**#Add more of the reasons why I made all these choices**

**# did I meet my timeline / goals**

**# add a conclusion**

**OSINT**

This tool is an OSINT (open-source intelligence) tool that goes out and gathers open-source information about a domain that you put into the tool. My reasoning for wanting to create this tool is every time I go to Pentest a domain for a bug bounties, I have to do all the OSINT information gathering by hand. Which after a while gets boring and tedious. So, I wanted to make a tools that can go out and gather a lot of that information for me. Now I am aware that tools like this already exist a good example being theHarvester. But I wanted to make my own tool to teach my self how a lot of OSINT information is gathered and to design a tool that is made for an open-source project.

**Design Goals for the Project:**

My design goals for this project’s design are to:

1. Modular application
2. Can Take in I/O from most sources
3. Easy for the end user to use

The application needs to be module because I want it to be easily extended/modified by other people in the future and act like an open-source tools.

The I/O requirements of the tool are because OSINT information can come in many forms and if I designed the tool to only take 1 or 2 forms of I/O then the tools would be limited.

The ease of use by the end user should always be a goal in application design because if no one wants to use the tool then why make it.

**Programing languages:**

The programing language I chose for this project is Python (latest version). The reason for this choice is:

1. The duck typing in Python makes it easier for I/O of all types
2. There are a lot of open-source helper classes that make the gather of data easier
3. It is a high-level language making programing easier

**SILO Design:**

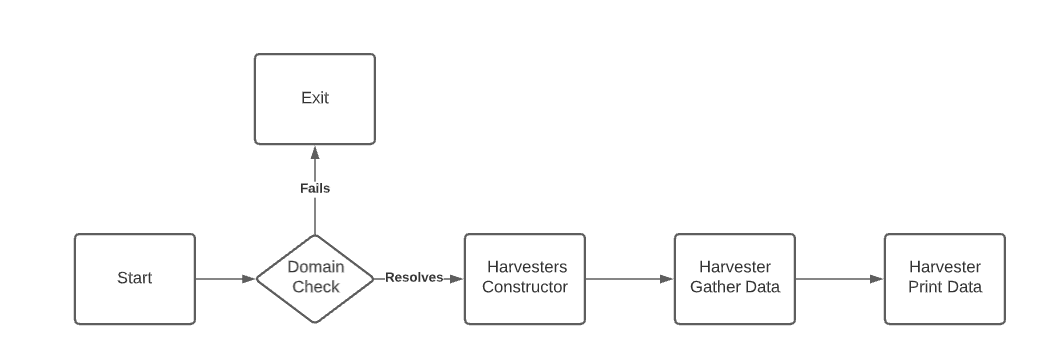
**TERMS:**

**SILO:** The SILO is the main class of the OSINT tool and the name of the OSINT tool.

**Harvester**: A harvester is a class that the SILO has which goes out and harvests data from a variety of sources.

The design of the SILO class has 4 main parts:

