Cyber Threat Intelligence: Cybercrime in the Clear

Thomas Birk Frederiksen

Masters of Science in Engineering

2023

STUDENT REPORT

AALBORG UNIVERSITY

AALBORG UNIVERSITY
STUDENT REPORT

Cyber Security Group
Aalborg University
A.C. Meyers Vænge 15
2450 Copenhagen SW
Denmark
http://www.aau.dk
vbn@aub.aau.dk

Title:
Cyber Threat Intelligence:
Cybercrime in the Clear

Theme:
Cyber Security

Project Period:
Spring Semester 2023

Project Group:
1

Participant(s):
Thomas Birk Frederiksen

Supervisor(s):
Dimitrios Georgoulias, PhD Researcher, Cyber Security, Aalborg University
Jens Myrup Pedersen, Professor, Department of Electronic Systems, Aalborg University

Copies: 1

Page Numbers: 120

Date of Completion:
May 22, 2023

Abstract:

The advent of the internet brings a plethora of advantages; however, it also provides a breeding ground for illegal activities, encompassing the trade of malicious software, sensitive data, and other illicit merchandise. Current research mainly investigates products on darknet markets or interactivity between actors on the forums. However, little attention is devoted to the realm of digital commodities found within illicit clear web forums. This thesis aims to address this by employing a custom-built web scraper to collect data from a suite of illicit clear web forums. Subsequently, large language models are fine-tuned to conduct natural language processing tasks aimed at extracting information concerning prices and predominant product categories. The resulting dataset reveals that illicit digital products spanning various categories sell for 25$ on average with prices occasionally plunging as low as 0.5$. Additionally, users of the forums generally request products 5$ cheaper than what they are available for. Lastly, the thread topics encountered on the investigated forums predominantly revolve around the sale of compromised accounts as well as various services, including account-banning services, botnet services and web hosting services.

Acknowledgements

During the development of this project, I have been given support from many angles. I would distinctively like to like to express my sincere gratitude to Dimitrios Georgoulias, PhD Researcher in Cyber Security, and Jens Myrup Pedersen, Professor in the Department of Electronic Systems, both from Aalborg University. Their invaluable guidance, expertise, and flexibility, have been instrumental in the successful completion of my research. Their insightful feedback and encouragement have significantly contributed to shaping my ideas and enhancing the quality of my work. Thank you for your support and for being exceptional mentors throughout this process.

Contents

List of Figures

List of Tables

List of Listings

Glossary

ML	Machine Learning
TP	True Positives
FP	False Positives
FN	False Negatives
TN	True Negatives
NLP	Natural Language Processing
LLM	Large Language Model
NER	Named Entity Recognition
TC	Text Classification
BL	Bulk Labelling
TN	Topic Modelling
DOM	Document Object Model
Div	Division Element
CSS	Cascading Style Sheets
HTML	HyperText Markup Language
CLI	Command Line Interface
CAPTCHA	Completely Automated Public Turing test to tell Computers and Humans Apart
RPC	Remote Procedure Call
CRUD	Create, Read, Update and Delete (operations)
TOR	The Onion Router
IP	Internet Protocol
DDoS	Distributed Denial-of-Service

1 Introduction

The internet has become an integral part of our lives and has opened up numerous opportunities for businesses, researchers, and consumers alike. However, the vastness of the web also provides a platform for illegal activities, including the sale of malware, database breaches, and other illicit products. The clear web, or the area of the internet that common web browsers and search engines can easily access consists of public websites indexed in search engines and accessed by millions of people worldwide every day.[1] This differs from the dark web and deep web, which are not indexed by search engines and, in the former case, require special permissions or software to access, commonly to mask the user's identity.[2]

Despite the anonymity the dark web provides, certain illicit forums still choose to offer their services on the clear web for their extended accessibility. These forums engage in illegal activities, such as selling ransomware, that can breach the security of systems and the privacy of users.[3],[4],[5] The easy access the clear web provides allows these illegal platforms to thrive and target a wide range of users to help perpetuate their illegal undertakings. At its peak, a single illicit forum may contain listings selling access to up to 1.5 million compromised computers (bots) and more than 80 million stolen account access credentials.[6] With the average price of a bot being around US $16[7] and the average price of a compromised account being US $23[8] forums like these can accumulate up to US $1.86 Billion in revenue.

The information found on these felonious forums has always been a significant contributing factor in disrupting cyber-criminal operations and is hence, a topic of great interest for law enforcement and researchers alike.[9] Specifically, the information these markets comprise - product listings, pricing, and available vendors can help researchers better understand how these markets operate on an economic scale. These forums, however, often have a brief lifespan, typically lasting from 1 to 3 years[10], before being rendered inaccessible as a result of law enforcement agencies infiltrating and confiscating them. Consequently, the task of analysing such forums remains a continuous process. Concurrently monitoring these forums represents a valuable strategy for researchers and law enforcement agencies to detect active malware, exposed databases, and other potentially harmful services. This, in turn, can help mitigate the impact of these products on consumers.

Despite the extensive research on the contents and ramifications of illicit forums and marketplaces on the dark web[11],[12],[13],[14], little attention has been directed towards digital products found on the forums operating on the clear web. This project seeks to narrow this scope by performing a targeted analysis of hacking, cracking, and leak-related clear web forums, with the aim of providing an up-to-date assessment of prices for exposed databases, malware, cracked software and other similar products. By doing so, this study aims to offer a more nuanced understanding of the scope and nature of criminal activities on the clear web specifically related to these topics. The main research question this thesis aims to answer is, therefore:

> ***What is the scope and nature of digital products related to hacking, cracking, and information-leaking for illicit forums operating on the clear web?***

To efficaciously answer this question, it is broken down into the following three sub-research-questions:

1. *How can a web scraper be developed to extract data from illicit clear web forums?*
2. *How can Natural Language Processing (NLP) techniques be applied to extract information from text data?*
3. *What are the common characteristics of products and services on these forums, and how do they vary across different forums?*

To answer these questions, firstly, web scraping techniques are applied to gather and analyse data from clear web illicit forums. The scraping process involves building a web crawler to extract data from the unlawful forums and store it in a structured format for analysis. To create the web crawler, modern Python packages for web scraping and HTML parsing are used, ensuring high performance without the need for specialised hardware.

Secondly, NLP techniques such as ***N**amed **E**ntity **R**ecognition* (NER) and ***T**ext **C**lassification* (TC) are applied to extract relevant information in the textual data. This will involve fine-tuning custom ***L**arge **L**anguage **M**odels* (LLMs) for each of the aforementioned NLP techniques based on state-of-the-art pre-trained LLMs. Thirdly, the collected data is analysed to provide a comprehensive overview of the products and services on each of the investigated forums throughout the entirety of 2023. The insights gained from this project, thereby, contribute to the field in the following ways:

- It generates a robust web crawler that effectively bypasses anti-scraping techniques when scraping illicit clear web forums
- It implements an NLP pipeline to parse data scraped from clear web illicit forums based on several state-of-the-art LLMs
- It provides a summarised quantitative analysis of four popular clear web hacking forums with a focus on digital assets

The contributions above will provide valuable information to help identify and track illegal activities and develop strategies to combat them. Additionally, the information gathered in this project can be used to inform policies and regulations aimed at reducing the scale of illegal activities on the clear web. To access the data and the developed scraper, interested parties can make a formal request to the final custodian of the data, scraper source code and NLP pipeline: Aalborg University (AAU).

The following structure is adopted for the organisation of the thesis. Firstly, in **Chapter 2**, a comprehensive review of background knowledge related to web scraping, data structure, natural language processing (NLP), and machine learning model (LLM) development is presented. Secondly, **Chapter 3** includes the research methods employed for the development of the thesis. **Chapter 4** provides an overview of the current methods and approaches utilised in academia to analyse illicit forums. Next, in **Chapter 5**, the system requirements for the developed scraper as well as the performance requirements of the fine-tuned LLMs used for NLP information extraction, are defined. Hereafter **Chapter 6** covers the overall structure of the developed system concerning the entire pipeline from scraping data to information extraction for analysis. Following this, in **Chapter 7**, a comprehensive description of how the system was implemented is described. **Chapter 8** concerns the analysis of the extracted and processed data, and in **Chapter 9**, a discussion takes place regarding the entire project and any future extensions of the work. Finally, **Chapter 10** provides a conclusion with a concise overview of the results of the project.

2 Background

The purpose of this section is to provide an overview of the theoretical framework behind creating the web crawler to scrape the illicit forums, as well as building an NLP pipeline to process the scraped data. As stated in **Chapter 1**, this will include the use of modern Python web scraping libraries, for efficiency and consistency. For the NLP pipeline, this will include NLP techniques like NER and fine-tuning of LLMs. In **Section 2.1** a thorough background to web scraping will be granted, followed by **Section 2.2**, where an overview of the underlying architecture used for storing and manipulating the scraped data is given. Lastly, **Section 2.3** provides all the necessary information needed to comprehend the NLP techniques applied in this project.

2.1 Scraping Techniques

Various techniques exist when it comes to extracting data from a website, each with its own advantages and caveats. In the following subsections, different approaches to web scraping are discussed.

2.1.1 REST API's

R*epresentational* ***S****tate* ***T****ransfer* ***A****pplication* ***P****rogramming* ***I****nterfaces* (REST APIs) are offered by many websites to provide programmatic access to their data. These APIs allow for a structured and efficient way of accessing data without the need for web scraping. Instead, developers can directly request the specific data they need, saving time and resources. An example of this could be Twitter's API, which enables programmatic access to Tweets on Twitter.[15] APIs function by sending ***C****reate*, ***R****ead*, ***U****pdate* and ***D****elete* (CRUD) that are subsequently executed by the receiver.[16] Unfortunately, not all sites offer REST APIs or don't make them publicly available, especially forums discussing illegal matters. Therefore two other methods are mainly used when scraping data from a website: *Static Web Scraping* and *Dynamic Web Scraping*.

2.1.2 Static Web Scraping

Static web scraping is typically used for websites where the data is not generated dynamically using JavaScript, which most modern websites are. Static web scraping involves parsing and extracting data from static HTML pages using libraries like `BeautifulSoup`, which is a popular Python library that enables parsing of HTML and XML documents, making it an ideal choice for web scraping. It offers a simple and elegant syntax that allows you to navigate and search the HTML tree structure of a webpage.[17] This is as simple as searching for a specific tag or class name as seen in **Listing 2.1**.

Listing 2.1: Fetching and parsing HTML with BeautifulSoup

```
# Get raw html from a website
url = "www.google.com"
response = requests.get(url)
# Convert to BeautifulSoup object
soup = BeautifulSoup(response.text, "html.parser")
# Get the title of the page
title = soup("title")[0].text
# Get the first div with the class "pagination"
soup.find("div",{"class":"pagination"})
# Get all the divs with the class "pagination"
soup.find_all("div",{"class":"pagination"})
```

Using this technique, one can easily parse raw HTML to find specific parts of a web page via CSS selectors, as seen in the last two lines of code. CSS selectors are generally easier to use and more readable than XPath expressions, though XPath expressions tend to be faster.[18]

2.1.2.1 Regular Expressions

To enhance the efficiency of data extraction from HTML pages, regular expressions (*regex*) can be employed in addition to the CSS and XPath selection techniques mentioned earlier. Regex expressions are patterns that match to some predefined targets and can be created utilising Python's regex library. This approach is particularly advantageous for websites with irregular or unorganised data formats, like forums. As depicted in **Listing 2.2**, the process of extracting links from the raw HTML of a website becomes remarkably simple with the use of regex.

Listing 2.2: Extracting links from HTML with regular expressions

```
import re
import requests
response = requests.get('https://www.example.com')
# Use regex to extract all the links starting with "http(s)://"
links = re.findall('href="(https?://.*?)"', response.text)
```

The regular expression `href="(https?://.*?)"` is a commonly used pattern for extracting URLs from HTML code and consists of three key elements. Firstly, the string `href="` matches the literal characters `href="`. This is due to the fact that URLs are typically enclosed within an HTML anchor tag (`<a>`) with an href attribute. Secondly, the pattern `https?://` matches any string starting with either *http://* or *https://*, where the question mark after the *s* character makes it optional. This allows the expression to handle both secure and insecure URLs. Finally, the `.*?` expression matches any sequence of characters until the first occurrence of the closing quotation mark. This ensures that the pattern matches the smallest possible string that includes the URL. These expressions are used extensively in this thesis to extract as much data as possible on the forums without having to manually define the CSS selector or XPath for every string on the site.

2.1.3 Dynamic Web Scraping

In contrast to static web scraping, dynamic web scraping is a technique employed to extract data from websites that do use JavaScript to generate content. This is often conducted with libraries such as `Selenium`[19] or `Nightmare.js`[20] to automate browser interactions and simulate user behaviour. By opening a browser window and programmatically navigating to the target page, these tools can interact with the page and extract data of interest. However, the use of dynamic web scraping tools introduces additional complexity, as they not only need to parse the website but also interact with its elements, leading to slower processing times. Additionally, since the website must visually load for the tool to interact with it, dynamic web scraping requires more computational resources.

Despite this, there are situations where dynamic web scraping is more suitable than static web scraping. One example is when dealing with complex authentication mechanisms that require users to log in before accessing specific data or features. In such cases, dynamic web scraping tools can automate the login process and extract data from the website after authentication, unlike cookie-based authentication, which can be handled by a combination of tools like the `requests` and `BeautifulSoup` Python packages. Moreover, packages such as `Selenium` can be integrated with various other tools to overcome challenges such as CAPTCHA verification,[21] thereby extending the functionality and applicability of the web scraper.

A useful criterion for selecting between dynamic and static web scraping tools is to consider whether the web page is server-side rendered or not. In cases where the website is not server-side rendered, dynamic tools are often more appropriate for scraping data, as much of the data is generated using JavaScript. Contrarily, if the website is server-side rendered, the entire HTML document is returned upon request, making it easier and faster to obtain and parse the document using local tools like `BeautifulSoup`.

2.1.3.1 Selenium & Undetected Chromedriver

The framework utilised for creating the scraper in this thesis is based on the aforementioned `Selenium` toolkit. The underlying architecture of Selenium is based on the implementation of *web drivers*, which serve as APIs for communication between `Selenium` and *web browsers*. The web driver, when instantiated, enables `Selenium` to launch a browser window, navigate to the intended webpage, and interact with its various elements programmatically.[19] Utilising `Selenium`, however, instantiates a set of JavaScript variables that are easily recognised by bot detection systems. For instance strings such as *Selenium* or *WebDriver* in any of the instantiated variables is an overt indicator.[22]

To surpass this limitation, a third-party module called `undetected_chromedriver`[23] has been developed as an extension to the `Selenium` package. This instantiates a Selenium driver with modified headers, closely resembling those of a regular user's browser, which as a result, mostly bypasses security measures such as CloudFlare and hCAPTCHA.[23] By incorporating this package, the features of `Selenium` can be leveraged to extract data from illicit forums while limiting the risk of being barred from the site.

2.2 Data Storage

When a web scraper has extracted data from a website, it must subsequently be stored for analysis. When storing the data, the choice of an appropriate data storage system is crucial to ensure the efficiency and scalability of data processing. *Relational* and *non-relational* databases are two primary paradigms used in the storage and management of data. The relational data model uses tables to store data in a structured format with predefined columns and rows. These tables can be interconnected through relationships, and data can be queried via a ***S****tructured* ***Q****uery* ***L****anguage* (SQL). In contrast, non-relational databases (also known as NoSQL databases) do not rely on a fixed schema or structure and use flexible data models such as documents, key-value pairs, and graphs to store and manage data. With this design, they are better equipped to handle unstructured data, providing greater scalability, performance, and fault tolerance than relational databases in many use cases.[24] A visual representation of the differences between a relational and non-relational database can be found in **Figure 2.1**.

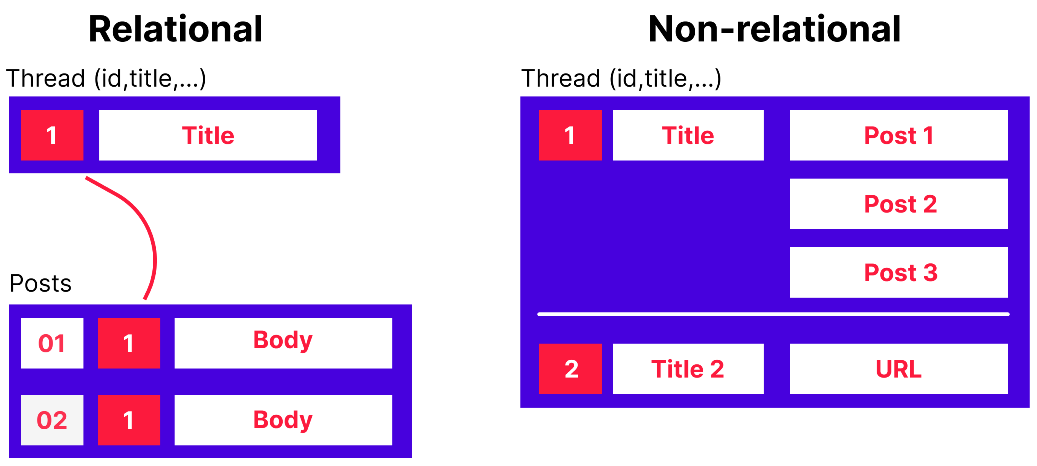


Figure 2.1: A visual representation of the difference between relational and non-relational database schemas (Created in Figma.[25] Inspiration: [24])

When it comes to forum data, everything is interconnected with users creating threads that contain replies which are subsequently written by users. To store this convoluted information efficiently, a non-relational database scheme is hence ideal. Moreover, text is inherently unstructured, further incentivising the use of a non-relational database paradigm. A thread entry in the dataset created for this thesis, therefore, looks similar to the non-relational entry in **Figure 2.1**, simply with many more key-value pairs.

2.2.1 SurrealDB

Common non-relational databases, like `Dgraph`[26], provide this non-relational database structure. However, this has some limitations in terms of query flexibility. `SurrealDB`[27] was therefore chosen for its immense flexibility and ease of use. `SurrealDB` allows the use of both an SQL-like and GraphQL-like querying language while seamlessly being able to function as a graph database as well as a regular relational database. Data is stored as *Records* (or vertices) that can be interconnected via edges. This then makes it possible to retrieve documents from any depth from multiple tables within the database efficiently, without the use of complex JOIN operations.[28] Apart from its features `SurrealDB` was utilised as it functions at a quite high level of abstraction, meaning efforts could be directed more towards the development of the web scraper and NLP pipeline.

2.3 NLP & Machine Learning

Natural Language Processing (NLP) is a subfield of computer science and artificial intelligence that deals with the interaction between computers and human languages. In recent years it has become a prominent research topic due to the increasing amount of text data available on the internet and the need for automated processing of this data.[29] One of the primary challenges in NLP is the inherent complexity and variability of human language, which makes it challenging to extract information and insights from. Machine Learning (ML), a subfield of artificial intelligence, plays a crucial role in NLP by providing algorithms and models that can learn from data and improve their performance over time. By training models on large amounts of annotated text data, algorithms can be developed that can perform a range of NLP tasks with high accuracy. The following subsections outline a selection of approaches for processing and extracting information from text data and present an overview of the pre-trained language models (LLMs) utilised in this thesis.

2.3.1 Feature Extraction/Encoding

Machine learning algorithms cannot work with raw text directly. The text must therefore be converted into numbers. Specifically, vectors of numbers. This process is called *feature encoding* or *feature extraction*. Not only must the text be converted to numbers, but it must also have a consistent length to be fed into the algorithms. A multitude of options exists for feature extraction, starting from the simpler variants such as *Bag of Words*, scaling up to the complex contextualised word embeddings used by state-of-the-art LLMs such as BERT.[30] To provide context, the following subsections will first introduce simpler encoders before covering the more complex encoding methods.

2.3.1.1 POS Tagging

P*art-****O****f-****S****peech* (POS) tagging is the process of assigning a grammatical tag to each word in a sentence, indicating its part of speech (e.g., noun, verb, adjective, etc.). This task is especially important in resolving the ambiguity of a word. For example, consider the word *bat*. Without determining its syntactic category, its meaning remains ambiguous. However, by examining the surrounding words, it is possible to ascertain whether the word in its context refers to the mammal or the sports equipment.[31] This type of inference can be achieved with software such as `spaCy`[32], a Python package containing pre-trained high-performing models for POS tagging. In the context of this thesis, this can be utilised to extract *items* being sold on the forums or *things* being requested by looking for words tagged as nouns. The implementation of this will be given in **Section 7.4.1**.

2.3.1.2 Contextual Embeddings

Simple feature encodings often represent text by describing the occurrence of words within a document. This involves two essential elements. Firstly, a predetermined vocabulary consisting of known words, and secondly, a measure of the presence of these known words within the document. This approach is termed a *Bag Of Words* (BoW) as it disregards any information about the order or structure of words within the document. The model is only concerned with whether known words occur in the document rather than their specific location within the text. While this can be used for simpler tasks such as document statistics or simple classification, it fails to properly grasp the rich semantic relationships and dependencies that exist within the text.[33]

To circumvent this *Word Embeddings* can be utilised. These function by initially building a global vocabulary of unique words in the document. The similarity between words is then learned by finding words that appear more frequently close to one another. However, this also comes with a few caveats. Namely, when building the global vocabulary, the meaning of words in different contexts is ignored. For instance, in the sentence *The bat flew over the baseball field while the child swung his bat at the pitch.*, only one representation of *Bat* is learned.

This brings us to the more powerful approach of *Contextual embeddings*. Contextual embeddings assign each word a representation based on the words around it, thereby capturing the uses of words across varied contexts and encoding knowledge that transfers across languages.[34] Take the word *Bug* for instance. Depending on the context, this word can mean an *Insect*, a *Coding Error* or a *Surveillance Device*. By utilising contextual embeddings, this word can be encoded according to its context into a feature space as depicted in **Figure 2.2**.

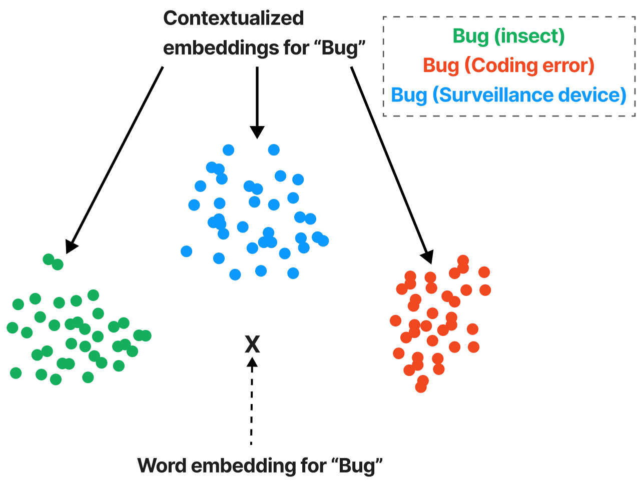

Contextualized embeddings for "Bug"
Bug (insect)
Bug (Coding error)
Bug (Surveillance device)
X
Word embedding for "Bug"

Figure 2.2: Fictional contextualized embedding vector points and clusters for the word *Bug* (Created in Figma.[25] Inspiration: [35])

With this approach, powerful LLMs that comprehend context much better can be implemented and subsequently be fine-tuned for a large range of NLP-related tasks.

2.3.2 Transformers & Attention

Many previous works employing contextual word embeddings have adopted architectures such as ***R**ecurrent **N**eural **N**etworks* (RRNs) and ***L**ong **S**hort **T**erm **M**emory* (LSTM) architectures. However, these architectures are inherently sequential, prohibiting the possibility of parallelisation, consequently inducing memory constraints during training. The sequential nature of these architectures in terms of natural language processing also means that for a given word in a sentence, they only try to encode context based on the words preceding that word. As a result, a part of the context for the word is lost, as no attention is given to the words following the word.[36]

To effectively surpass these constraints, researchers at Google[37] introduced the concept of *Attention* in 2017.[36] The key concept behind attention is that given an input sequence, it can *attend* to another sequence for context. Further improving this, the concept of *Self-attention* emerged for the *Transformer* machine learning architecture, allowing the transformer model to attend to different parts of the same input sequence for context. In the context of text, given an input sentence, each word can look at every other word for context, allowing the model to effectively comprehend the sentence.[36] This invention led to a series of remarkably capable LLMs such as BERT.

2.3.2.1 BERT: Not Just a Sesame Street Character Anymore

***B**idirectional **E**ncoder **R**epresentations from **T**ransformers*), or *BERT* for short, is a Machine Learning model based on transformers, likewise developed by Google, that has gained a lot of popularity since its inception in 2018. BERT was trained to simply pay *attention* to what is most important for understanding the context of words in a given sentence. Given that it is bidirectional, this allows it to look at the whole context of a sentence. For instance, in the sentence *She left her left shoe at home*, BERT can grasp the semantics of the word *left* in both instances to correctly understand the first occurrence of *left* as a verb, and the second as an adjective. Additionally, BERT was trained on 3.3 Billion words,[30] making it exceptionally good at understanding context in a wide spectrum of input sequences. This pre-trained version of BERT is available via the `transformers`[38] Python package, granting easy access to fine-tuning this high-performing LLM to tasks such as text prediction, text summarisation and many more.

To effectively employ BERT for the tasks in this thesis, the input to BERT has to be tokenised via a *word-based* tokeniser. In practice, this entails the following 3 pre-processing steps:

- The model requires a `[CLS]` token at the beginning of each sentence (ID 101) and a `[SEP]` token at the end of each sentence (ID 102)
- The model's input layer has a certain size, and hence the input should be formatted to a specific length. This can be achieved by padding the tokenised text (adding `[PAD]` tokens (ID 0) so the vectors always have the same length. The maximum length allowed by the base BERT model is 512 tokens

- An attention mask should be generated, consisting of a list of 0's and 1's that indicate whether a given token, given by its ID, should be considered when learning their contextual representation in the text

Visually this can be represented as illustrated in **Figure 2.3**.

