



School of Computer Science and Engineering

Vellore Institute of Technology

Vellore

July, 2022

**Course: Cyber Security (CSE4003) EPJ**

**Slot: A1+A2+TA1+TA2**

**Class Group: Special Term II**

**Project Report**

**Review - 3**

**Team-8**

**19BCE2119 – Gaurav Kumar Singh**

**19BCE0689 – Piyush Rajput**

# **Sensitive Data Detection in Website's Public Domain using Web Mining**

## **Abstract**

Sensitive data exposed to the wider internet is a huge ordeal of most web pages. This sensitive data may be embedded as data visible in the public domain as JavaScript files or in the webpage structure itself. Sensitive data such as access keys, secret keys, passwords, etc.; when exposed to malicious authorities can compromise a webpage's integrity and accessibility. When deploying a web page, it must be taken care that none of the sensitive information is revealed to the public. Our aim is targeted toward detecting and classifying the mentioned sensitive data which can be used to exploit the web application so that the developers can take the required safety measures. We are going to develop an application that will assist the developers in identifying and securing any vulnerable access codes and sensitive data that is visible in the public domain and thus can be used to exploit the web application in question. We also want to show the extent of the harm that some of these flaws may do by performing some of these exploits in a supervised and permitted setting. We will be using web mining tools and techniques for detecting sensitive data. Our application will take the URL of the web page which has to be analysed for sensitive data. Web mining and data mining frameworks such as scrapy. Using this application, user will be able to find any sensitive data exposed to public domain along with its metadata which includes type of sensitive data found, location and the data discovered efficiently.

**Keywords:** Sensitive data, web mining, web application, Python, Regular Expressions

## **Introduction**

The main idea behind up taking this project is to enable developers and security analysts an application to help them to discover and secure any vulnerable access codes and sensitive data that is visible in the public domain and therefore can be used to exploit the web application at hand. These applications appear in many forms, from small homemade to large-scale commercial and private services (e.g., YouTube, Twitter, Facebook, GeeksForGeeks). However, web applications have been plagued with security problems. These security problems are often enough to incur huge losses to big firms if they are discovered by malicious parties. There are a lot of standard exploits that the hacker tests on the websites in a hope that the developer might have missed securing an end whereas the major threat is when the developer accidentally enables the hacker to launch a targeted attack on the website. By targeted attacks, we mean the cyber-attacks which are initiated by a hacker knowing that the exploit exists rather than trial and error of different attacks. It gives the developers less time to react to the attack launched and at the same time, it doesn't necessarily mean that the fix of the problem will be quick.

We also aim to demonstrate the scope of damage some of these vulnerabilities can incur by carrying out some of these exploits ourselves in a supervised and authorized environment to demonstrate the efficacy of the application in development environment which can help to avoid such threats. The sensitive data, and its corresponding metadata, will be provided in the results so that the developer may address the vulnerabilities that are open to the public. The leading URLs can be included in the website's input data, which are good to ignore during web mining since they are not directly related to the application backend or because the volume of data in such modules is too enormous to analyse in a reasonable amount of time. For logging and future reference, an option to export the discovery result together with the restrictions and time can be supplied. In most cases, web application vulnerabilities are caused by coding errors. Around half of leaks may lead to disclosure of account credentials, including for third-party resources. One example is configuration files (with passwords and access keys stored inside) that are accessible to all users. For large firms with stringent coding norms, machine learning algorithms can be used as well provided the context of the web application is well defined but for more diverse use, machine learning algorithms will not input accurate data classification due to lack of context and classification on training dataset and unsupervised machine learning algorithms are usually less accurate than supervised machine learning algorithms. Also, supervised machine learning algorithms makes the classifier blind to some contexts on which the classifier hasn't been trained.

Many similar ideas have used Natural Language Processing, Deep Learning Techniques, Machine Learning Algorithms etc. but the problem with such approaches is the fact that webpages are inconsistently structured set of context-free documents. Natural Language Processing and neural networks can be used effectively only when algorithm can be made aware of the context of the document and also the accuracy or hit-rate of such approaches vary widely across different documents. It also takes a lot of time to train NLP/ML models to accurately classify a string corpus and it is difficult to add new patterns in the string corpus. Since, nature of sensitive data also varies widely depending on the development software in use, being vague in description of what is considered as sensitive data with easy scalability and update of the approach is crucial which is not the case with machine learning or NLP based approaches.

This application can be used by web developers to test that no sensitive data is directly accessible through a website's public domain since this approach works on locally hosted webpages as well. It can inform the developers about the sensitive data which is identified, in which file it was found and what kind of sensitive data it is expected to be. This can help mitigate data leaks at deployment level by identifying such leaks during the testing phase of the application itself. Since scrapy can mine several domains concurrently, it makes the application suitable for quick and easy to use for developers as an additional piece of software to check their web application's public domain for potential data leaks with a feature to add their own set of keywords for identification of sensitive data. There are little to none reliable software which can be used to detect sensitive data in a webpage's public domain which can cause sensitive information leaks such as access codes, passcodes, etc. if hardcoded into one of the out-linking files. We also use Nostril phase classifier which is a recently made python framework to distinguish and classify the phrases as meaningful or not-meaningful which can

be used to filter out content based on the characteristics of the string being randomly generated or not.