1. The INIT and domain check
2. The Harvester constructor
3. The Harvester gather data
4. The Harvester print data

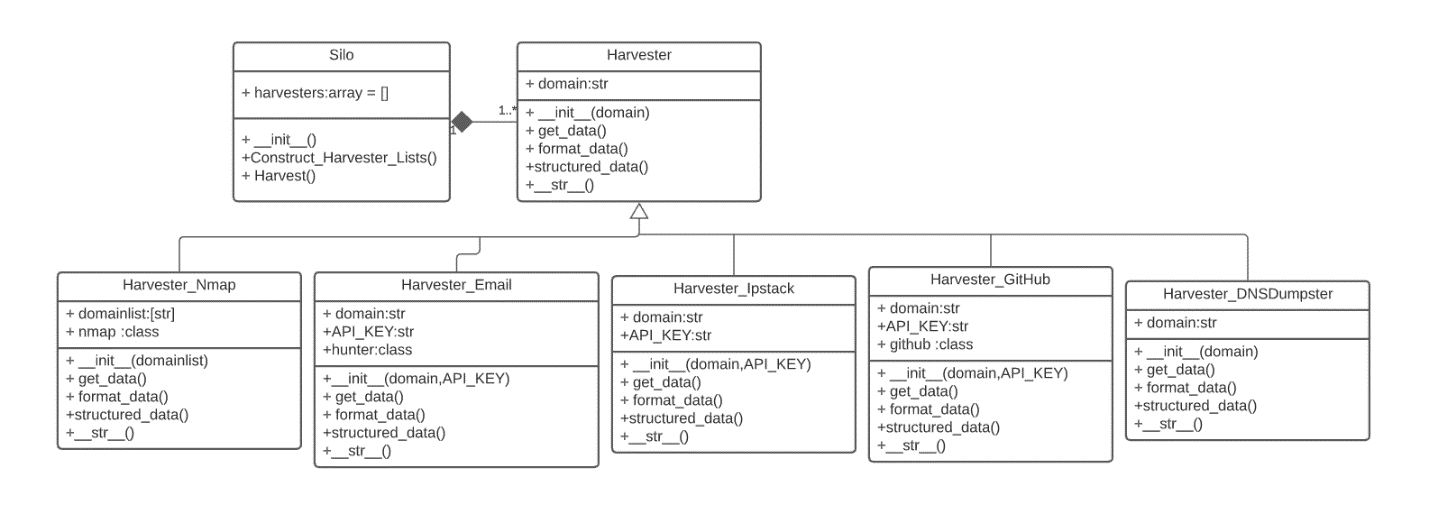


The SILO class INIT has the functionality of initializing all the variables that are needed before the harvesters can run. This includes the API keys, argparse output, and the verification that domain exists. After the Domain is check in the program could exit or move on to the constructor faze of the harvesters.

In the harvester constructor phase the harvesters are added to the harvesters list. Most of the harvesters have some sort of check in front of them to make sure the application will not crash. All the harvesters with API Keys check a check before them to make sure there is an APIKEY in the json file. This does not check to see if the API key works though because that would require calls in the SILO class and that is not supposed to happen. Another example of a check before the harvester is added to the list is the Nmap function. The Nmap function checks the argparse variables to make sure that -a flag is set.

The last section of the SILO code is the Harvest function. This function performs the Gathering of the data form the harvests list and prints the data that is gathered from the harvesters list. This section of the SILO is just two for loops that are looping though the harvesters’ lists. All the work is done by the harvester subclasses which is the next section.

**Harvesters Design:**

For this section I will be discussing every harvester class that can be used by the SILO to gather data about the specified domain. With the exception on Nmap because Nmap must resolve the domains before scanning.

This UML diagram shows the relation ship between the Silo class, the harvester and sub harvesters. The Silo class has a harvester, and the harvester is the super class for the sub harvesters.

**TERMS:**

**Table**: A table is a class using the rich forming class in Python that makes the output data look better.

**Harvester Classes:**

**Harvester:**

The base harvester super class is the main structure for every other Harvester. There are 4 functions to this super class. The functions are get\_data, formatted\_data, structured\_data, and \_\_str\_\_.

**get\_data:** function will be called in the SILO and will go out and harvest the data that is needed for the user. This function is called in the \_\_init\_\_ for most harvesters because it initializes data that is needed for the function of the formatted\_data and structured\_data functions.

**formatted\_data:** function will return the data in a form that is iterable in Python. This function was designed for the use case of making it so that the SILO program would have access to the raw data that is output from the get\_data function for use if needed by the SILO.

**structured\_data:** function will print out the data to the terminal in a form that is presentable to the end user. This function was designed to output a rich version of the output data that is formatted to be presented to the end user.

**\_\_str\_\_:** function for every harvester class will print out a string casted version on the output from the formatted\_data function. This function is more of a backup function to the structured\_data function and is never used by the SILO program as of this implementation.

**Harvester\_DNSDumpster:**

The DNSDumpster harvester is designed to gather some of the subdomains for the domain.

**INPUT:**

API key, Domain

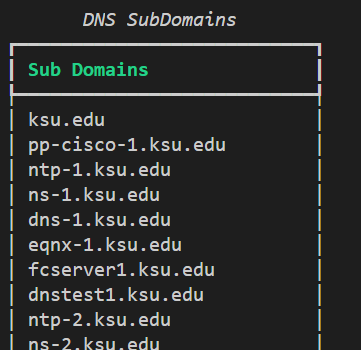
**PROCESSING:**

Making a call to the DNSDumpster API and formatting the JSON output from the DNSDumpster API.