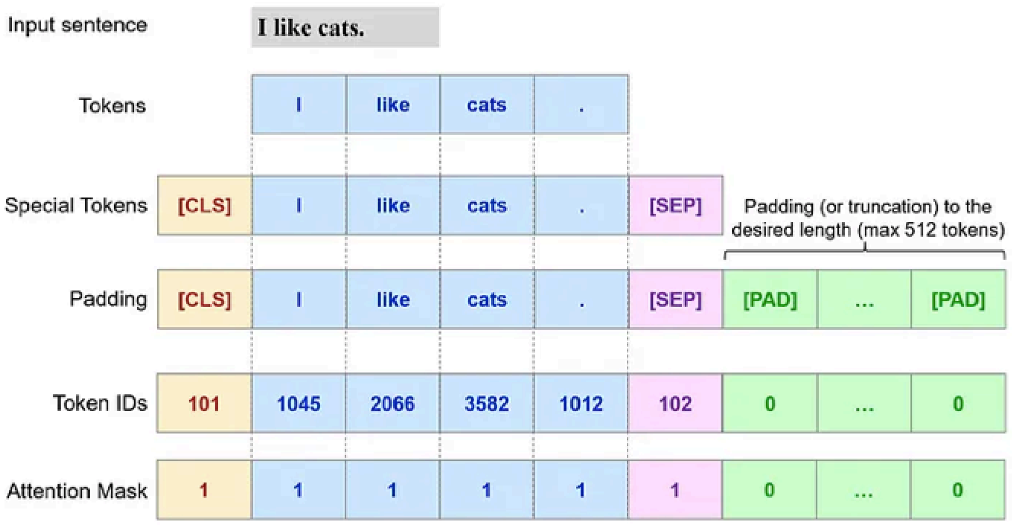


Figure 2.3: Visual representation of the tokenisation of text for the BERT architecture (Source: [39])

The figure clearly depicts, how a given input sentence such as *I like cats*, is converted into a sequence of numbers, with each word or special character having a special id. Additionally adding the `[CLS]` and `[SEP]` token the model can comprehend when a sentence starts, and when a sentence stops, further increasing its ability in understanding the context the sentence tries to convey.

2.3.2.2 XLnet

When BERT is trained, one of the tasks it is given is to fill in masked input, i.e. fill in the missing word in a sentence. However, as it is an *autoencoding* model, it does not consider how each mask in a sentence might be connected. For instance, in the sentence *She went to the [MASK] to buy [MASK]* an appropriate solution would be that *She went to the* ***pharmacy*** *to buy* ***medicin***. However, seeing as BERT does not consider the connection between the two masks, *She went to the* ***pharmacy*** *to buy* ***popcorn***, is also a valid solution despite it being quite unlikely. BERT has therefore been bested in many NLP tasks by *XLNet*, an *autoregressive model*, which does consider the permutation of these masks. As with BERT, the input to XLNet must be tokenised, however, via a slightly different subword-based tokenisation algorithm.[40]

2.3.2.3 RoBERTa

R*obustly* ***O****ptimized* BERT, or RoBERTa, is a reimplementation of BERT with a slight difference in the masking task explained in **Section 2.3.2.2**. With BERT, the number of different ways a sentence is masked is fixed to 10, while for RoBERTa, the generation of masks from the input sequence is done dynamically, potentially increasing the number of different masking variations. Consequently, Yinhan Liu et al. could report RoBERTa introduced an increase in performance when compared to BERT. As the model is based on BERT, but simply trained differently, RoBERTa uses the same tokeniser as BERT.[41]

2.3.2.4 DistilBERT

The *distilled* version of BERT, or DistilBERT, is, as the name implies, a smaller, cheaper, faster and lighter version of BERT. By reducing the number of parameters utilised in the model by 40%, DistilBERT runs up to 60% faster than BERT while preserving over 95% of the performance. These metrics have popularised the model, as it allows for similar performance while reducing computational foodprint, consequently enabling its use in constrained devices such as recent smartphones. Despite the reduction in parameters, the model, as RoBERTa follows the same tokeniser structure as BERT.[42]

2.3.3 spaCy NLP Pipelines

`spaCy`[32] is a popular open-source NLP framework that allows the easy creation of pipelines, which represent a sequence of processing steps that transform raw text into meaningful linguistic annotations. For a basic NLP pipeline, spaCy first tokenises the text according to the model specified, which can then subsequently be passed through NLP models such as POS taggers, entity recognisers or text classifiers. Each processing step is defined in an initial configuration file along with their respective hyperparameters. Additionally, it comes with a set of pre-trained word-vectors that encapsulate general comprehension of various languages. In this thesis, their English word vectors were therefore utilised for reduced processing time and increased accuracy.

2.3.4 Machine Learning Metrics

To effectively evaluate the performance of machine learning models fine-tuned for the NLP tasks of this thesis, a selection of metrics are calculated while training the models. These metrics provide quantitative measures to assess the accuracy, precision, recall, and overall effectiveness of the models. Each of these measures is calculated based on four variables produced during training: the number of *True Positives* (TP), the number of *False Positives* (FP), the number of *False Negatives* (FN), and the number of *True Negatives* (TN). These variables represent the outcomes of the model's predictions compared to the ground truth labels.

Precision measures the proportion of correctly classified positive instances out of all instances predicted as positive. Consequently, this measure is of particular importance in situations where minimising false positives is crucial, such as for medical diagnoses. *Accuracy* measures the proportion of correctly classified instances, both positive and negative, out of the total number of instances. As such, accuracy is only an indicative performance metric when the dataset is balanced. I.e. if the dataset consists of 90% label 1 and 10% label 2, the model still achieves 90% accuracy when always predicting label 1, regardless of the model's ability to accurately classify instances of label 2.

Recall, also known as sensitivity or true positive rate, measures the proportion of correctly classified positive instances out of all actual positive instances. As such, recall is useful when the consequence of missing positive instances is significant, such as in disease detection. Finally, the *F1-score* combines precision and recall and is, therefore, especially valuable when dealing with imbalanced datasets, given its ability to consider the trade-off between identifying positive instances (recall) and minimising false positives (precision). Thus it ensures that the model's performance is adequately evaluated even on the minority class.

2.3.4.1 Loss

In machine learning, *loss functions* quantify the dissimilarity between the predicted and actual labels during the training process. *Training loss* therefore serves as an optimisation objective, allowing the model to update its parameters during training to minimise the loss and improve performance. As such, it provides a measure of how well a model is learning during each iteration (epoch), which, when visualised, provides insight into the model's capability to learn from the dataset. *Validation loss*, on the other hand, serves as an indicator of how well the model generalises to new, unseen examples. The validation loss is calculated on a separate validation dataset that is distinct from the training data. This is done to ensure the model does not become *Overfit* to the dataset, i.e. to avoid having it memorise the training data rather than learning the underlying patterns.

After the model has been trained and fine-tuned on the training and validation set, its performance must be evaluated on an entirely independent dataset. This dataset is unseen and representative of real-world scenarios and thus provides an unbiased estimate of how the model will perform when making predictions on new, unseen instances. A common way to split the train, validation and test set is 80% training, 10% validation and 10% test.[43]

3 Methodology

The primary objective of this project is to obtain comprehensive information regarding the scope and character of illicit activities prevalent on clear web forums. As stated in the introduction, to accomplish this, web scraping techniques will be utilised to gather data from these forums, and NLP will be employed to analyse it. This section will outline the organisational structure of the project and the methodologies employed to achieve the research objectives.

3.1 Project Management: Agile Methodology

The project was managed using an Agile approach, breaking it down into smaller sprints with specific goals and objectives. At the end of each sprint, progress was reviewed, and the plan was adjusted based on the supervisors' feedback. This approach allows for flexibility and adaptation to changes, ensuring the data gathered is relevant, timely, and accurate. Each sprint concerned a site being scraped or a dataset being analysed, with lessons learned implemented in the scraper and NLP models for future iterations.[44] That isn't to say work was conducted sequentially. While a sprint with the goal of setting up the scraper for one site was completed, concurrently, the scraper was set up for another site, and the same approach was taken for the training of NLP models.

Although the Agile methodology has many benefits, such as increased flexibility and faster time-to-market, it is important to acknowledge its limitations. **Harleen K. Flora et al.**[45] stated that one of the main drawbacks of Agile is that the constant changing of requirements can lead to never-ending projects and a lack of predictability, making it difficult to establish a clear vision for the final product and manage schedule, budget, and scope. For instance, the target data for selected forums may change as different analysis approaches are taken, requiring extensive rewriting and rerunning of the entire scraper for each site. To counteract this, when creating the scraper, it was set up to gather as much visible data as possible, regardless of if it may currently be of interest. This may include info such as awards granted to the user or who most frequently visits the user's page.

Additionally, as Agile requires minimal documentation, it can also make it challenging for new developers to understand the software's development process, as the internal design changes frequently based on changing requirements after every iteration. This can result in a lack of detailed documentation of design and implementation, making it difficult for new developers to pick up where the previous team members left off. To battle this, `Jupyter Notebooks`[46] were utilised, providing a clear, interactive description of each step in running both the scraper and the NLP pipeline.

3.2 Selection of Target Forums

The project's initial stage will involve a comprehensive literature review and a thorough investigation of online hacking forums to identify the unlawful hacking forums operating on the clear web. The chosen forums must reflect the current state of these activities and be selected based on factors such as the nature of illicit goods or services being offered, the forum's magnitude, and its general popularity within the online community.

Locating these forums and forums can be challenging and potentially risky due to their covert nature. Hacking-related forums, such as Reddit's /r/hacking[47] or /r/BlackHatHackers[48], are common places to find links to these forums. Once a forum has been found, it must be examined in more detail by creating an account, browsing available goods and services, and potentially interacting with other users.

However, it is crucial to approach this process with care and ethical considerations, as the potential harm and legal implications of engaging with illicit forums must be considered. Any data collected must be used confidentially and appropriately while maintaining transparency in research practices. These considerations will be elaborated upon in **Section 3.8**.

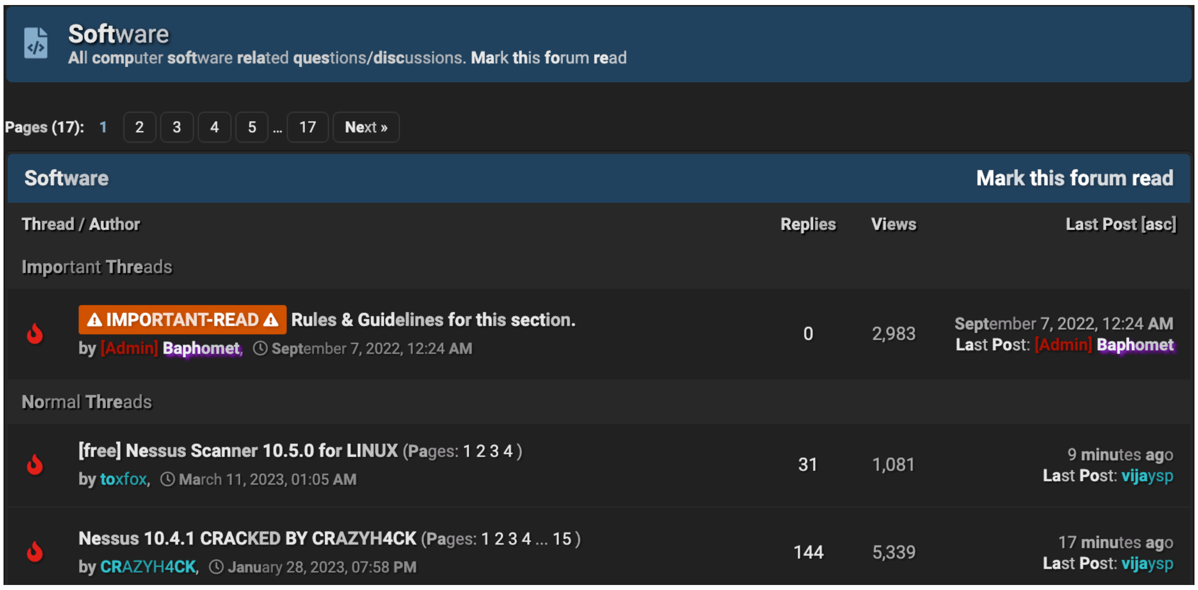
Security precautions are essential to ensure the safety of the investigator and the integrity of the investigation. Therefore, to effectively ensure the anonymity of the researcher, the following precautions were taken:

- A Virtual Private Network (VPN) was used to obscure the investigator's IP address and location
- A secure (Brave[49]) was used to ensure the possibility of browser fingerprinting and drive-by downloads were limited
- Utilised devices were regularly backed up
- The anti-virus software on the utilised devices was kept up to date

After extensive research taking the aforementioned approaches, the sites investigated were boiled down to `Breached.vc`, `Sinister.ly`, `Hackforums.net` and `Leakzone.net`, given their popularity and size as described by **Yuval Shini**[50], and the **Feedspot Media Database Team**.[51] Although there were other notable contenders such as `xss.is`[52] or `mpgh.net`[53] these were excluded from the selection due to language barriers and a lack of hacking focus, respectively The succeeding subsections will provide a brief description of each of the chosen sites.

3.2.1 Breached.vc

`Breached.vc` or BreachForums, is a forum containing anything from doxing and malware to cracked software and database leaks. The site contains over 964K posts, 47K threads and 337K members[54] as of 19/03/2023 and has been noted as the successor to the infamous RaidForums.[55]. A screenshot of some of the threads found on breached, related to software tools, can be seen in figure **Figure 3.1**. As seen, the listings commonly concern cracked software, in this case Nessus[56] (a professional vulnerability scanner).



Software
All computer software related questions/discussions. Mark this forum read

Pages (17): 1 2 3 4 5 ... 17 Next »

Software — Mark this forum read

Thread / Author	Replies	Views	Last Post [asc]
Important Threads			
⚠ IMPORTANT-READ ⚠ Rules & Guidelines for this section. by [Admin] Baphomet, September 7, 2022, 12:24 AM	0	2,983	September 7, 2022, 12:24 AM Last Post: [Admin] Baphomet
Normal Threads			
[free] Nessus Scanner 10.5.0 for LINUX (Pages: 1 2 3 4) by toxfox, March 11, 2023, 01:05 AM	31	1,081	9 minutes ago Last Post: vijaysp
Nessus 10.4.1 CRACKED BY CRAZYH4CK (Pages: 1 2 3 4 ... 15) by CRAZYH4CK, January 28, 2023, 07:58 PM	144	5,339	17 minutes ago Last Post: vijaysp

Figure 3.1: Software threads subforum on Breached.vc (Source: [54])

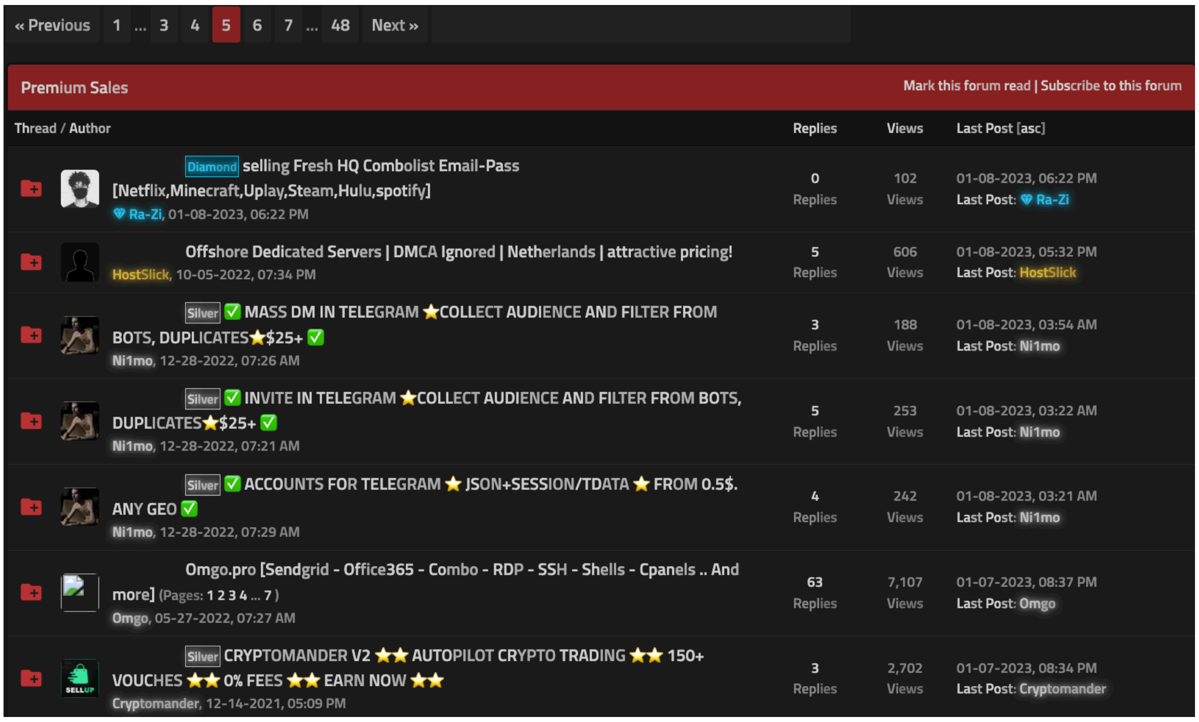
3.2.1.1 The FBI Intervenes

On the 15th of March 2023, the FBI and Department of Health and Human Services Office of Inspector General (HHS-OIG) conducted a disruption operation that caused BreachForums to go offline. In the process, they arrested the alleged leading site operator Conor Brian Fitzpatrick, a 20-year-old from New York. Fitzpatrick is now charged with conspiracy to commit access device fraud and facing up to five years in prison if convicted.[57]

The takedown of a site that provided illegally obtained datasets with sensitive information is a significant achievement for law enforcement. However, it has limited the output of the project, as the scraper's initial design was based on BreachForums, making a large part of the work in building the scraper unnecessary. Nevertheless, this was not a total loss since the development of bypassing anti-scraping measures, iterating over the site structure, and inducing rate-limiting and IP rotation was established, which could be used for scraping other websites in the future. Unfortunately, BreachForums was seized before the scraper managed to scrape any usable dataset. The remaining data mostly consisted of links and the number of threads in subforums, which while still useful is not sufficient to provide a comprehensive understanding of the forum's activities. As a result, this dataset was not included in any NLP models and was not used in the analysis. Nonetheless, the development conducted during this project related to BreachForums will still be discussed, as it was still an important part of the project's progression.

3.2.2 Sinister.ly

`Sinister.ly` is a forum for discussing topics related to hacking, cracking, and cryptocurrency. With over 1.1M posts, 153K threads and 190K users as of 30/03/2023[58], it allows for sharing of cracked combo lists, tutorials, hacking solutions, and configurations. The website was created in 2014 and also has sections related to sales of malware, hacking tools or other *sinister* services as depicted in **Figure 3.2**.



« Previous 1 ... 3 4 5 6 7 ... 48 Next »

Premium Sales — Mark this forum read | Subscribe to this forum

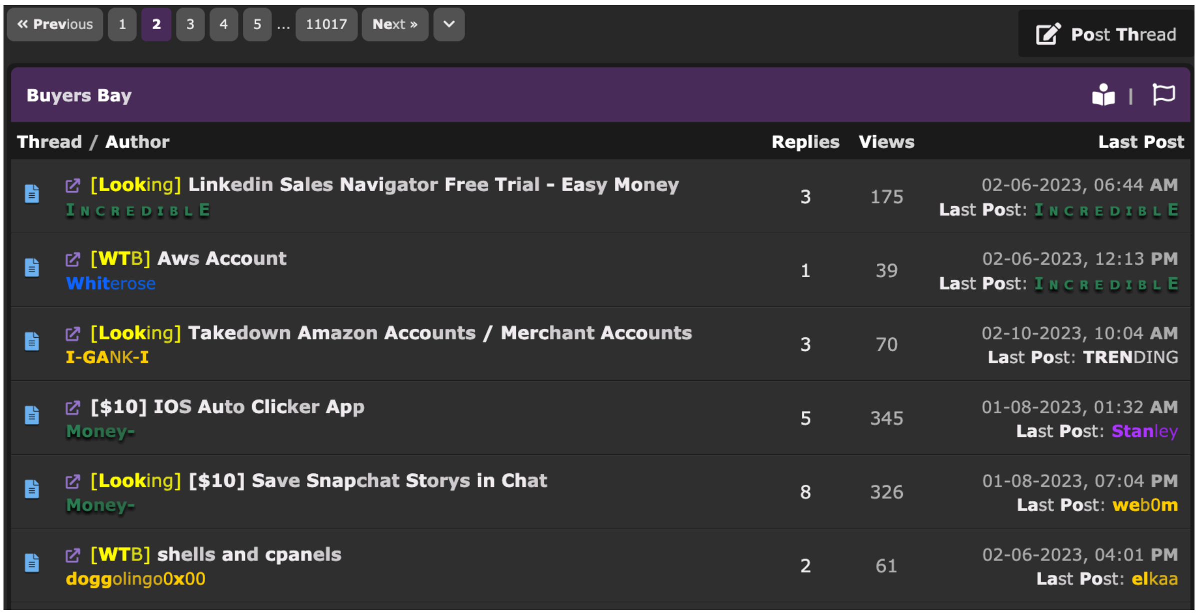
Thread / Author	Replies	Views	Last Post [asc]
Diamond selling Fresh HQ Combolist Email-Pass [Netflix,Minecraft,Uplay,Steam,Hulu,spotify] Ra-Zi, 01-08-2023, 06:22 PM	0 Replies	102 Views	01-08-2023, 06:22 PM Last Post: Ra-Zi
Offshore Dedicated Servers | DMCA Ignored | Netherlands | attractive pricing! HostSlick, 10-05-2022, 07:34 PM	5 Replies	606 Views	01-08-2023, 05:32 PM Last Post: HostSlick
Silver ✅ MASS DM IN TELEGRAM ⭐COLLECT AUDIENCE AND FILTER FROM BOTS, DUPLICATES⭐$25+ ✅ Ni1mo, 12-28-2022, 07:26 AM	3 Replies	188 Views	01-08-2023, 03:54 AM Last Post: Ni1mo
Silver ✅ INVITE IN TELEGRAM ⭐COLLECT AUDIENCE AND FILTER FROM BOTS, DUPLICATES⭐$25+ ✅ Ni1mo, 12-28-2022, 07:21 AM	5 Replies	253 Views	01-08-2023, 03:22 AM Last Post: Ni1mo
Silver ✅ ACCOUNTS FOR TELEGRAM ⭐ JSON+SESSION/TDATA ⭐ FROM 0.5$. ANY GEO ✅ Ni1mo, 12-28-2022, 07:29 AM	4 Replies	242 Views	01-08-2023, 03:21 AM Last Post: Ni1mo
Omgo.pro [Sendgrid - Office365 - Combo - RDP - SSH - Shells - Cpanels .. And more] (Pages: 1 2 3 4 ... 7) Omgo, 05-27-2022, 07:27 AM	63 Replies	7,107 Views	01-07-2023, 08:37 PM Last Post: Omgo
Silver CRYPTOMANDER V2 ⭐⭐ AUTOPILOT CRYPTO TRADING ⭐⭐ 150+ VOUCHES ⭐⭐ 0% FEES ⭐⭐ EARN NOW ⭐⭐ Cryptomander, 12-14-2021, 05:09 PM	3 Replies	2,702 Views	01-07-2023, 08:34 PM Last Post: Cryptomander

Figure 3.2: Accounts, software and services sold on Sinister.ly (Source: [58])

From the figure, it is clear that the listings found on the site are quite diverse, ranging from account sales to combo lists as well as hacking (`omgo.pro`) and cryptocurrency software.

3.2.3 Hackforums.net

`Hackforums.net` is a long-standing and extensive community of hackers that can be easily accessed through the surface web. The forum grew in popularity after the release of the Mirai botnet source code on the site and is considered one of the larger hacking communities still ongoing.[3] As of 09/05/2023, it holds 5.4M members, 62M posts and 6.3M threads[59] and provides various services as well as hacking or leaking related products as depicted on **Figure 3.3**.



Buyers Bay

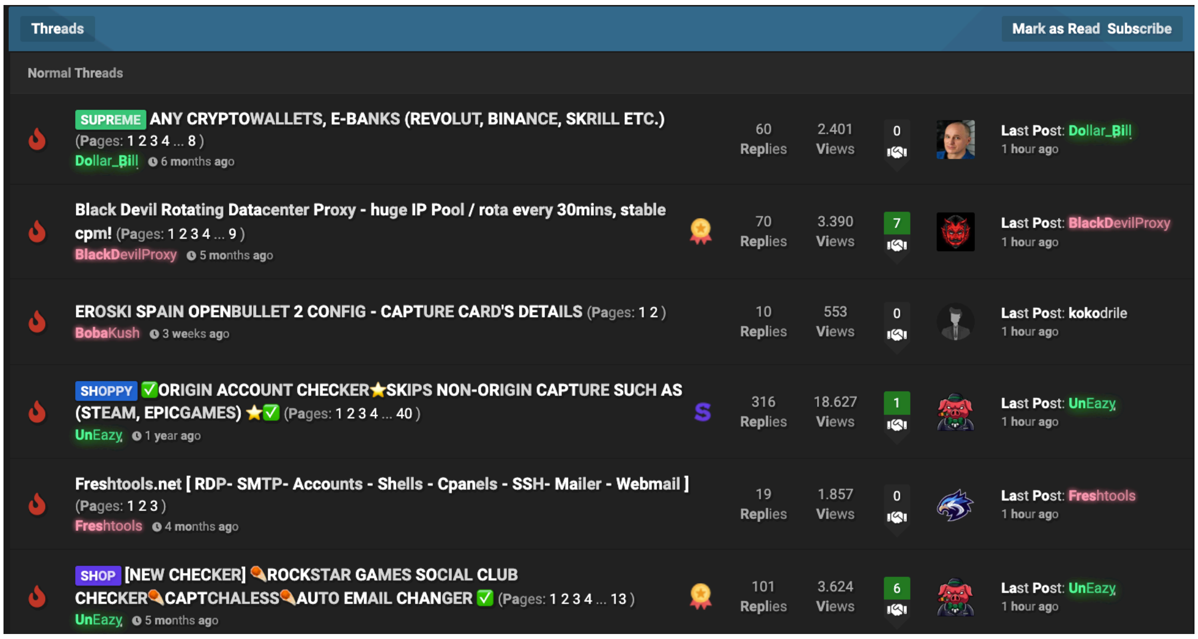
Thread / Author	Replies	Views	Last Post
[Looking] Linkedin Sales Navigator Free Trial - Easy Money INCREDIBLE	3	175	02-06-2023, 06:44 AM Last Post: INCREDIBLE
[WTB] Aws Account Whiterose	1	39	02-06-2023, 12:13 PM Last Post: INCREDIBLE
[Looking] Takedown Amazon Accounts / Merchant Accounts I-GANK-I	3	70	02-10-2023, 10:04 AM Last Post: TRENDING
[$10] IOS Auto Clicker App Money-	5	345	01-08-2023, 01:32 AM Last Post: Stanley
[Looking] [$10] Save Snapchat Storys in Chat Money-	8	326	01-08-2023, 07:04 PM Last Post: web0m
[WTB] shells and cpanels doggolingo0x00	2	61	02-06-2023, 04:01 PM Last Post: elkaa

Figure 3.3: Malware and accounts sold on Hackforums.net (Source: [59])

As with the other sites, it can be seen that the selection of listings is quite broad, as they can concern anything from the sharing or takedown of accounts to obtaining shells on servers.

