به نام خدا

این پروژه از دو مرحله تشکیل شده است:

Pre Process
Main Process

Pre Process

در این مرحله با استفاده از داده های train ، الگوها و pattern های ویروس شناسایی میشوند
به این ترتیب که الگوهایی که در فایل های ویروس تکرار شده اند و در فایل های بی خطر وجود ندارند به عنوان pattern های ویروس شناخته شده و در فایل Viruses.txt نوشته میشوند.
چند نکته لحاظ میشود:
برای صرفه جویی در زمان و حافظه، صرفا الگوها نیازی نیست که در فایل های بی خطر به تعداد صفر بار تکرار شوند و کمی خطا را نیز پذیرفته ایم.
برای صرفه جویی در زمان و حافظه و افزایش کارایی برنامه، الگوهایی که در بیش از 20 درصد فایل های ویروس یک فولدر تکرار می شوند به عنوان پترن های ویروس آن دسته بندی (فولدر) شناخته می شوند.
هر فولدر حاوی فایل های ویروس حاوی الگوهای خاص و مشابه خود می باشند ، بدین ترتیب فایل های دریافتی در مرحله ی main process ممکن است حاوی هر یک از این دسته بندی ویروس ها باشند.
با بررسی و تحلیل فایل های ویروس train متوجه میشویم که تنها بخشی از آن حاوی اطلاعات مفید می باشد به طوری که تقریبا از نیمه ی دوم فایل ها، پترن و الگوی جدید یافت نمی شود و اطلاعات مفیدی وجود ندارد یا اگر هم وجود دارد به دلیل برتری زمان و هزینه و سرعت(trade off بین زمان و دقت) از آن صرف نظر میشود.
با استفاده از آرایه ی Virus_Patterns الگوهای ویروسی که شرایط ذکر شده را دارا باشند، ذخیره می کنیم و در فایل Viruses.txt ذخیره می شوند.
با استفاده از تابع Searching_for_virus که دو آرگمان directory (آدرس فولدر های فایل های ویروسی به عنوان آموزش و یافتن الگوهای ویروسی) و Virus_Patterns (آرایه ی الگوها برای ذخیره ی الگو های ویروس) ورودی می گیرد و در نهایت الگوهای ویروس را در Viruses.txt می نویسد.

```
# Pre-Process

Virus_patterns = []
directory = "C:\git\Anti_Virus\Train\Malware Sample\\"
find_pattern(directory, Virus_patterns)
```

در تابع find_pattern با پیمایش 20 فولدر فایل های ویروسی، تمام الگوهایی را که با شرایط را ارضا میکنند، پیدا کرده و در آرایه ذخیره می کنیم.
از آنجایی که هر فولدر پترن های یونیک مخصوص خود را دارد، پس پترن های هر فولدر را در آرایه ای به نام patternsOfrow ذخیره میکنیم.
طبق مشاهدات و آنالیز فایل های ویروسی مربوط به هر فولدر، در میابیم که فایل های پشت سر هم (معمولا 15 تا بیشتر) الگوها و پترن های بسیار مشترکی با یکدیگیر دارند پس با بررسی و مقایسه ی هر 15 فایل پشت هم ویروس های مشترک را پیدا کرده و این دسته بندی را با فاصله های 15 تایی نیز از هر دسته تکرار می کنیم.

```
def find_pattern(directory, Virus_patterns):
    for i in range(20):
        directory = directory + str(i)
        j = 0
        patternsOfrow = [] #similar substring (pattern) in a row(consequtive files) since conse
        while(j<len(os.listdir(directory))-1): #check files in a row of size 15 since the conse
            file2chek_add1 = os.path.join(directory,os.listdir(directory)[j])
            file2chek_add2 = os.path.join(directory,os.listdir(directory)[j+1])
            Find_similar_substring(file2chek_add1, file2chek_add2, patternsOfrow)
            j+=1
            if(j%15==0):
                j += 30
            check_founded_pattern(patternsOfrow, j)
            Virus_patterns.append(patternsOftwo)
```

