Encryption Recommendation for SMEs

Project Report

by

Kewin Skuza

Student ID: C00237361

Project Supervisor: Christopher Staff

Date: 25th April 2022

Abstract

The final year project had its ups and downs. I've encountered plenty of problems most of which were either detection related, or front-end development related. There was not a single problem I didn't learn from. Getting to know more services and languages is always helpful. Even with some features missing in the end I am happy with my achievements. Most sensitive information is detected and neatly displayed to screen. Throughout development I had to deviate from the original flawed design, that's what learning is about. We change and evolve constantly. With every new function I was a smarter person. In the end the tests of the application and seeing that the testers can use and understand it, makes me think that my application is successfully developed.

Acknowledgments

There are a lot of people I would like to thank. Firstly, my classmates for helping me brainstorm some ideas. They gave me some pointers and recommendations on how to go about development. I would like to thank all lecturers for providing learning resources and materials to make this possible. I can't forget about my testers that gave me some helpful feedback to make the application as easy to use as it currently is. And finally, my supervisor Christopher Staff for the weekly meetings and countless questions answered.

Table of Contents

Contents

Problems Encountered

Scope of analysis

Every project developer will encounter problems throughout the development process. I am no exception. I encountered an issue even before I started developing the application. The scope of analysis is huge. Sensitive information is a broad term that encompasses a lot of data, ranging from person names, addresses, passwords, access keys etc. Finding a solution to detect it all was a challenge. At first I tried to teach a machine learning algorithm to identify Personal Identifiable Information (PII) but was unable to find a dataset large enough to make the algorithm reliable. I had to look for an alternative. This left me with two options; either find a pre-trained algorithm or stick to rule-based patterns. Ruled-based patterns had a lot of limitations, so I decided to go with the first option.

Finding a PII detection solution

Deciding on the approach was not the end. Finding a pre-trained algorithm was more difficult than first anticipated. No one will simply upload a pre-trained model for everyone to use, at least I was unable to find one. Eventually I found out that Amazon Web Services (AWS) has a solution called Macie. It detects sensitive information and has a Software Development Kit (SDK) for python named boto3. This was a viable option, so I went with it.

AWS syntax

Even though the algorithm was pre-trained, interacting with it was still a challenge. The requests and responses for AWS are complex. This took me a while to get used to. Even at the end of the development process I had to refer to the documentation and perform my own test on how to parse the responses. Eventually I became adept in flying back and forth between my code and boto3 documentation, making my development process more efficient with every function.

Table resizing

My front-end solution posed some issues as well. I decided to use Tkinter for python. This package is included with python and not too different from other GUI solutions, so I decided to go with it. Everything was going fine until I started working with tables. This package does not handle dynamic table development very well. The table solution provided by Tkinter is called Treeview. My goal was to make the table a certain size and implement vertical and horizontal scrollbars to view data that is out of sight. Also, I would have a button for every database table to allow the user to switch between them. Everything worked fine until the number of columns for every table was different. This made the table resize when you switch between tables. Eventually the table would resize and go out of the screen boundary, despite the fact that I explicitly set the boundaries. When you first view the table the horizontal scrollbar works fine. It's just after switching between differently sized tables that the bug happens. I've spent a long time trying to find solutions but was unable to find it. It looked like it is simply a limitation of Tkinter. The user can restart the application and view the large table first and then go to the smaller tables to view all the information as a temporary fix. A more temporary solution would involve probably switching to a web-based application or trying out alternative GUI libraries.

Detection coloring

This was not the end of my treeview table issues. Initially I wanted to color in the whole column for every sensitive column. From what I could find treeview can only color in the rows but not columns. I was able to find a calendar application which highlights the specific column when you press it. It was using a widget called canvas. It displayed a canvas of the desired color with the dimensions of the cell, take the cell text and display it on the canvas piece. I was trying to implement that but as a column. After hours of work, I was unable to implement it. I was on the other hand able to find an alternative solution. Instead of coloring the whole column I simply created a second blank treeview, of the same size and place it behind the main one. The user would only be able to see the top title row of this treeview just barely above the one with data. I found out that instead of text in the title row you can display an image. So, I got two images, one red and one green. After appropriate resizing I was able to create a small row with detection colors above the main data treeview.