**OUTPUT:**

The two outputs of this harvester are the JSON file returned by the DNSDumpster API and the table that is formatted in the structured\_data function.

**EXAMPLE:**



**Harvester\_Email:**

The Email Harvester is designed to gather a list of names and emails that are related to the domain the user searched.

**INPUT:**

API\_KEY, domain, limit (limits the number of results the API will output)=5, offset=2

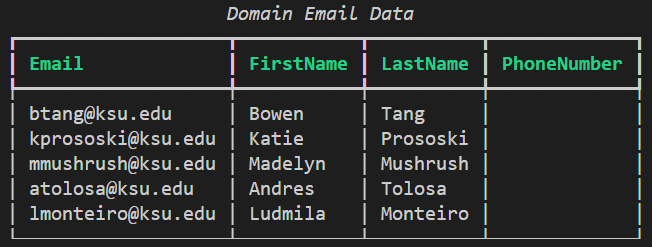
**PROCESSING:**

The processing of this harvester starts with getting the data from the PYhunter API. This API is attached to a database of names and emails associated with a domain. Then a table is constructed out of the data gathered from the API.

**OUTPUT:**

The output of this harvester is the JSON returned from the API and the table constructed from that data.

**EXAMPLE:**



**Harvester\_GitHub:**

The GitHub harvester is used to find GitHub repos that are related to the domain that you are searching.

**INPUT:**

API\_KEY, domain

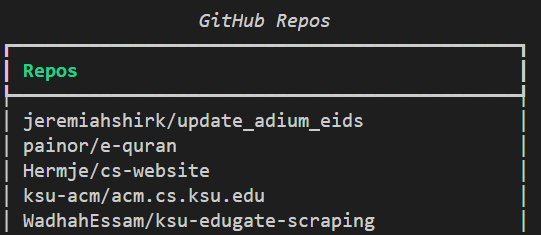
**PROCESSING:**

The processing for this harvester is a call to the GitHub API and the formatting of the output data in the formatted\_data and structured\_data functions.

**OUTPUT:**

The output data is a list of GitHub repos that are related to the domain that the user is searching for.

**EXAMPLE:**



**Harvester\_Nmap:**

**INPUT:**

Domain\_list

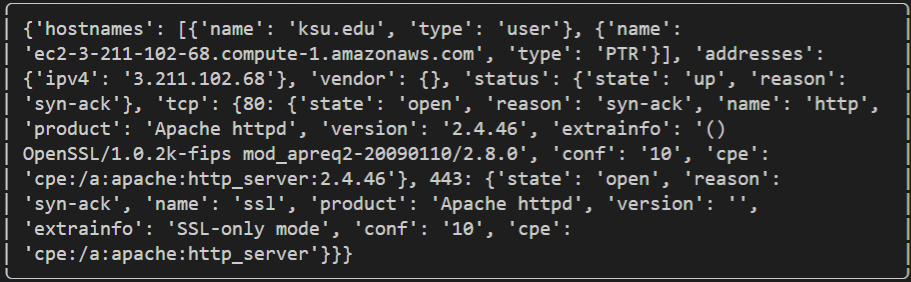
**PROCESSING:**

The processing for this harvester involves calling the Nmap Python class on the domains that are listed in the domain list variable. Then formatting that output into a list of results that can be used by the user.

**OUTPUT:**

The output for this harvester is a JSON formatted results list that is put into a Panel variable to be displayed by rich. The current output for the Nmap harvester is not formatted well because it does not have a standard for that can easily be put in to a table.

**EXAMPLE:**



**Harvester\_Ipstack:**

This harvester’s function is to get the IP and other related information about the domain.

