Summer_Hunters_Data_Science_Instructions

February 17, 2021

1 HoxHunt Summer Hunters 2021 - Data - Home assignment

1.1 Assignment

In this assignment you as a HoxHunt Data Science Hunter are given the task to extract interesting features from a possible malicious indicator of compromise, more specifically in this case from a given potentially malicious URL.

This assignment assumes that you are comfortable (or quick to learn) on using Jupyter Notebooks and Python. You are free to use any external libraries you wish. We have included an example below using the Requests library.

Happy hunting!

1.2 Interesting research papers & resources

Below is a list of interesting research papers on the topic. They might give you good tips what features you could extract from a given URL:

Know Your Phish: Novel Techniques for Detecting Phishing Sites and their Targets

DeltaPhish: Detecting Phishing Webpages in Compromised Websites

PhishAri: Automatic Realtime Phishing Detection on Twitter

More or Less? Predict the Social Influence of Malicious URLs on Social Media

awesome-threat-intelligence

1.3 What we expect

Investigate potential features you could extract from a given URL, and implement extractors for the ones that interest you the most. The example code below extracts one feature, but does not store it very efficiently (just console logs it). Implement a sensible data structure using some known data structure library to store the features per URL. Choose one feature for which you can visualise the results. What does the visualisation tell you? Also consider how you would approach error handling, if one of the feature extractor fails?

Should you make it to the next stage, be prepared to discuss the following topics: what features could indicate the malicousness of a given URL? What goes in to the thinking of the attacker when they are choosing a site for an attack? What inspired your solution and what would you develop next?

1.4 What we don't expect

- That you implement a humangous set of features.
- That you implement any kind of actual predicition models that uses the features to give predictions on malicousness at this stage.

1.5 Tips

- Keep it tidy - a human is going to asses your work :)
- Ensure your program does not contain any unwanted behaviour
- What makes your solution stand out from the crowd?

1.6 Indicators from the URL string

Let's start with low hanging fruit in directly analyzing the potentailly malicious URL. According to research in the PhishAri paper the most significant features are number of dots (marked by ".") and subdomains (marked by "/") and length of the URL, however a paper titled Feature-based Malicious URL and Attack Type Detection Using Multi-class Classification listed over 40 features that has been used in literature and proposed 16 new ones so there are defintly more things that could be added but for now the most interesting are the amount of certain symbols and the length of url.

1.7 Examining the content

Let's visit the website and see what IoCs we can get from the content. According to the Know Your Phish paper the phishing websites might load content from the actual webpage they're pretending to be which means there are a lot of external links in the html source. Another indicator is that they try to redirect bots to the actual website or somewhere else than browser in order to hide their phishing attempt from blacklists in their robots.txt. So let's use a webdriver to get the html and landing url that a browser would get and use the requests library to check if the bot would have the same landing url.

The features looked for here are; if the webpage redirects browser and requests to different urls,requests statuscode, how many redirections a request gets, amount of external and internal links, how many input fields there are, and how many images come from external and internal sources.

[1]:
```
import requests
import json
from urllib.parse import urlparse
from selenium import webdriver
from bs4 import BeautifulSoup as bs
import logging
from importlib import reload
import csv
import pandas
import plotly.express as px
```

Given functions

[2]:
```
def get_domain_age_in_days(domain):
    show = "https://input.payapi.io/v1/api/fraud/domain/age/" + domain
    data = requests.get(show).json()
    return data['result'] if 'result' in data else None

def parse_domain_from_url(url):
    t = urlparse(url).netloc
    return '.'.join(t.split('.')[-2:])
#Moved analyze url and example urls over to saving block
```

Examining the content functions

[3]:
```
#Setting up logging and webdriver
reload(logging)
logformat = "%(levelname)s %(asctime)s : %(message)s"
logging.basicConfig(filename='ThreatAnalysisLog.log', level=logging.WARNING,␣
 ↪format=logformat)
logger = logging.getLogger()
options = webdriver.ChromeOptions()
options.add_argument("--log-level=3")
options.add_argument("window-size=1200x600")
options.add_argument("headless")
driver = webdriver.Chrome(options=options)



def try_conn(url):
    """Checks if connection can be established"""
    try:
        return requests.get(url).status_code
    except requests.exceptions.ConnectionError as e:
        logger.warning(e)
        return 404
    except Exception as e:
        logger.error(e)
        return 'unk'

def get_page_browser(url):
    """Uses the webdriver to get the html code"""
    driver.get(url)
    html = driver.page_source
    return html

def get_links_ratio(source, domain):
    """Outputs external,internal link amounts"""
    soup = bs(source, "html.parser")
    internal, external = 0, 0
```

```
    try:
        for link in soup.find_all("a"):
            href = link.get("href")
            try:
                if "#" in href[0]:
                    continue
                if domain == parse_domain_from_url(href):
                    internal += 1
                    continue
                if "/" == href[0]:
                    internal += 1
                    continue
                external += 1
            except IndexError as e:
                continue
        return (external, internal)

    except TypeError as e: #Type error is raised when no links are found
        logger.warning(e)
        return (external, internal)

def input_fields(source):
    """Returns amount of input fields"""
    soup = bs(source, "html.parser")
    input_fields = soup.find_all("input")
    return len(input_fields)


def images(source, domain):
    """Outputs external,internal image amounts"""
    soup = bs(source, "html.parser")
    images = soup.find_all("img")
    external,internal = 0, 0
    for image in images:
        try:
            src = image.get("src")
            if src is not None:
                src_domain = parse_domain_from_url(src)
                if src_domain != domain:
                    external += 1
                else:
                    internal += 1
        except Exception as e:
            logger.warning(e)
    return (external, internal)

def compare_urls(url):
```

```
    """Checks if a request gets same url as the spoofed browser"""
    r = requests.get(url, allow_redirects=True)
    if driver.current_url == r.url:
        return 1
    else:
        return 0

def history_length(url):
    """returns history so you can check if there were many redirections"""
    hist =  requests.get(url).history
    return hist
```