### Literature Survey / Related Works

S.No.	Research Paper	Methods Used	Conclusion
1.	Identification of Sensitive Content in Data Repositories to Support Personal Information Protection. Briand A., Zacharie S., Jean-Louis L., Meurs MJ. (2018)	This paper describes a two-step process for finding sensitive information within documents. The proposed algorithm first identifies the document's context before identifying the sensitive data. The article uses NLP and pattern matching to detect and classify sensitive information in the documents.	The first step of the two-step process proposed here is context identification which is not feasible if the domain of contexts to choose from is basically limitless. In the conducted research out of kNN, SVM, RandomForest, MLPClassifier, Naive Bayes, MLP Classifier had the highest F1 Score of 0.927 followed by Naive Bayes at 0.905. Training the NLP model requires well-framed datasets to train the model which is not readily available for our project's domain of work so we have to come up with other ways to identify and classify sensitive data.
2.	Nostril: A nonsense string evaluator written in Python. Hucka, (2018).	Nostril is a Python 3 module that can be used to classify the string as either meaningful or nonsense. It can classify data as such with an adequate margin of error and reliability even when the words are somewhat jumbled, joint, broken, or semantically meaningless.	The nostril framework is a great fit for our application's use case as we do not focus on the context or semantics of the documents. It is precisely the tool that can be used to identify randomly generated strings that are extremely less likely to be classified as meaningful which can help to reduce false negatives from the discovery list.
3.	Detecting Sensitive Information of Unstructured Text	This paper discusses the criticality of exposure of sensitive information from	Text CNN (Convolutional Neural Network) is primarily used for sentiment

	Using Convolutional Neural Network. G. Xu, C. Qi, H. Yu, S. Xu, C. Zhao, and J. Yuan, (2019)	unstructured documents. The paper suggests using Text-CNN instead of recurrent neural networks which rely on the context of the document to accurately predicting the sensitivity of the document.	analysis/text classification of text data. It works well for data that are expected to have some context bound to them but in our use case, context is not very relevant, rather the structure of words is to be used to realize if a particular phrase is sensitive or not.
4.	Detection and Prevention of Sensitive Data from Data Leak using Shingling and Rabin Filter Sushma Gaikwad; Shilpa Chougule; Shrikant Charhate (2014)	The proposed method implements a prevention mechanism to upgrade the system performance. If someone takes data from a pen drive or a hard disc, and the owner discovers the identical material on the internet or on someone's laptop, the owner employs the likelihood approach to track down the leaker.	In all domains, preventing sensitive data from being compromised is an essential and practical research topic. Everyone wants their communication to be private, whether it's personal or professional. The algorithms mentioned in this work, Shingling and Rabin filter, produced superior results. The system tested and deployed a novel data- leak detection system that allows the data owner to identify and prevent data leaks. The results of the evaluation show that this approach can support accurate detection with a low number of false alarms and can be effectively implemented in a variety of organizations. However, rigorous testing on various data divisions of such methods will be required to implement the same in important sectors such as defense and other large establishments.
5.	Automatic Anonymization of Textual Documents: Detecting Sensitive Information via Word Embeddings Hassan,	Based on the concept of word embedding, the paper presents a broader method to text anonymization. All of the entities in the page are	In comparison to traditional approaches to text anonymization based on named entity recognition, the results demonstrate a

	Fadi; Sanchez, David; Soria-Comas, Jordi; Domingo-Ferrer, Josep (2019)	represented as word vectors that capture their semantic relationships. Then, by deleting other entities co-occurring in the document with vectors that are identical to the particular entity's vector, a specific entity can be automatically safeguarded.	considerable improvement in detection recall.
6.	Effective Implementation of Data Segregation & Extraction Using Big Data in E-Health Insurance as a Service Kumar, K. Manoj; Tejasree S, Swarnalatha, S. (2016)	The data acquired from hospital databases includes patient information such as illness information, treatment procedures, and billing information. Insurance businesses analyze the data using the Infinispan Big Data platform, and the important information is extracted using the map-reduce methodology and a mapping method.	The output of the mapping is fed into the reducer. The reducer technique is used to compute the rates of success and failure. Finally, the new patient will be able to look at the status of certain ailments that the hospital is dealing with. As a result, the analysis of insurance datasets in E-health insurance provides successful application of the Infinispan and Map-reducer principles. The data is extracted and separated.
7.	Privacy-Preserving Detection of Sensitive Data Exposure. X. Shu, D. Yao, and E. Bertino (2015)	In this paper, authors propose a sensitive data detection methodology using a special set of sensitive data digest to enable the owner of the data to carry out the detection operation in a safe manner on third party services without exposing their sensitive data to the service providers.	It suggests a privacy-preserving data leak detection concept and its implementation. The exposure of sensitive data is limited to a minimum during detection by using special digests. This method is a sensitive data leak prevention mechanism but it requires acceptance of proposed architecture by the service providers over their existing monolithic architecture which is hard to implement for well-established service providers.
8.	Detecting Sensitive	The goal of the paper is to provide a system for	In comparison to approaches based on