Scrollbar

Now that I had a second treeview a new issue arose. With the horizontal scrollbar implemented, the data treeview would move but the detection coloring treeview would not. I was able to find a guide to bind one scrollbar to multiple treeviews. Now with the scrollbar both would move the exact same amount.

Waiting for processing

When the classification job is being performed the application will be displayed as not responding and the user will not be able to perform any actions until processing is finished. This is an issue due to the amount of time it takes to wait for the job to finish (upwards of 15 minutes, depending on amount of data to classify). This issue arises because I'm using a sleep statement inside a while True loop. While inside this loop no other processing can be done in the background. I was not able to fix this issue with the amount of time given for the project.

Achievements

Desktop application

Before this project I only ever developed web-based applications. This was the first time I attempted to make something that works on desktop. At the beginning I was lost. I had no idea what libraries and packages to use and where to start. I had to do some research and get pointers from the online python community. People online seemed to like the tkinter package so that's what I went with. Finding myself in the world of desktop application development was difficult but I was able to pull though and have a working and clear GUI. The user interface I developed looks clean and intuitive. I wanted to make sure that people of varied technical background where able to use it. I put some time into documenting and writing guides within the application so that users do not have to go back and forth between the user manual.

AWS integration

A good GUI is nothing if the application does not have any functionality. I was able to integrate AWS into my application while making sure that some security features are enabled. To give an example, if you create a s3 bucket using boto3 it is automatically set to public. This is not mentioned, only in documentation. I was able to create a second method that sets the bucket to private to make sure no one can view it.

AWS services seem to be working correctly and the responses are parsed and displayed to the user in a neat manner. This is a huge achievement considering the complexity of the responses you are given back.

Secure by design & code readability

I also was able to develop with security in mind. It is a bad idea to use your root credentials to perform requests. When the user logs in another IAM user is created with minimum privileges (just enough to perform the job they are created for) to perform the requests required. Alongside the user of minimum privilege, I also create temporary access keys that are deleted when the application processing is finished ensuring that they cannot be stolen. At the end of processing, I also delete the file uploaded from the s3 environment and disable Macie, clearing the findings. All of this to make sure minimum amount of information is stored and free to take by threat actors.

Dynamic application

An achievement I'm proud from is the fact that I used minimum hardcoding. The application is generated dynamically no matter how many fields are in the database and how much information needs to be processed. If there are some limitations the application does not crash but displays an error message informing the user of the issue. On top of that the different functions are separated into two files. A file with all the AWS functions and a file with some other detections and scripts. This makes my code cleaner and easier to read.

Rule-Based Patterns

I was able to implement some rules-based pattern solutions to catch some low hanging fruit that my main solution does not detect. I created a function to detect home addresses, password fields and email addresses. The email address solution was implemented into Macie as a custom identifier while the other two into code as functions.

Does as designed

My application is able to perform the tasks it is supposed to perform. It recommends which fields should be encrypted, it gives an estimate on how long the encryption process would take if the user decided to do so and informs the user on encryption and security best practices while maintaining application security of its own.

What's Missing

With a limited timeframe I was unable to develop some functionality. Although the main feature of sensitive information detection is developed there are a few sub features that I did not have enough time for.

Paired detection

Originally I wanted to go through all the data, detect PII remove the detected items from the list. Then I would go through the data again and compare it to other bits of data to see if any paired PII was detected. Because of the fact that I am using a pre-trained model I was unable to change that.

Window resizing

I had big ambitions of developing the application and allowing the user to resize the window as they please. This is a helpful feature but one that is extremely difficult to perform. When I attempted to make it resizable the tables and fields would still break, and the GUI would become unreadable. I decided to stick to making the window a certain size and not allow the user to change it.

Limited support

The application only supports one filetype of one administration tool. Every database and every administration tool have different formats. It is difficult to develop something that is compatible with all of them. With limited time to make this project I just choose one and stuck with it. At the moment my application only works on Windows operating systems. It also only accepts JSON files exported from PhpMyAdmin administration tool. I choose this format as it's the one that I could test on my home device. Thankfully if the user includes a different file the application wont crash, it will display an error message and allow the user to resubmit the correct file.