در دسته بندی های 15 تایی هر فایل را با فایل مجاور مقایسه می کنیم و در نهایت ویروس یافته شده را در صورت عدم وجود در لیست، به لیست اضافه می کنیم برای اینکار از تابع Find_similar_substring استفاده می کنیم.

```python
def Find_similar_substring(file_ad1, file_ad2, patterns):
    file_str1 = unicode2hex_cleaned(file_ad1)
    file_str2 = unicode2hex_cleaned(file_ad2)
    Domain = min(len(file_str1),len(file_str2))
    lofss = 50 # length of substring to check
    similarity = ['0'*lofss] # check similarity , goes in periods of 50 characters
    min_pattern_length = 5 # min pattern length
    max_pattern_lenght = 20 # max pattern lenght
    i = 0
    while(i<D-50):
        similarity = ['1' for j in range(lofss) if(file_str1[i+j] == file_str2[i+j])]
        for k in range(min_pattern_length, max_pattern_lenght): # range of allowed length for pattern
            if('1'*k in similarity):
                index = similarity.index('1'*k)
                pattern = file_str1[i + index : i + index + k]
                if(pattern not in patterns):
                    patterns.append(pattern)


        i+=1
```

در این تابع با استفاده از تابع unicode2hex_cleaned محتویات مفید آدرس ها را دریافت می کنیم و آن ها را با هم مقایسه می کنیم.
ثابت Domain مینیمم طول فایل ها می باشد.
حال یک بازه ی با طول 50 را به طور همزمان از ابتدای دو فایل تا انتهای آن ها جلو می بریم به طوری که در هر مرحله کاراکتر های مشترک را به صورت بیت در آرایه ی similarity ذخیره می کنیم. حال کاراکتر های مشابه های متوالی به صورت 11...1 در لیست قرار می گیرند.
حال یک کران بالا (max_pattern_length) و یک کران پایین(min_pattern_length) قرار می دهیم که ماکزیمم و مینیمم طول الگوی ویروس را تعیین می کنند.
به ازای طولی از یک بین این دو بازه پترن هایی خواهیم داشت که که ایندکس های متناظر آن ها را از محتویات فایل ها بدست آورده و در صورتی که در لیست وجود نداشت به patterns اضافه می کنیم .

```python
def unicode2hex_cleaned(file_add):

    no_use_chars = [str(hex(i))[2::] for i in range(20,47)]
    no_use_chars.extend([str(hex(i))[2::] for i in range(123, 192)])
    no_use_chars.append(str(hex(0))[2::])
    file_hex_str = ""
    with open(file_add, "rb") as file:
        file_hex_data = binascii.hexlify(file.read()) # convert to hex equivalent for simpli
        file_hex_str = file_hex_data.decode('utf-8')
        file_hex_str = file_hex_str[int(0.1*len(file_hex_str)):int(0.6*len(file_hex_str))] #
        for char in no_use_chars:
            file_hex_str = hex_str.replace(char, '')

    return file_hex_str
```

تابع unicode2hex_cleaned استرینگ ورودی file_add را به عنوان آدرس فایل میگیرد و آن را در مورد باینری می خواند سپس برای سهولت در محاسبات و پردازش آن را در مبنای 16 (hexadecimal) محاسبه می کند.
همانطور که در بالاتر ذکر شد، تمام محتویات فایل ها مفید نمی باشد پس فقط از ده درصد ابتدایی تا شصت درصد ابتدایی را در نظر می گیریم. و تمام کاراکتر های بلا استفاده در تشخیص پترن را از آن (با توجه به معادل هگزای آن ها) حذف می کنیم.
معادل هگزای این کاراکترها در no_use_chars ذخیره می کنیم.
حال رشته ی نهایی را برمی گردانیم.

```python
def check_founded_pattern(patterns, index,step):
    # Check if virus files have patterns in common
    directory = f"Train\\Malware Sample\\{i}"
    start = j-step
    end = j
    pattern_similarity_occurred = [0 for i in range(len(patterns))]
    for i in range(start, end):
        file_add_2check = os.path.join(directory,os.listdir(directory)[j])
        file_str1 = unicode2hex_cleaned(file_add_2check)
        for j in range(len(patterns)):
            if(sub_string_match(patterns[j],file_str1)):
                pattern_similarity_occurred[j] += 1

    patterns = [patterns[i] for i in range(len(patterns)) if(pattern_similarity_occurred[i]>= int(0.2*len(step)))] # if a pattern exist in 20
    # Check if non-virus files have patterns not in common
    directory = "C:\git\Anti_Virus\Train\Benign"
    pattern_similarity_occurred = [0 for i in range(len(patterns))]
    for files in listdir(directory):
        file_add_2check = os.path.join(directory,files)
        file_str1 = unicode2hex_cleaned(file_add_2check)
        for j in range(len(patterns)):
            if(sub_string_match(patterns[j],file_str1)):
                pattern_similarity_occurred[j] += 1

    patterns = [patterns[i] for i in range(len(patterns)) if(pattern_similarity_occurred[i] < int(0.2*len(step)))] # if a pattern exist in mo
```