	Information from Textual Documents: An Information-Theoretic Approach	detecting sensitive information from textual sources in a domain-independent manner. It evaluates the degree of sensitivity of phrases based on the amount of information they give, using the Information Theory and a corpus as vast as the Web.	trained classifiers, preliminary findings demonstrate that the strategy greatly enhances detection recall.
9.	Detecting Sensitive Data Disclosure via Bidirectional Text Correlation Analysis Huang, Jianjun; Zhang, Xiangyu; Tan, Lin (2016)	The paper focuses on developing BIDTEXT, a novel static technique to detect sensitive data disclosures. BIDTEXT approaches the problem as a type system, with variables being typed according to the text labels they see. If a sensitive text label is typed on a parameter at a sink point, data disclosure is reported.	BIDTEXT detects text labels in both code and UI, treats them as types, associates them with the associated variables, bidirectionally propagates the types across data flow, and finally attaches them to sink points that could possibly reveal sensitive data.
10.	PrivacyBot: Detecting Privacy Sensitive Information in Unstructured Texts Welderufael B. Tesfay; Jetzabel Serna; Kai Rannenber (2019)	This paper presents PrivacyBot, a machine-learning-based proof-of-concept that detects PSI in user-generated unstructured texts. The method can detect PSI with an accuracy of up to 95%, according to a series of rigorous tests.	The findings are encouraging and point to the possibility of integrating such tools to assist people in making educated privacy related decisions while disclosing PSI on the internet.
11.	SIDD: A Framework for Detecting Sensitive Data Exfiltration by an Insider Attack Yali Liu; Cherita Corbett and Ken Chiang; Rennie Archibald, Biswanath Mukherjee and Dipak Ghosal (2016)	The method is based on using statistical and signal processing techniques to produce signatures and/or extract features for classification purposes from the data flow. The suggested framework intends to address ways for detecting, deterring, and preventing a trusted insider from intentionally	To improve system performance, the proposed solution employs a preventative mechanism. It also tries to track out the data leaker using the methods listed below.

		or unintentionally distributing sensitive content outside the business using the organization's system and network resources.	
12.	Automated identification of sensitive data from implicit user specification Ziqi Yang; Zhenkai Liang (2018)	The method takes into account all semantic, syntactic, and lexical information, with the goal of identifying sensitive material based on the semantics of its description texts. The notion of concept space is introduced in the study to express the user's sense of privacy, with the goal of allowing the technique to enable flexible user needs in defining sensitive material.	It achieves an average precision of 89.2%, and an average recall of 95.8% in identifying sensitive data.
13.	Side-Channel Leaks in Web Applications: A Reality Today, a Challenge Tomorrow Shuo Chen; Rui Wang; XiaoFeng Wang; Kehuan Zhang (2010)	The paper shows that despite encryption, applications' internal information flows are inevitably exposed as a realistic and serious threat to user privacy. Detailed sensitive information is being leaked out from a number of high-profile, top-of-the-line web applications in healthcare, taxation, investment, and web search.	The study provides a detailed examination of the difficulties in mitigating such a threat, highlighting the need for a disciplined engineering process for side-channel mitigations in future web application development.
14.	A survey on data breach challenges in computing security: Issues and threats. R. Barona and E. A. M. Anita, (2017)	The paper discusses various concepts of cloud architecture and lists potential security threats such as Data Breach, Service Traffic Hijacking, Insecure APIs DOS, Insider Attacks, Cloud Service Abuse, Shared, Technology Vulnerabilities. The	The paper gives an abstract idea about the potential security threats faced on the internet, specifically the cloud computing environment which in recent years had become more and more popular and essential. It also discusses proper SLA policies to be followed by a cloud service provider and



		authors also discuss solutions and challenges faced while resolving the issues.	monitoring mechanisms to prevent data breaches and limit the damage if the security is compromised.
15.	An Automated Recognition Model for Sensitive Information. Dongdan Guo, Hao Sun, Tao Zhu, and Chang Cai (2020)	In this paper, the authors describe an automatic sensitive information identification model to identify sensitive data associated with the passengers of the Civil aviation business in structured and unstructured files. They classify the structured data according to their sensitivity and use pattern matching, keyword recognition, and NLP recognition to automatically identify and classify the data.	The classification of structured data can be automated however, the solutions proposed do not give much insight if the documents are unstructured relying on the idea of NLPs again in those cases which is what we are dealing with.

## Proposed System

### Overview

We will use Python's web mining framework Scrapy to mine the data which can be publicly accessed to analyses for any sensitive data exposure. Scrapy is faster to mine data from websites when compared to similar frameworks such as bs4, Headless-Selenium etc. It can handle concurrent requests easily and have a lot more configurations available to use compared to other frameworks. The logic data (primarily JavaScript) from the website is scraped and exported using scrapy because usually JavaScript in a webpage has higher chances of leaking confidential data. After the data is exported from the miner process, we will use standard regular expression classifiers to classify the data. Many sensitive access keys or passwords follow a certain set of rules to be identified as one. We will use this premise to identify and classify along with the kind of classification it is. The data after being filtered is passed through these regular expression identifiers to find any matches and the output is forwarded to the Nostril phrase classifier for further filtering.