Traffic light design

Originally I wanted to have a traffic light design for displaying my results. Red would be fields that are sensitive, green fields would be safe to leave unencrypted and amber fields would be fields that my application is unsure about. I was able to implement both the red and green coloring. Amber coloring pose a bigger challenge. With the pre-trained algorithm, I've chosen I was unable to implement this functionality.

Lessons Learned

Web > Desktop

Desktop application development is more challenging than web-based application development. At the start of the development process, I thought that the two approaches are similar in nature. As the development process went I realized more and more that they differ tremendously. It is much more difficult to scale the widgets in desktop applications than web-based applications. It is much easier to display information neatly in the browser with helpful languages like CSS or JavaScript. Even though it was challenging I learned how to use tkinter to develop front-end applications in python.

JavaScript Object Notation

Before this project I was unaware on how to use the JSON file format. It was the first time I had to parse results from JSON. It felt intimidating at first but the more I worked on it the more I got familiar. Now I fully understand JSON and how to work with it.

AWS

Learning certain amazon web services is definitely advantageous. Even though I most likely will not use these services again, unless hired by amazon, it still gave me an insight into learning new libraries and SDKs. Just like learning another spoken language, the more you know the easier it is to learn another one.

Importance of research

When I first viewed the brief I thought that it will be easy. With additional research I quickly realized how difficult this task is becoming. Data detection and classification is very difficult. Also teaching a machine learning algorithm is more challenging that I first anticipated. I learned that research is one of the most important aspects in application development. If I didn't research into machine learning and alternatives I would be stuck with a project that barely identifies any PII.

Python

I also learned the python programming language. This language was not part of our course. I knew only a small bit from my placement project last year. I brushed up on it through this year and became an intermediate python developer.

Python as a programming language is not the best solution for GUI development. Although the front-end capabilities I felt were lacking it did make up in the backend. Pythons' automation and data parsing capabilities are superb. In the future I will definitely choose a different language to do the front-end and keep python in the background.

Changes if Started from Scratch

Using python to do the front-end of the application was not the correct choice in the end. It posed a challenge to make it dynamically resize the table depending on the number of columns in it. If I were to start again I would choose a different language to do the GUI. Another option would include doing a web-based application altogether. HTML and CSS have better capabilities at portraying information. I think that would have been a better option.

A useful thing that I could have done is create more regular expressions (regex). There still are a few types of sensitive data that can be detected using regex. Identifying these and creating regexes might have cut down on execution time. At the moment it takes from upwards of 15 minutes to execute, creating more regexes would definitely be an advantage.

Finding an alternative detection solution to Amazon Macie would be better. Amazon services are not free, you require to pay a certain amount for every couple thousand requests and gigabytes of storage used. Finding an alternative would be better to not force the user to spend money on an external service. Amazon Macie also does not detect some other bits of sensitive information that it should. Home addresses and email addresses were not detected so I had to create alternatives to do so. If I was starting again I could have attempted to teach my own machine learning algorithm or find something other than Macie.

The GUI that I have created does not look ideal. If starting again I would put more effort into making the application look nicer. At the moment it has a navy, easy on the eyes, background with white writing. Other desktop applications use a multitude of colors and styles to create a professional look that my application lacks.

Deviation from Earlier Design

Results display

The way I display my results kept changing throughout the development process. At the beginning I wanted to simply display my findings in writing. This would have been the easiest option but one that is not very user friendly. When we display database data we usually have it in a table. This gave me an idea to display it in a table and color in the fields that are sensitive in red and fields that are not sensitive in green. I gave an attempt to do so but tkinter has its limitations. In the end I had an additional row on top of the table that is colored depending on sensitivity of the information.

Toolbar position

Position of the toolbar with buttons to switch between pages has changed as well. Before this project I ever developed web applications. In these the toolbar is usually north, on top of the screen. When I added in the buttons it looked a little odd. In the end I decided to change the position to west of the screen, which looked better. That allowed me to have the title of the page on top where it belongs.