3.2.4 Cracked.io

`Cracked.io` is a community forum that provides cracking tutorials, tools, and leaks, while also functioning as a distribution channel for various cracked software and malware. The site received over 5.4 million requests in March 2023, proving its popularity and mainly receives visitors from the US, Germany and Spain.[60] As of 09/05/2023 it holds 25M posts, 845K threads and 3.76M users[61] and provides products and services as the ones seen in **Figure 3.4**,



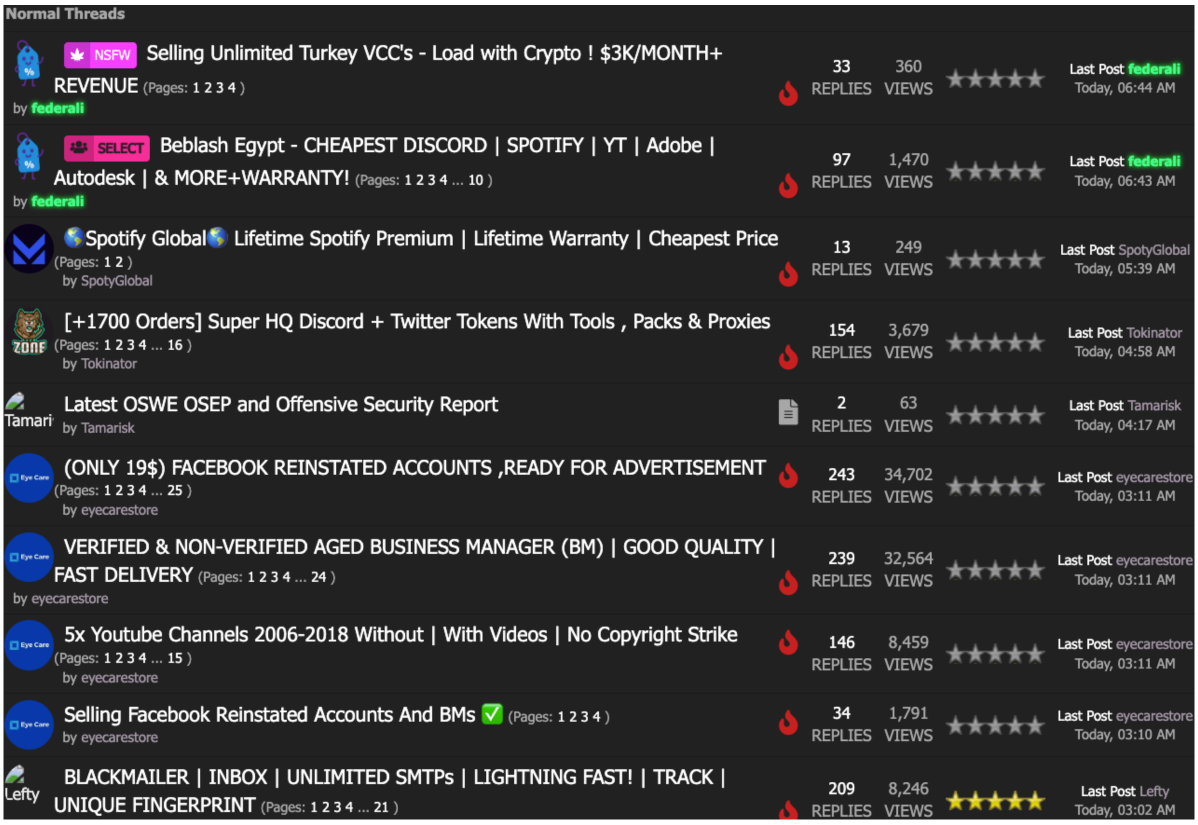
Thread	Replies	Views		Last Post
SUPREME ANY CRYPTOWALLETS, E-BANKS (REVOLUT, BINANCE, SKRILL ETC.) (Pages: 1 2 3 4 ... 8) Dollar_Bill 6 months ago	60 Replies	2.401 Views	0	Last Post: Dollar_Bill 1 hour ago
Black Devil Rotating Datacenter Proxy - huge IP Pool / rota every 30mins, stable cpm! (Pages: 1 2 3 4 ... 9) BlackDevilProxy 5 months ago	70 Replies	3.390 Views	7	Last Post: BlackDevilProxy 1 hour ago
EROSKI SPAIN OPENBULLET 2 CONFIG - CAPTURE CARD'S DETAILS (Pages: 1 2) BobaKush 3 weeks ago	10 Replies	553 Views	0	Last Post: kokodrile 1 hour ago
SHOPPY ✅ORIGIN ACCOUNT CHECKER⭐SKIPS NON-ORIGIN CAPTURE SUCH AS (STEAM, EPICGAMES) ⭐✅ (Pages: 1 2 3 4 ... 40) UnEazy 1 year ago	316 Replies	18.627 Views	1	Last Post: UnEazy 1 hour ago
Freshtools.net [RDP- SMTP- Accounts - Shells - Cpanels - SSH- Mailer - Webmail] (Pages: 1 2 3) Freshtools 4 months ago	19 Replies	1.857 Views	0	Last Post: Freshtools 1 hour ago
SHOP [NEW CHECKER] 🍭ROCKSTAR GAMES SOCIAL CLUB CHECKER🍭CAPTCHALESS🍭AUTO EMAIL CHANGER ✅ (Pages: 1 2 3 4 ... 13) UnEazy 5 months ago	101 Replies	3.624 Views	6	Last Post: UnEazy 1 hour ago

Figure 3.4: Malware and accounts sold on Cracked.io (Source: [61])

In contrast to the other sites, it can be seen that Cracked.io has listings related to capturing card details (carding) as well as offering web hosting and other hacking services.

3.2.5 Leakzone.net

`Leakzone.net` is a hacking forum with a specific focus on leaking & cracking and, as of 09/05/2023, consists of 35K threads, 216K posts and 78K members.[62] As seen on **Figure 3.5**, the selection of items and services being sold on Leakzone.net is quite broad and quite similar to the other chosen sites.



Normal Threads

Thread	Replies	Views	Last Post
NSFW Selling Unlimited Turkey VCC's - Load with Crypto ! $3K/MONTH+ REVENUE (Pages: 1 2 3 4) by federali	33 REPLIES	360 VIEWS	Last Post federali Today, 06:44 AM
SELECT Beblash Egypt - CHEAPEST DISCORD | SPOTIFY | YT | Adobe | Autodesk | & MORE+WARRANTY! (Pages: 1 2 3 4 ... 10) by federali	97 REPLIES	1,470 VIEWS	Last Post federali Today, 06:43 AM
Spotify Global Lifetime Spotify Premium | Lifetime Warranty | Cheapest Price (Pages: 1 2) by SpotyGlobal	13 REPLIES	249 VIEWS	Last Post SpotyGlobal Today, 05:39 AM
[+1700 Orders] Super HQ Discord + Twitter Tokens With Tools , Packs & Proxies (Pages: 1 2 3 4 ... 16) by Tokinator	154 REPLIES	3,679 VIEWS	Last Post Tokinator Today, 04:58 AM
Latest OSWE OSEP and Offensive Security Report by Tamarisk	2 REPLIES	63 VIEWS	Last Post Tamarisk Today, 04:17 AM
(ONLY 19$) FACEBOOK REINSTATED ACCOUNTS ,READY FOR ADVERTISEMENT (Pages: 1 2 3 4 ... 25) by eyecarestore	243 REPLIES	34,702 VIEWS	Last Post eyecarestore Today, 03:11 AM
VERIFIED & NON-VERIFIED AGED BUSINESS MANAGER (BM) | GOOD QUALITY | FAST DELIVERY (Pages: 1 2 3 4 ... 24) by eyecarestore	239 REPLIES	32,564 VIEWS	Last Post eyecarestore Today, 03:11 AM
5x Youtube Channels 2006-2018 Without | With Videos | No Copyright Strike (Pages: 1 2 3 4 ... 15) by eyecarestore	146 REPLIES	8,459 VIEWS	Last Post eyecarestore Today, 03:11 AM
Selling Facebook Reinstated Accounts And BMs ✅ (Pages: 1 2 3 4) by eyecarestore	34 REPLIES	1,791 VIEWS	Last Post eyecarestore Today, 03:10 AM
BLACKMAILER | INBOX | UNLIMITED SMTPs | LIGHTNING FAST! | TRACK | UNIQUE FINGERPRINT (Pages: 1 2 3 4 ... 21)	209 REPLIES	8,246 VIEWS	Last Post Lefty Today, 03:02 AM

Figure 3.5: Malware and accounts sold on Leakzone.net (Source: [62])

The figure confirms Leakzone.net's dedication to the sales of leaked accounts, as opposed to hacking tools.

3.3 Data Collection

Once the target forums had been identified, the next step was to collect data from these sites. This involved web scraping techniques to extract information such as the vendor names, the type of goods or services being sold and the prices. A web scraping tool was created in Python with the packages described in **Section 2.1.3** to extract the relevant data from the HTML source code of the forums. The developed tool was able to run on standard hardware, and the data collected was converted to a structured format for further analysis. A more detailed explanation of the development of this tool can be found in **Section 7.1**.

3.4 Data Storage

The data was stored locally in a database on the system that collected and extracted it, which will be elaborated upon in **Section 7.2**. After being provided to the university, the data will be erased from the author's system within three months of project completion, no later than August 22, 2023. The university will be free to maintain the data, and it will only be accessible upon request, subject to their terms.

3.5 Data Cleaning

Once the data has been collected, it will be necessary to clean and pre-process the data to ensure that it is in a format that can be easily analysed. This will involve removing any irrelevant information, correcting erroneous data, and transforming the data into a consistent format. In this project, the data is comprised of raw text, and to effectively filter out irrelevant information and group similar data together, various NLP techniques such as *Bulk Labelling* (BL) and *Text Classification* (TC) will be utilised, alongside *dimensionality reduction*, to cluster similar data together. The implementation of these techniques will be described in **Section 7.4**.

3.6 Data Analysis

The next step will be to analyse the data collected from the illicit forums. This will entail the use of descriptive statistical techniques to uncover discernible patterns and trends within the data. Specifically, these techniques will be leveraged to visually represent the information extracted from the text data, thereby offering a comprehensive overview of the user base and the scale of the forums. The extracted information will additionally allow for a comprehensive understanding of the types of malware or other digital services offered on clear web illicit forums. The insights obtained via this information will be valuable for both law enforcement agencies and researchers concerned with the unlawful forums found on the internet. Not only would it assist law enforcement in developing effective enforcement strategies, but it would also allow researchers to better comprehend the inner workings of illicit forums on the clear web.

3.7 Machine Learning Models

To ensure the data cleaning, as well as information extraction, is accurate, supervised machine learning models based on pre-trained LLMs will be developed from manually annotated segments of the raw data. The pre-trained models chosen will be based on the current state-of-the-art LLMs available at the time of writing (10/09/2023), as described in **Section 2.3.2**. The models developed will consist of TC models for data segmentation and categorisation and an NER model for extracting price information. These combined techniques will allow for efficient and accurate extraction of insights from the raw text data.

3.8 Ethical Issues

Web scraping on illicit forums raises several ethical issues, as it involves collecting and using data from unauthorised sources without proper consent or permission. One of these issues is the violation of privacy. While illegal forums often limit the amount of sensitive and personal information present for an individual, they still host data that can help infer the identity of both buyers and sellers, as well as what they buy or sell, similar to how third-party cookies are used to identify individuals. Additionally, users across forums were cross-referenced during the analysis. This action was performed as the risk of identifying the users' identities solely based on their usernames was deemed low by the author. While the identification of criminal individuals may excite law enforcement, it is not the goal for researchers to infringe on individuals' privacy. Care will be taken to ensure that the methods used for data collection and analysis do not interfere with the normal functioning of the forum or compromise the security of the individuals involved, despite the illegal actions that may be conducted on these sites. As a researcher, the goal is simply to understand the inner workings of these forums.

3.8.1 Availability and Scraped Data

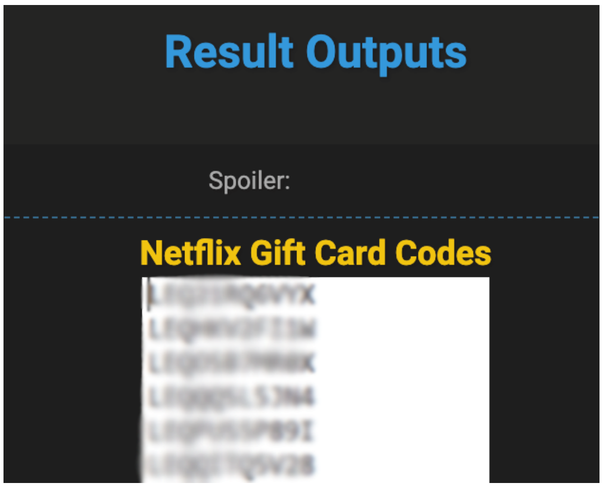
Although web scraping is not inherently illegal, it may be perceived as unethical and potentially unlawful. However, the research conducted in this thesis is confined to gathering information that is already accessible to the general public through the navigation of the selected illicit forums. This information includes site structure, raw thread text data, posting date, and user information such as username and date joined. Since this information is publicly available, its study is unlikely to cause harm and instead may facilitate scientific progress.[63] Furthermore, while the scraping system does run in a non-headless mode, meaning the entire page to be scraped, including images, is loaded, the system does not store these images. Simply their links, hence disallowing sensitive information encoded in images to be stored.

Moreover, rate-limiting is enabled, both to reduce the risk of the system being banned by the server as well as fostering equity across the network, i.e. limiting the impact on the availability of the crawled sites. In practice, this meant a request rate of no more than one request every 5 seconds, or 0.2 requests/sec. Considering that a machine with a basic Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz, which is not considered a high-end server CPU by current standards, can handle about 100 requests per second[64], the scraping system is not likely to limit the availability of the illicit forums.

Additionally, for the system to operate effectively and have access to the data of interest on the different sites, premium accounts were purchased on three of the sites: Cracked.io, Leakzone.net and Sinister.ly. While monetarily contributing to these services is ethically dubious, it was deemed necessary to extract interesting data for analysis. The feeble amount each of these sites was given for this access (20€, 15£ and 15$ respectively) is quite low, and hence the contribution it brings to the illegal activities they may conduct is considered quite limited.

3.8.2 Risks to The Researcher

Engaging in web scraping activities on illegal forums may potentially infringe upon intellectual property rights, given that these online platforms are notorious for trading counterfeit merchandise, pirated content, and other unlawful goods that are safeguarded by copyright and trademark laws. Although this content is seldom provided free of charge, sellers commonly furnish samples to entice buyers, as illustrated in **Figure 3.6**. Nonetheless, scraping such samples can result in the researcher possessing trademarked, copyrighted, and other illicit items, constituting a violation of intellectual property rights.



[redacted]

Figure 3.6: A sample being presented in a listing offering a Netflix gift card generator (Source: [65])

As the custodian of this data, the researcher has a legal obligation to provide it as potential evidence in legal proceedings.[66] However, in doing so, it is the researcher's responsibility to ensure that the data is not stripped of its identifiability, as the de-identification of data without lawful justification is widely recognized as a criminal offence under the privacy and data protection laws of numerous institutions and countries, such as the European Union and the United States[67]. The degree of sanctions levied on violators of this law may differ based on the gravity of the offence and the legal framework of each jurisdiction. However, maximum penalties for such violations may reach up to 20 years imprisonment or more.[68] Steps are therefore taken to ensure that the data collected for this thesis is not immediately deleted, and as mentioned in **Section 3.4**, the data will responsibly be handed over to the university for any future use, processing or possible legal proceedings.

4 Related Works

The proliferation of the internet and e-commerce has led to the growth of illicit forums that engage in illegal activities such as the sale of malware, DDoS attacks and other criminal digital goods.[69] Various studies have been conducted to assist researchers in understanding the operations of these forums and their impact on society. In this section, a comprehensive overview of the literature on web scraping, illicit forums, and the analysis of their data will be presented. In doing so their key contributions and findings will be examined to shed light on the current state of knowledge.

4.1 Illicit Forums & Web Scraping

While not all content or items posted on underground forums are necessarily illegal, their source or purpose may be questionable. Examples of this could be sales of Amazon gift cards or the selling of illegally obtained digital goods such as unique items in Massively Multiplayer Online (MMO) games. The analysis of these forums is commonly conducted by developing a Web scraper.

Web scraping is a common research topic and has produced crawlers for web forums such as *iRobot* by **Rui Cai et al.**[70], *FoCUS* by **Jiang, Jingtian et al.**[71] and *Vigi4Med* by **Audeh, Bissan et al.**[72] These crawlers, however, are mostly focused on scraping general online forums, that do not necessarily employ anti-crawling techniques. Due to the illicit nature of the forums that are to be investigated in this thesis, a crawler that can actively combat anti-scraping techniques must be developed.

These anti-scraping techniques, as described by **Alkhatib, Bassel et al.**[73] encompass dealing with login functionality, bypassing CAPTCHAS and gaining privileged access via registration. In some cases, the website administrators even incorporate functionality that disables inactive accounts or accounts with barely any visible behaviour on the site as well as accounts that display any form of suspicious (non-human) behaviour.[73] For the scraper developed in this thesis, the techniques for scraping data on the sites will be a combination of the techniques employed by**Alkhatib**[73]. I.e. HTML Parsing, DOM parsing and Xpath traversing, seeing as these generally work quite well at extracting specific data on a site. Additionally, several measures will be taken to efficiently handle anti-scraping techniques employed by the forums. These techniques will be elaborated upon in **Section 4.1.2**.

4.1.1 Dark Web Forums

Many of the illicit forums present today are located on the so-called *Dark Web*, only accessible via The Onion Router (TOR) protocol. Online anonymous markets, which started as mainly drug-oriented markets in 2011, have become a significant part of the cybercrime world. Their supply of digital goods, both in quantity and diversity, has steadily increased over the years, and a single top-tier market can generate over 200,000 US dollars daily, as found by **van Wegberg**.[74] The illegal products and services offered on these markets range from physical goods like passports to digital goods like stolen credit cards or malware packages. These markets are used not only for retail transactions (e.g., small drug quantities or a few compromised Netflix accounts) but also for wholesale transactions (e.g., bulk drug sales or large databases of compromised email accounts).[74]

Many previous studies were conducted on the dark web market *Silk Road*, the first cryptomarket that appeared in 2011, which offered both business-to-customers (B2C) and business-to-business (B2B) transactions. Investigations by **Broséus J. et al.**[12] showed that while there were over 1000 vendors active with annual sales reaching almost 90 million USD, the site was seized by law enforcement after just two years in 2013, as given by **Buskirk, Joe et al.**[75], which is often the case with unlawful online forums. As stated by the **US Department of Justice**[76] Silk roads successor Alpha Bay also had a modest time to live, being spun up in December 2014 and suffering a similar fate when taken down in 2017.

This proves to show that these forums are of ever-changing nature and research results in the field are hence short-lived. In other words, despite the vast amount of studies on illicit forums (24400 as of 12/02/2023 on Google Scholar), continued research is required to accurately follow the development and inner functions of these forums, as well as understand the scale at which they function.

4.1.2 Clear Web Forums

As described in **Section 4.1.1** many infamous hacking forums are hosted on the *dark web*. However, the aim of this study is to investigate forums on the *clear web*, though with a more recent state and expanded palette of forums.

Alice Hutchings et al.[3] developed *Crimebot* for automatically collecting data from online clear web forums and analysing it for useful insights. Each of these resulted in a dataset that, when analysed, provided a comprehensive overview of the products sold on these forums. *Crimebot* actively attempted to evade the anti-scraping techniques found on illicit forums. Specifically, it was developed to manage session cookies obtained from previous registrations via its session management module to circumvent repetitive CAPTCHA solving to access privileged information reserved for registered users. Essentially, a non-headless browser is opened from which a human operative can solve the CAPTCHA and log in, allowing the Crimebot to browse and scrape data without being apprehended by a login wall.

Additionally, CrimeBot was developed for modularity, making it easy to add additional scrapers for different web toolkits or custom sites. These modules are then combined with a general approach to displaying human behaviour, such as varying navigation times and patterns, changing HTTP headers to resemble non-headless browsers and limiting the crawling time to mimic a human browsing session.[3]

The scraper developed in this thesis will employ similar functionality as Crimebot, with the difference being the added functionality of handling sites, not investigated by **Alice Hutchings et al.**[3] such as `Sinister.ly`, `Cracked.io` and `Leakzone.net`. Additionally, the scraper developed in this thesis will also be able to bypass certain CAPTCHAs, specifically hCAPTCHAs, automatically.

4.2 Analysing Forum Data

At scale, NLP and ML methods are commonly used to analyse forums. With these, intent can be inferred from messages, and posts can be related to transactions, products and pricing, which, when combined, can help identify the supply chain of products on online forums as described by **Anh V. Vu et al.**[4] Comparing the work of this thesis with previous efforts, it is clear that there is little research conducted on the digital products found on these forums. In the following subsections, current work analysing unstructured text data will be outlined.

4.2.1 Topic Modelling

A common approach when analysing unstructured text data is to use *Topic Modelling* (TM). TM is an unsupervised learning method used to identify the main topics or themes present in a collection of text data. **Middleton, Stuart E**[77] utilised this approach to cluster extracted information from online forums related to illegal trades of endangered plant species. The model extracted this data via a set of predefined search terms, in this case, the plant species names. For the work of this thesis, no specific terms are defined as a term, such as *malware* is an umbrella term. Each specific piece of malware usually has custom names completely unrelated to the malware term, such as *Agent Tesla* or *NanoCore*.[78] Conversely, this study will adopt a targeted methodology involving manual labelling of a subset of the data and the development of customised models for the categorisation of thread types, identification of sales-oriented threads, and determination of the corresponding product pricing. This is done to ensure the labelled data is accurate and its subsequent analysis is indicative of the true nature of the dataset.

4.2.2 POS Tagging

The goal of web scraping projects is to extract information about actors, products, prices, and other relevant details found on a site. In order to perform this extraction effectively, POS tagging is often utilised. With POS tagging, words are categorised into their respective parts of speech, such as nouns, verbs, adjectives, and adverbs. Utilising this method, it becomes possible to identify which words are most relevant to the extraction process. **Pastrana, Sergio et al.**[79] performed POS tagging to extract nouns from text to help classify actors based on their main topics of conversation. Similarly, in this project, POS tagging is employed to extract nouns, as nouns in this context often refer to the key actors, products, and other entities involved in such forums. However **Pastrana, Sergio et al.**[79] then fed these POS tags into a topic analysis model, where in this case, the POS tags will be used directly for analysis, via *word clouds*, to grant an overview of the main topics discussed on the investigated forums. Furthermore, the extracted POS tags will be used for smart labelling of data.

4.2.3 Named Entity Recognition

NER is an essential NLP task that involves identifying and categorising named entities, such as people, locations, and organisations, in textual data and significantly simplifies the process of information extraction from unstructured textual data. **Eduardo Fidalgo et al.**[80] employed this technique to identify categories of named entities such as location, person and corporation in onion domains on the Tor network. Similar approaches will be taken in this thesis, with a specific focus on extracting *Monetary* entities, i.e. at what price and in which quantities are products being sold on the illicit forums being investigated.

4.2.4 Large Language Models

The development of LLMs has been one of the most significant recent advances in NLP. LLMs are neural network-based language models that can learn to represent text data in a high-dimensional vector space, allowing them to perform a range of NLP tasks, such as language generation, text classification, and machine translation. The introduction of LLMs has revolutionised the field of NLP, enabling researchers to achieve state-of-the-art results on many NLP benchmarks. Today LLMs are available pre-trained on a large corpus of data, making the task of creating models for a specific task much easier as done by **Dogu Tan Araci**.[81] **Araci** utilised the BERT LLM, as described in **Section 2.3.2.1**, to fine-tune a model for tackling NLP tasks in the financial domain, such as analysing sentiment from financial text.

This project will take a similar approach to **Araci**[81] to finetune a state-of-the-art LLM. However, this work will focus on creating TC and NER NLP models for classifying forum thread data and extracting prices for items potentially being sold in the thread, respectively.

5 Requirements

The purpose of this chapter is to outline the requirements for developing the web scraper to extract data from the chosen illicit clear web forums, as well as the defining a set of performance requirements for the LLM models used for information extraction. **Section 5.1** will specify the requirements of the former, while **Section 5.2** will provide the performance requirements of the latter.

5.1 Web Scraping Requirements

For the development of an effective web scraper for extracting data from the chosen illicit clear web forums, a set of functional and non-functional requirements must be defined. *Functional Requirements* (FR), as described by **Martin Glinz**[82], are requirements that define functionalities in the system, while *Non-Functional Requirements* (NFR), concern performance, attributes or constraints. For the web scraper to function and deliver reliable data for the analysis while still ensuring the scraped forum is not overloaded, the following requirements should be met:

Functional Requirements

1. **Robustness**: The web scraper should be robust and able to handle site structures and layouts of each of the chosen forums as well as handle any obstructions and request timeouts.
2. **Data quality**: The web scraper should ensure high-quality data by performing data validation and cleansing during the scraping process. This can include tasks such as removing emojis, unsupported Unicode characters, or irrelevant ASCII art. This will reduce the likelihood of errors or inconsistencies in the extracted data.
3. **Data Parsing:** After obtaining a specific page on a forum's HTML structure, the web scraper must be capable of effectively parsing the *Document Object Model* (DOM) structure for each of the selected forums to extract any valuable data.
4. **Data storage**: The web scraper should store the extracted data in a structured format that is easy to manipulate and analyse using data analysis tools in Python.
5. **IP Rotation**: To effectively counteract IP banning, the scraper should be able to change IP address, either per request or per a given time interval.

6. **Bypassing Anti-scraping techniques**: The scraper should be able to bypass any anti-scraping techniques employed by each of the respective forums. This includes IP banning (see above), DDoS blocks and CAPTCHA blockades. Should the scraper be encountered by an (h)CAPTCHA, the system should automatically solve the CAPTCHA, thereby bypassing it and granting the scraper access to the forum.

Non-Functional Requirements

1. **Scalability**: The web scraper should be scalable and able to handle large volumes of data efficiently. This will ensure that the scraper can extract data from multiple forums and store it in a structured format for subsequent analysis.

2. **Efficiency**: The web scraper should function as efficiently as possible, utilising minimal computational resources. As such the scraper should be to run on consumer hardware, as well as run for several sites at once in parallel.

3. **Rate Limiting**: To ensure the scraper does not affect the availability of the forums while scraping, rate limiting must be induced, ensuring a balance between speed and fairness over the network.

