter 를 거칠 수 있도록 설정

```
# sendmail.cf 파일에 아래 두 줄 추가

vim /etc/mail/sendmail.cf

O InputMailFilters=pythonfilter
Xpythonfilter,          S=local:/home/[사용자 이름]/pythonsock

# sendmail 재시작, 정상 작동 확인
systemctl restart sendmail
```

- selinux 가 설정되어 있으면 서비스가 정상 작동하지 않기 때문에 해제

```
# selinux 해제

vim /etc/selinux/config

SELINUX=disabled

# OS 재시작
reboot
```

- sample.py 파일을 사용. (mymilter.py 로 파일 이름 변경)
- body 함수 내에서 바디 텍스트는 utf-8 로 디코딩 하며, re 라이브러리를 사용하여 URL 추출

```
import re

# sample.py 의 body 함수에서 내용 수정
# utf-8 디코딩 텍스트 추출
text = chunk.decode('utf-8').strip()

# http 혹은 https 문자를 가지고 있는 부분을 list 로 저장
urls = re.findall(r'(https?:₩/₩/[^₩s]+)', text)
```

- Feature 추출 이전에 URL 이 동작하는지 확인하는 코드 작성

```
import requests

# body 함수에 아래 내용 작성

```

```
# header 는 website 에서 bot 으로 인지해 접근을 거부하는것을 방지하기 위해 전달되는 정보
header = {'User-Agent' : 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/94.0.4606.61 Safari/537.36'}
for url in urls:
    try:
        # 요청 무한로딩 방지를 위해 timeout 2 초 설정
        res = requests.get(url, headers=header, timeout=2)
    except:
        pass
```

- mymilter.py 는 header 함수와 body 함수를 커스텀하여 사용

```
#__init__ 함수에 아래 변수 추가
self.isphishing = False
self.subject = ""

# header 함수에 아래 내용 추가
if lname == 'subject':
    self.subject = val
```