Results legend

Initially I wanted the legend, explaining results coloring, to be below the table. That was something I had to change the second I started development. When I realized that the treeview table resizes itself depending on the amount of data in the table, I could not risk the legend to be pushed off screen. A solution involved having it simply above the table.

Encryption approximation
Since I recommend which fields should be encrypted I decided to add an approximation of how long it would take to encrypt the user's data. Since the user is using my application, it is safe to assume they need to encrypt it and any additional information could be helpful to them.

Input
Another deviation was how to user enters their data. Initially I wanted the user to login using their database credentials and I would get the information myself. After some thought I realized that a normal user in a business would not have the credentials ready on hand. An alternative was for the administrator to export the database into a file and just include that file within the application, so that's what I went with.

Additional Research
Even after my initial research phase I had to do extra research when I started development. There is always something you require more information on.

AWS services
All amazon services are used in the AWS console. They have their own GUI that a user could use. There is a lot to unpack there. Each service requires three or more other services to work. My application had to join these services together, so the user does not have to. I had to put in a lot of research into how to use the boto3 SDK and which AWS services are required for Macie to function. Even with the required services I had to research best practices on handling access keys that AWS provides for you to authenticate a request.

JavaScript Object Notation
With my decision to use a JSON file as input I had to put some research into the JavaScript Object Notation itself. With no previous experience with JSON, I had to put some time and effort to parse it and work with it. Thankfully python has a build in package named JSON which can load the file into a variable. From there it was just a case of parsing the lists and dictionaries to obtain the data I'm after.

Tkinter
Tkinter is like its own programming language. It has a lot of functions and syntaxes that you need to take into consideration. When developing my application, I had to constantly refer to the documentation and research methods on how to perform certain tasks. A lot of time came into looking up how exactly a treeview table works and what changes are possible to make.

Module Descriptions

Login Screen

Once the user enters their credentials the application sets up the work environment. Firstly, like any other application, it checks if the security credentials entered are correct. If not, a message indicating so will be displayed. To ensure highest level of security possible a custom user with lowest privilege is created. Using root credentials for normal requests can be dangerous.

The application looks for a user named “rec-encrypt_user”. If the user is not created yet it will create it. At the moment the user has no policy. We must create it. A policy named “rec-encrypt_policy” is created if it does not exist yet and bound to the user. The policy only allows the user to call s3, Macie, and IAM functions. There is no need for allowing them to use any other services.

Now with a valid user we generate the access key pair for them. Access keys are required for every AWS request. We make sure to delete these keys on application exit. With access keys generated we need to remember the user id to allow for classification jobs to be performed.

Classification

Setup

After login and when the user chooses their file, more setup and the actual classification takes place. Amazon Macie classifies information stored inside of s3 buckets, so our first goal is to check if a specific bucket is made and if not create it. My create_bucket() function also sets the bucket to private so no one can access the contents except for the user.

With the bucket created and secure the application uploads the file to it while making sure that it is the correct file format (JSON). With the file uploaded we enable Macie to be configured.

Unfortunately, Macie does not detect email addresses. Thankfully it does have an option to create and attach custom identifiers. They can either be keywords or regular expressions (regex). So, the first thing I do is check if the custom data identifier named EMAIL_ADDRESS exists. If it does not then I create it and keep track of the identifiers id.

The job

Creating a classification job in Macie requires a lot of specific information. First the job type. Macie allows for jobs being performed once a week or another specified amount of time. I want the job to be performed only once. So, the jobType option is set to ONE_TIME. We include the email_identifier id we kept track of and tell the system to identify all sensitive information. If we were looking for something specific we could limit the search area. We name the job and point it to the bucket we created. SamplingPercentage option specifies how much of the file we want to check. For some jobs you might only want to check half of the document, for us the whole thing; so we set it to 100.

If we created the job and tried to display the results right away, we would crash our program. I had to create a function that waits for the job to finish and then move on with the execution. In order to do that I made a While loop that checks the job status. If it is “COMPLETE” then the job

is finished. If its “PAUSED” or “CANCELLED” we also want to stop execution. Sending these requests every couple of milliseconds is a good way to overwhelm you AWS console. With the job taking more than 15 minutes I sent a request every minute checking the status. If completed the program execution can continue.