How each of these requirements will be addressed will be covered in **Sections 7.1**, **7.2.3** and **7.3** respectively.

5.2 NLP and LLM Performance Requirements

When the data has been collected, it has to be subsequently passed through an NLP pipeline for analysis. This will ensure the data is categorised, and any data of interest in the unstructured text data will be gathered for analysis. To fine-tune the custom state-of-the-art LLM models for each of the NLP tasks used for information extraction, the following requirements must be enforced:

Functional Requirements

1. **Input Flexibility**: The fine-tuned LLM models must be able to handle text inputs of any length and, in any case, as well as being able to process all types of characters, including special characters.

Non-Functional Requirements

1. **High Performance**: The fine-tuned LLM models should provide accurate results in the task they are fine-tuned for. Hence they should achieve a reasonable F1 score. This metric was selected for its effectiveness in comparing models and depicting a fair score with unbalanced datasets as described in **Section 2.3.4**. Demanding a strong F1-Score will ensure that the extracted data is reliable and consistent for the subsequent data analysis.

2. **Efficiency**: The fine-tuned LLM models should be efficient in terms of both training and prediction time. This will ensure that the models can be trained and used in a timely and cost-effective manner.

6 Design

This chapter provides an overview of the system architecture, which, as previously stated, is composed of two main components: the web scraper and the NLP pipeline. The web scraper is responsible for collecting and parsing data from illicit clear web forums, while the NLP pipeline processes the textual data to extract relevant features for analysis, enabling insights into the products, services, and users on the forums.

Section 6.1 includes an overview of the scraper architecture, including its different components and how they are integrated to achieve the project objective. Additionally, **Section 6.2** outlines the hierarchy of the gathered data as well as how it is stored and connected in the database. Finally, in **Section 6.3** an overview of the NLP pipeline design is given, highlighting the various techniques used to extract features from the textual data, including connecting the various models used in the pipeline.

6.1 Web Scraper Design

The web scraper runs off a plethora of functions divided into the following modules:

1. Scraping functions (`scraping.py`)
2. Parsing/extraction functions subdivided in:
   a) Forum parsing/extraction functions (`forum_extraction.py`)
   b) Thread parsing/extraction functions (`thread_extraction.py`)
   c) User parsing/extraction functions (`user_extraction.py`)
3. Database functions (`db_func.py`)
4. Utility functions (`utils.py`)

The web scraper uses the modules mostly in the order presented above, with the exception of the utility functions that are used throughout each module. A visual representation of the scrapers flow can be found in the sequence diagram in **Figure 6.1**.

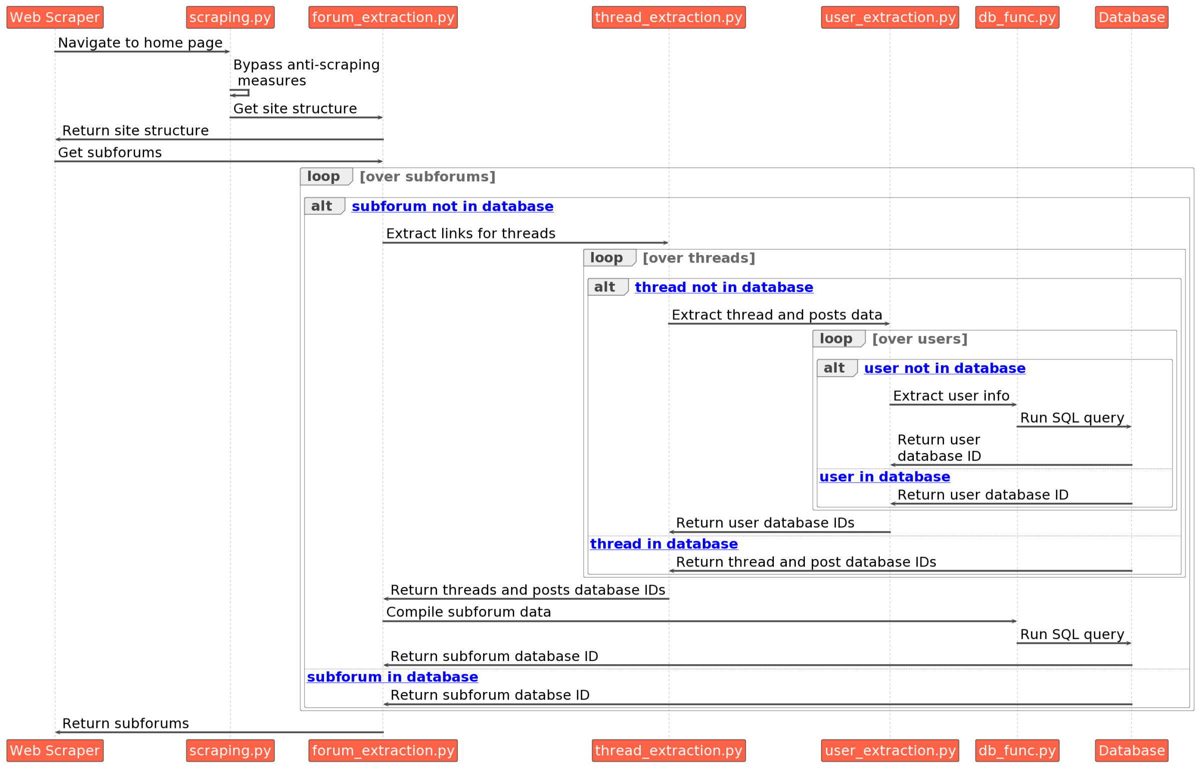


Figure 6.1: Sequence diagram of the flow the scraper operates by when scraping the illicit clear web forums (Generated with PlantUML[83])

Following the flow illustrated in **Figure 6.1** the scraper begins by utilising functions from the `scraping.py` module to navigate to the home page. Additionally, it performs a thorough examination to confirm whether it has effectively circumvented any anti-scraping measures implemented by the website, including DDoS protection, hCAPTCHA blockades, and IP bans. The methods for how the scraper will achieve this is described in more detail in **Section 7.3**. Once it has successfully bypassed these techniques, the scraper uses functions from the `forum_extraction.py` module to obtain the site structure.

After obtaining the site structure, the scraper begins a loop iterating over the subforums on the site using functions from the same module. A subforum in this context would for example be a forum *Java* contained within the parent forum *Software*. Within each iteration, the scraper gathers links to all the threads within the subforum, provided the subforum is not already present in the database. The scraper will then iterate over each thread link, calling functions from the `thread_extraction.py` module to check if the thread already exists in the database, and navigates to the thread link if it does not. If a new thread is encountered a database entry is created for each of its posts (thread responses), while linking the post to the thread in the database.

Additionally, the scraper initiates another loop over the links to the users posting in the thread. In this process, the `user_extraction.py` module is invoked to navigate to the profile of each user and retrieve their information, should the user not already be present in the database. Subsequently, every user is appropriately linked in the database to the thread and/or posts they have created.

Once all threads, their posts and their associated users have been extracted, the IDs of the threads and posts in a given subforum are returned to the `forum_extraction.py` module. The generation of these IDs is based on the extracted data and is explained in further detail in **Section 7.2.1**. The `forum_extraction.py` module then compiles the thread IDs and their post IDs into a dictionary, which subsequently is converted into a query for the creation of the subforum data entry in the database. This process repeats until all subforums have been scraped and their data have been saved in the database.

In addition to its main functions, the scraper can also be given inputs to allow it to start scraping from a specific subforum, thread, or user page, depending on the desired starting point. For the purposes of generating the datasets used in this thesis, the scraper was given the input starting point of the illicit forums' main *Marketplace* page, i.e. the subforum where sales and requests take place. The implementation of the various modules and their inner workings will be discussed in **Chapter 7**.

6.2 Database Design

The data acquired by the web scraper exhibits a structural resemblance to that of a graph database. Specifically, the type of graph database commonly used for social networks. In particular, users can initiate threads, which can encompass posts authored by other users. Additionally, threads are associated with specific forums or subforums, which can, in turn, be subforums of other forums. This topology is visually represented in Figure **Figure 6.2**, providing a concise top-down depiction of the hierarchical relationship between sites, forums, subforums, threads, posts and users.

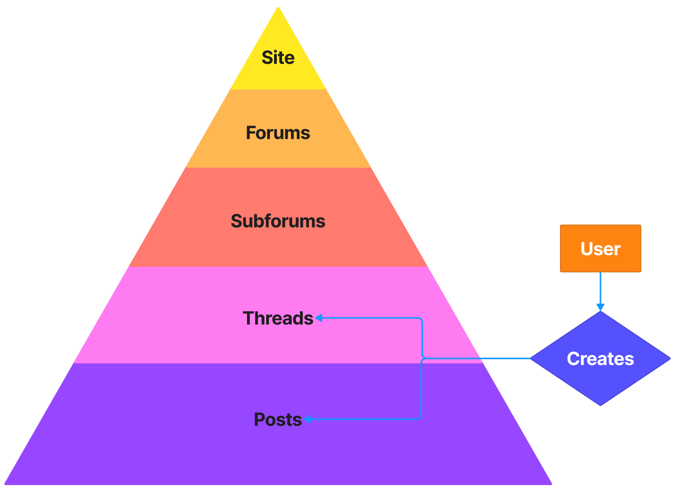


Figure 6.2: Topology of the forums investigated

To effectively store this information, an interconnected database schema must be generated. Each of the aforementioned entities is therefore interconnected in the database as described before. The connections between each of the entities in the database are presented in **Figure 6.3**. Note certain attributes for the various objects are left out, indicated by the [...], for simplification.

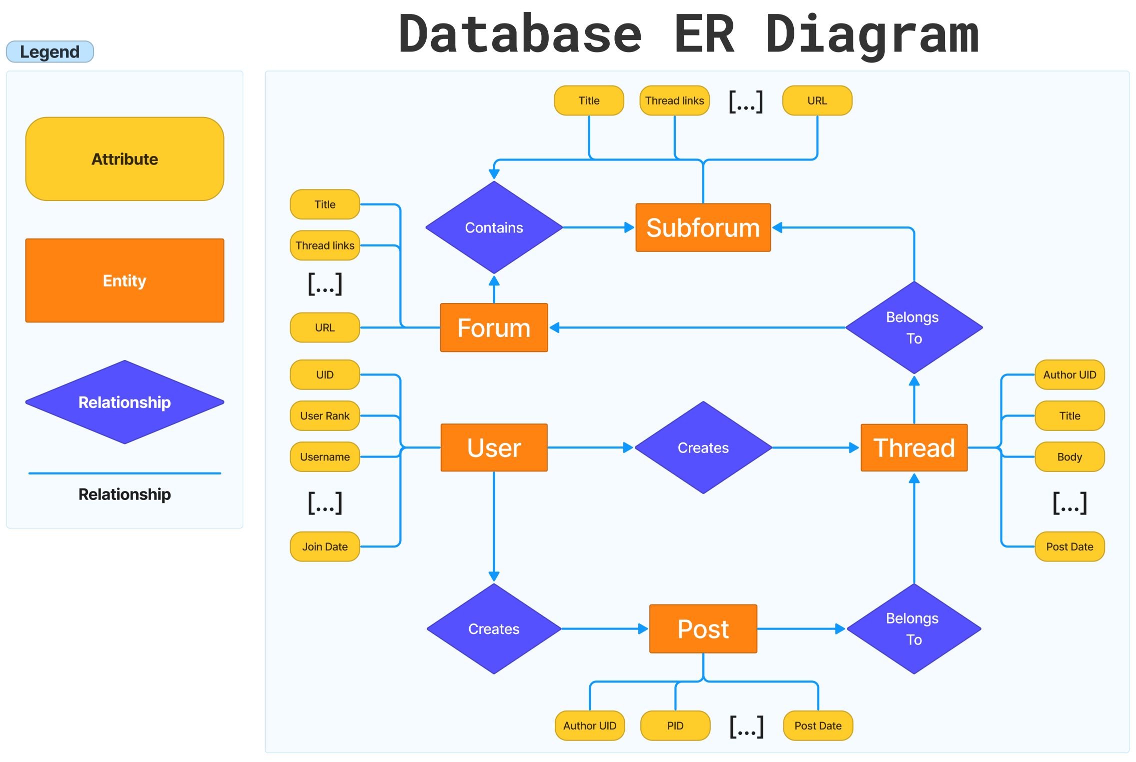


Figure 6.3: ER diagram granting an overview of the database structure

It should be noted that as indicated in **Chapter 1**, only data from 2023 was gathered, simply as anything more was not considered achievable within the time frame of the project. As such, the database did not require any extensive capabilities in terms of handling large datasets but should simply be well-structured and efficient.

6.3 NLP Pipeline Design

The extracted data from the scraper is further processed through an NLP pipeline, which is comprised of two text classifiers and an NER classifier, before being exported to a final dataset for analysis.

The first text classifier is designed to determine if a given thread (and its posts) concerns a `Sale`, an `Request` or is simply a `Discussion`. I.e. what *Intent* the text has. The classifier is based on the latest developments in LLMs to ensure efficiency as well as accuracy. The second text classifier is designed to identify which *category* the specific thread belongs to. I.e. is it related to *malware*, *database leaks*, *cracked software* etc. Specifically, it will classify the text into one of the following categories:

- Malware
- Cracking
- Software
- Account/Database/Combolist
- Ban-Service
- Gift cards
- Gaming-cheats
- Web-hosting
- Cryptocurrency
- Botnet
- Bot
- Money-earning-guide

Hereof, *Bot* refers to automation bots such as scrapers for extracting emails for spamming, while *Botnet* refers to hacked devices used in DDoS campaigns. *Combo lists* are a list of usernames/emails and passwords. *Malware* functions as an umbrella term for viruses, ransomware etc. while *Cracking* denotes the cracking of software to bypass license restrictions. *Ban-services* are services used for actively banning someone else's account or unbanning one's own account on various platforms. *Gaming-cheats* specify hacks used in games to give the player an advantage over others. *Software*, like *Malware*, is an umbrella term for leaked software, while *Web-hosting* refers to services such as proxies, website hosting or cloud computing resources. *Gift cards* are, as the name implies, the sale of digital gift cards (e.g. Amazon gift cards), which are often used for money laundering.[84] *Cryptocurrency* simply refers to any threads related to crypto-topics in any conceivable way. Finally, *Money-earning-guide* denotes the sale of methods that can be used to *Make Money Fast* online. The choice of these categories is based on preliminary investigations of the forums threads and is described further in **Section 7.4.1**. The model is similar to the first in the sense that it is based on a state-of-the-art LLM, though fine-tuned on a different dataset.

The NER classifier is designed to identify and extract the prices of the product being sold. Should the thread, for example, has been classified by the first text classifier as **Request**, this would be the price a certain product is requested at. Similarly, for a **Sale** thread, this would be the price the product in the thread is sold at. Threads marked as **Discussion** by the first classifier are passed to the second for categorisation. However, they are not passed to the NER classifier, as there is nothing being sold or requested in the thread, and hence there is nothing for the NER classifier to find. As with the other classifiers, this also utilises a pre-trained LLM model that is fine-tuned for the specific task. Each of the classifiers is given an input consisting of the combination of the main thread text, as well as its posts. The overall pipeline is visualised in **Figure 6.4**.

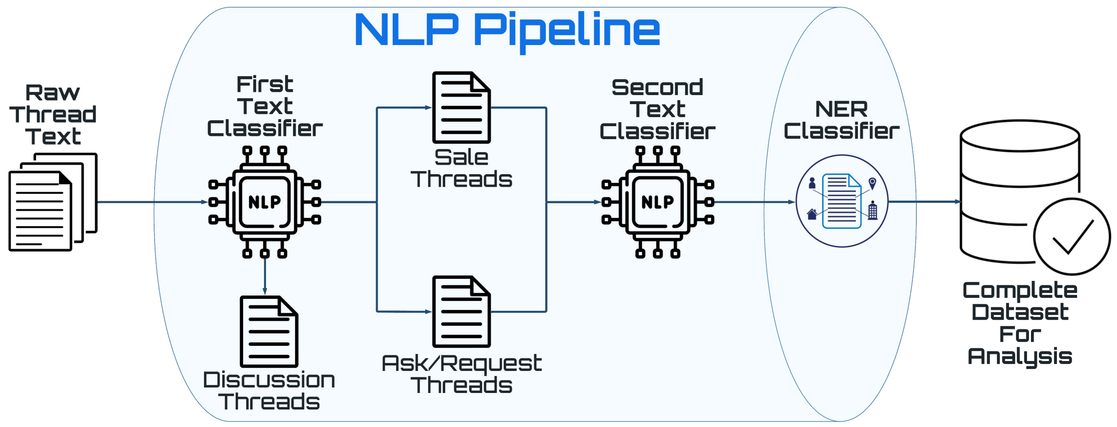


Figure 6.4: Flow diagram for the NLP pipeline (Generated with Lucid[85]. Icon sources: [86, 87, 88, 89, 90])

After passing through the text and NER classifiers, the extracted data is exported to a final dataset for analysis. This dataset includes the original input text data, the output of the two text classifiers, and the NER annotations. This is then used for further analysis, such as identifying product categories and their prices. The development and implementation of this NLP pipeline will be described in further detail in **Section 7.4**.

7 Implementation

This chapter provides a comprehensive overview of the technical aspects involved in developing a web scraper. In **Section 7.1** an in-depth discussion of the choice of programming language, libraries, and frameworks used in the development process of the scraper is presented along with the inner workings of the scraper. Furthermore, in **Section 7.2**, the methods used for implementing data storage is discussed. Moreover, **Section 7.4** covers the process of developing the NLP pipeline. It describes the approach taken to generate a labelled dataset for training the NLP models and explains the rationale behind the choice of pre-trained LLMs for developing the different models. Additionally, the chapter presents the performance of the developed models and compares them to other state-of-the-art models in the field.

7.1 Scraping

As outlined in **Section 2.1**, there exists a variety of tools and techniques that can be utilised for web scraping. Although the illicit marketplaces under investigation, employ server-side rendering Beautiful Soup and REST APIs did not prove sufficient for the task. Dynamic web scraping techniques, utilising Selenium, proved to be more effective as this allowed for circumventing the anti-scraping measures implemented by the marketplaces, more effortlessly. The succeeding subsections will provide a comprehensive overview of the scraper's general structure and workflow, as defined in section **Section 6.1**. Additionally, it will detail how the scraper was modified to handle all of the investigated forums.

7.1.1 Illicit Forum Backend: MyBB

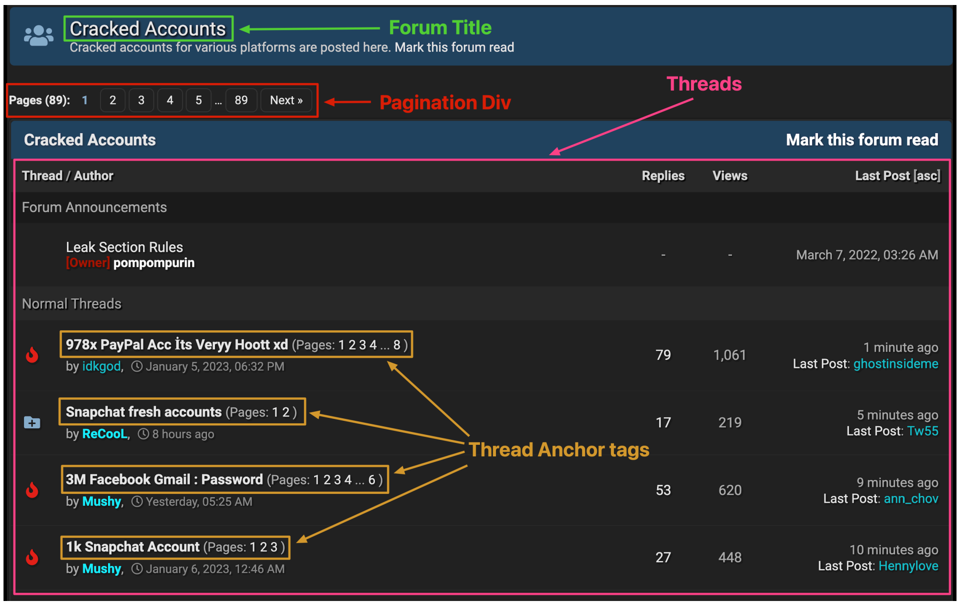
All of the investigated forums utilise the open-source `MyBB`[91] forum software to run their site. As it is open-source, it is quite frequently seen used for hosting these types of marketplaces, with similar forums such as `Nulled.to` and `Mpgh.net` likewise using this software for their sites. The structure of the investigated forums is, therefore, fortunately quite consistent. Specifically, whenever a page concerns a forum the URL of the site will look like `http://breached.vc/Forum-<Forum-name>` i.e. `http://breached.vc/Forum-General`. Should a forum contain several pages appending `?page=<page_number>` will provide the page specified by the page number. This pagination feature is present on all the investigated forums. The same holds true for the threads where `Forum` is simply replaced by `Threads-` followed by the title of the thread. Finally, each user on the site has their own page that likewise follows the URL scheme, i.e. consisting of the main site URL + `/User-<username>`.

The only slight alteration to this structure is for Hackforums.net where forums have a URL of the type `https://hackforums.net/forumdisplay.php?fid=1`, with the fid indicating the forum id. The same accounts for the thread pages where `forumdisplay.php?fid=1` is simply replaced by `showthread.php?tid=1` and for user pages it is replaced by `member.php?action=profile&uid=1`. Hereof the tid and uid refer to the thread and user id respectively. Additionally the `/User-` part of the URL is removed on Cracked.io, leaving just the username (i.e. `https://cracked.io/ExactGoat`).

The scraper utilises this setup allowing it to run separate actions to extract data for each of these types of URLs. Specifically, to determine what approach to take, the scraper simply uses string matching to determine if a specific page is a *Forum*, *Thread* or *User* page. The strings it matches by are, therefore, `/Forum` or `forumdisplay.php?fid=`, `/Thread` or `showthread.php?tid=` and `/User` or `?action=profile&uid=` respectively. For Cracked.io user pages, it was simply checked that `/Thread` and `/Forum` were not in the URL. In the following subsections, the methods used to extract data for each of the respective page types are discussed.

7.1.1.1 Forum Pages

A Forum page will generally contain a table wherein each row contains a link to a thread or a subforum as seen in **Figure 7.1**.



Cracked Accounts
Cracked accounts for various platforms are posted here. Mark this forum read

Pages (89): 1 2 3 4 5 ... 89 Next »

Cracked Accounts — Mark this forum read

Thread / Author	Replies	Views	Last Post [asc]
Forum Announcements			
Leak Section Rules [Owner] pompompurin	-	-	March 7, 2022, 03:26 AM
Normal Threads			
978x PayPal Acc İts Veryy Hoott xd (Pages: 1 2 3 4 ... 8) by idkgod, January 5, 2023, 06:32 PM	79	1,061	1 minute ago Last Post: ghostinsideme
Snapchat fresh accounts (Pages: 1 2) by ReCooL, 8 hours ago	17	219	5 minutes ago Last Post: Tw55
3M Facebook Gmail : Password (Pages: 1 2 3 4 ... 6) by Mushy, Yesterday, 05:25 AM	53	620	9 minutes ago Last Post: ann_chov
1k Snapchat Account (Pages: 1 2 3) by Mushy, January 6, 2023, 12:46 AM	27	448	10 minutes ago Last Post: Hennylove

Figure 7.1: A forum page on a MyBB site (source: https://breached.vc)

As seen the forum can sometimes be paginated. To determine if the forum is paginated the scraper searches for any *Division Element* (div) with the CSS class *pagination*.

Once found, it finds the maximum pagination page number by simply using a regex expression to find all numbers in this div, and choosing the largest one as seen in **Listing 7.1**.

Listing 7.1: Finding the maximum number in a string with regex expressions

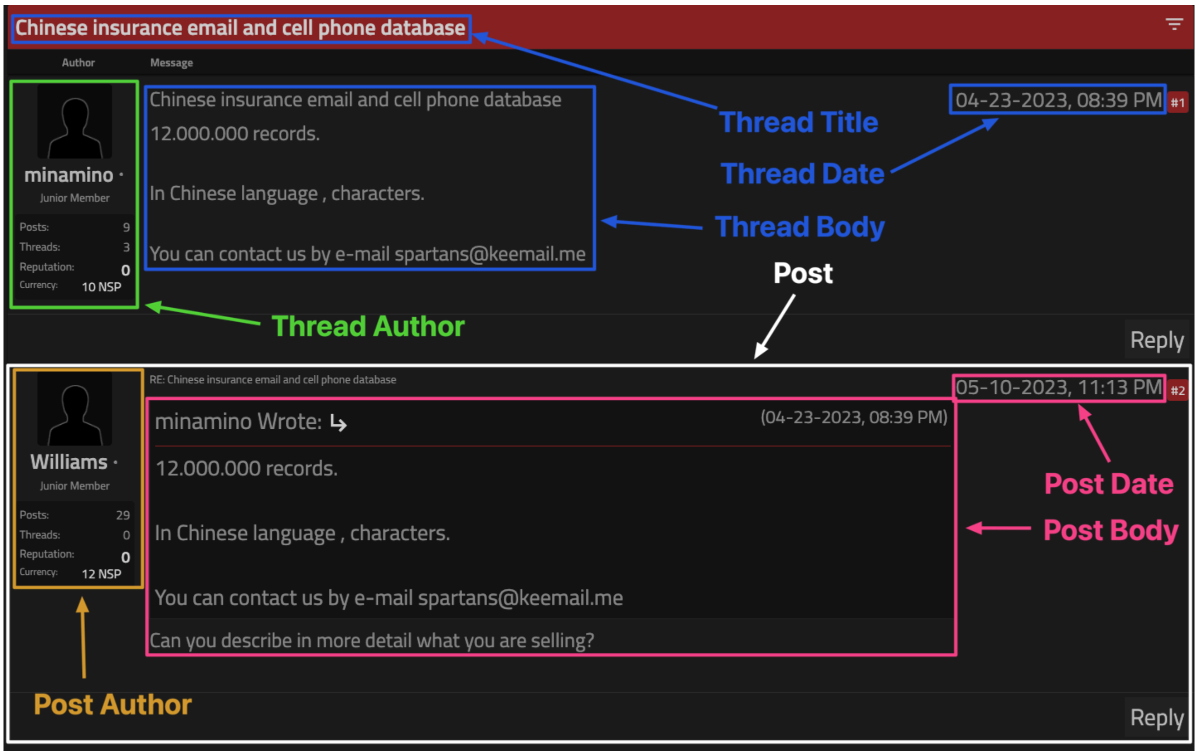
```
html = BeautifulSoup(driver.page_source)
pagination_div = html.find("div",{"class":"pagination"})
max_page = max([int(n) for n in re.findall('\d+',pagination_div.text)])
```

Hereof, the `driver.page_source` is simply the raw HTML of the fetched site stored in a Selenium `WebDriver` object.[92] To extract all links (anchor tags) from each pagination page, the raw HTML is first converted into a `BeautifulSoup` object, which provides a convenient interface for parsing HTML documents. The `find_all('a')` method is then used to extract all anchor tags, from which the source URL (href value) of each link is obtained. The process is performed recursively through each pagination page until reaching the maximum page number or encountering threads older than a specified date if provided.