تابع check_found_pattern دو ورودی pattern به عنوان الگوهای یافت شده و index به عنوان ایندکس انتهایی فایل های مربوط به pattern و step را به اندازه ی هر دسته بندی، ورودی میگیرد و در نهایت مشخص می کند کدام pattern ها معتبر می باشند که با دو شرط این مهم بررسی می شود.
ابتدا با توجه به نکات ابتدایی که گفته شد تمام پترن هایی که در 20 درصد یا بیشتر فایل های ویروس آن فولدر، وجود دارند نگه داشته می شوند.
سپس با پیمایش تمام فایل های بی خطر، تمام پترن هایی که در کمتر از 20 درصد فایل های بی خطر تکرار می شوند نیز ذخیره می شوند.

```
def sub_string_match(pattern, file_str):
    L =  len(pattern)
    pattern_hash = 0
    for k in range(L):
        pattern_hash = (pattern[k] + pattern_hash*10) % L
    sub_string_hash = 0
    for k in range(L):
        sub_string_hash = (ord(file_str[L-1-k]) + sub_string_hash*10) % L
    for t in range(len(input_file)-L+1):
        if(sub_string_2check == sub_string_hash): # hash matched
            o = 0
            for o in range(L): # check if they are identical in characters
                if(input_file[t+o] != pattern[o]): break # not identical
            if(o == L): # each of their characters matched -> it's a virus
                return True

        sub_string_2check += (-int(str(sub_string_2check)[0])*(10**(L-1))*10 + file_str[t])


    return False
```

تابع sub_string_match یک الگو و رشته ی یک فایل را به عنوان ورودی می گیرد و با الگوریتمstring matching با کمک hash function و در مرتبه زمانی خطی، در تمام رشته ی فایل پیمایش می کند و وجود زیر رشته ی ورودی را در آن بررسی می کند.

Main Process

```
# Main-Process
directory = input()
Searching_for_virus(directory, Virus_patterns)
```

ابتدا آدرس فولدری که فایل های آن قرار است، اسکن شوند به عنوان ورودی از ترمینال به صورت رشته دریافت می کند.

با استفاده از تابع Seaching_for_virus تمام فایل های ویروس را در فولدر Malware_files کات میکنیم.

دو ورودی directory و virus_patterns که آدرس فایلی است که تمام الگوهای یافت شده در آن نوشته شده اند.

```
def searching_for_virus(directory, Virus_patterns):

    Virus_patterns = []
    with open('Viruses.txt') as file:
        Virus_patterns = file.readlines()

    for filename in os.listdir(directory):
        file2chek_add = os.path.join(directory,filename)

        file_hex_str = unicode2hex_cleaned(file2chek_add)

        if(Is_Virus(file_hex_str, Virus_patterns)):
            try:
                shutil.move(file2chek_add, f'.\Malware_files\{filename}')
            except:
                os.mkdir('\Malware_files')
                shutil.move(file2chek_add, f'.\Malware_files\{filename}')
```

در این تابع ابتدا رشته های فایل virus_patterns را استخراج می کنیم و در آرایه ی virus_patterns ذخیره می کنیم سپس به ازای تمام فایل های موجود در directory آلوده بودن آن را با تابع IS_Virus بررسی می کنیم و در صورتی که حاوی ویروس بود، در صورت نبودن فولدر Malware_files آن را ایجاد می کنیم و فایل حاوی ویروس را به آن منتقل می کنیم.

```
def Is_Virus(input_file, Virus_patterns):
    for i in range(len(Virus_patterns)):
        # Hashing & Matching
        if(sub_string_match(Virus_patterns[i], input_file)):
            return True
    return False
```

این تابع با دریافت محتویات رشته ی فایل و الگوهای ویروس، ویروسی بودن آن را با تابع sub_string_match که در بالاتر توضیح داده شده بود،به ازای هر الگو، بررسی می کند.
و در صورتی که هیچ کدام از پترن های ویروس در آن، وجود نداشت، آن را فایل بی خطر تشخیص می دهد.

علی شیخ عطار
استاد عبدی
الگوریتم
دانشکده ی مهندسی کامپیوتر علم و صنعت