Alternative detection

Now we can do some custom identification on fields that Macie does not lookout for. I’ve noticed that address fields and password/username fields are not detected. These must be detected for best security practices. I created two functions check_for_address_field() and check_for_password_field().

Password and username detection was a challenge, and even now it is not fully reliable. Identifying the field by looking at the data would be impossible. A password could literally be anything, from random numbers to coherent words. My solution involves looking at the name of the field. I compiled a list of commonly used password field names and look out for them.(names include password, pword, pswrd, pass etc.) If one of these keywords is a field name, we know it is a sensitive field and should be secured.

Address detection solution involves uploading the potential address to google and scraping the webpage. In the results I would look out for a keywords like address, rent or property. If a certain amount of these keywords come up, we can be sure it is an address. I also made sure to take in the already sensitive fields as input to the function. This makes sure that the already identified sensitive fields are not uploaded and reclassified.

Result parsing

AWS classification job response is huge and bloated with information that are not useful to the user. I had to parse it. I knew the results are under certain field names. At the end there should be a field called JsonPath which has the directions to where the sensitive information is stored. Once it is found I parse the path to something I can use to display the results to screen. My general syntax is a dictionary of {“field_name” : “table_name”}. If I had a table name as the key There would be a lot of duplicate keys which would break the application.

Encryption approximation

Since I recommend which fields should be encrypted I also created a function which estimates how long such encryption would take. I do this by taking the system time, encrypting the data myself using a recommended algorithm AES and taking the system time after encryption. Now if we subtract those times we get an approximation of how long it took.

Cleanup

Like in every job we need to clean up after ourselves. I simply delete the file submitted to the bucket and disable Macie. As mentioned before the temporary access keys are deleted when the user quits the application.

Data Structures

JSON & AWS

Throughout the project development I had to work with a lot of data structures. Some were common, like strings or integers, and some were complex in nature. The most important data structure within my project was the JavaScript Object Notation (JSON). If we dissect this structure we can notice that it is a bunch of dictionaries and lists within each other. After familiarizing myself with it, using it was enjoyable. It is simple to work with data in the JSON format, especially with python's prebuilt module called JSON. I used this module to read in a JSON file and then I was able to work with it like any object.

Similarly, the responses AWS gives back are JSON like. Referring to the documentation and familiarizing myself with the different result components was crucial to parse the data and take in the important bits.

Classification jobs can look at normal text files or specifically JSON files. When a sensitive field is detected it marks the path to it using JsonPath. I have never heard of this term before viewing the results. Thankfully it is not too complicated to understand. The Macie generated JsonPath is separated into three components using full stops. First component was the index of the file. PhpMyAdmin exports start with a header (which is index 0) then the database name (index 1) and every other table has its own index. That is what is shown within the first chunk of a JsonPath. Next part is data[index]. All information (e.g., username, address etc.) is stored withing the data section of your table. So, the first entry would be data[0] and so on. This points us to where is the actual sensitive field. The last part is the actual field name. If the field within the table was called "Name" then the JsonPath would be ".Name". So, with all this put together if we had a JsonPath structured $[2].data[3].Address, we could deduct that sensitive information is at index 2 of the file, inside the data block at index 3 and the field called address. At first glance it seems complicated but when you actually try to work with it you can get accustomed swiftly.

Tkinter widgets

Tkinter has a lot of different objects named widgets I had to work with. Every button, label, frame, page etc. was an object with different properties. In the grand scheme of the application there is a lot to concern yourself with. I decided to make every page an object. By iterating though each page using a for loop and displaying all of it to screen as a frame I was able to create multiple pages within a single window. Positioning everything within these frames was the biggest challenge in the front-end side of things. With each button and label having different properties and the grid method of displaying widgets, it was difficult to place everything where I envision them. Changing the position of one widget could change the position of another. This took a lot of trial and error to place everything in the correct spots.

Testing

Detection

After testing with a multitude of dummy data from different datasets most of sensitive fields is detected with ease. There is an odd number that Amazon Macie does not detect. The fields that Macie does detect are processed correctly into a sensitive field dictionary.

Other attempts at detecting sensitive data were made to cover areas in which Macie lacks. Macie was not able to detect home addresses, passwords and usernames.