- 최종 pymilter 코드

```
from __future__ import print_function
import feature_extraction as fe # our own feature module

# Author: Stuart D. Gathman <stuart@bmsi.com>
# Copyright 2001 Business Management Systems, Inc.
# This code is under GPL.    See COPYING for details.

import sys
import os
try:
  from io import BytesIO
except:
  from StringIO import StringIO as BytesIO
import mime
import Milter
import tempfile
from time import strftime
# import tensorflow as tf
```

```
import numpy as np
import re
import requests
import pickle
# from sklearn.externals import joblib
import joblib
from sklearn.metrics import accuracy_score
from sklearn import ensemble
from sklearn.multioutput import MultiOutputClassifier
import warnings
warnings.filterwarnings('ignore')

class myMilter(Milter.Milter):
  "Milter to replace attachments poisonous to Windows with a WARNING message."

  def log(self,*msg):
    print("%s [%d]" % (strftime('%Y%b%d %H:%M:%S'),self.id),end=None)
    for i in msg: print(i,end=None)
    print()

  def __init__(self):
    self.tempname = None
    self.mailfrom = None
    self.fp = None
    self.bodysize = 0
    self.id = Milter.uniqueID()
    self.user = None
    self.isphishing = False
    self.subject = ""

  # multiple messages can be received on a single connection
  # envfrom (MAIL FROM in the SMTP protocol) seems to mark the start
  # of each message.
  @Milter.symlist('{auth_authen}')
  @Milter.noreply
  def envfrom(self,f,*str):
    "start of MAIL transaction"
    self.fp = BytesIO()
```

```
    self.tempname = None
    self.mailfrom = f
    self.bodysize = 0
    self.user = self.getsymval('{auth_authen}')
    self.auth_type = self.getsymval('{auth_type}')
    if self.user:
        self.log("user",self.user,"sent mail from",f,str)
    else:
        self.log("mail from",f,str)
    return Milter.CONTINUE

def envrcpt(self,to,*str):
    # mail to MAILER-DAEMON is generally spam that bounced
    if to.startswith('<MAILER-DAEMON@'):
        self.log('DISCARD: RCPT TO:',to,str)
        return Milter.DISCARD
    self.log("rcpt to",to,str)
    return Milter.CONTINUE

def header(self,name,val):
    lname = name.lower()
    if lname == 'subject':
        self.subject = val

    # log selected headers
    if lname in ('subject','x-mailer'):
        self.log('%s: %s' % (name,val))
    if self.fp:
        self.fp.write(("%s: %s₩n" % (name,val)).encode())          # add header to buffer
    return Milter.CONTINUE

def eoh(self):
    if not self.fp: return Milter.TEMPFAIL      # not seen by envfrom
    self.fp.write(b'₩n')
    self.fp.seek(0)
    # copy headers to a temp file for scanning the body
    headers = self.fp.getvalue()
    self.fp.close()
```

```
        self.tempname = fname = tempfile.mktemp(".defang")
        self.fp = open(fname,"w+b")
        self.fp.write(headers)  # IOError (e.g. disk full) causes TEMPFAIL
        return Milter.CONTINUE

    def body(self,chunk):                   # copy body to temp file
        header = {'User-Agent' : 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/94.0.4606.61 Safari/537.36'}
        if self.fp:
            self.fp.write(chunk)  # IOError causes TEMPFAIL in milter
            self.bodysize += len(chunk)

        # url extraction with decoding
        text = chunk.decode('utf-8').strip()
        urls = re.findall(r'(https?:₩/₩/[^₩s]+)', text)
        print('******')
        print('url list: %s' %urls)
        print('******')

        # url check(get request 200 -> ok, else: error)
        for url in urls:
            try:
                res = requests.get(url, headers=header, timeout=1)
                print("-> %s: Status %s₩n" %(url, res))

                # feature extraction
                feature = fe.FeatureExtraction(url)
                feature_score = np.array([feature.run_process()])
                print('Feature_score')
                print(feature_score)
                print()

                # call model and integrate feature_score
                model = joblib.load('./phishing_model.pkl')
                prediction = model.predict(feature_score)
                print('Prediction result: %d₩n₩n' %int(prediction))
                if int(prediction) == -1:
                    self.isphishing = True
```

```
            break
        except:
            print('url not found or timed out')
            pass
    print('******')

    return Milter.CONTINUE

def _headerChange(self,msg,name,value):
    if value:       # add header
        self.addheader(name,value)
    else:# delete all headers with name
        h = msg.getheaders(name)
        cnt = len(h)
        for i in range(cnt,0,-1):
            self.chgheader(name,i-1,'')

def eom(self):
    if not self.fp: return Milter.ACCEPT
    if self.isphishing == True:
        self.addheader('subject', '[PHISHING]'+self.subject, idx=-1)
    self.fp.seek(0)
    msg = mime.message_from_file(self.fp)
    msg.headerchange = self._headerChange
    if not mime.defang(msg,self.tempname):
        os.remove(self.tempname)
        self.tempname = None        # prevent re-removal
        self.log("eom")
        return Milter.ACCEPT        # no suspicious attachments
    self.log("Temp file:",self.tempname)
    self.tempname = None            # prevent removal of original message copy
    # copy defanged message to a temp file
    with tempfile.TemporaryFile() as out:
        msg.dump(out)
        out.seek(0)
        msg = mime.message_from_file(out)
        fp = BytesIO(msg.as_bytes().split(b'₩n₩n',1)[1])
        while 1:
```

```
            buf = fp.read(8192)
            if len(buf) == 0: break
            self.replacebody(buf)       # feed modified message to sendmail
        return Milter.ACCEPT            # ACCEPT modified message
    return Milter.TEMPFAIL

  def close(self):
    sys.stdout.flush()                  # make log messages visible
    if self.tempname:
      os.remove(self.tempname)    # remove in case session aborted
    if self.fp:
      self.fp.close()
    return Milter.CONTINUE

  def abort(self):
    self.log("abort after %d body chars" % self.bodysize)
    return Milter.CONTINUE

if __name__ == "__main__":
  #tempfile.tempdir = "/var/log/milter"
  #socketname = "/var/log/milter/pythonsock"
  socketname = os.getenv("HOME") + "/pythonsock"
  Milter.factory = myMilter
  Milter.set_flags(Milter.CHGBODY + Milter.CHGHDRS + Milter.ADDHDRS)
  print("""To use this with sendmail, add the following to sendmail.cf:

O InputMailFilters=pythonfilter
Xpythonfilter,            S=local:%s

See the sendmail README for libmilter.
Milter startup""" % socketname)
  sys.stdout.flush()
  Milter.runmilter("pythonfilter",socketname,240)
  print("sample milter shutdown")
```