As explained in **Section 7.1.1**, the links to both (sub)forums and threads follow a consistent structure. Therefore, extracting all links that contain the string *thread* will gather the thread links, while links containing *forum* will gather the (sub)forum links. In the case of Hackforums.net, the search is modified to include links with the strings *tid* and *fid* in the href attribute to collect the thread and forum links, respectively. After processing all paginated pages, the relevant data is extracted and stored in a database entry, following the flow of the scraper described in **Section 6.1**. This data comprises forum entities such as the title, number of pages, and links to all its threads. The complete function running this workflow is presented in **Appendix A2**.

7.1.1.2 Thread Pages

After collecting all thread and subforum links of a given forum, the scraper proceeds to scrape each thread and subforum iteratively. In the case of a thread, the page structure typically comprises a table, as illustrated in **Figure 7.2**. Each row in the table represents a reply to the thread, which, as previously mentioned, is also referred to as a *post*.


Chinese insurance email and cell phone database

Author | Message

minamino · Junior Member
Posts: 9
Threads: 3
Reputation: 0
Currency: 10 NSP

Chinese insurance email and cell phone database
12.000.000 records.

In Chinese language , characters.

You can contact us by e-mail spartans@keemail.me

04-23-2023, 08:39 PM #1

Reply

Williams · Junior Member
Posts: 29
Threads: 0
Reputation: 0
Currency: 12 NSP

RE: Chinese insurance email and cell phone database

minamino Wrote: ↳ (04-23-2023, 08:39 PM)
12.000.000 records.

In Chinese language , characters.

You can contact us by e-mail spartans@keemail.me

Can you describe in more detail what you are selling?

05-10-2023, 11:13 PM #2

Reply

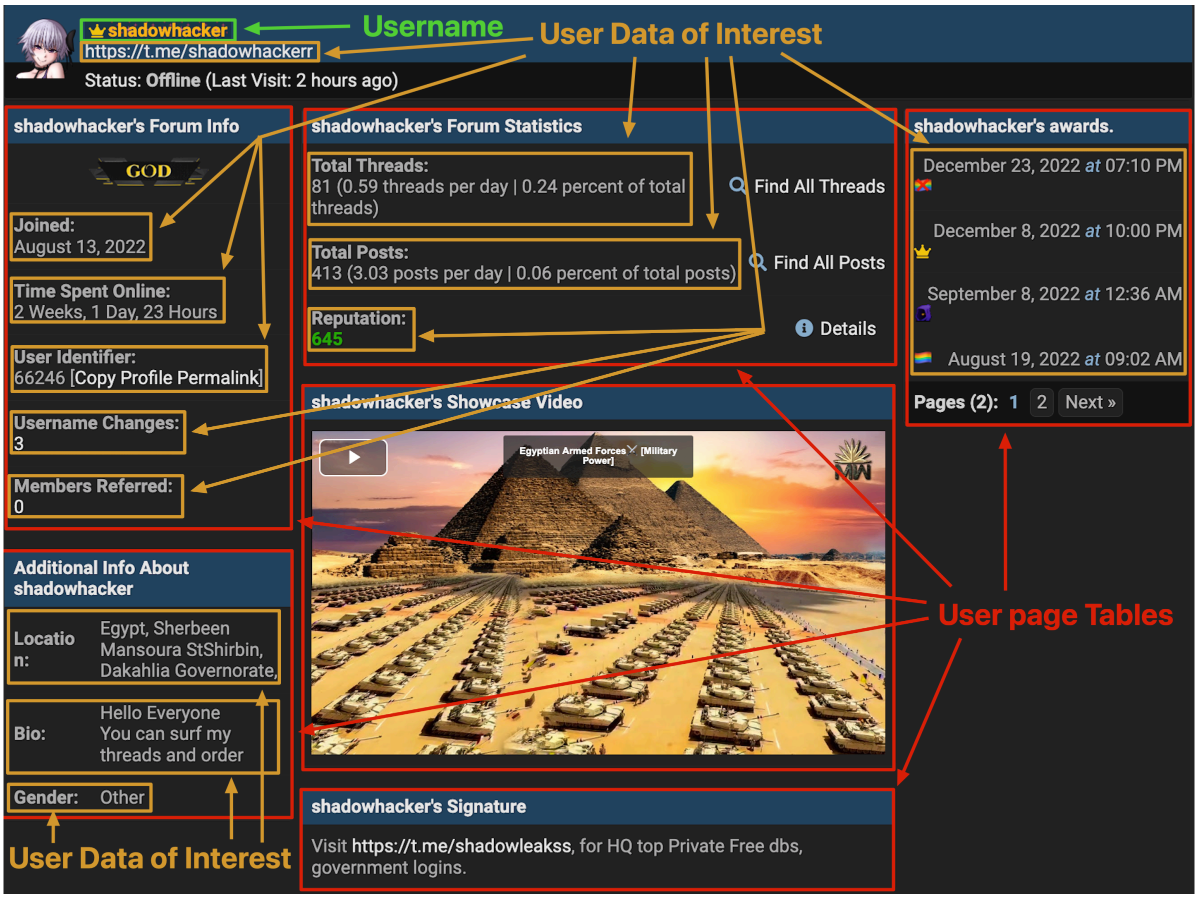
Figure 7.2: A thread page on a MyBB site (source: [58])

As with the forum pages, the thread can be paginated. Should that be the case, the same approach as described in **Section 7.1.1.1** will be taken. Once the paginated pages have been gathered, each page will be processed by finding the main table on the page by the div class or div id, depending on the site. The tables on a thread page are processed by iterating over their rows, which are defined by a list of divs contained within the table. During the iteration, each post is converted into a row in a `Pandas DataFrame`. Since the structure of these tables remains consistent, the data is extracted directly from each row. As depicted on **Figure 7.2** The first row contains the author of the post as well as the title and the date of posting for the thread. All subsequent rows represent *posts* to the thread and are extracted as such. For each post, the uid of the user who created the post and the *post identifier* (pid) of the post is extracted from the metadata of the link to the user and post, respectively. Additionally, the date of posting, as well as the post content (including any external links or image links), are extracted by the div that contains them.

Once all posts and pages of a thread have been gathered, an entry is created in the database for each post and one for the thread itself, with the database IDs of each of the posts it contains. Additionally, threads are linked to the forum section they belong to and are along with the posts it contains *related* to the user that created them in the database. I.e. a link between the nodes is created in the database as described in **Section 6.2**.

7.1.1.3 User Pages

To relate a thread or post to a specific user, a corresponding user entry must be present in the database. If it is not, following the flow of the sequence diagram in **Figure 6.1**, the page for the relevant user is fetched via inserting the uid gathered from the thread or post into the URL: `https://<domain-name>/.php?action=profile&uid=<uid>`. A user page largely consists of several tables arranged in a grid-like fashion within a div as displayed in **Figure 7.3**.



shadowhacker
https://t.me/shadowhackerr
Status: Offline (Last Visit: 2 hours ago)

shadowhacker's Forum Info

GOD

Joined:
August 13, 2022

Time Spent Online:
2 Weeks, 1 Day, 23 Hours

User Identifier:
66246 [Copy Profile Permalink]

Username Changes:
3

Members Referred:
0

shadowhacker's Forum Statistics

Total Threads:
81 (0.59 threads per day | 0.24 percent of total threads)
Find All Threads

Total Posts:
413 (3.03 posts per day | 0.06 percent of total posts)
Find All Posts

Reputation:
645
Details

shadowhacker's awards.

December 23, 2022 at 07:10 PM
December 8, 2022 at 10:00 PM
September 8, 2022 at 12:36 AM
August 19, 2022 at 09:02 AM

Pages (2): 1 2 Next »

shadowhacker's Showcase Video

Egyptian Armed Forces [Military Power]

Additional Info About shadowhacker

Location:	Egypt, Sherbeen Mansoura StShirbin, Dakahlia Governorate
Bio:	Hello Everyone You can surf my threads and order
Gender:	Other

shadowhacker's Signature

Visit https://t.me/shadowleakss, for HQ top Private Free dbs, government logins.

Figure 7.3: A user page on a MyBB site (source: [54])

Similar to the approach used for the forum and thread pages, the container housing these tables is first extracted, and each table within this container is subsequently processed. As the tables on these sites may contain various types of information, the tables are simply extracted as raw data to a list of `DataFrames`. Subsequently, the data in each of these `DataFrames` is scanned for specific keywords such as *Join Date*, *Date of Birth*, *Reputation* etc. Whenever a keyword is encountered during the scanning process, the associated value is extracted and added to a dictionary. Once all keywords have been searched for, the dictionary is compiled into a user entry to be stored in the database and linked to the posts or threads generated in the previous step.

7.2 Data Storage: SurrealDB

As mentioned in the **Section 2.2.1**, the selection of the database for the project was based on its ability to model graph-related data while also enabling individual nodes in the graph to store documents or tables. The chosen SurrealDB database allows each node in the database to contain information in the form of key-value pairs, such as username, post date, or text content, stored as documents or tables. Each document/table can then be interconnected with each other to preserve the relation each scraped entity has to each other. In the following subsections, the implementation of the database and its connection to the scraper will be described.

7.2.1 Using SurrealDB in Python

To utilise SurrelDB, their provided Python package[93] was installed and integrated into the script that runs the scraper. Once the scraper has gathered data, it is sent to the SurrealDB to be created, ensuring that existing data is not duplicated but updated with the most up-to-date value. Utilising the aforementioned Python package, a simple function named `create_database_entry` was constructed which as the name implies creates a database entry in the database. The code for the function can be found in **Appendix B1**.

When data needs to be stored in the database, and the `create_database_entry` function is used, it internally calls the `create_with_id` function. The `create_with_id` function, in turn, uses the SurrealDB Python package's `db.create` function to initialise a Remote Procedure Call (RPC) to the SurrealDB.

Having distinct object types in the database necessitates the creation of different unique ID values. The function `get_unique_id` was therefore created for the purpose of generating a unique SHA256 hash using the Python `hashlib` package. For user entries, the hash is derived from the user's ID, title/rank, and the specific forums they are associated with. Having this approach guarantees the maintenance of uniqueness among users within the database, even when data from multiple sites is coalesced. Consequently, the occurrence of duplicate data is thereby effectively prevented.

For threads, a unique hash is constructed by combining the thread identifier (`tid`) with the corresponding author's user identifier (`uid`). Posts are likewise assigned a unique identifier by merging the post identifier (`pid`) with the respective post author's user identifier. Forum entries are identified by a combination of the forum URL and title. As with the unique ID generated for users, the other objects' unique hash is also based on the main site URL, to ensure uniqueness across sites.

Additionally, the `create_with_id` function is wrapped in a try/except clause within the `create_database_entry`. This approach handles the scenario where the code attempts to create an item in the database with an ID that already exists. If such a situation occurs, the SurrealDB Python package raises a `SurrealPermissionException` indicating that the object already exists. In response, the code performs a comparison between the timestamps of the new entry and the existing entry and inserts the entry that was scraped most recently.

7.2.2 SurrealDB Security

The default settings for SurrealDB's local hosting specify that it will only accept requests coming from the localhost space (127.0.0.1/32), and the default port is set to 0.0.0.0:8000. In other words, it will only accept requests coming from the machine it is running on. As the scraper simply runs on a laptop and saves the data to the SurrealDB locally, setting up complex (but safer) authentication mechanisms is hence not necessary as the only device that can make alterations to the database is the laptop itself. Had the project been expanded to concern more forums, the database might have needed to be set up on a server with more storage, but given the scale of the websites investigated in this thesis, the storage capacity of the laptop proved sufficient.

7.2.3 Handling Special Characters (Unicode / Emojis)

Unfortunately, the SurrealDB RPC client is not configured to escape special Unicode characters (such as emojis) when making queries to the database. If a special Unicode character is passed, it simply crashes the database service. Hence these characters had to be removed with the function found in **Listing 7.2**.

Listing 7.2: Function for removing emojis and other special Unicode characters

```python
import demoji
import re
        def remove_emojies(obj): # Source: https://tinyurl.com/mpfc7kf2
    """ Removes all emojis from a string"""
    emoji_pattern = re.compile("["
        u"\U0001F600-\U0001F64F"  # emoticons
        u"\U0001F300-\U0001F5FF"  # symbols & pictographs
        u"\U0001F680-\U0001F6FF"  # transport & map symbols
        u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
        u"\U0001F1F2-\U0001F1F4"  # Macau flag
        u"\U0001F1E6-\U0001F1FF"  # flags
        u"\U0001F600-\U0001F64F"
        u"\U00002702-\U000027B0"
        u"\U000024C2-\U0001F251"
        u"\U0001f926-\U0001f937"
        u"\U0001F1F2"
        u"\U0001F1F4"
        u"\U0001F620"
        u"\u200d"
        u"\u2640-\u2642"
        "]+", flags=re.UNICODE)
    if isinstance(obj,str):
        return demoji.replace(emoji_pattern.sub(r'',obj),'').replace('--','
   ').replace("\"","\'").replace("’","\'").replace("\\","\\\\")
    elif isinstance(obj,(list,tuple,set)):
        return type(obj)(remove_emojies(elem) for elem in obj)
    elif isinstance(obj,dict):
        return {remove_emojies(key):remove_emojies(value) for key,value in
   obj.items()}
    else:
        return obj
```

The function simply uses regular expressions to match any Unicode characters in specific ranges, mainly the ones concerning emojis. Additionally, when two hyphens (--) appear together, it also makes the database fail, as when making a call to the database, an SQL query is performed, and -- in SQL queries are used for commenting out code. This was therefore replaced with a space. Similar approaches were taken for characters such as ’ and escape characters like \.

7.2.4 Handling Large Requests

Some posts on the illicit marketplaces being investigated contain an immense corpus of text, which unfortunately makes the function making RPC calls to the database fail, as the payload is too big and it simply times out. In those cases sending a manual `CREATE` query string is needed, which surpasses this implementational error in the SurrealDB Python package. This is achieved with the `KeyError` clause in the `create_with_id` function in **Appendix B1** by simply converting the dictionary values to be saved into one long string of the form `CREATE <table>:<id> SET x=1, y=2`.

7.3 Surpassing Access Limits

Websites, as previously stated, often implement measures to restrict website scraping, as it can result in a high load on their servers without providing any benefit to the website itself. Such measures include blocking IP addresses that make too many requests in a short period of time or using CAPTCHAs to verify that the requests are made by humans. However, there are ways to bypass these restrictions and scrape the website's data. One approach is to use a proxy server or rotating IP addresses, which makes it difficult for the website to track the source of the requests. Another method is to use web scraping frameworks that mimic human browsing behaviour, such as changing the headers of the requests or adding random delays between requests. For optimal effect, these measures were combined to avoid being blocked when scraping the illicit forums. In the following subsections, the steps taken to counter these anti-scraping techniques are discussed.

7.3.1 Proxies

In order to address the issue of IP blocking, proxies were initially employed as a solution. Proxies serve as intermediaries that route internet traffic through different IP addresses, creating the appearance that the traffic originates from those locations. This approach shares similarities with VPNs (Virtual Private Networks), although it lacks the same level of security provided by VPNs, as these also encrypt the transmitted data. By utilising proxies and dynamically changing the IP address for each request, the scraper successfully circumvented IP blocking measures implemented by the targeted website in the majority of cases.

However, the free proxies available are often unreliable, slow and heavily used. **Perino, D. et al.**[94] found that less than 1% of available free proxies in 2018 provided reliable performance and 10% of them even exhibited malicious behaviour like ad injection or TLS interception as well as monitoring traffic to sell to third parties. Additionally, the IP addresses of free proxy services are often blocked or blacklisted by most websites and many free proxies do not allow HTTPS connections, meaning the connection is entirely unencrypted, which in turn could expose the researcher conducting the scraping.[94]

7.3.2 Successful Solution: VPN IP Rotation

For the sake of security and reliability, the choice therefore turned to VPNs. Although reliable VPNs are typically not free of charge, they offer a dependable connection that surpasses the security level provided by proxies. The rationale behind this shift is rooted in the fact that repeatedly sending a high volume of requests to a website from a single or a few IP addresses would trigger suspicion in the website's traffic monitoring system. To circumvent this a vast amount of IPs, with substantially different geographical locations, is needed, which while effective is not offered for free.

7.3.2.1 Chosen VPN: NordVPN

Among the available options, *NordVPN*[95] emerged as the final choice for VPN service based on several compelling factors. Notably, NordVPN boasts an extensive global network of over 5500 servers, surpassing the server count offered by most other VPN providers. Furthermore, NordVPN is also renowned for its robust security measures, stringent logging policies, impressive connection speeds, and supplementary features such as split tunnelling and double VPN functionality[96]. These attributes played a vital role in addressing the challenges outlined in **Section 7.3.1** and a basic monthly subscription to the service was therefore purchased for the scraping task.

Changing the VPN location does however incur performance degradation of the scraper. Each time a new connection is made, the Python code running the scraper has to wait until the connection is established successfully. To minimise this, the scraper was set up to only change the VPN location after a certain amount of time. Initially, 15 min was attempted, as this is roughly the amount of time users spend on average on social network platforms per visit.[97] However this proved to be inefficient as the scraper was still being blocked rather often. Several values below 15 min were tried and 5 min seemed to provide the best balance between speed and avoiding IP blocking. To rotate IP addresses NordVPN was utilised along with the `nordvpn-switcher` Python package, which allows for changing VPN location from a Python script through the code found in **Listing 7.3**.

Listing 7.3: Swithcing IP address location with nordvpn_switcher

```python
from nordvpn_switcher import initialize_VPN,rotate_VPN
vpn_settings = initialize_VPN(stored_settings=1)
last_vpn_switch_time = datetime.datetime.now()
rotate_VPN(vpn_settings)
```

As seen the setup is as simple as initialising the VPN and calling the `rotate_VPN` function which will handle changing location, and waiting till the connection is established successfully. Should it fail to connect to the first location it will attempt to connect to another location. When the VPN is initialised using the `initialize_VPN` function, the `stored_settings` flag is set to 1. This flag indicates that the code will search for a nordvpn-switcher settings file, which is simply a `.txt` file that contains which operating system the code is running on, and a list of locations to connect to. When running the scraper the global `last_vpn_switch_time` variable is checked against the current time, and the IP is rotated if more than 5 min has passed.

7.3.3 Selenium and Undetected Chromedriver

The scraper runs on Selenium and is created with the `undetected-chromedriver`[23] Python package, as described in **Section 2.1.3.1**. Furthermore, the scraper is set to wait a random amount of time between 5 to 15 seconds between each page it gets to imitate user behaviour. The code for instantiating the Selenium driver can be found in **Listing 7.4**.

Listing 7.4: Instantiating an `undectected_chromedriver` instance with hCAPTCHA solving capabilities

```python
# Source: https://github.com/QIN2DIM/hcaptcha-challenger
import nest_asyncio
nest_asyncio.apply()
import hcaptcha_challenger as solver
from urllib.parse import urlparse
url_parsed = urlparse(url_main)
driver = solver.get_challenge_ctx(silence=False, lang="en")
driver.get(url_main)
```

As seen the driver is firstly initialised from the `hcaptcha_challenger` Python package, whose `get_challenge_ctx` function returns a Selenium driver object from the `undetected_chromedriver` package. Subsequently, the relevant site is fetched. The instantiation of the Selenium driver from the `hcaptcha_challenger` module, as opposed to directly from the `undetected_chromedriver` package, serves the purpose of granting the scraper the capabilities to bypass hCAPTCHAs. To detect if the scraper has been met with an hCAPTCHA the following approach is taken. Firstly, whenever the scraper navigates to a site it employs a waiting mechanism that delays its execution for up to 5 minutes, or until the page title is no longer identified as *DDoS-Guard*. This is done as certain websites utilise `DDoS-Guard`[98] to restrict the volume of incoming requests. Should the title of the page no longer be *DDoS-Guard* after 5 min, it is assumed the scraper has bypassed the *DDoS-Guard* security mechanism. This occurred in most cases.

However, if the scraper remains stuck on the DDoS-Guard page after the designated time frame, it indicates that the DDoS-Guard system has likely flagged the scraper as a bot. Consequently, the DDoS-Guard system prompts the scraper to solve an hCAPTCHA challenge. The scraper then utilises the functions the `hcaptcha-challenger`[21] provides, in order to gain access to the site. If these functions fail to solve the hCAPTCHA challenge, the scraper simply refreshes the page, allowing for a subsequent attempt to solve the hCAPTCHA. This is repeated up to 15 times before the scraper admits defeat and starts a waiting period of 30 min, before attempting to scrape from the site again.

Initially, a smaller time frame was utilised however after testing various values it was found that 30 min commonly un-marked the scraper as a bot, allowing it to proceed to the site at the next attempt, without being blocked by an hCAPTCHA. **Listing 7.5** displays a partial view of the implementation of these hCAPTHCA solving steps. The complete code for the main function running the the scraper can be found in **Appendix A1**.

Listing 7.5: Rotating IP addresses and bypassing CAPTCHAS with the `nordvpn-switcher` and `hcaphtca_challenger` packages

```
[...]
try:
    datetime_now = datetime.datetime.now()
                if ((datetime_now - last_vpn_switch_time).total_seconds() /
   60.0) > 5 and not is_retry:
                    rotate_VPN(vpn_settings)
                    last_vpn_switch_time = datetime.datetime.now()
                scrape_time = datetime_now.strftime("%Y-%m-%d %H:%M:%S")
                driver.get(request_url)
                WebDriverWait(driver, 300).until_not(EC.
   title_contains("DDoS-Guard"))
                time.sleep(randint(5,15))
[...]
except Exception as e:
            try:
                h_captcha = driver.find_element(By.ID,"h-captcha")
                if h_captcha:
                    await solve_h_capthca(site=request_url,ctx=driver)
            except NoSuchElementException as e2: # no Captcha found on site
                [...]
                html = BeautifulSoup(driver.page_source)
                title = html("title")[0].text
                if is_recursive:
                        return {"scrape_time":scrape_time,"title":
   title,"url":request_url,"response":driver.page_source}
                else:
                    return responses
```

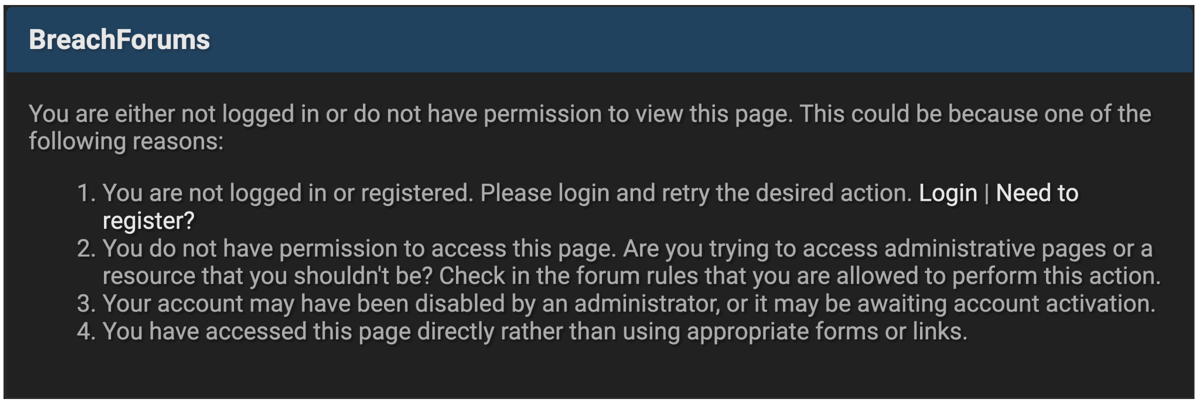
As seen in the listing, in the event that the `WebDriverWait` produces a `TimeOutException` after a waiting period of 5 minutes (300 seconds), the scraper will attempt to locate the hCAPTCHA on the site by its element id: `h-captcha`. Subsequently the scraper transfers control to the `solve_h_catpcha` function to solve the hCAPTCHA on the illicit forum before proceeding. If the scraper fails to locate the hCAPTCHA element, it is assumed that either the scraper is being blocked through other means, or the implementation of the DDoS-Guard system has been modified, necessitating a revision of the code.

7.3.4 Forum Specific Access Restrictions

Some content is unfortunately not available unless you create an account on the illicit forums and log in as seen in figure **Figure 7.4**. Additionally, each of the illicit forums has proprietary rules for what level of access different users have. For instance, specific forums may mandate a minimum number of posts, a specified membership duration, or a certain rank before granting access to restricted areas. Some of these restrictions can be bypassed by purchasing a premium account for the illicit forum. For the sake of the author's security and anonymity an anonymous email and VPN service (NordVPN[95] and Proton mail[99]) were utilised to both directly access the scraped illicit forums as well as obtain verified accounts for the illicit forums. The following subsections will outline each of the illicit forums' proprietary access rules, as well as how they are surpassed.

7.3.5 Access Restrictions: Breached.vc

On `Breached.vc` quite a large section of the website is available publicly without any requirements for registration. However when navigating to certain parts of the illicit forum, such as when attempting to use the search functionality you are met with a blockade as seen in **Figure 7.4**



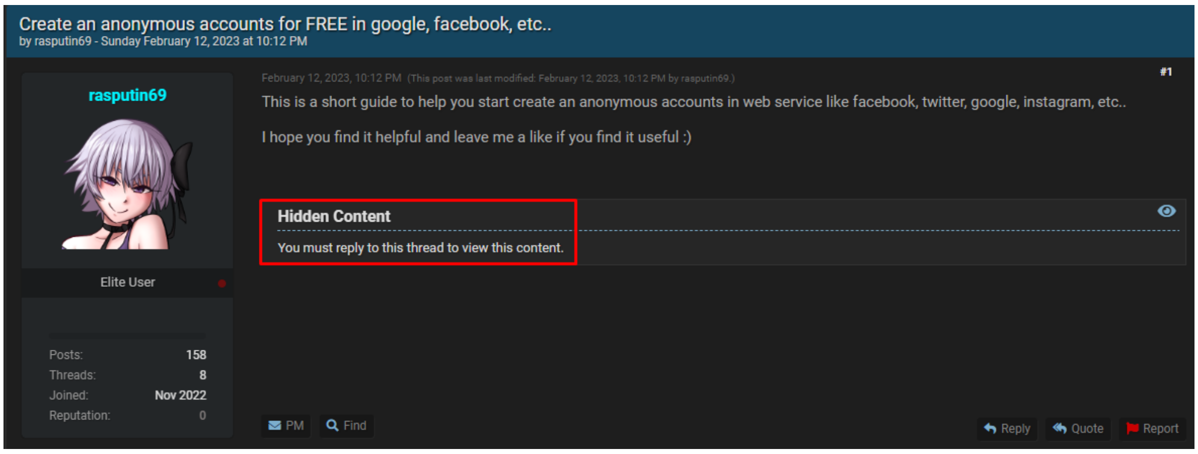
BreachForums

You are either not logged in or do not have permission to view this page. This could be because one of the following reasons:

1. You are not logged in or registered. Please login and retry the desired action. Login | Need to register?
2. You do not have permission to access this page. Are you trying to access administrative pages or a resource that you shouldn't be? Check in the forum rules that you are allowed to perform this action.
3. Your account may have been disabled by an administrator, or it may be awaiting account activation.
4. You have accessed this page directly rather than using appropriate forms or links.

