Detectarea potențialelor aplicații malițioase prin analiza comportamentului utilizatorilor pe social media

31.05.2018

Olarasu Loredana

Stoleru Ingrid

Unghianu Anda

Vasilache Mihai

Facultatea de Informatica, Universitatea “Alexandru Ioan Cuza”

Overview

Scopul acestui proiect constă în a realiza o analiză a comportamentului utilizatorilor pe social-media, pentru a putea deduce pe baza acestei analize potențialul unui user de a face release la aplicații malițioase pe platforme open source. În particular, am ales să analizăm tweet-urile unui utilizator și să realizăm o analiză sentimentală pe baza acelor comentarii. Ulterior, oferim utilizatorului aplicației posibilitatea de a descărca de pe Github codul uploadat de emițătorul tweet-urilor și de a realiza o scanare a fișierelor, în urma căreia se va oferi un verdict asupra fișierelor.

Studii pe marginea subiectului

Specificații

În vederea implementării funcționalităților aferente, am realizat requesturi la mai multe api-uri. Pentru a vizualiza tweet-urile unui utilizator, am realizat requesturi la Twitter. În ceea ce privește analiza sentimentală a tweeturilor, am utilizat librăria TextBlob pentru procesarea textelor. Am realizat, de asemenea, requesturi la Github pentru a descărca sursele acestor utilizatori, pe care le-am stocat în cloud, prin intermediul Azure Storage Service. Nu în ultimul rând, am realizat requesturi la Virus Total, pentru a realiza o analiză a fișierelor descărcate.

Detalii de implementare

I. Overview

Proiectul este realizat în framework-ul Django, iar pentru partea de front-end am utilizat Bootstrap. Pentru implementarea funcționalităților de bază am realizat requesturi la Github, Twitter, Virus Total și Azure Storage Service. Aplicația a fost deployata pe Azure. Pentru serviciul de autentificare, respectiv stocare a informațiilor, am utilizat baza de date integrată în Django.

II. Funcționalități

1. **Logare si Signup**

La prima pornire a aplicației, userul are posibilitatea de a-și crea un cont nou sau de a se loga, utilizând contul de Google. Ulterior creării contului, acesta poate să se logheze cu datele contului nou creat. La nivel de implementare, am folosit componentele de autentificare din Django. La signup, utilizatorul trebuie să ofere datele specifice formului clasic din Django, respectiv userul, numele, emailul și parola. Pentru autentificarea prin contul Google am utilizat *Google OAuth* din librăria *python-social-auth*. Userul poate intra în contul propriu și în consecință, accesa următoarele pagini ale aplicației doar dacă este autentificat.

User model:

```
class Meta:
model = User
fields = ('username', 'first_name', 'last_name', 'email', 'password1', 'password2', )
```

Functia de login:

```
def post(self, request, **kwargs):
  username = request.POST['username']
  password = request.POST['password']
  user = authenticate(request, username=username, password=password)
  if user is not None:
        login(request, user)
        return render(request, "home.html")
  else:
        return render(request, self.template_name)
```

2. **Tweets Retrieval**

Pentru a face retrieve la tweet-urile unui utilizator, numele accountului aferent este necesar. Acesta este introdus din interfață, din secțiunea *myaccount*. Libraria twitter din python conține functii ce pot fi apelate pentru extragerea informațiilor aferente utilizatorilor. Extragerea tweet-urilor a fost realizată utilizând comanda: **t = api.GetUserTimeline(screen_name, count)**, unde screen_name reprezintă numele accountului, iar count numărul de tweeturi ce vor fi retrieved. Pentru ca requestul să fie autorizat, aplicația a fost înregistrată pe Twitter Developer Platform, iar Twitter requestul a primit ca parametri: consumer key-ul, secret key-ul,

access_token key-ul si access_token secret-ul. Outputul requestului va fi afișat într-o tabelă definită în bootstrap:

<table id="item_table_2" class="table table-striped table-sm table-hover table-bordered">

<tbody class="table_body_2" id = "tbody_2" style="display:block; height:280px; overflow-y:auto; overflow-x:auto" >{% include 'table_body.html' %}</tbody></table>

3. Sentimental analysis

Pentru simplificarea procesului de analiză a sentimentelor pe baza contentului primit de la Twitter, am utilizat o librărie dedicată procesării de texte, respectiv *TextBlob*. Analizatorul se bazează pe algoritmul Bayes Naiv, clasificând procentul de pozitivitate și negativitate specific tweetului analizat, în funcție de modul în care a apărut acea componentă în corpusul de antrenament. Utilizatorul trebuie să introducă din interfață tweetul ce urmează să fie analizat. În urma analizei, va fi dat un verdict asupra tweetului, ce poate fi pozitiv, negativ sau neutru. Acest rezultat va fi afișat într-un modal.

Funcția de sentimental analysis a unui tweet:

```
def cleanTweet(tweet):
        return ' '.join(re.sub("(@[A-Za-z0-9]+)|([^0-9A-Za-z \t])|(\w+:\/\/\S+)", " ", tweet).split())
def getTweetSentiment(tweet):
  if tweet == "nada":
      return 'No tweet found! Please provide with us with a tweet!'
  analysis = TextBlob(cleanTweet(tweet))
  if analysis.sentiment.polarity > 0:
    return 'This denotes positive feelings!:)'
  elif analysis.sentiment.polarity == 0:
    return 'This denotes neutral feelings!'
  else:
    return 'This denotes negative feelings!:('
```

4. Github files retrieval

Pentru funcția de files retrieval, s-a realizat un request la Github. În prealabil, aplicația a fost înregistrată pe Github Developer API, astfel încât requestul a primit ca parametri key-ul generat la înregistrare și un token generat din interfata de Github. Utilizatorul aplicației trebuie să introducă din interfață numele accountului, numele proiectului ce va fi downloadat și limita de fișiere. Aceste fișiere vor fi stocate în cloud, funcționalitate ce a fost implementată prin Azure Storage Service. De asemenea, pentru acest serviciu, a fost necesar un token, ce a fost dat ca parametru la request.

=== Funcția de retrieve a proiectelor unui user ===

```
def getUserProjects(user):
resp = requests.get('https://api.github.com/users/{}/repos'.format(user),
headers={'Authorization': 'token {}'.format(OAUTH_TOKEN)})
    if resp.status_code != 200:
        return []
    resp = resp.json()
    projects = [project['name'] for project in resp]
    return projects
```

=== Funcția de retrieve a commit-urilor unui utilizator ===

```
def getProjectLastCommit(user, project):
resp = requests.get('https://api.github.com/repos/{}/{}/commits'.format(user, project),
headers={'Authorization': 'token {}'.format(OAUTH_TOKEN)})
    if resp.status_code != 200:
        return []
    resp = resp.json()
    commit = resp[0]['sha']
    return commit
```

```
=== Funcția de retrieve a tuturor numelor de fișiere din cadrul unui proiect ===
def getProjectFilenames(user, project, limit):
    sha = getProjectLastCommit(user, project)
    files = []
    getProjectFilenamesRecursively(user, project, sha, files, limit)
    return files

def getProjectFilenamesRecursively(user, project, sha, files, limit):
    if len(files) == limit:
        return
    resp = requests.get('https://api.github.com/repos/{}/{}/git/trees/{}'.format(user, project,
sha), headers={'Authorization': 'token {}'.format(OAUTH_TOKEN)})
    if resp.status_code == 200:
        resp = resp.json()
        tree = resp['tree']
        for content in tree:
            if len(files) == limit:
                return
            if content['type'] == 'blob':
                files.append(content['path'])
            elif content['type'] == 'tree':
                dir_sha = content['sha']
                getProjectFilenamesRecursively(user, project, dir_sha, files, limit)

def getFileContent(url):
    resp = requests.get(url)
    if resp.status_code != 200:
        return ''
```

```
    resp = resp.json()
    if resp['encoding'] == 'base64':
        return base64.b64decode(resp['content'])
    else:
        print(resp['encoding'])
        return resp['content']
```