⑤ Feature 추출

- URL 의 특징을 파악하여 Feature 점수를 부여하는 방식. 총 13 개의 feature 를 파악

- Library 호출

```
from bs4 import BeautifulSoup
import requests
import urllib
import whois
import ssl, socket
from dateutil.parser import parse
from tld import get_tld
```

- 클래스 선언 및 사용할 함수 정의(1 == 정상, 0 == 피싱 의심, -1 ==피싱 사이트로 점수 계산)

```
# 클래스 선언
class FeatureExtraction:
    def __init__(self, url):
        self.url = url
        self.feature_list = []
        self.header = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/94.0.4606.61 Safari/537.36'}

    # phishing = -1, suspicious = 0, normal = 1

    def get_domain_url(self, url):
        domain = ''
        cnt = 0
        for i in url:
            if i == '/':
                cnt += 1
            if cnt == 3:
                break
            if cnt == 2:
                domain += i
        domain = domain[1:]
        return domain
```

- 1 번째 feature. 문자열 길이가 75 보다 길면 피싱으로 판단

```
def count_characters(self):
    if len(self.url) > 75: # when phishing
        self.feature_list.append(-1)
    elif len(self.url) >= 54 and len(self.url) < 75:   # when suspicious
        self.feature_list.append(0)
    else: # when not fishing
        self.feature_list.append(1)
```

- 2 번째 feature. URL 에 '@' 문자 포함 여부 확인

```
def contain_at(self):
    if '@' in self.url:
        self.feature_list.append(-1)
    else :
        self.feature_list.append(1)
```

- 3 번째 feature. URL 에 '-' 문자 포함 여부 확인

```
def contain_dash(self):
    if '-' in self.url:
        self.feature_list.append(-1)
    else :
        self.feature_list.append(1)
```

- 4 번째 feature. URL 에 'https.' 포함 여부 확인

```
def contain_HTTPS_dot(self):
    temp = self.url.lower()
    if 'https.' in temp:
        self.feature_list.append(-1)
    else:
        self.feature_list.append(1)
```

- 5 번째 feature. 외부 요청 URL 비율 확인(사이트의 link, href 태그가 외부 URL 로 요청을 하는지 파악)

```
def find_link_href(self):
    try:
        html = requests.get(url, headers=self.header, timeout=4.0).text
        soup = BeautifulSoup(html, "html.parser")
        find_link = soup.find_all('link')
```

```
                find_href = soup.find_all('href')
                link_cnt = 0
                href_cnt = 0

                for i in range(len(find_link)):
                    if self.get_domain_url(self.url) not in str(find_link[i]):
                        link_cnt += 1

                for i in range(len(find_href)):
                    if self.get_domain_url(self.url) not in str(find_href[i]):
                        href_cnt += 1

                if self.link_in_href_rate(link_cnt, href_cnt) == 0:
                    feature5 = 1
                elif self.link_in_href_rate(link_cnt, href_cnt) >= 61:
                    feature5 = -1
                elif self.link_in_href_rate(link_cnt, href_cnt) >= 22 and self.link_in_href_rate(link_cnt,
href_cnt) < 61:
                    feature5 = 0
                else:
                    self.feature_list.append(1)
            except requests.exceptions.Timeout:
                feature5 = 0
            except requests.exceptions.TooManyRedirects:
                feature5 = -1
            except requests.exceptions.RequestException:
                feature5 = -1
            except:
                feature5 = -1
            self.feature_list.append(feature5)


        def link_in_href_rate(self, link_cnt,href_cnt):
            if link_cnt ==0 and href_cnt ==0:
                return 0
            elif href_cnt >= 1 and link_cnt ==0:
                return 70
            else:
```

```
            link_in_href_rate_result = (href_cnt/link_cnt)*100
        return link_in_href_rate_result
```