**INPUT:**

API\_KEY, domain

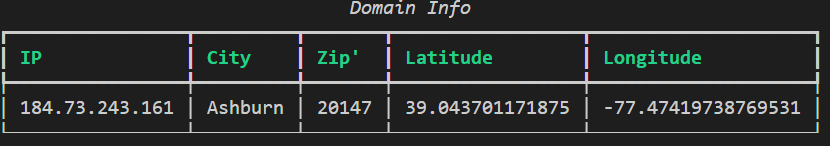
**PROCESSING:**

The processing for this harvester is a call to the Ipstack API. Which is then formatted by the structed\_data function.

**OUTPUT:**

The output of this harvester is a table that has the IP, City, ZIP and Log/Lat of the server that the domain points to.

**EXAMPLE:**



**Harverster\_haveibeenpwned:**

This harvester did not get implemented because the API that controlled the data was not free. This does not mean that it would not be a good API to add to the tool, but I wanted to make this version of the tool 100% free and not paid for. I also could have done web scraping on the haveibeenpwned.com website, but I did not want to breach their Terms of Service with this tool.

**Harvester\_Linkedin:**

This harvester did not get implemented because its functionality would have been very similar to the Email harvester. Also, there where time restrictions that made implementing this harvester a non-priority.

**How to Extend Silo:**

To extend the Silo tool there are only two main things you must do. This is to create a sub harvester class that extends from the main harvester and add that harvester to the harvesterlist in the harvester constructor section of the code. When adding a harvester to the construction section of the code make sure you check that the harvester has all the variables it needs to run present in the code. Also when implementing a harvester make sure in the structured\_data function to return a Rich.Table or a Rich.Panle to keep the forming of the output the same. But in general, there is not much to adding more functionality to the Silo tool.

**Time management:**

My original timeline at the start of the semester was:

**February**: pick idea /start of design process

**March**: finalize design for app / start development

**April**: development / testing

**May**: finish testing and dev

Overall, I stayed true to this over all timeline. I have no statistics on how much coding I did or what the timeline was for that. I pushed my code every Monday because that is when I meet with my professor and have very little to offer past that.

**Challenges in Creation:**

The hardest part in the creation of the SILO tool was finding the data sources to get the data needed for every domain. A good example of this is for the Harvester\_Ipstack. This Harvester was not hard to implement in code but the process of finding the Ipstack API took me a week of searching for a 100% free API that gave me the information that I needed. This process was the same for most of the harvesters where implementation took me less then a day but finding the API took me much longer.

Another challenge was coming up with the best design for the SILO main class and the harvesters that it uses. As with most programing projects the design phase is the hardest because you must consider all of the requirements the project needs to meet. In the case of the SILO app, I needed to make a design that was flexible and could take I/O from most sources. Eventually I got to the design that there currently is, but it was one of the harder parts of the project.

**Bugs and Testing:**

Because there is limited user input there is very little bug testing needed. The main bug testing I have done is with the input of domains and how it can crash the application. I tested domains that do not resolve, words that are not domains, and not inputting a domain. All these cases have messages that will be sent to the user telling them how they messed up their input.

Another case of user input bugs is the API key file that the user inputs through the APIKEY.json file. I had to do bug testing on this by inputting malformed API keys and malformed JSON formatting. Both of those cases will be handled correctly by the code.

**Conclusion:**

**Future of the SILO Tool Goals:**

1. **Add more user input with argparse for more customization of the tool**
2. **Add more harvesters**
3. **Fix Nmap output and add more customization**
4. **Add an open-source license to the project**

**Links to APIs Used**

<https://docs.github.com/en/rest>

<https://ipstack.com/documentation>

<https://hunter.io/api-documentation/v2>

**Python interfaces:**

To make my work easier for this project I used a lot of helper Python classes that do the gathering that I needed for my tool.

List of helper classes: nmap, github, pyhunter, dnsdumpster, argparse, socket, json, sys, rich, ipstack.

**Links:**

<https://pypi.org/project/python-nmap/> , <https://github.com/PyGithub/PyGithub> , <https://pypi.org/project/hunter/> , <https://pypi.org/project/dnsdumpster/> , <https://docs.python.org/3/library/argparse.html> , <https://docs.python.org/3/library/socket.html> , <https://docs.python.org/3/library/json.html> , <https://docs.python.org/3/library/sys.html>, <https://rich.readthedocs.io/en/stable/introduction.html> , <https://pypi.org/project/ipstack/>