=== Extragerea tuturor fișierelor din cadrul unui proiect ===

```
def getProjectFiles(user, project, limit):
    sha = getProjectLastCommit(user, project)
    files = []
    getProjectFilesRecursively(user, project, sha, files, limit)
    for i in range(len(files)):
        content = getFileContent(files[i]['url'])
        files[i]['content'] = content
    return files
```

```
def getProjectFilesRecursively(user, project, sha, files, limit):
    resp = requests.get('https://api.github.com/repos/{}/{}/git/trees/{}'.format(user, project,
sha), headers={'Authorization': 'token {}'.format(OAUTH_TOKEN)})
    if resp.status_code == 200:
        resp = resp.json()
        tree = resp['tree']
        for content in tree:
            if len(files) == limit:
                return
            if content['type'] == 'blob':
                files.append({'path': content['path'], 'url': content['url'], 'size': content['size'], 'sha':
content['sha']})
            elif content['type'] == 'tree':
```

```
            dir_sha = content['sha']
            getProjectFilesRecursively(user, project, dir_sha, files, limit)
```

=== Funcția de storage ===

```
def saveProjectFilesToAzure(user, project, limit):
    def saveProjectFilesRecursively(user, project, sha, path, files, dirName, limit):
        print('https://api.github.com/repos/{}/{}/git/trees/{}'.format(user, project, sha))
        resp = requests.get('https://api.github.com/repos/{}/{}/git/trees/{}'.format(user,
project, sha), headers={'Authorization': 'token {}'.format(OAUTH_TOKEN)})
        print(resp.status_code)
        if resp.status_code == 200:
            resp = resp.json()
            tree = resp['tree']
            for content in tree:
                if len(files) == limit:
                    return
                if path != '':
                    newPath = '{}\\{}'.format(path, content['path'])
                else:
                    newPath = content['path']
                if content['type'] == 'blob':
                    if content['path'] in [".gitignore", "LICENSE", "README.md"]:
                        continue
                    print("here" + newPath)
                    files.append(newPath)
                    saveFile(newPath.replace('\\', '-'), getFileContent(content['url']), dirName)
                elif content['type'] == 'tree':
                    dir_sha = content['sha']
                    saveProjectFilesRecursively(user, project, dir_sha, newPath, files, dirName, limit)
    sha = getProjectLastCommit(user, project)
```

```
    print('sha: ', sha)
    files = []
    dirName = '{}-{}'.format(user, project)
    createDir(dirName)
    saveProjectFilesRecursively(user, project, sha, '', files, dirName, limit)
    return files

def getAzureFileReport(githubUser, githubProject, fileName):
    def scan(fileName, fileText, wait=False):
resp =
requests.post('https://www.virustotal.com/vtapi/v2/file/scan?apikey={}'.format(apiKey),
files={'file': (fileName, fileText)})
        if resp.status_code == 204:
            return 'api limit'
        if resp.status_code != 200:
            return None
        resp = resp.json()
        if resp['response_code'] != 1:
            return None
        fileSha = resp['sha1']
        if wait == True:
            return getReportWait(fileName, fileText)
        return 'loading'

    def getReportNoWait(fileName, fileText):
        h = hashlib.sha1()
        h.update(fileText)
        fileSha = h.hexdigest()
```

```
resp =
requests.post('https://www.virustotal.com/vtapi/v2/file/report?apikey={}&resource={}'.form
at(apiKey, fileSha))
        if resp.status_code == 204:
            return 'api limit'
        if resp.status_code != 200:
            return None
        resp = resp.json()
        if resp['response_code'] == 1:
            return resp
        elif resp['response_code'] == -2:
            return 'loading'
        else:
            return scan(fileName, fileText, False)
    dirName = '{}-{}'.format(githubUser, githubProject)
    file_ = getFile(dirName, fileName.replace('\\', '-'))
    if file_ == None:
        return 'file not in azure'
    fileText = file_.content.encode('utf-8')
    f = True
    while f:
        report = getReportNoWait(fileName, fileText)
        if report in ['api limit', 'loading']:
            time.sleep(20)
            continue
        f = False
    if report['positives'] > report['total'] // 2:
        return 'infected'
    return 'clean'
```