- 6 번째 feature. HTML 의 소스 코드 길이를 확인

```
def domain_in_length(self):
    try:
        html = requests.get(self.url, headers=self.header, timeout=5).text
        if len(html) < 5000:
            feature6 = -1
        elif len(html) < 50000:
            feature6 = 0
        else:
            feature6 = 1
    except:
        feature6 = -1
    self.feature_list.append(feature6)
```

- 7 번째 feature. SSL 인증서 검증

```
def SSLfinal_State(self):
    try:
        domain = self.get_domain(self.url)
        s = self.https_connect(domain)
        if s == 0:
            feature7 = -1
        else :
            cert = s.getpeercert()
            issuer = dict(x[0] for x in cert['issuer'])
            issued_by = issuer['organizationName']

            trusted_issuer_list = self.get_trusted_issuer()
            for trusted_issuer in trusted_issuer_list:
                if trusted_issuer == issued_by:
                    break
                else:
                    feature7 = 1

            notAfter = cert['notAfter']
            notBefore = cert['notBefore']
```

```
                init_date = parse(notBefore)
                expiration_date = parse(notAfter)
                total_days = (expiration_date.date() - init_date.date()).days
                if total_days >= 365:
                    feature7 = -1
                else:
                    feature7 = 1
        except:
            feature7 = -1
        self.feature_list.append(feature7)

    def https_connect(self):
        try:
            TIMEOUT = 4.0
            socket.setdefaulttimeout(TIMEOUT)
            ctx = ssl.create_default_context()
            s = ctx.wrap_socket(socket.socket(), server_hostname=self.url)
            s.connect((self.url, 443))
            return s
        except:
            s = 0
            return s

    def get_trusted_issuer(self):
        f = open("CA_list.txt", "r")

        trusted_issuer = []
        for line in f:
            issuers = line.strip('₩n')
            trusted_issuer.append(issuers)
        return trusted_issuer
```

- 8 번째 feature. 도메인 생성 기간으로 탐지

```
    def domain_registration_period(self):
        try:
            total_date = self.get_total_date(self.url)
            if total_date <= 365:
                feature8 = -1
```

```
            else:
                feature8 = 1
        except:
            feature8 = -1
        self.feature_list.append(feature8)

    def get_total_date(self):
        try:
            TIMEOUT=3.0
            socket.setdefaulttimeout(TIMEOUT)
            domain = whois.whois(self.url)
        except:
            return 0
        if domain.expiration_date is None:
            return 0
        if domain.updated_date is None:
            return 0
        if type(domain.expiration_date) is list:
            expiration_date = domain.expiration_date[0]
        else:
            expiration_date = domain.expiration_date

        if type(domain.updated_date) is list:
            updated_date = domain.updated_date[0]
        else:
            updated_date = domain.updated_date

        total_date = (expiration_date - updated_date).days
        return total_date
```

- 9 번째 feature. Alexa 랭킹 기준으로 탐지

```
    def check_alexa_rank(self):
        try:
            domain = self.get_domain(self.url)
            rank_str =
BeautifulSoup(urllib.request.urlopen("http://data.alexa.com/data?cli=10&dat=s&url="+
domain).read(), 'xml').find("REACH")['RANK']
            rank_str1 = int(rank_str)
```

```
            if rank_str1 > 100000:
                feature9 = 1
            else:
                feature9 = -1
        except:
            feature9 = -1
        self.feature_list.append(feature9)
```