Figure 7.4: Page displayed on Breached.vc when trying to access pages that are restricted to certain users

Additionally, certain threads demand not only that you are logged in to see the content as seen in **Figure 7.5**, but also that you reply to the thread before the content will be visible as displayed in **Figure 7.6**.


Create an anonymous accounts for FREE in google, facebook, etc..
by rasputin69 - Sunday February 12, 2023 at 10:12 PM

rasputin69
Elite User
Posts: 158
Threads: 8
Joined: Nov 2022
Reputation: 0

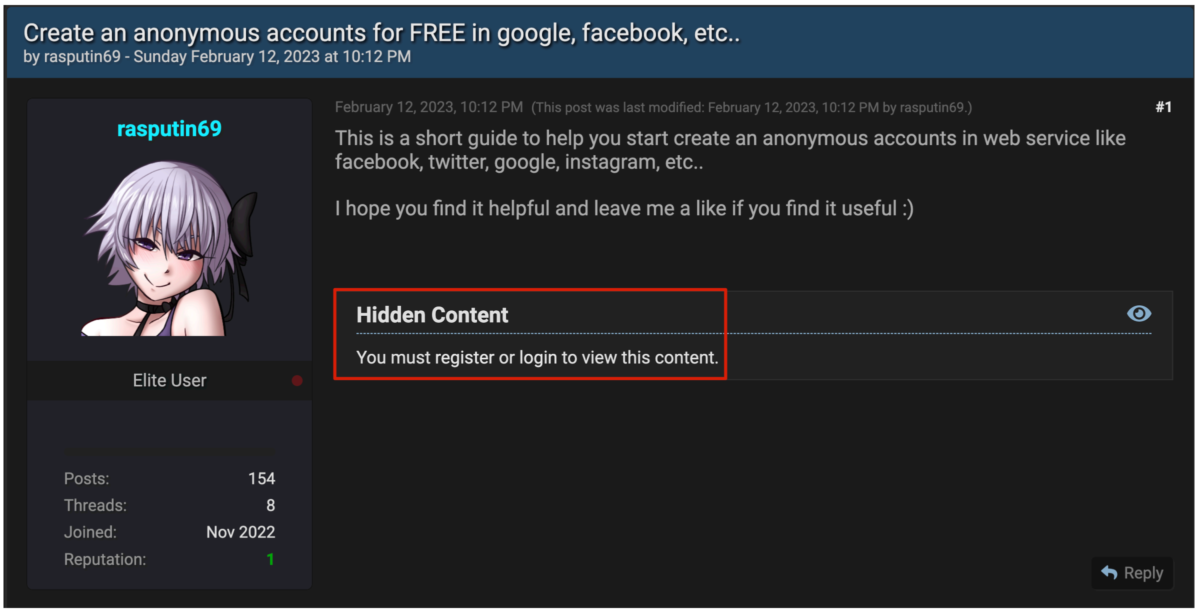
February 12, 2023, 10:12 PM (This post was last modified: February 12, 2023, 10:12 PM by rasputin69.) #1

This is a short guide to help you start create an anonymous accounts in web service like facebook, twitter, google, instagram, etc..

I hope you find it helpful and leave me a like if you find it useful :)

Hidden Content
You must reply to this thread to view this content.

Figure 7.5: Thread requiring that the logged-in user replies to the thread before content will be displayed


Create an anonymous accounts for FREE in google, facebook, etc..
by rasputin69 - Sunday February 12, 2023 at 10:12 PM

rasputin69
Elite User
Posts: 154
Threads: 8
Joined: Nov 2022
Reputation: 1

February 12, 2023, 10:12 PM (This post was last modified: February 12, 2023, 10:12 PM by rasputin69.) #1

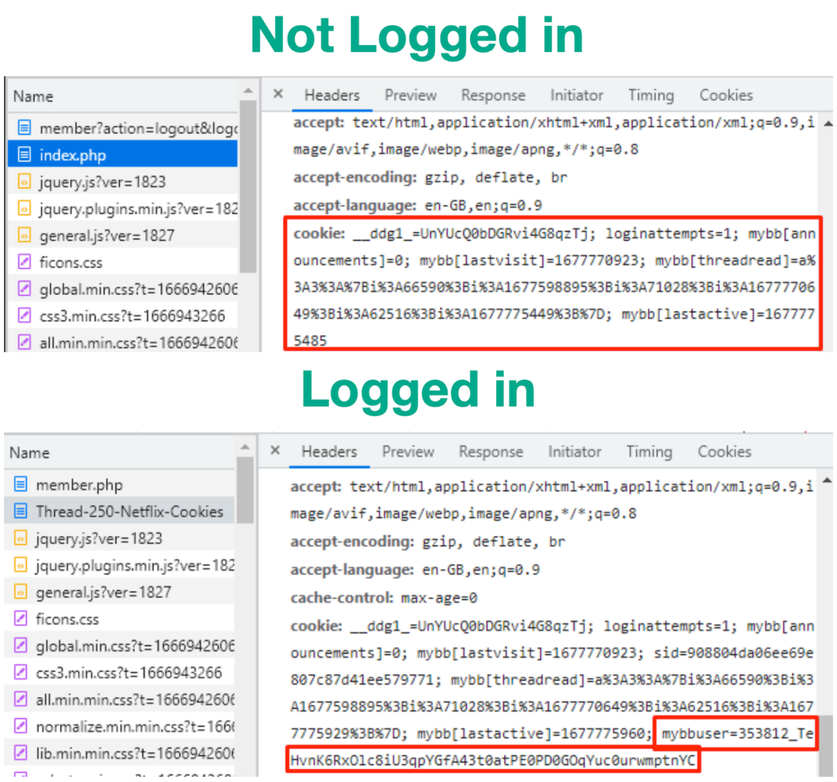
This is a short guide to help you start create an anonymous accounts in web service like facebook, twitter, google, instagram, etc..

I hope you find it helpful and leave me a like if you find it useful :)

Hidden Content
You must register or login to view this content.

Figure 7.6: Thread requiring authentication before content will be displayed

Luckily, by simply creating an account the first issue is resolved. The second obstacle can be tackled by either having the scraper reply to a thread, to see the content, or upgrading the account to a *VIP* account, which allows the user to see thread content without having to reply to the thread first. For simplification, the latter solution was used to bypass this restriction. To replicate the login mechanism in the scraper, the request made to `https://breached.vc`, after logging in was captured. In the headers of this request, the cookies were found, wherein the key `mybbuser` appears after logging in as seen in **Figure 7.7**.


Not Logged in
Name
member?action=logout&logo
index.php
jquery.js?ver=1823
jquery.plugins.min.js?ver=182
general.js?ver=1827
ficons.css
global.min.css?t=166694260
css3.min.css?t=1666943266
all.min.min.css?t=166694260
Headers
Preview
Response
Initiator
Timing
Cookies
Logged in
Name
member.php
Thread-250-Netflix-Cookies
jquery.js?ver=1823
jquery.plugins.min.js?ver=182
general.js?ver=1827
ficons.css
global.min.css?t=166694260
css3.min.css?t=1666943266
all.min.min.css?t=166694260
normalize.min.min.css?t=166
lib.min.min.css?t=166694260
Headers
Preview
Response
Initiator
Timing
Cookies

```
accept: text/html,application/xhtml+xml,application/xml;q=0.9,i
mage/avif,image/webp,image/apng,*/*;q=0.8
accept-encoding: gzip, deflate, br
accept-language: en-GB,en;q=0.9
cookie: __ddg1_=UnYUcQ0bDGRvi4G8qzTj; loginattempts=1; mybb[ann
ouncements]=0; mybb[lastvisit]=1677770923; mybb[threadread]=a%
3A3%3A%7Bi%3A66590%3Bi%3A1677598895%3Bi%3A71028%3Bi%3A16777706
49%3Bi%3A62516%3Bi%3A1677775449%3B%7D; mybb[lastactive]=167777
5485
```

```
accept: text/html,application/xhtml+xml,application/xml;q=0.9,i
mage/avif,image/webp,image/apng,*/*;q=0.8
accept-encoding: gzip, deflate, br
accept-language: en-GB,en;q=0.9
cache-control: max-age=0
cookie: __ddg1_=UnYUcQ0bDGRvi4G8qzTj; loginattempts=1; mybb[ann
ouncements]=0; mybb[lastvisit]=1677770923; sid=908804da06ee69e
807c87d41ee579771; mybb[threadread]=a%3A3%3A%7Bi%3A66590%3Bi%3
A1677598895%3Bi%3A71028%3Bi%3A1677770649%3Bi%3A62516%3Bi%3A167
7775929%3B%7D; mybb[lastactive]=1677775960; mybbuser=353812_Te
HvnK6RxOlc8iU3qpYGfA43t0atPE0PD0GOqYuc0urwmptnYC
```

Figure 7.7: Request headers for fetching `https://breached.vc` before and after logging in

Conveniently, the value associated with the `mybbuser` cookie remains constant across logins. In other words, if someone possesses your `mybbuser` cookie, they can freely log in as you without encountering any restrictions. This value was therefore simply added to the Selenium driver, with the `add_cookie` function as seen in **Listing 7.6**. This allows the scraper to operate as if it were logged in as the `mybbuser` identified in the cookie, enabling seamless retrieval of the text from restricted threads.

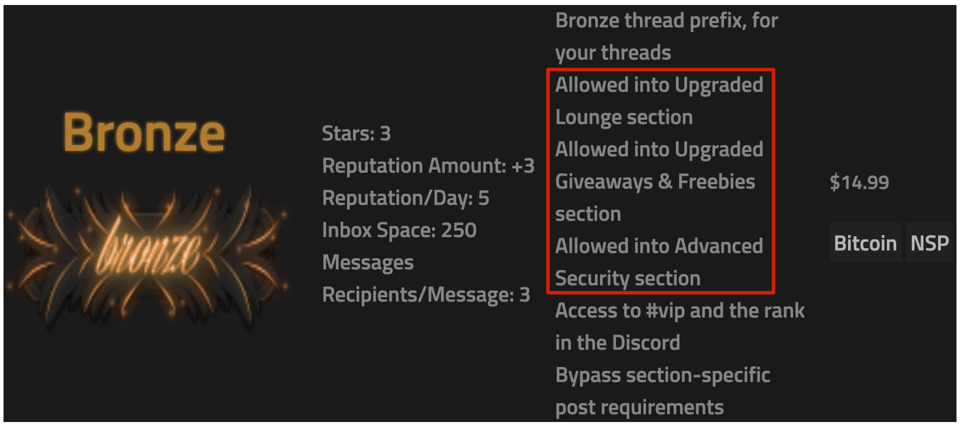
Listing 7.6: Adding a cookie to the Selenium driver

```
cookie = {"name":"mybbuser","value":user_cookie,"domain":url_parsed.netloc}
driver.add_cookie(cookie)
```

Hereof the `user_cookie` value is naturally set elsewhere to the value depicted in **Figure 7.7**.

7.3.6 Access Restrictions: Sinster.ly

Of all the forums scraped `Sinister.ly` seemed to have the least anti-scraping techniques in place. Staying with the same IP and removing the randomised waiting period between each request had no effect on being blocked by the site. In fact Sinister.ly simply seemed not to block scrapers entirely unless an running the scraper for a prolonged period of time. Simply waiting for 10 min before continuing fixed this issue. However, similarly to Breached.vc certain areas of the site are inaccessible to non-VIP members or in this case *Bronze* or above members as seen in **Figure 7.8**.


Bronze

Stars: 3
Reputation Amount: +3
Reputation/Day: 5
Inbox Space: 250 Messages
Recipients/Message: 3

Bronze thread prefix, for your threads
Allowed into Upgraded Lounge section
Allowed into Upgraded Giveaways & Freebies section
Allowed into Advanced Security section
Access to #vip and the rank in the Discord
Bypass section-specific post requirements

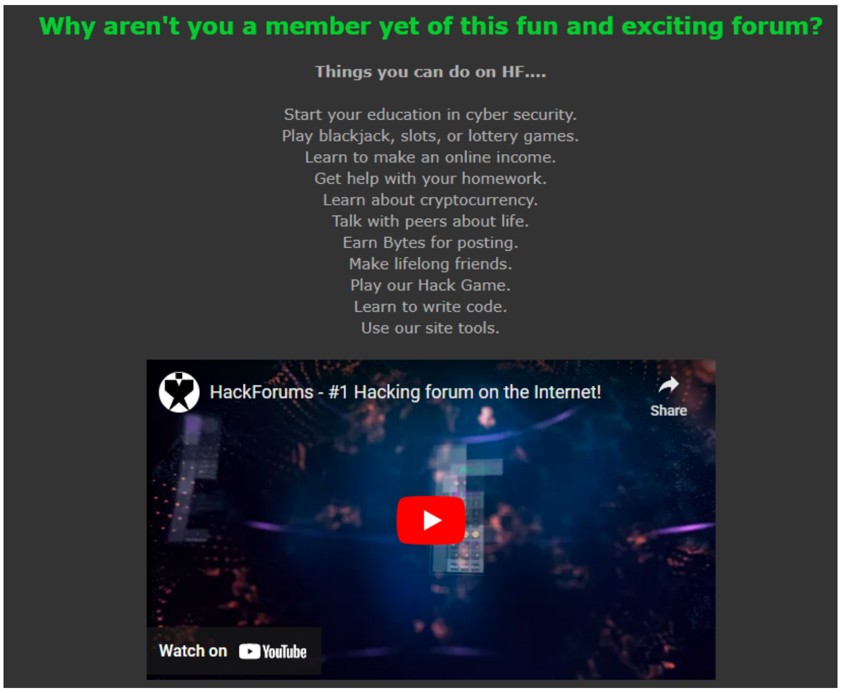
$14.99

Figure 7.8: Bronze membership benefits on Sinister.ly

A Bronze membership was therefore purchased and these access restrictions were lifted, allowing broadened access to specific forums as well as utilising the search functionality of the site. To have the scraper act as a logged-in user, the same approach as in **Section 7.3.5** was taken.

7.3.7 Access Restrictions: Hackforums.net

As with the other forums certain areas of `Hackforums.net` are accessible to everyone however most forums do demand that you have an account as displayed in figure **Figure 7.9**


Why aren't you a member yet of this fun and exciting forum?

Things you can do on HF....

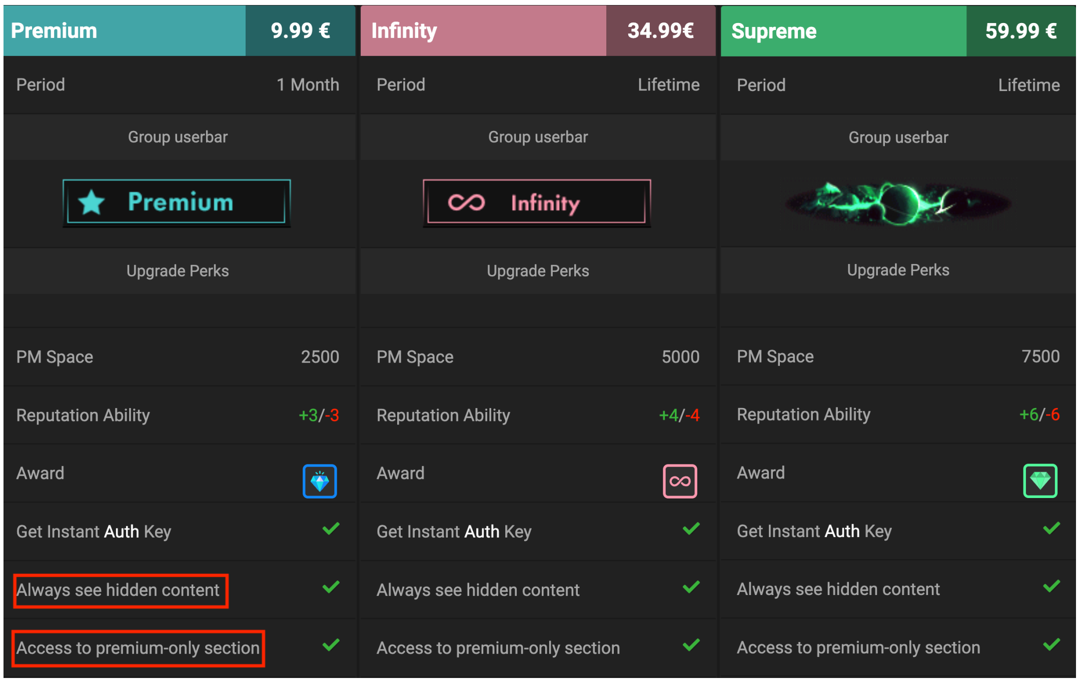
Start your education in cyber security.
Play blackjack, slots, or lottery games.
Learn to make an online income.
Get help with your homework.
Learn about cryptocurrency.
Talk with peers about life.
Earn Bytes for posting.
Make lifelong friends.
Play our Hack Game.
Learn to write code.
Use our site tools.

Figure 7.9: Access restrictions on Hackforums.net when not logged in

Though in comparison to the other sites, Hackforums.net seem to have more open forums for basic users. However, certain features, such as seeing all available forums and a list of users, only come with their premium *L33t* membership. This was however now needed for the data to be extracted by the scraper and hence was not purchased. Out of all the forums, Hackforums.net was by far the most aggressive in terms of anti-scraping techniques. Cloudflare was always met, whenever making the initial request to the forum, which was mostly handled by the `undetected_chromedriver`. Additionally, after a few hours of scraping, the scraping account was frequently banned, necessitating the creation of a new account and repeating the process. The frequency of bans varied significantly depending on the specific day.

7.3.8 Access Restrictions: Cracked.io

`Cracked.io` had, like Hackforums.net, CloudFlare employed which sometimes blocked the scraper. Additionally, certain content could be hidden; however, this could be bypassed with a premium account as depicted in **Figure 7.10**. Hence for simplification, an account was likewise purchased here.



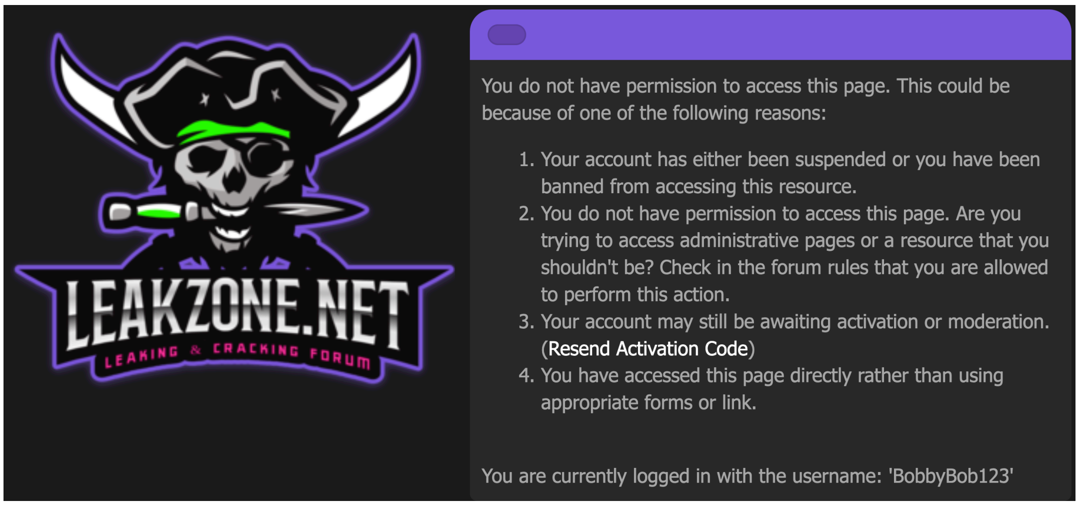
Premium	9.99 €	Infinity	34.99€	Supreme	59.99 €
Period	1 Month	Period	Lifetime	Period	Lifetime
Group userbar		Group userbar		Group userbar	
Premium		Infinity			
Upgrade Perks		Upgrade Perks		Upgrade Perks	
PM Space	2500	PM Space	5000	PM Space	7500
Reputation Ability	+3/-3	Reputation Ability	+4/-4	Reputation Ability	+6/-6
Award	💎	Award	∞	Award	💎
Get Instant **Auth** Key	✓	Get Instant **Auth** Key	✓	Get Instant **Auth** Key	✓
Always see hidden content	✓	Always see hidden content	✓	Always see hidden content	✓
Access to premium-only section	✓	Access to premium-only section	✓	Access to premium-only section	✓

Figure 7.10: Selection of purchasable ranks on Cracked.io

The moderators of Cracked.io were much less aggressive in banning, and the purchased account was fortunately only banned at the very end of the scraping period. At that point, the scraper had already scraped the relevant forums and was simply focusing on scraping additional user data, which did not require a premium account.

7.3.9 Access Restrictions: Leakzone.net

`Leakzone.net` is rather similar to Hackforums.net in the sense that most areas of the site are completely inaccessible without an account. When attempting to access restricted areas of the forum the message depicted in **Figure 7.11** is displayed.



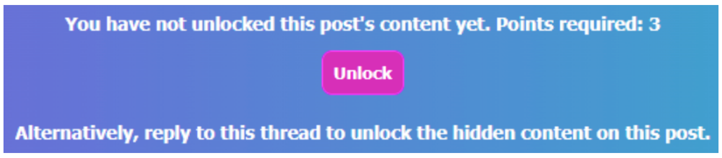
You do not have permission to access this page. This could be because of one of the following reasons:

1. Your account has either been suspended or you have been banned from accessing this resource.
2. You do not have permission to access this page. Are you trying to access administrative pages or a resource that you shouldn't be? Check in the forum rules that you are allowed to perform this action.
3. Your account may still be awaiting activation or moderation. (Resend Activation Code)
4. You have accessed this page directly rather than using appropriate forms or link.

You are currently logged in with the username: 'BobbyBob123'

Figure 7.11: Message displayed on when trying to access restricted areas on Leakzone.net

Additionally, several sections of the marketplace are inaccessible without a premium account, including the marketplace section. Furthermore when browsing forums that were available certain content would not be visible unless a premium account was purchased as illustrated in **Figure 7.12**.

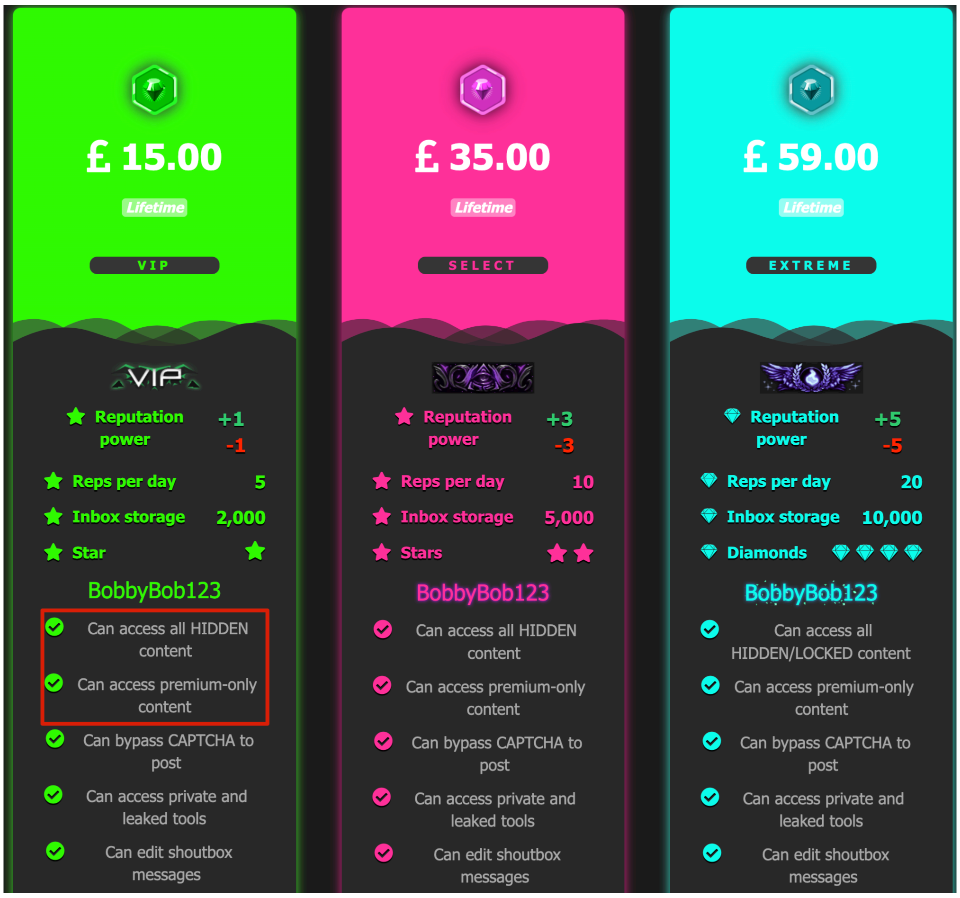


You have not unlocked this post's content yet. Points required: 3

Alternatively, reply to this thread to unlock the hidden content on this post.