The issue with using regular expression for classification is the number of false positives it can give due to the vague nature of some regular expressions of being too flexible with what can be considered valid match by these regular expressions. Also, few meaningful clauses can be classified as sensitive data while most access keys tend to be randomly generated resulting in

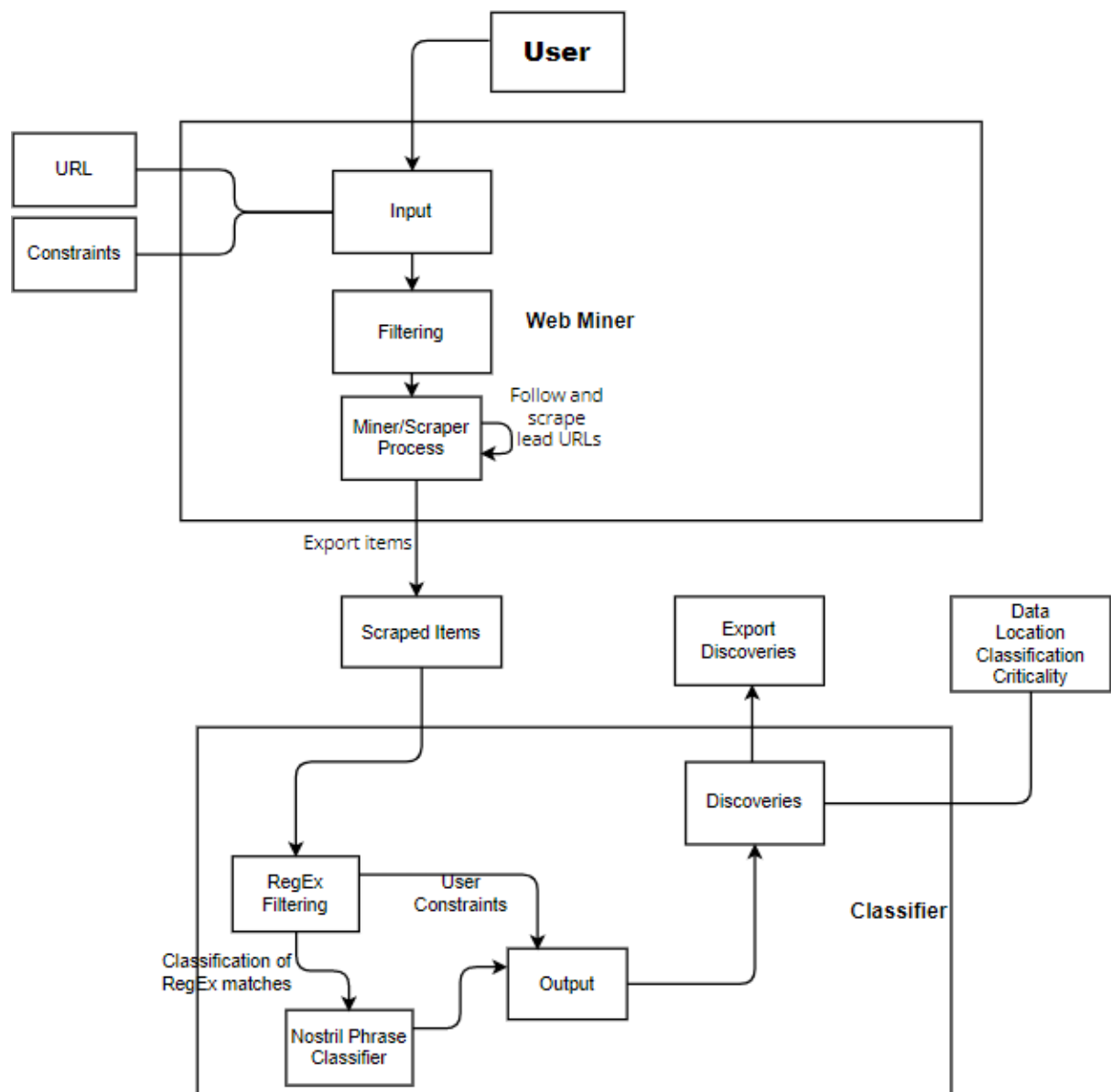
more false positives. To solve the issue, we use Nostril phrase classifier to categorize the data as meaningful or 'nonsense'. The nonsense data is classified sensitive and the metadata for that string is recorded and exported as output. The advantage of using this approach is that it does not require context of the document it is processing to accurately identify sensitive data and is fast to execute since we are not training any data. The disadvantage is the consistent requirement of updating the regular expression classifiers according to the needs.