- 10 번째 feature. 사이트 포트 열림 확인

```
#10 사이트 포트 열림 확인
    def port_open_check(self):
        feature10 = 1
        try:
            ip = self.get_domain(self.url)
        except:
            feature10 = -1

        ports = [80, 21, 22, 23, 443, 445, 1433, 1521, 3306, 3389]
        for port in ports :
            socket.setdefaulttimeout(1.5)
            s = socket.socket()
            if port == 80 and 443 :
                try:
                    s.connect((ip, port))
                    s.close()
                    feature10 = 1
                except:
                    feature10 = -1
            else:
                try :
                    s.connect((ip, port))
                    s.close()
                    feature10 = -1
                except :
                    feature10 = 1
                    pass
        self.feature_list.append(feature10)
```

- 11 번째 feature. 서브 도메인 개수 파악

```
def having_sub_domain(self):
    try:
        url = self.remove_www(self.url)
        domain = get_tld(url, as_object=True)
        dot = domain.subdomain.count('.')
        if domain.subdomain == "":
            feature11 = 1
        elif dot == 0:
            feature11 = 0
        else:
            feature11 = -1
    except:
        feature11 = -1
    self.feature_list.append(feature11)

def remove_www(self):
    if "www." in self.url[:12] :
        url = self.url.replace("www.", "")
    return url
```

- 12 번째 feature. '//'를 사용하는 리다이렉션 체크

```
def double_slash_redirecting(self):
    parse = urllib.parse.urlparse(self.url)
    path = parse.path
    if '//' in path:
        feature12 = -1
    else:
        feature12 = 1
    self.feature_list.append(feature12)
```

- 13 번째 feature. URL 단축 서비스

```
def shortening_service(self):
    try:
        response = requests.get(self.url,headers=self.header,timeout = 3)
        if response.status_code== 301 or response.status_code == 302   :
            feature13 = -1
        else:
```

```
                feature13 = 1
        except:
            feature13 = -1
        self.feature_list.append(feature13)
```

- 실행 함수. 스코어를 feature_list 에 담아 반환

```
    def run_process(self):
        self.count_characters() # 1
        self.contain_at() # 2
        self.contain_dash() # 3
        self.contain_HTTPS_dot() # 4
        self.find_link_href() # 5
        self.domain_in_length() # 6
        self.SSLfinal_State() # 7
        self.domain_registration_period() # 8
        self.check_alexa_rank() # 9
        self.port_open_check() # 10
        self.having_sub_domain() # 11
        self.double_slash_redirecting() # 12
        self.shortening_service() # 13

        return self.feature_list
```

⑥ 머신러닝 모델 학습

- 2 만개(1 만개 정상, 1 만개 피싱) URL 데이터를 사용하여 랜덤포레스트 모델링 적용

- 랜덤포레스트를 구현할 라이브러리 호출

```
from sklearn.metrics import accuracy_score
import numpy as np
from sklearn import ensemble
from sklearn.multioutput import MultiOutputClassifier
import tld
```

- 데이터 호출 및 인덱스 순서 섞기

```
training_data = np.genfromtxt("./test.csv", delimiter=',' , dtype = np.int32 )

np.random.shuffle(training_data)
```

- 트레이닝 데이터과 정답 데이터 분리

```
inputs = training_data[:,:-1] #트레이닝
outputs = training_data[:,-1] #정답

training_inputs = inputs[: 16000]
training_outputs = outputs[: 16000]
testing_inputs = inputs[16000 :]
testing_outputs = outputs[16000 :]
```