Figure 7.12: Hidden content in a thread for non-premium users on Leakzone.net

As the marketplace section was the section of interest for this project, a premium VIP account was purchased. This allowed these restrictions to be bypassed as given in **Figure 7.13**


£ 15.00
Lifetime
VIP
Reputation power +1 -1
Reps per day 5
Inbox storage 2,000
Star ★
BobbyBob123
✓ Can access all HIDDEN content
✓ Can access premium-only content
✓ Can bypass CAPTCHA to post
✓ Can access private and leaked tools
✓ Can edit shoutbox messages

£ 35.00
Lifetime
SELECT
Reputation power +3 -3
Reps per day 10
Inbox storage 5,000
Stars ★★
BobbyBob123
✓ Can access all HIDDEN content
✓ Can access premium-only content
✓ Can bypass CAPTCHA to post
✓ Can access private and leaked tools
✓ Can edit shoutbox messages

£ 59.00
Lifetime
EXTREME
Reputation power +5 -5
Reps per day 20
Inbox storage 10,000
Diamonds 💎💎💎💎
BobbyBob123
✓ Can access all HIDDEN/LOCKED content
✓ Can access premium-only content
✓ Can bypass CAPTCHA to post
✓ Can access private and leaked tools
✓ Can edit shoutbox messages

Figure 7.13: Premium ranks on Leakzone.net

7.4 Natural Language Processing

As described in **Section 2.3** natural language processing has revolutionised the way data is analysed, offering powerful tools to extract meaningful insights from large and complex datasets. The preceding subsections focus on the crucial preprocessing steps necessary to efficiently apply NLP techniques to the gathered data. Additionally, the specifics of how different NLP techniques were implemented will be outlined.

7.4.1 Data Preprocessing

To efficiently apply NLP techniques to the gathered data, it is essential to have a clear understanding of the main topics of interest in the data, i.e. which words are most common. To avoid having this list of words contain common so-called *stop-words*, i.e. commonly used words such as *the*, *and* etc., the data is firstly pre-processed with the `nltk` Python package as seen in **Listing 7.7**

Listing 7.7: Removing stop-words in the thread titles with `nltk`

```python
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import re
import numpy as np
# Downloading the stopwords and punkt resources
nltk.download('stopwords')
nltk.download('punkt')

text = " ".join([thread["title"] + " " + thread["body"] for _, thread in
   threads_df.iterrows()])
text = text.lower()
# Tokenizing the text data
tokens = word_tokenize(text)
tags = nltk.pos_tag(tokens)
# Filter for nouns
nouns = [word for word,pos in tags if (pos == 'NN' or pos == 'NNP' or pos ==
   'NNS' or pos == 'NNPS')]
# Removing stopwords, non-alphabetic characters and empty strings
stop_words = set(stopwords.words('english'))
filtered_tokens = [re.sub(r'[^a-zA-Z]', '', token) for token in nouns if not
   token in stop_words and token != ' ']
filtered_tokens = [token for token in filtered_tokens if token != '']
```

As seen, all the data was firstly coalesced into one string and subsequently tokenised, as described in **Section 2.3.1**. Hereafter the data was filtered for stop-words with the `stop_words` corpus from the `nltk.corpus` package. Additionally, any non-alphabetic characters, empty strings etc. were filtered out and each word was assigned a **P**art **O**f **S**peech (POS) tag, as described in **Section 2.3.1.1**. This was then utilised to extract just the nouns, as the aim is to find *items* of digital products being sold. Lastly, a lemmatiser is used, to combine words such as *Accounts* and *Account* into their base form (*Account*). The filtered tokens could then subsequently be processed, with the `pandas.DataFrame.value_counts()` function to generate a table of unique words and their occurrence as depicted in **Table 7.1** and **Table 7.2** respectively.

Word	Occurrence
Account	1064
Service	949
Money	728
Method	514
Day	440
Discord	301
Month	267
Crypto	265
Sale	247
Bot	232

Table 7.1: Top 10 most popular words in thread titles

2-Gram	Occurrence
Make money	246
Per day	123
Earn money	114
Checker v	103
Disputed contract	101
Make day	73
Quick love	71
Get free	69
Dispute resolved	68
Contract dispute	68

Table 7.2: Top 10 most popular 2-grams in thread titles

As seen accounts and services are the most popular terms found in thread titles. This is rather self-evident as these forums are mostly *Leak* and *Hacking service* sites. Additionally, when looking at the N-grams (set of N words occurring together), with N=2 the more popular phrases are concerning *making* or *earning* money. With this list of words serving as guidance the task of generating a labelled dataset could begin.

7.4.2 Bulk Labelling

To efficiently classify all threads on the various forums, a suite of NLP models was required. To ensure the precision of these models, a labelled dataset, was required to train and evaluate the models' performance on. In order to avoid manually labelling large amounts of the data, as redundant data may be in abundance, as well as to ensure a balanced dataset, several techniques can be utilised. One would be to shuffle the data and simply select random portions of data from each dataset, However, this approach provides little guarantee that the majority of the data, will be of interest. Instead a *Bulk Labelling* technique was used. ***Bulk***[100] is a tool that provides an interface wherein the user can visualise their data in a 2-dimensional space, and inspect and save subsets of it. Text however is regrettably not 2 dimensional, and therefore has to be transformed into a 2D representation of itself to be used with this tool.

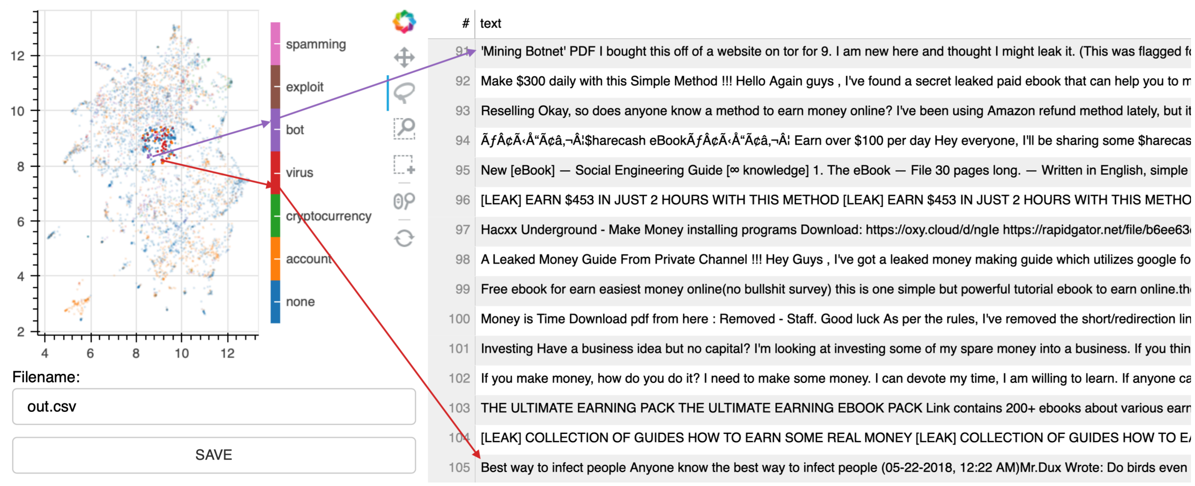
7.4.2.1 Uniform Manifold Approximation and Projection (UMAP)

One way to transform high dimensional data to a 2D representation is to use dimension reduction algorithms such as ***U****niform* ***M****anifold* ***A****pproximation* and ***P****rojection* (UMAP). UMAP is a mathematically complex algorithm that constructs a high-dimensional graph representation of the input data and then optimises a low-dimensional graph to be as structurally similar as possible.[101] The UMAP algorithm does not accept text directly, and hence the text must first be transformed into a vector representation of itself. One way to do that is to use sentence-transformer networks like *MPNet*. Fortunately, the Python `sentence-transformers` package provides pre-built models to transform sentences into their dense vector representation counterparts. The code to implement this can be seen in **Listing 7.8**.

Listing 7.8: Projecting text data to 2 dimensions with the UMAP algorithm

```
import pandas as pd
from umap import UMAP
from sentence_transformers import SentenceTransformer
# Load the universal sentence encoder
model = SentenceTransformer('all-mpnet-base-v2') # best all-around model
df = pd.DataFrame(texts, columns=["text"])
sentences = texts
X =  model.encode(sentences)
# Reduce the dimensions with UMAP
umap = UMAP()
X_tfm = umap.fit_transform(X)
# Apply coordinates
df['x'] = X_tfm[:, 0]
df['y'] = X_tfm[:, 1]

df.to_csv("2D_projection.csv")
```

After the text is projected into 2 dimensions, it is subsequently saved to a file. This can then be used with the `Bulk` tool with a list of keywords to highlight clusters concerning specific topics such as *account*, *virus*, *bot* etc. These are inspired by the keywords in **Table 7.1** and **Table 7.2** respectively, with a few manual additions. Using the lasso tool clusters are selected, inspected and saved if they contain data of interest. In this case, if the data concerns the sale of either malware, account hacks etc. **Figure 7.14** displays the tool interface with a cluster concerning botnets and viruses selected.



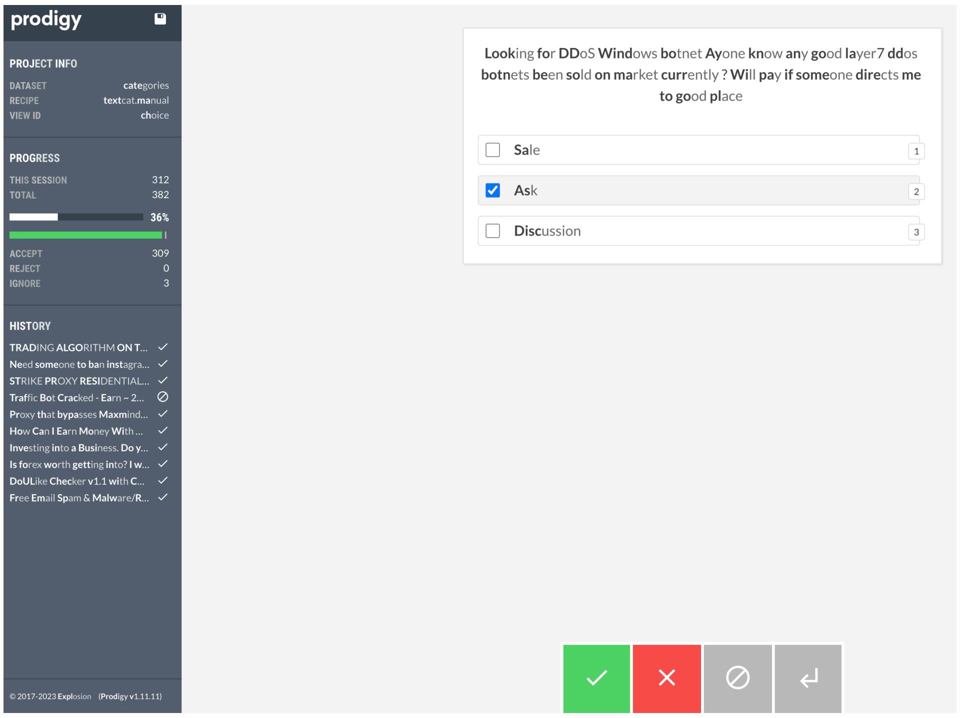
#	text
91	'Mining Botnet' PDF I bought this off of a website on tor for 9. I am new here and thought I might leak it. (This was flagged f
92	Make $300 daily with this Simple Method !!! Hello Again guys , I've found a secret leaked paid ebook that can help you to m
93	Reselling Okay, so does anyone know a method to earn money online? I've been using Amazon refund method lately, but it
94	ÃƒÂ¢Ã‹Å“Ã¢â‚¬Â¦$harecash eBookÃƒÂ¢Ã‹Å“Ã¢â‚¬Â¦ Earn over $100 per day Hey everyone, I'll be sharing some $harecas
95	New [eBook] — Social Engineering Guide [∞ knowledge] 1. The eBook — File 30 pages long. — Written in English, simple
96	[LEAK] EARN $453 IN JUST 2 HOURS WITH THIS METHOD [LEAK] EARN $453 IN JUST 2 HOURS WITH THIS METHO
97	Hacxx Underground - Make Money installing programs Download: https://oxy.cloud/d/ngle https://rapidgator.net/file/b6ee63
98	A Leaked Money Guide From Private Channel !!! Hey Guys , I've got a leaked money making guide which utilizes google fo
99	Free ebook for earn easiest money online(no bullshit survey) this is one simple but powerful tutorial ebook to earn online.th
100	Money is Time Download pdf from here : Removed - Staff. Good luck As per the rules, I've removed the short/redirection lin
101	Investing Have a business idea but no capital? I'm looking at investing some of my spare money into a business. If you thin
102	If you make money, how do you do it? I need to make some money. I can devote my time, I am willing to learn. If anyone ca
103	THE ULTIMATE EARNING PACK THE ULTIMATE EARNING EBOOK PACK Link contains 200+ ebooks about various earn
104	[LEAK] COLLECTION OF GUIDES HOW TO EARN SOME REAL MONEY [LEAK] COLLECTION OF GUIDES HOW TO E
105	Best way to infect people Anyone know the best way to infect people (05-22-2018, 12:22 AM)Mr.Dux Wrote: Do birds even

Figure 7.14: Interface used for bulk labelling of data

The process was repeated for each forum dataset to generate a balanced dataset of relevant threads (e.g. sale of malware, database leaks) and irrelevant discussions (e.g. favourite programming language).

7.4.3 Prodigy

With a subset of the data now selected, the literal labelling process can begin. ***Prodigy***[102] is a Python CLI tool that works by calling a Python package or *recipe* for a specific model, and then utilising that model for semi-automatic *smart* labelling of data. This greatly reduces the overall amount of labelling work and allows for quick iteration of models. An illustration of the interface that the Prodigy tool provides can be found in **Figure 7.15**



PROJECT INFO

DATASET	categories
RECIPE	textcat.manual
VIEW ID	choice

PROGRESS

THIS SESSION	312
TOTAL	382
	36%
ACCEPT	309
REJECT	0
IGNORE	3

HISTORY

- TRADING ALGORITHM ON T... ✓
- Need someone to ban instagra... ✓
- STRIKE PROXY RESIDENTIAL... ✓
- Traffic Bot Cracked - Earn ~ 2... ⊘
- Proxy that bypasses Maxmind... ✓
- How Can I Earn Money With ... ✓
- Investing into a Business. Do y... ✓
- Is forex worth getting into? I w... ✓
- DoULike Checker v1.1 with C... ✓
- Free Email Spam & Malware/R... ✓

© 2017-2023 Explosion (Prodigy v1.11.11)

Looking for DDoS Windows botnet Ayone know any good layer7 ddos botnets been sold on market currently ? Will pay if someone directs me to good place

- [ ] Sale 1
- [x] Ask 2
- [ ] Discussion 3

Figure 7.15: Annotating data with the Prodigy interface

7.4.4 Intent Text Classification Model

As previously mentioned, the threads of the hacking forums can concern anything between selling hacking-related items, asking for services or simply discussing various topics. To efficiently determine if a thread concerns a **Sale**, **Ask/Request** or **Discussion** a text classification model was developed. Using Prodigy a labelled dataset containing these three classes was created. Initially, it was attempted to train a model with a labelled dataset of around 5% of the total gathered data. The Prodigy tool also functions as a wrapper around the **`spaCy`**[32] training API, as introduced in **Section 2.3.3**. This allows for creating models at a higher level of abstraction, by having the API automatically handle tokenisation of the text, as explained in **Section 2.3.1**. Furthermore, Prodigy automatically handles the splitting of the data into training and validation sets and basing the model to be trained on an industrial-strength text classification model. Assuming the labelled text classification dataset is saved in the prodigy database *categories* the command in **Listing 7.9** will start training a model, and save it in the `./models/textcat/categories folder`.

Listing 7.9: CLI command to train a text classification model with Prodigy

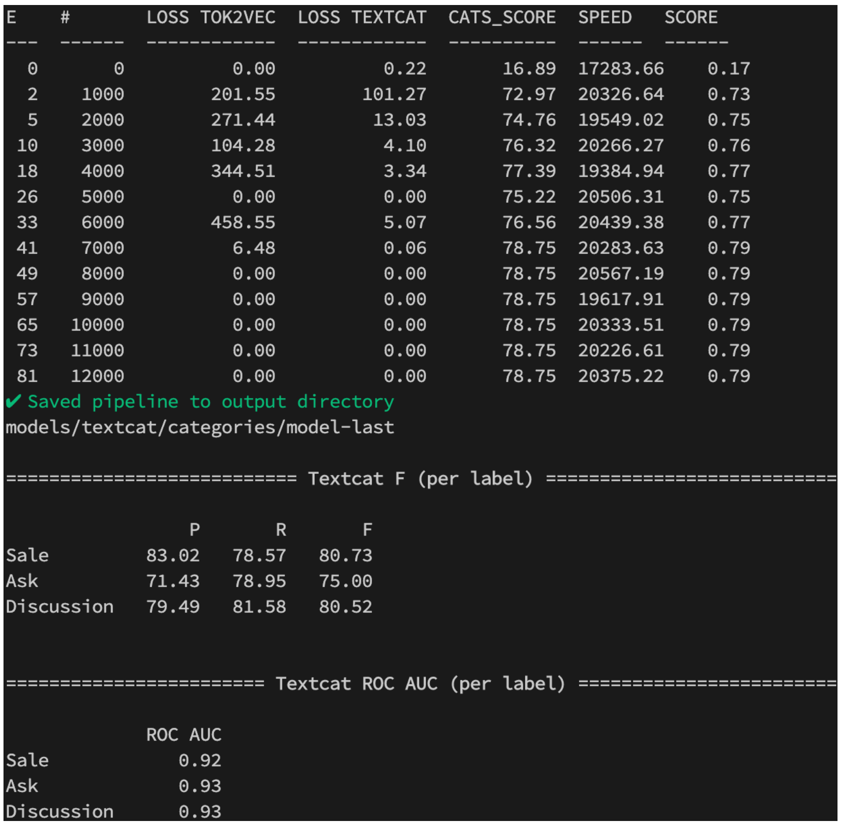
```
$ prodigy train –textcat categories –base-model en_core_web_lg –eval-split 0.2 –gpu-id 0
–label-stats ./models/textcat/categories
```

Hereof the `--base-model` parameter is set to en_core_web_lg which is the pre-trained pipeline provided by `spaCy` for the English language as given in **Section 2.3.3**.

It comes in various sizes, this being the largest. Additionally, the `--eval-split` parameter defines how much of the data should be used for validation, which in this case is set to 20% - a common division of training and validation data. Regrettably, the software does not allow for the introduction of a test set as well, meaning the scores produced are based on the validation set, i.e. data it has technically seen before, introducing the risk of overfitting. The other two metrics `--gpu-id` and `--label-stats` simply run the command on the GPU and outputs metrics for each of the classification labels respectively. The default text classification recipe, utilises the following hyperparameters[103]:

- **Batch Size**: 128
- **Learning Rate**: 5e-05
- **Optimiser (Adam)**:
  - **Warm-up steps**: 250
  - **Total Steps**: 20000

The command ran for around 5 min and produced the output found in **Figure 7.16**



```
E    #       LOSS TOK2VEC  LOSS TEXTCAT  CATS_SCORE  SPEED    SCORE
---  ------  ------------  ------------  ----------  ------  ------
  0       0          0.00          0.22       16.89  17283.66    0.17
  2    1000        201.55        101.27       72.97  20326.64    0.73
  5    2000        271.44         13.03       74.76  19549.02    0.75
 10    3000        104.28          4.10       76.32  20266.27    0.76
 18    4000        344.51          3.34       77.39  19384.94    0.77
 26    5000          0.00          0.00       75.22  20506.31    0.75
 33    6000        458.55          5.07       76.56  20439.38    0.77
 41    7000          6.48          0.06       78.75  20283.63    0.79
 49    8000          0.00          0.00       78.75  20567.19    0.79
 57    9000          0.00          0.00       78.75  19617.91    0.79
 65   10000          0.00          0.00       78.75  20333.51    0.79
 73   11000          0.00          0.00       78.75  20226.61    0.79
 81   12000          0.00          0.00       78.75  20375.22    0.79
✔ Saved pipeline to output directory
models/textcat/categories/model-last

=========================== Textcat F (per label) ===========================

                 P       R       F
Sale         83.02   78.57   80.73
Ask          71.43   78.95   75.00
Discussion   79.49   81.58   80.52


======================== Textcat ROC AUC (per label) ========================

             ROC AUC
Sale            0.92
Ask             0.93
Discussion      0.93
```

Figure 7.16: Performance metrics of the Prodigy intent text classification model during training

The tool conveniently displays the weighted average F1-score over the different intent categories at various numbers of epochs (E) and iterations (#). Additionally, the speed, given in words processed per second, is given along with the `CATS_SCORE`, i.e. the F1 score on the validation set. `SCORE` is the weighted average of all the components in the NLP pipeline (see **Section 2.3.3**) which in this case is the same as `CATS_SCORE` but normalised between 0 and 1. Finally, the value of the loss function for each component in the NLP pipeline is given, which in this case is simply the text tokenizer and text classifier.

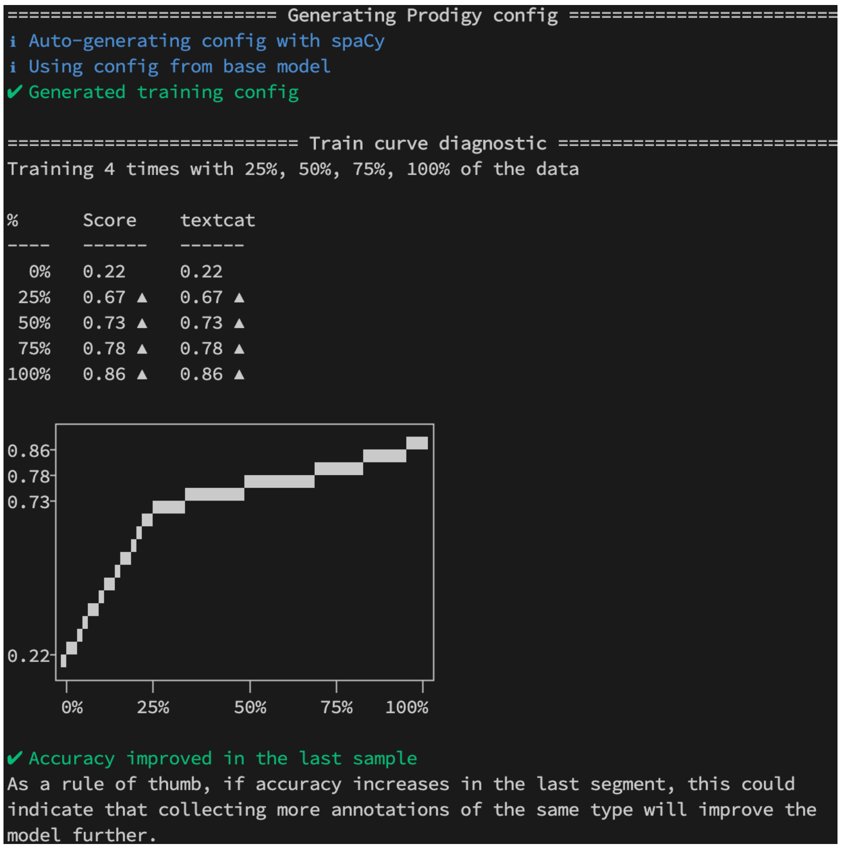
From the numbers, it can be seen that already after 2 epochs the score has jumped to 73% and hardly increase in the following epochs. At the bottom, it can be seen that the model averages an F1-Score across the labels of around 79%, though the accuracy for each of the labels is fairly high and nearly identical. However, as described in section **Section 2.3.4**, accuracy is hardly a great metric to evaluate the model on, as it cannot be guaranteed that the dataset is completely balanced. Likewise as explained in **Section 2.3.4**, in machine learning the goal is to minimize the loss function, i.e. bring the computed output as close to the expected as possible. In the output in **Figure 7.16** the loss function stays at 0 after the 49th epoch untill the training finishes, indicating that the model is simply unable to learn any new information from the given dataset.

7.4.4.1 Improving The Model

To try and improve the model, a common approach is to generate a learning curve as described in **Section 2.3.4**. This will help determine if adding more labelled data to the dataset will help improve the model's performance. Prodigy conveniently provides a `train-curve` command that will train with 0%, 25%, 50%, 75% and 100% of the data respectively, and report the performance at each step. Should the score increase in the last segment it could indicate that providing more data would improve the model. The command is rather similar to the `train` as seen in **Listing 7.10**, the only difference being the `--show-plot` parameter, that produces a visual representation of the learning curve as depicted in **Figure 7.17**.