## **System Architecture**

The workflow of the project starts with a web miner. The application will ask for the website URL, and custom search strings for inputs alongside the predefined constraints. These inputs will be filtered and the filtered inputs will be pipelined to the web scraper. We will use the Scrapy module in python for mining the input URL's source. The relevant out-links of JavaScript in the source will be separately mapped along with their URLs and the extracted output will be exported into a csv file for the classifier to read from. In the classification phase of the application, the data is read from the exported scrapy file and the data is filtered again, and passed through a bunch of Regular expression classifiers and the output from these classifiers then pipelined to nostril phrase classifiers to determine randomness of the string. If a string passes the classifiers, it is marked as sensitive data and the data along with its metadata which includes location, type of data is recorded.

Nostril is a python module that is used to classify strings as meaningful and randomly generated. Since much of the sensitive data such as access codes, passwords, etc. are usually randomly generated, it will help us to minimize false positive outcomes in cases when the sensitive data is supposed to be randomly generated. Custom strings can be directly pushed into output without Nostril filtering if needed by turning it off for the particular classifier. The output result is then exported as CSV or JSON by the user to maintain logs.

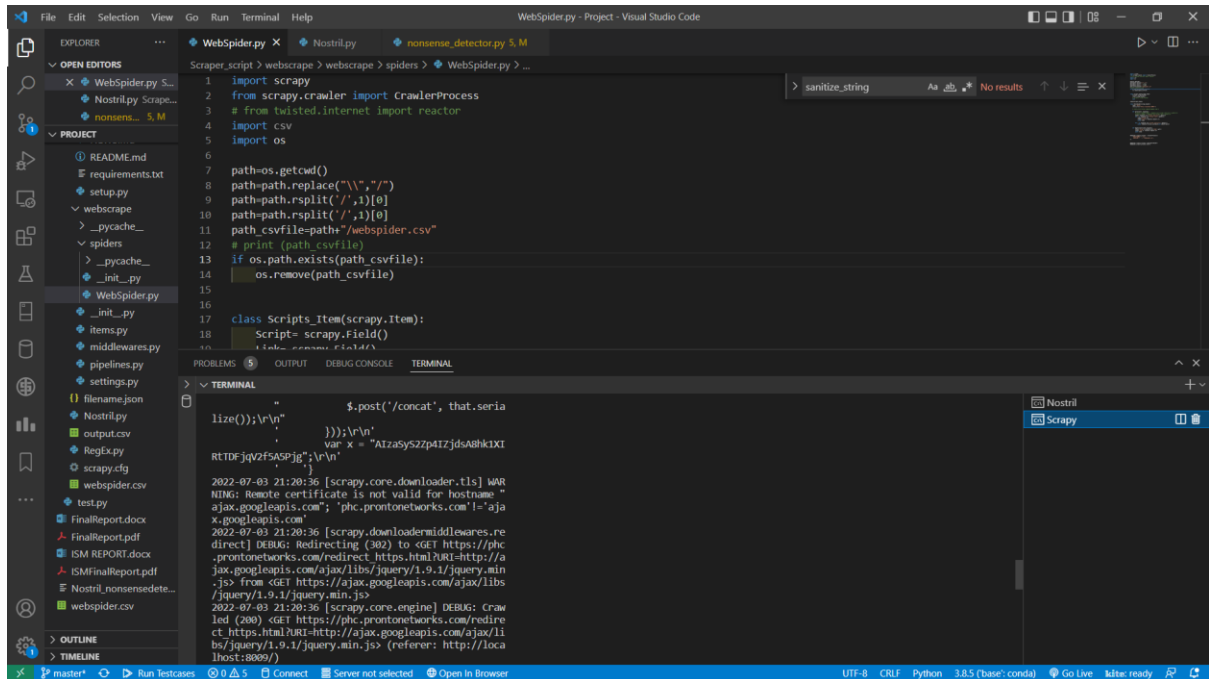
## Modular Design



## Implementation Details and Analysis

### Web Miner

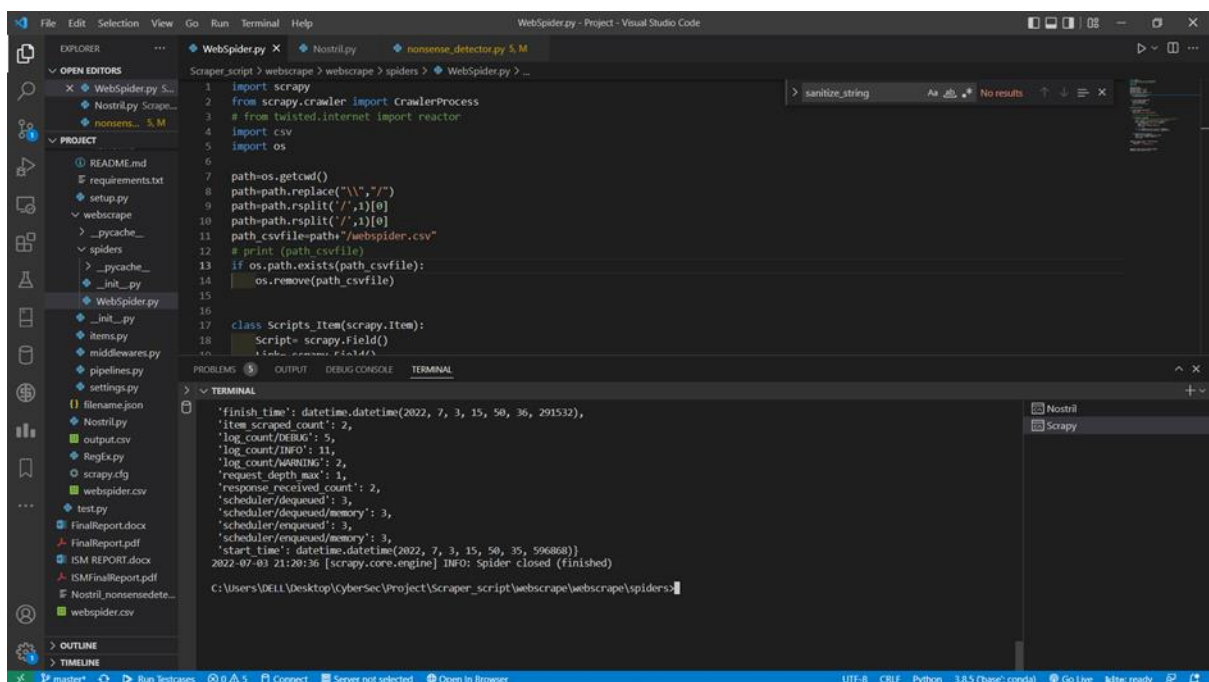