1.8 Running the extractors and saving

analyze_url now extracts the features into a dict which it sends to write_to_CSV function which uses the keys as headers for the columns and outputs a csv. Containing the extracted data.

[4]:
```
def write_to_CSV(analysis, file):
    with open(file+".csv", "a", newline="") as csvfile:
        writer = csv.DictWriter(csvfile, analysis.keys())
        if csvfile.tell() == 0: #checks if file already has header
            writer.writeheader()
        writer.writerow(analysis)


def analyze_url(url, file="analyzed_urls"):
    """Runs all the feature extractors for given url outputs a dict"""
    res = {}
    res['url'] = url
    # First feature, if domain is new it could indicate that the bad guy has␣
 ↪bought it recently...
    domain = parse_domain_from_url(url)
    age = get_domain_age_in_days(domain)
    res['age']  = "No info" if age == None else age
    res['length'] = len(url)
    symbols = [".", "-", "_"] #list of symbols to be counted from the url string
    for symbol in symbols:
        res[symbol] = url.count(symbol) #count symbols amounts from url
    res['status_code'] = try_conn(url)
    if res['status_code'] in [404, 'unk']:
        #If no access to content or unexpected error then just add fields as␣
 ↪negative.
        res['images_internal'] = res['images_external'] = res['redirections'] =␣
 ↪res['links_external']= -1
        res['links_internal'] = res['input_fields'] = res['same_landing_url'] =␣
 ↪-1
    else:
```

```
        page_html = get_page_browser(url)
        res['images_external'], res['images_internal'] = images(page_html,␣
↪domain)
        res['redirections'] = len(history_length(url))
        res['links_external'],res['links_internal'] =␣
↪get_links_ratio(page_html, domain)
        res['input_fields'] = input_fields(page_html)
        res['same_landing_url'] = compare_urls(url)
    write_to_CSV(res, file)
```

[ ]:
```
# Note some of these urls are live phishing sites (as of 2021-02-05) use with␣
↪caution! More can be found at https://www.phishtank.com/
example_urls = ["http://mdflandres.com/balouteli/",
    "http://gulfsynergy.ram-fsip.com/doneds/ing/",
    "https://oviendooreal.com/next2.php",
    "https://mycompositi.com/5698/",
    "http://basqfar.maninfinity.com/bavchjs",
    "https://portale-dati-creval.com/"
                ]
for url in example_urls:
    analyze_url(url)
```

[ ]:
```
#Analyze urls straight from a csv containing only one column with the urls
file = "phishes"
with open(file + ".csv", "r") as csvfile:
    for row in csvfile:
        analyze_url(row.strip(), file+"_extracted")
```

1.9 Let's visualize!

As data I'm using top 25 from alexas top 1m websites list, and 10 sites from phishtank.org that were still online when I extracted the features. Let's vizualize the internal and external links.

First I import the csvs to panda dataframe then I add phish column to differentiate which ones are from the top25 csv and which are from the phish csv.

[8]:
```
df1 = pandas.read_csv('top25_extracted.csv', header=0,delimiter=',') #read data␣
↪from csv contain the extracted features.
df2 = pandas.read_csv('phishes_extracted.csv', header=0,delimiter=',')
df1['phish'] = "No" #add labels to dfs which are from the phish csv and which␣
↪are not.
df2['phish'] = "Yes"
df3 = pandas.concat([df1,df2.loc[df2['status_code'] != 404]]) # Let's not␣
↪visualize 404 returing phishing links
                                                              #as they have␣
↪-1,-1 for internal and external links
```

[9]:
```
fig = px.strip(df3, x="links_internal", y="links_external", color="phish",␣
↪hover_name="url")
fig.show()
```

Based on my dataset internal vs external links seems like a very good feature as the phishing ones have almost no internal links or no links at all while the legitimate websites had way more links especially internal ones. However, it should be noted that my dataset is very small and not randomized which might make it seem like a better feature than it really is.

1.9.1 Error handling

Most common errors are just handled with try and except that log the error for in a txt file for debugging. It's not a very good approach long-term but it works for this purpose.

[ ]:
```

```