- 앙상블 랜덤포레스트 적용 코드

```
# 의사 결정 가지 수: 20
classifier = ensemble.RandomForestClassifier(n_estimators=20)
classifier.fit(training_inputs, training_outputs)

predictions = classifier.predict(testing_inputs)

accuracy = 100.0 * accuracy_score(testing_outputs,predictions)
print("RandomForest accuracy = ", str(accuracy))
```

<그림 6. 모델 정확도 결과 확인>

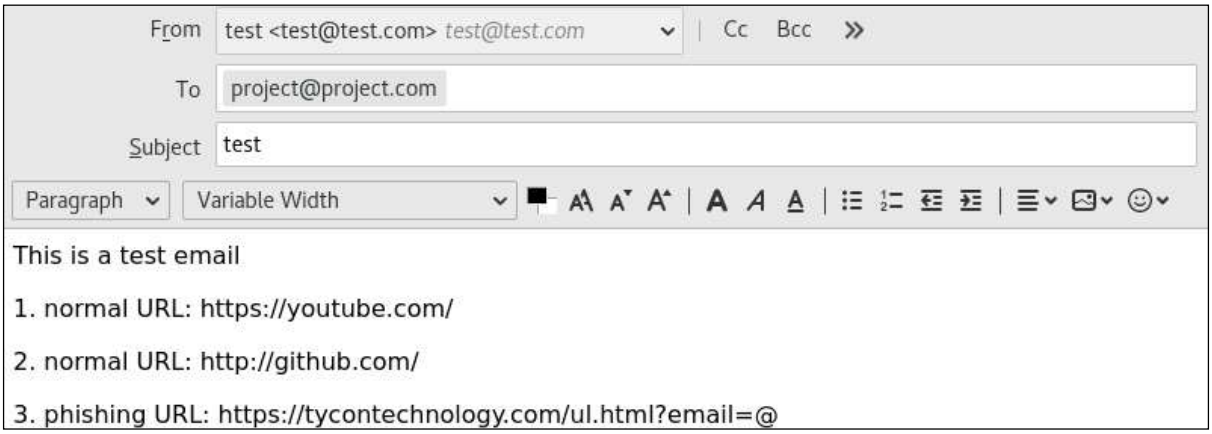
```
predictions = classifier.predict(testing_inputs)

accuracy = 100.0 * accuracy_score(testing_outputs,predictions)
print("RandomForest accuracy = ", str(accuracy))
```

```
RandomForest accuracy =  95.52238805970148
```