The result of this phase of the application is an csv file (webspider.csv) with a list of JavaScript codes as text and their respective domain URL. Create a scrapy project with the code in it and directly execute the file as python file. The code has logic to run the code directly as a python file without using scrapy clause in terminal.



```
1 import scrapy
2 from scrapy.crawler import CrawlerProcess
3 # from twisted.internet import reactor
4 import csv
5 import os
6
7 path=os.getcwd()
8 path=path.replace("\\","/")
9 path=path.rsplit('/',1)[0]
10 path=path.rsplit('/',1)[0]
11 path_csvfile=path+"/webspider.csv"
12 # print (path_csvfile)
13 if os.path.exists(path_csvfile):
14     os.remove(path_csvfile)
15
16
17 class Scripts_Item(scrapy.Item):
18     Script= scrapy.Field()
19     link= scrapy.Field()
```

Terminal output:

```
2022-07-03 21:20:36 [scrapy.core.downloader.tls] WARNING: Remote certificate is not valid for hostname 'ajax.googleapis.com'; 'phc.prontonetworks.com' != 'ajax.googleapis.com'
2022-07-03 21:20:36 [scrapy.downloadermiddlewares.redirect] DEBUG: Redirecting (302) to <GET https://phc.prontonetworks.com/redirect_https.html?url=http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js> from <GET https://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js>
2022-07-03 21:20:36 [scrapy.core.engine] DEBUG: Crawled (200) <GET https://phc.prontonetworks.com/redirect_https.html?url=http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js> (referer: http://localhost:8009/)
```



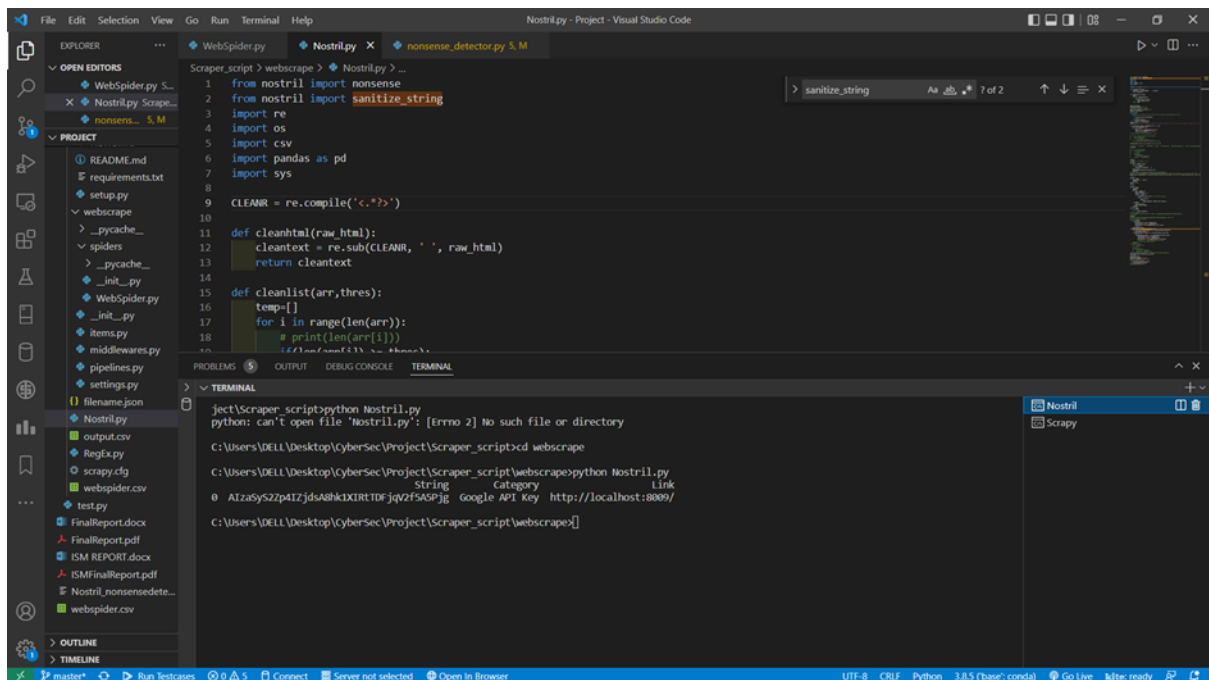
```
1 import scrapy
2 from scrapy.crawler import CrawlerProcess
3 # from twisted.internet import reactor
4 import csv
5 import os
6
7 path=os.getcwd()
8 path=path.replace("\\","/")
9 path=path.rsplit('/',1)[0]
10 path=path.rsplit('/',1)[0]
11 path_csvfile=path+"/webspider.csv"
12 # print (path_csvfile)
13 if os.path.exists(path_csvfile):
14     os.remove(path_csvfile)
15
16
17 class Scripts_Item(scrapy.Item):
18     Script= scrapy.Field()
19     link= scrapy.Field()
```