- Addresses detection
    - Most address test cases were successful. The function was able to highlight the fields in which the addresses were stored
    - If the sample size is low names, and email addresses can potentially be detected as address. This does not pose an issue as these fields are also sensitive.
    - The more data is in the table the less false positives there are.
- Password and username fields detection
    - Since it is a simple match keyword function there is not much testing to do. If field name matched the keyword list the field successfully marked it as sensitive.
- Email address detection
    - A regular expression was uploaded to Macie as a custom data identifier. This regex was tested and successful identifies all email addresses.
- All results are compiled into one list allowing for extra detection functions to be added in the future.

User interface

The application is usable. The user can switch between pages successfully without any drawbacks. The user can always go back to the “how to use” page to guide them through the application.

If the database export has more than 7 buttons (more than can fit in one row on screen) the subsequent buttons are pushed down dynamically to be displayed correctly.

The results are portrayed in an easy-to-understand manner. A table with all information is generated and a traffic light approach is used. Red in fact came up for fields that were deemed sensitive and green for those who were not.

To ensure the user understands that something did not go as planned, intuitive error messages are displayed to inform them of the issue.

File compatibility

Since the application by design only allows JSON files exported from PhpMyAdmin I had to test what happens if incorrect files are supplied. For starters, the browse file window defaults to display JSON files and directories only. The user will not have the option to choose a different file. For purposes of extensive testing this feature was disabled.

- Tests performed:
    - The cancel button was pressed (no file chosen):
    - Results:
    - The browse file window vanished and a message telling the user to choose a file displayed (no crash)
- A non-JSON file chosen:
    - Results:
    - The browse file window vanished and a message telling the user to choose a JSON file displayed (file not chosen, no crash)
- JSON file not exported from PhpMyAdmin:
    - Results:
    - The browse file window vanished and a message telling the user that only PhpMyAdmin JSON export are allowed (file not chosen, no crash)
- JSON file exported from PhpMyAdmin:
    - Results:
    - The browser file window closes, all previous error messages vanish and choose table buttons appear to begin processing

Security

The application has a login screen that requires a valid access key pair from AWS. If the incorrect key pair is inserted an error message is displayed. The login system seems to work. When logged in a new user with minimum privileges is created and access keys for that user are generated. That user by default does not have access to the AWS web console. Temporary access was given to check if indeed the policy applied, and the access key was generated.

Results:

- The Access keys were generated
- The policy is made and bound to the user
- The new user can perform all application tasks but nothing else

Temporary credentials should be deleted after application close. The application was closed using the x button from every page in the application and the access keys were indeed deleted every time.

The login screen also masks the secret access key input field with “***” successfully.

The file uploaded to the s3 bucket is successfully deleted right after processing. Also, Macie is turned off to delete all findings and results from the web console after use.

User testing

The application was given to four users for testing. They were divided into two groups based on their experience with using a computer.
These groups were:

- Avid users (users that use the computer on a daily basis)
- Novice users (users that rarely use a computer)

Both groups have no experience in the field of encryption and data security.

Avid users were able to log in and use the application with ease. The results were clear to them and were able to identify which fields would be considered sensitive.

Novice users struggled to use the AWS web console, so I added a guide on the login page on how to generate security credentials. After login they struggled to understand the file upload mechanism. This cannot be changed as the file browse menu is standard for windows devices. When the results were displayed they could easily understand which fields are sensitive and should be encrypted and which fields don't.
When stuck, novice users did not go back and refer to the "how to use" menu explaining everything. I added small bits of information on each page indicating what is happening and what the user requires.

DECLARATION

*I declare that all material in this submission e.g. thesis/essay/project/assignment is entirely my/our own work except where duly acknowledged.

*I have cited the sources of all quotations, paraphrases, summaries of information, tables, diagrams or other material; including software and other electronic media in which intellectual property rights may reside.

*I have provided a complete bibliography of all works and sources used in the preparation of this submission.

*I understand that failure to comply with the Institute's regulations governing plagiarism constitutes a serious offence.

Student Name: (Printed) Kewin Skuza

Student Number(s): c00237361

Signature(s): Kewin Skuza

Date: 25th April 2022