5. Malware analysis

Userul va oferi ca input numele fișierului pe care dorește să îl scaneze, acesta putând fi ales din lista de fișiere ce au fost downloadate de pe Github. Funcționalitățile sunt în cascadă două câte două, astfel încât outputul primei funcționalități (tweets retrieval) reprezintă inputul celei de a doua (sentimental analysis), iar outputul celei de a treia funcționalități (Github files retrieval) reprezintă inputul celei de a patra (malware analysis). Analiza se realizează printr-un request la Virus Total. Asemenea celorlalte API-uri, acesta a necesitat un API key. În cazul în care fișierul scanat nu există în baza de date Virus Total, raportul este oferit cu un delay.

```
def getReportWait(filePath, fileSha=None):
    if not fileSha:
        h = hashlib.sha1()
        with open(filePath, 'rb', buffering=0) as f:
            for b in iter(lambda : f.read(128*1024), b''):
                h.update(b)
        fileSha = h.hexdigest()
resp =
requests.post('https://www.virustotal.com/vtapi/v2/file/report?apikey={}&resource={}'.form
at(apiKey, fileSha))
    if resp.status_code == 204:
        time.sleep(30)
        return getReportWait(filePath, fileSha)
    if resp.status_code != 200:
        return None
    resp = resp.json()
    if resp['response_code'] == 1:
        return resp
    elif resp['response_code'] == -2:
        time.sleep(20)
```

```
        return getReportWait(filePath, fileSha)
    else:
        return scan(filePath, True)
```

=== Funcția de scanare ===

def scan(filePath, wait=False):

```
resp =
requests.post('https://www.virustotal.com/vtapi/v2/file/scan?apikey={}'.format(apiKey),
files={'file': (os.path.basename(filePath), open(filePath, 'rb'))})
    if resp.status_code == 204:
        return 'api limit'
    if resp.status_code != 200:
        return None
    resp = resp.json()
    if resp['response_code'] != 1:
        return None
    fileSha = resp['sha1']
    if wait == True:
        return getReportWait(filePath)
    return 'loading'
```

==== Funcția de preluare a răspunsului de la VT, în cazul în care fișierul este deja în baza de date ===

def getReportNoWait(filePath):

```
    h = hashlib.sha1()
    with open(filePath, 'rb', buffering=0) as f:
        for b in iter(lambda : f.read(128*1024), b''):
            h.update(b)
    fileSha = h.hexdigest()
```

```
resp =
requests.post('https://www.virustotal.com/vtapi/v2/file/report?apikey={}&resource={}'.format(apiKey, fileSha))
    if resp.status_code == 204:
        return 'api limit'
    if resp.status_code != 200:
        return None
    resp = resp.json()
    if resp['response_code'] == 1:
        return resp
    elif resp['response_code'] == -2:
        return 'loading'
    else:
        return scan(filePath, False)
```

==== Funcția de preluare a răspunsului de la VT, în cazul în care fișierul nu este în baza de date (așteptarea rezultatului) ===

```
def getFileReport(filePath):
    f = True
    while f:
        report = getReportNoWait(filePath)
        if report in ['api limit', 'loading']:
            time.sleep(20)
            continue
        f = False
    if report['positives'] > report['total'] // 2:
        return 'infected'
    return 'clean'
```