Terminal output:

```
'finish_time': datetime.datetime(2022, 7, 3, 15, 50, 36, 291532),
'log_count/DEBUG': 9,
'log_count/INFO': 11,
'log_count/WARNING': 2,
'request_depth_max': 1,
'response_received_count': 2,
'scheduler/dequeued': 3,
'scheduler/dequeued/memory': 3,
'scheduler/enqueued': 3,
'scheduler/enqueued/memory': 3,
'start_time': datetime.datetime(2022, 7, 3, 15, 50, 35, 596868)}
2022-07-03 21:20:36 [scrapy.core.engine] INFO: Spider closed (finished)
C:\Users\DELL\Desktop\cybersec\Project\Scrapper_script\webscrape\webscrape\spiders>
```

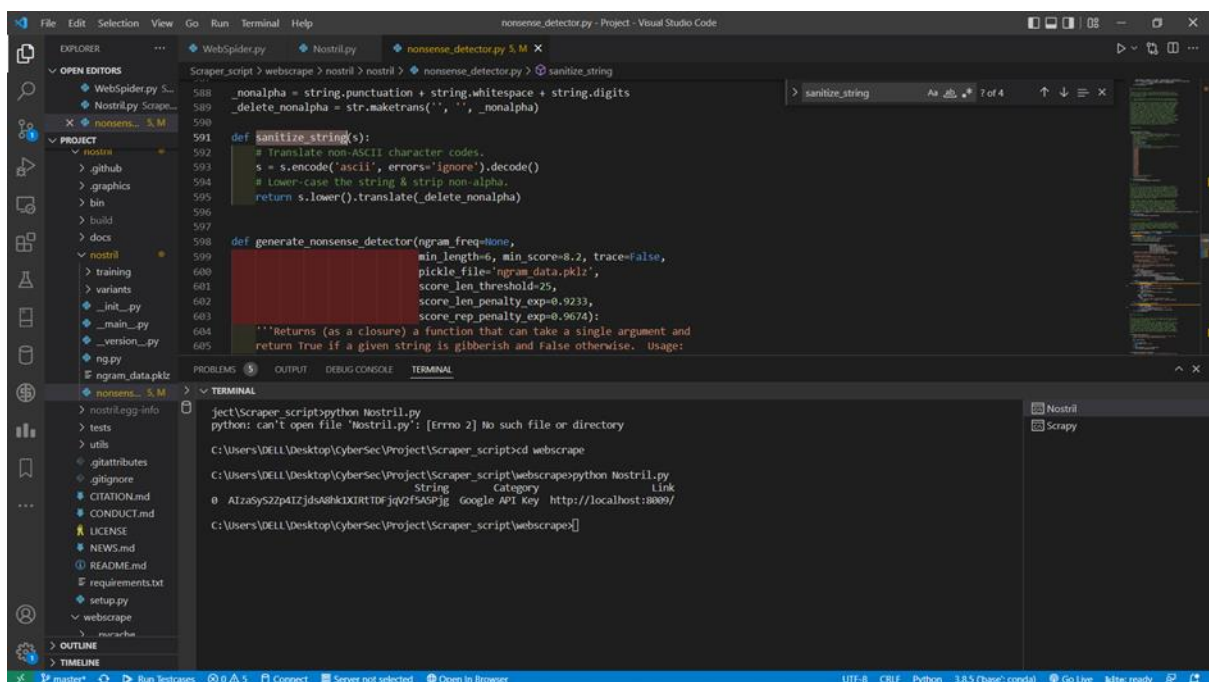
## Classifier

The output for this phase of application is a csv file (output.csv) with a clean list of sensitive data exposed through the JavaScript files along with their domain and Category.