- 동작 확인

<그림 7. test 서버의 test 계정에서 피싱 메일 발송>



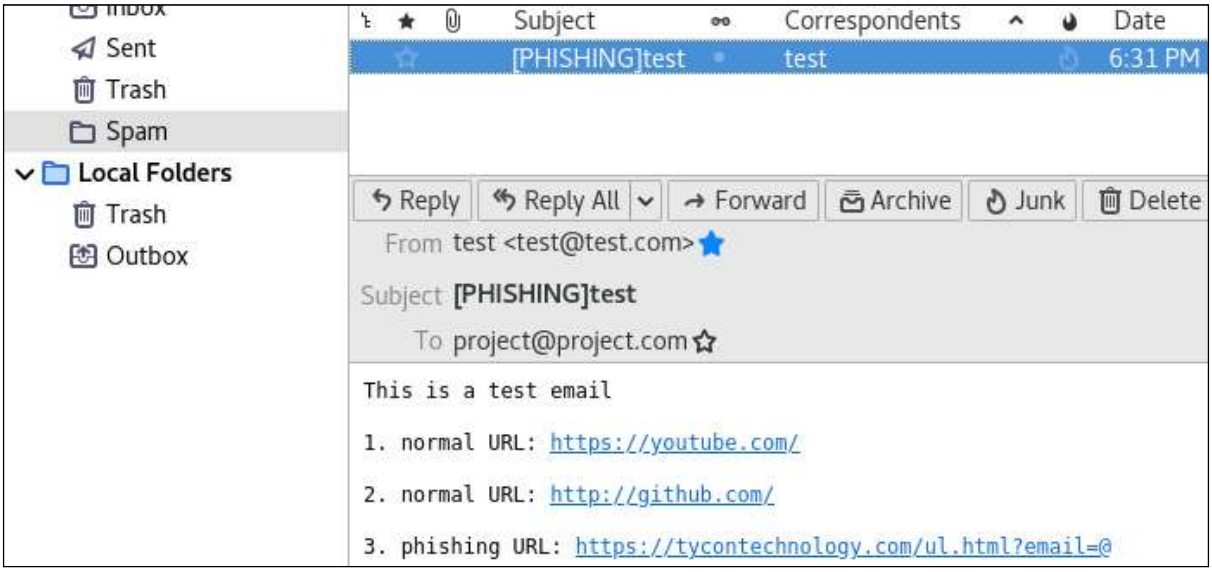
<그림 8. project 서버의 project 계정에서 피싱 URL 판별>

```
******
url list: ['https://youtube.com/', 'http://github.com/', 'https://tycontechnology.c
om/ul.html?email=@']
******
-> https://youtube.com/: Status <Response [200]>
Feature_score:  [[ 1  1  1  1  1  1 -1 -1 -1  1 -1  1  1]]
Prediction result: 1

-> http://github.com/: Status <Response [200]>
Feature_score:  [[ 1  1  1  1  1  1 -1 -1 -1  1 -1  1  1]]
Prediction result: 1

-> https://tycontechnology.com/ul.html?email=@: Status <Response [404]>
Feature_score:  [[ 1 -1  1  1  1 -1 -1 -1 -1  1 -1  1  1]]
Prediction result: -1
```

<그림 9. Project 계정의 스팸 메일함>



3.4 활용 방안

① 중소기업의 메일 서버와 연동하여 피싱 메일 분류 자동화

② 기업 내 메일 어플리케이션을 지원하는 회사에서 활용 가능

4. 기대 효과

4.1 기대 효과

① 피싱 메일 URL 분석 자동화
② 분류된 피싱 메일 수집 및 데이터 활용
③ 기업끼리의 메일을 통한 무역 사기를 사전에 방지
④ 중소기업의 비용 부담 감소

4.2 향후 개선 사항

① 다중 사용자가 하나의 메일 서버에서 서비스를 사용 가능하도록 코드 추가 필요
② 피싱 메일 탐지 시 메일 바디부분 경고문 삽입 기능 추가
③ 스팸 필터링 기능을 추가하여 스팸/피싱 차단 서비스로 업그레이드 방법 모색

5. 개발 후기

성명	후기
정병헌	공부하는 마음으로 프로젝트를 시작하게 되서 정말 많이 도움이 되었습니다. 특히 논문을 찾아보는 능력과 모르는 부분이 있으면 찾아내는 능력을 키울 수 있어서 좋았습니다. 또한 팀의 리더로써 어떻게 팀원들을 관리하고 프로젝트를 진행해야 하는지 알 수 있어서 좋았습니다.
연람희	서버 구축과 개발을 하며 실제 상용 프로그램을 다룰 수 있겠다는 자신감이 들었습니다. 그리고 인공지능 모델과 서비스를 연동하는 기술을 더 익혀야겠다고 느낌과 동시에 인공지능 모델 구축 또한 공부가 많이 필요하다고 생각되었습니다. 이 프로젝트의 연장선으로 다른 여러 가지 프로젝트들에 대한 영감을 얻을 수 있었습니다. 제 자신의 능력을 한껏 끌어낼 수 있었던 프로젝트였습니다. 더 구체적으로 구현하지 못해 아쉬움이 남는 부분도 있지만 프로젝트가 끝까지 마무리될 수 있었고 성취라는 결과를 주었기에 많이 기억에 남을 것으로 생각됩니다. 팀원분들 모두 고생 많으셨습니다.
권채연	개발을 하면서 리서치를 통해 프로젝트 주제에 대한 이해도를 높이고 필요성을 알게 되었습니다.