The screenshot shows the Visual Studio Code interface with the 'Nostril.py' file open. The file contains Python code for sanitizing strings and cleaning HTML. The terminal output shows the execution of the script, which successfully processes the data and outputs a CSV file named 'output.csv'. The output data is as follows:

String	Category	Link
0 AIzaSySzp4IZjdsA8hk1XIRTDFjqV2fSASPjg	Google API Key	http://localhost:8009/



The screenshot shows the Visual Studio Code interface with the 'nonsense\_detector.py' file open. The file contains Python code for sanitizing strings and generating a nonsense detector. The terminal output shows the execution of the script, which successfully processes the data and outputs a CSV file named 'output.csv'. The output data is as follows:

String	Category	Link
0 AIzaSySzp4IZjdsA8hk1XIRTDFjqV2fSASPjg	Google API Key	http://localhost:8009/

## Analysis

From the test case used which is a webpage hosted on local machine with custom and general classifiers, it can be seen that the classification is indeed working correctly and disregards

context of document in favour of our specific use case as it allows the application to work consistently over a large array of webpages over the internet.

It is however limited to webpages which can't be scraped by scrapy module implying, dynamically loaded webpages such as google cloud services home domain for example can't be used in root URL.

The aim of this application is to prompt developers about potential sensitive data leak during the development phase of the applications before they are deployed.

## **Conclusion and Future Work**

Significance of this application is to act as a safeguard between the development and deployment phase of the web application for developers to use to test if any private keys or sensitive data will be accidentally exposed to the public domain without the need of the developer to manually go through each and every line of the application. It is encouraged to properly set file permissions of configuration files before the application is deployed and avoid hard-coding any sensitive data directly into the backend logic or comments. The results obtained through this project are stored for the developers to keep logs of the mistakes made and correct those mistakes by using more secure coding practices and techniques.

The efficiency of the application in terms of runtime is average since web mining as a process is time consuming but scrapy is one of the fastest web crawlers there are for python and the classifier itself is efficiently programmed. The output result accuracy is tough to determine for the project because of lack of context in most web applications but from the tests performed locally, it performed perfectly. The primary issue faced during implementing the project was to find the proper and cohesive set of tools to implement the workflow we had planned for. There were a few dependency issues involved and the Nostril framework was to be analysed thoroughly to debug the application according to how the framework works. It is a problem associated with using already built frameworks is that, it is not recommended to actually change the logic in the framework's file since the changes would be tough to clone or edit in other devices.

For future work, the 2 phases of the application can be integrated into one by creating a flask application and take the inputs for URLs and constraints from the frontend or uploading a text file in a specific format for the program to read. More classifiers can be included in the classifier list and in future it can be used to provide classification for the web applications it works accurately on to enable scope of utilising machine learning algorithms because we can create classified data for the machine learning algorithms to train.

## **References**

- 1) Antoine Briand, Sara Zacharie, Ludovic Jean-Louis and Marie-Jean Meurs, "Identification of Sensitive Content in Data Repositories to Support Personal Information Protection," 2018.
- 2) Michael Hucka, "Nostril: A nonsense string evaluator written in Python," 2018.
- 3) G. Xu, C. Qi, H. Yu, S. Xu, C. Zhao and J. Yuan, "Detecting Sensitive Information of Unstructured Text Using Convolutional Neural Network," 2019.
- 4) Sushma Gaikwad, Shilpa Chougule and Shrikant Charhate, "Detection and Prevention of Sensitive Data from Data Leak Using Shingling and Rabin Filter," 2014.
- 5) Fadi Hassan, David Sánchez, Jordi Soria-Comas and Josep Domingo-Ferrer, "Automatic Anonymization of Textual Documents: Detecting Sensitive Information via Word Embeddings," 2019.
- 6) K. M. Kumar, Tejasree S and S. Swarnalatha, "Effective Implementation of Data Segregation & Extraction Using Big Data in E-Health Insurance as a Service," 2016.
- 7) X. Shu, D. Yao, and E. Bertino, "Privacy-Preserving Detection of Sensitive Data Exposure," 2015.
- 8) David S´anchez, Montserrat Batet, and Alexandre Viejo, "Detecting Sensitive Information from Textual Documents: An Information-Theoretic Approach," 2012.
- 9) Jianjun Huang, Xiangyu Zhang and Lin Tan, "Detecting Sensitive Data Disclosure via Bi-directional Text Correlation Analysis," 2016.
- 10) Yali Liu, C. Corbett, Ken Chiang, R. Archibald, B. Mukherjee and D. Ghosal, "SIDD: A Framework for Detecting Sensitive Data Exfiltration by an Insider Attack," 2016
- 11) Ziqi Yang and Zhenkai Liang, "Automated identification of sensitive data from implicit user specification," 2018
- 12) Shuo Chen, Rui Wang, XiaoFeng Wang and Kehuan Zhang, "Side-Channel Leaks in Web Applications: A Reality Today, a Challenge Tomorrow," 2010
- 13) R. Barona and E. A. M. Anita, "A survey on data breach challenges in computing security: Issues and threats," 2017
- 14) Dongdan Guo, Hao Sun, Tao Zhu, and Chang Cai, "An Automated Recognition Model for Sensitive Information," 2020
- 15) Xiyuan Wang and Yong Wang, "Educational Sensitive Information Retrieval: Analysis, Application, and Optimization," 2017