Listing 7.10: CLI command to produce a learning curve with Prodigy

```
$ prodigy train-curve –textcat categories –base-model en_core_web_lg –eval-split 0.2 –gpu-
id 0 –show-plot
```

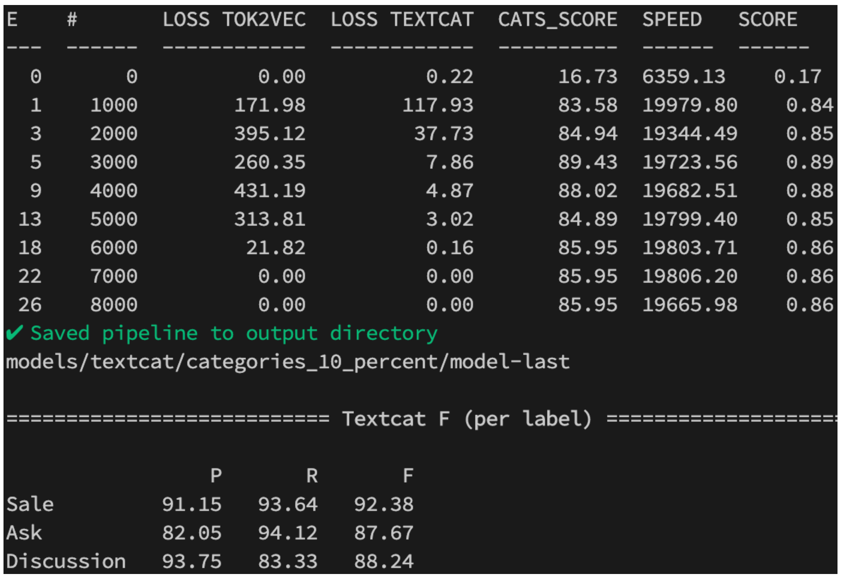


```
======================== Generating Prodigy config ========================
ℹ Auto-generating config with spaCy
ℹ Using config from base model
✔ Generated training config

========================== Train curve diagnostic ==========================
Training 4 times with 25%, 50%, 75%, 100% of the data

%       Score    textcat
----    ------   ------
  0%    0.22     0.22
 25%    0.67 ▲   0.67 ▲
 50%    0.73 ▲   0.73 ▲
 75%    0.78 ▲   0.78 ▲
100%    0.86 ▲   0.86 ▲
```

```
✔ Accuracy improved in the last sample
As a rule of thumb, if accuracy increases in the last segment, this could
indicate that collecting more annotations of the same type will improve the
model further.
```

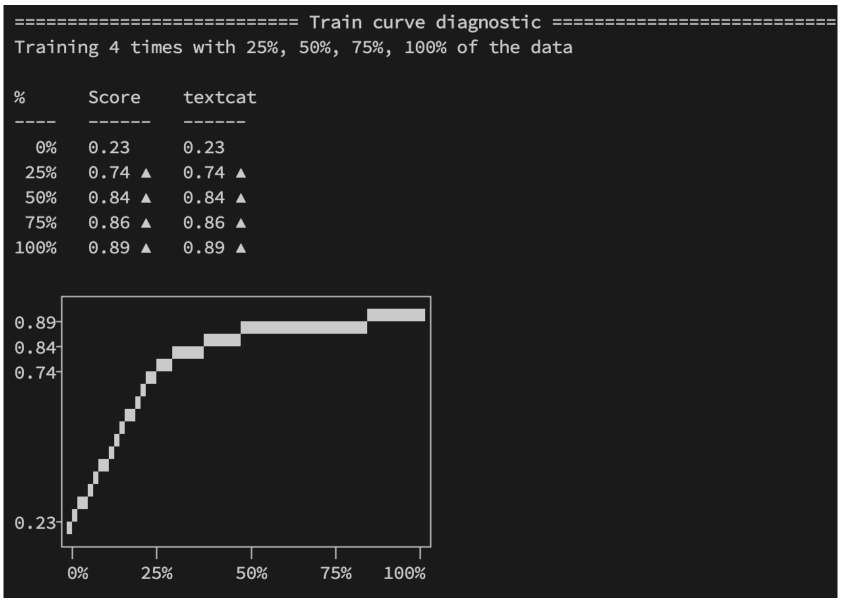
Figure 7.17: Output of the Prodigy `train-curve` comand

From the figure, it can be seen that the performance of the model increases at every step as more data is added, especially in the last step. This is a clear indication that adding more labelled data to the dataset would have a fair chance of improving the model's performance. Consequently, the procedure followed in **Section 7.4.3** was reenacted by manually labelling another 5% of the total data gathered. To ensure only new data was added, Prodigy provides an `--exclude` parameter that allows for excluding data from previous datasets for labelling. With the labelled dataset now hosting 10% of the total gathered threads a new model was trained with the same approach as before. The output of the models training and its learning curve is depicted in **Figure 7.18** and **Figure 7.19** respectively.



```
E    #       LOSS TOK2VEC  LOSS TEXTCAT  CATS_SCORE  SPEED    SCORE
---  ------  ------------  ------------  ----------  ------   ------
  0       0          0.00          0.22       16.73  6359.13    0.17
  1    1000        171.98        117.93       83.58  19979.80    0.84
  3    2000        395.12         37.73       84.94  19344.49    0.85
  5    3000        260.35          7.86       89.43  19723.56    0.89
  9    4000        431.19          4.87       88.02  19682.51    0.88
 13    5000        313.81          3.02       84.89  19799.40    0.85
 18    6000         21.82          0.16       85.95  19803.71    0.86
 22    7000          0.00          0.00       85.95  19806.20    0.86
 26    8000          0.00          0.00       85.95  19665.98    0.86
✔ Saved pipeline to output directory
models/textcat/categories_10_percent/model-last

=========================== Textcat F (per label) ===================

                P       R       F
Sale        91.15   93.64   92.38
Ask         82.05   94.12   87.67
Discussion  93.75   83.33   88.24
```

Figure 7.18: Output of the Prodigy tool when training the intent classifier with 10% labelled data of the total dataset



```
========================== Train curve diagnostic ==========================
Training 4 times with 25%, 50%, 75%, 100% of the data

%      Score    textcat
----   ------   ------
  0%   0.23     0.23
 25%   0.74 ▲   0.74 ▲
 50%   0.84 ▲   0.84 ▲
 75%   0.86 ▲   0.86 ▲
100%   0.89 ▲   0.89 ▲
```

Figure 7.19: Learning curve for the intent text classification model trained on 10% labelled data of the total dataset

As evident in the figures, the overall score improved by about 6% while the loss function started converging much earlier (epoch 18) than the previous model. This would be a clear indication that the model simply needed more data to improve its performance, as it now converges much quicker. Likewise, the F1-Score improved quite significantly for each of the classes, the sales class especially. To verify that the model's

0.86 F1-score is truly competitive, however, it must be compared to other models in the field.

7.4.4.2 Comparing to State-of-the-art LLMs

To ensure that the text classification model developed is on par with state-of-the-art LLMs, the model developed by the Prodigy software will be compared with 4 of the most renowned LLMs on the market: BERT, XLNet, RoBERTa and DistilBERT. These models were chosen based on both their popularity as well as proclaimed performance. In the following subsections, the development of each of these models will be discussed and a comparison of their performance will be given.

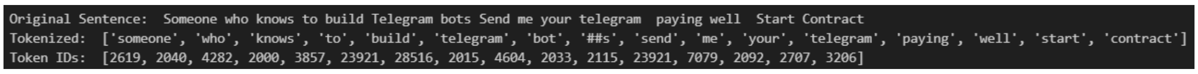
BERT

As described in **Section 2.3.2.1** BERT is a commonly used transformer model in NLP. It would therefore be suitable to test if using a pre-trained BERT model as a base can improve the performance of the text classifier. To utilise a pre-trained BERT model, the text first has to be encoded into the right format as outlined in **Section 2.3.2.1**. Conveniently the `transformers` library provides a pre-trained BERT model along with a `BertTokenizer` that will automatically encode text to the BERT-supported format as seen in **Listing 7.11**

Listing 7.11: Using the transformer library's `BertTokenizer` to tokenize the dataset.

```
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased',
   do_lower_case=True)
```

The tokenizer now contains the text in tokenised format as well as each of their respective token IDs as seen in **Figure 7.20**



```
Original Sentence:  Someone who knows to build Telegram bots Send me your telegram  paying well  Start Contract
Tokenized:  ['someone', 'who', 'knows', 'to', 'build', 'telegram', 'bot', '##s', 'send', 'me', 'your', 'telegram', 'paying', 'well', 'start', 'contract']
Token IDs:  [2619, 2040, 4282, 2000, 3857, 23921, 28516, 2015, 4604, 2033, 2115, 23921, 7079, 2092, 2707, 3206]
```

Figure 7.20: Example of text tokenised for a BERT model

To add the special tokens (CLS, SEP, PAD) as described in **Section 2.3.2.1**, the tokeniser provides an `encode_plus` method, transforming the data into a vector with these tokens added. Finally, the labelled dataset is divided into a training, validation and test set, and converted into a `TensorDataset` so it can be passed into the pre-trained BERT model. When fine-tuning the model a linear scheduler was utilised to adjust the learning rate as the model trains. This was instantiated with zero warm-up steps and the number of training steps set to the size of the training set times the number of epochs, ensuring each epoch covers the entire training dataset. The code utilised to conduct these steps as well as train the model can be found in **Appendix C1**. As described in the code comments, **Jacob Devlin et al.**[30], the creators of BERT, claimed the following hyperparameters worked well across all tasks:

- **Batch size**: 16,32
- **Learning Rate (Adam)**: 5e-5, 3e-5, 2e-5
- **Number of Epochs**: 2,3,4

To find the optimal model, a combination of all parameters was therefore tested, and the best parameters were saved along with the model it produced. The models were trained on a server with an NVIDIA A10 GPU with 24GB of onboard RAM, an Intel Xeon Processor (Icelake) with 60 cores and 120 GB system RAM, though the entire training barely utilised anything but the GPU. It was found that the best parameters for this use case were:

- **Batch size**: 16
- **Learning Rate (Adam)**: 2e-05
- **Number of Epochs**: 4

This produced a final weighted average F1-score of 0.92, which while not a a significant increase from 0.86 is still a quite high score in its own right. To ensure this truly is the optimal model, it was tested against the other state-of-the-art pre-trained LLMs.

XLNet
As with BERT, a pre-trained version of XLnet is available via the `transformers` Python package. The model was created with the same suite of parameters utilised for the BERT model, with the only difference being that the BERT pre-trained model was an *uncased* model. I.e. it does not differentiate between cased and uncased text. The only available pre-trained XLNet model available at the time of writing was a *cased* model. After running the model with the various set of hyperparameters the best performance achieved was an F1-score of 0.91 with the following hyperparameters:

- **Batch size**: 32
- **Learning Rate (Adam)**: 2e-05
- **Number of Epochs**: 4

Surprisingly it performed slightly worse than BERT, despite XLNet supposedly outperforming BERT in many other tasks.[40] This may be caused by the fact that the XLNet considers the case in the text when classifying. As a lot of threads don't strictly present perfect grammar etiquette, such as having entire threads in all caps, this may be the cause of the model's slightly reduced performance.

RoBERTa
RoBERTa is likewise only available in a case-sensitive version from the `transformers` package. Testing the suite of hyperparameters utilised for the previous models, achieved a maximum weight average F1-score of 0.87 with the same optimal hyperparameters as the XLNet model. Surprisingly, it did not outperform BERT despite the fact that RoBERTa is supposed to be a ***R**obust* ***O**ptimised* version of BERT.[41] Compared to BERT, RoBERTa was only trained on a masking task, whereas BERT was trained on a masking task and next-sentence prediction task. From the achieved performances this could mean that the BERT model better handles the data present in this task, as the dataset consisted of a combination of thread title and body, for which BERT's next sentence prediction capabilities may have granted it an advantage.

DistilBERT
DistilBERT is, as explained in **Section 2.3.2.4**, essentially a smaller and faster version of BERT. It is therefore expected that the model achieves a similar performance. The optimal hyperparameters for this model were however found not to be the same as the BERT model but the same as the RoBERTa and XLNet model, achieving a maximum weighted average F1-score of 0.89. This should be considered quite impressive as the model was twice as fast to train as BERT and much faster than any of the other models, as displayed in **Table 7.3**.

Model Type	Prodigy	BERT	DistilBERT	XLNet	RoBERTa
Epoch	26	4	4	4	4
Learning Rate	5e-05	2e-05	2e-05	2e-05	2e-05
Batch Size	128	16	32	32	32
Training Time	5m 17s	2m 9s	**1m 3s**	1h 6m 20s	3m 10s
Weighted Avg F1	0.86	**0.92**	0.89	0.90	0.87

Table 7.3: Performance comparison of LLMs for the intent text classification model

From **Table 7.3** it should also be noted that the training time for XLNet is not truly indicative of the time it takes to train. As the model is larger than the others, the 24GB of onboard GPU RAM was simply not enough to host the model during training with a batch size of 32. Hence the XLNet model with those parameters was trained on a CPU. This only further advertises the superiority of the BERT and DistilBERT models, as these were trainable with 24GB of RAM, and achieved better performances than the XLNet and RoBERTa models.

Looking at the training and validation loss for each of the models, as depicted in **Figure 7.21**, it can also be seen that all the models except XLNet start converging after just two epochs, indicating that further training using the same training dataset would not produce significantly better results. The XLNet model converges slightly later, at 3 epochs, and is still quite far from zero. Seeing as it starts to converge, despite being far from zero, indicates that the model is simply not fit for this specific NLP task. Note the Prodigy model's data is not included in the graph due to its prolonged number of epochs, which would render the remaining models' data unreadable.

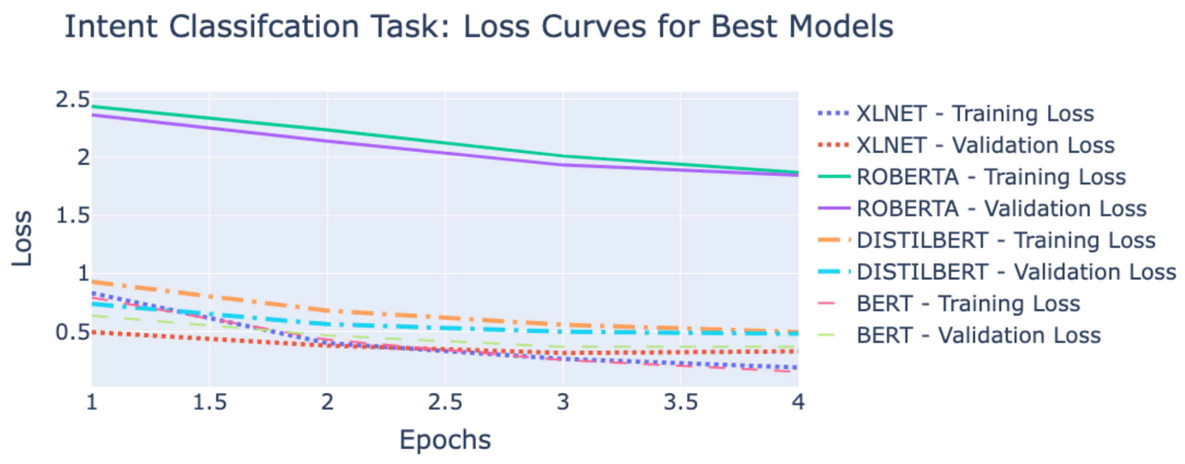


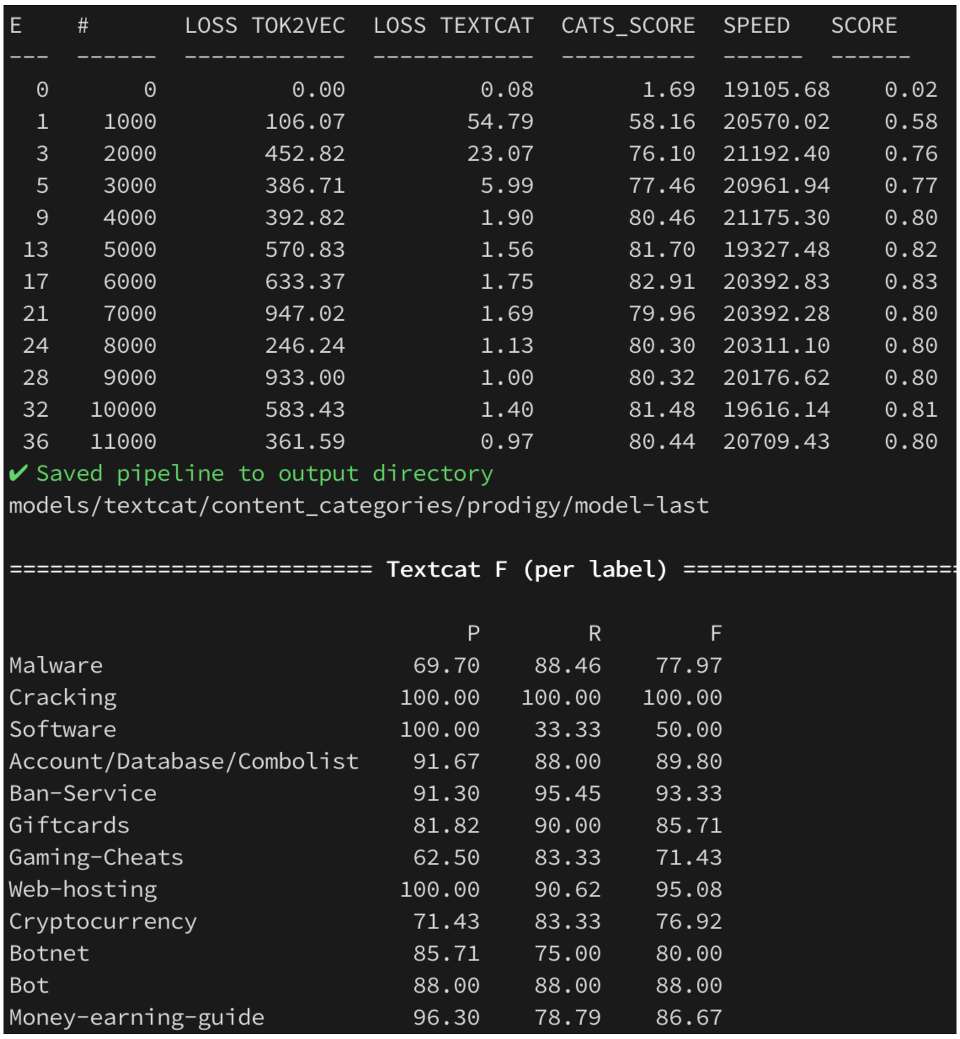
Figure 7.21: Training and validation loss for the best performing models for the intent text classification task

The best-performing model, in this case, the BERT model, was naturally selected for integration into the NLP pipeline. With the first model created, the focus could turn to the next model for the NLP pipeline.

7.4.5 Category Text Classification Model

To develop the next text classifier the same approach as with the previous model was taken. 10% of the total dataset was labelled, according to the classes defined in **Section 6.3**. Both Prodigy and the previously tested models were trained on this dataset, utilising the same hyperparameters as before.

Prodigy revealed comparable results as with the previous model, averaging an F1-score of 0.8. It especially struggled on the *Software* (F1=0.5) and *Gaming-Cheats* (F1=0.7) tags, as depicted in **Figure 7.22**, indicating that the generated labelled dataset is unbalanced.



```
E    #       LOSS TOK2VEC  LOSS TEXTCAT  CATS_SCORE  SPEED    SCORE 
---  ------  ------------  ------------  ----------  ------  ------
  0       0          0.00          0.08        1.69  19105.68    0.02
  1    1000        106.07         54.79       58.16  20570.02    0.58
  3    2000        452.82         23.07       76.10  21192.40    0.76
  5    3000        386.71          5.99       77.46  20961.94    0.77
  9    4000        392.82          1.90       80.46  21175.30    0.80
 13    5000        570.83          1.56       81.70  19327.48    0.82
 17    6000        633.37          1.75       82.91  20392.83    0.83
 21    7000        947.02          1.69       79.96  20392.28    0.80
 24    8000        246.24          1.13       80.30  20311.10    0.80
 28    9000        933.00          1.00       80.32  20176.62    0.80
 32   10000        583.43          1.40       81.48  19616.14    0.81
 36   11000        361.59          0.97       80.44  20709.43    0.80
✔ Saved pipeline to output directory
models/textcat/content_categories/prodigy/model-last

=========================== Textcat F (per label) ====================

                                  P        R        F
Malware                       69.70    88.46    77.97
Cracking                     100.00   100.00   100.00
Software                     100.00    33.33    50.00
Account/Database/Combolist    91.67    88.00    89.80
Ban-Service                   91.30    95.45    93.33
Giftcards                     81.82    90.00    85.71
Gaming-Cheats                 62.50    83.33    71.43
Web-hosting                  100.00    90.62    95.08
Cryptocurrency                71.43    83.33    76.92
Botnet                        85.71    75.00    80.00
Bot                           88.00    88.00    88.00
Money-earning-guide           96.30    78.79    86.67
```

Figure 7.22: Performance metrics of the Prodigy category classification model during training

Training the suite of transformer models also did not improve the performance significantly, with the best-performing model (XLNet) achieving an F1 score of 0.84. An overview of the performance for each of the trained models is depicted in **Table 7.4**, along with their accompanying loss curves in **Figure 7.23**.

Model Type	Prodigy	XLNet	DistilBERT	BERT	RoBERTa
Epoch	36	4	4	4	4
Learning Rate	5e-05	2e-05	2e-05	2e-05	2e-05
Batch Size	128	16	16	16	32
Training Time	8m 57s	4m 40s	**1m 4s**	2m 6s	2m 3s
Weighted Avg F1	0.8	**0.84**	0.81	0.79	0.74

Table 7.4: Performance comparison of LLMs for the category text classification model

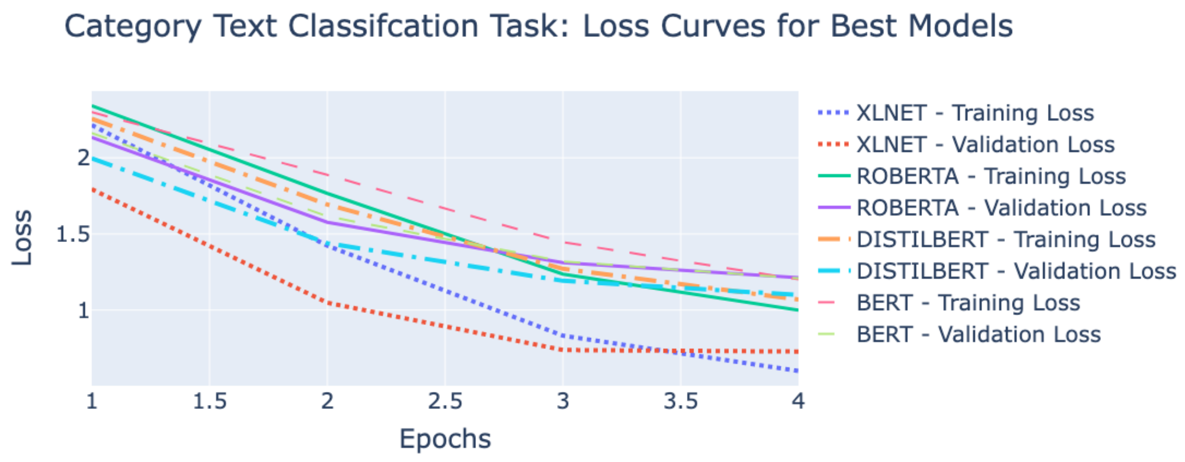


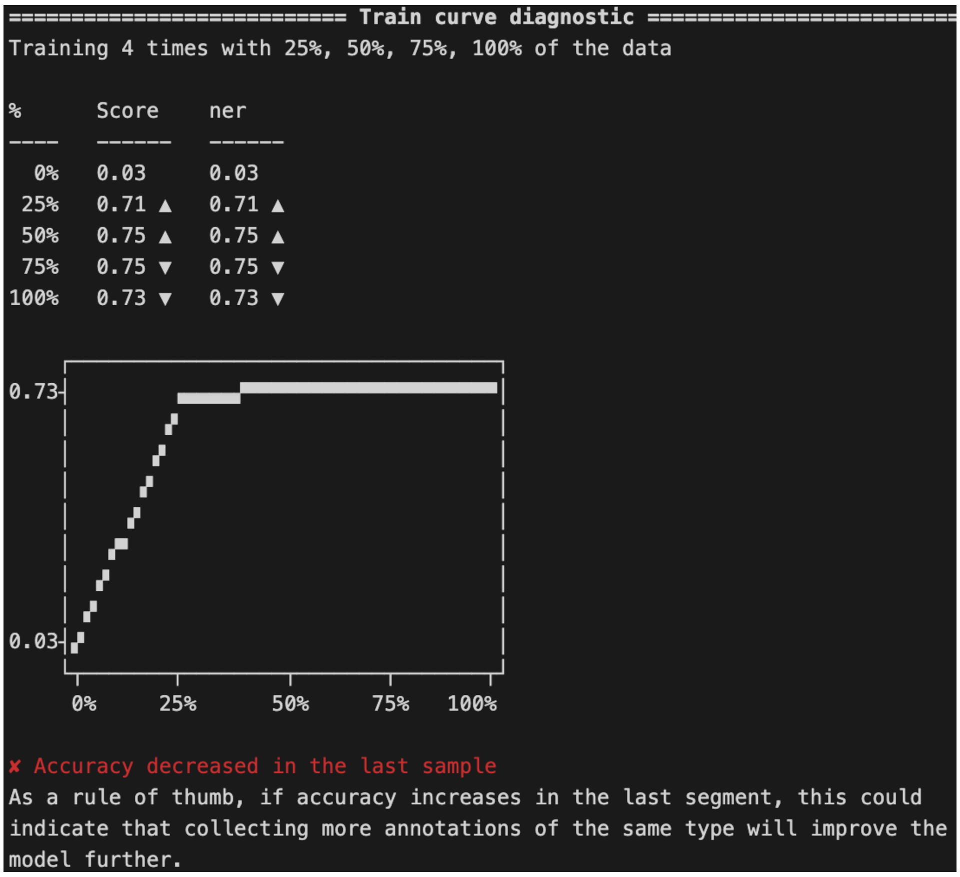
Figure 7.23: Training and validation loss for the best performing models for the category text classification task

The loss curves clearly illustrate that the majority of the models' validation loss starts to converge after the third epoch, with DistilBert showing convergence even earlier, after the second epoch. This observation strongly suggests that increasing the number of epochs will not improve the models' performance. While an attempt could be made at adding more data, due to the small size of the overall dataset it is likely that it would only induce marginal performance improvements. Efforts were therefore directed more towards the other models as well as the analysis of the data they created. Consequently the XLNet model was chosen to operate as the category classifier in the NLP pipeline, as it performed the best among all the models developed.

7.4.6 NER Classification Model

Having segregated the data into intent and category classes, the next step is to extract the prices of items being sold or discussed in the threads using an NER classifier. Similar to the two text classifiers, the development of the NER classifier required annotating a portion of the dataset to create a labelled dataset for training. This annotation process was once again facilitated by using Prodigy, allowing for the straightforward selection of spans of text, in this case, the prices mentioned in the text.

Initially, only 5% of the dataset was labelled for training the NER classifier, which resulted in a relatively low average F1-score of 0.64 when trained using Prodigy. In an attempt to improve the performance, an additional 5% of the data was labelled, increasing the size of the training set to match that of the previous models. Unfortunately, this incremental increase had a minimal impact, with the F1-score only improving to 0.73. To check if adding more data would improve the model further, the learning curve was generated through the same approach as for the text classifiers, producing the graph found in **Figure 7.24**.



```
========================= Train curve diagnostic =========================
Training 4 times with 25%, 50%, 75%, 100% of the data

%      Score    ner
----   ------   ------
  0%   0.03     0.03
 25%   0.71 ▲   0.71 ▲
 50%   0.75 ▲   0.75 ▲
 75%   0.75 ▼   0.75 ▼
100%   0.73 ▼   0.73 ▼
```

```
✘ Accuracy decreased in the last sample
As a rule of thumb, if accuracy increases in the last segment, this could
indicate that collecting more annotations of the same type will improve the
model further.
```

Figure 7.24: The learning curve for the NER model when using 0, 25, 50, 75 and 100% of the training dataset

As more data is added, the model's performance improves up to 25%, but the rate of improvement after this plateaus and actually decreases indicating that adding more data would not necessarily improve the model's performance. A likely cause of this could be that the model has already learned most of the relevant patterns in the data, and additional data would not provide significant new information. As with the other models, the model created by Prodigy was compared to a suite of state-of-the-art transformer LLMs, resulting in the metrics found in **Table 7.5**.

Model Type	Prodigy	XLNet	RoBERTa	BERT	DistilBERT
Epoch	24	4	2	2	3
Learning Rate	2e-05	5e-05	3e-05	5e-05	3e-05
Batch Size	128	16	32	16	32
Training Time	8m 29s	4m 5s	3m 15s	2m 52s	**1m 33s**
Weighted Avg F1	**0.73**	70.76	70.34	68.97	72.26

Table 7.5: Performance comparison of LLMs for the NER classification model

From the table, it is evident that regardless of the model used, the performance does not surpass an F1-score of 0.73. Surprisingly DistilBERT performs nearly identically to Prodigy, despite its much shorter training time. To accurately depict how well the models improve during their training the loss curves were produced as illustrated in **Figure 7.25**

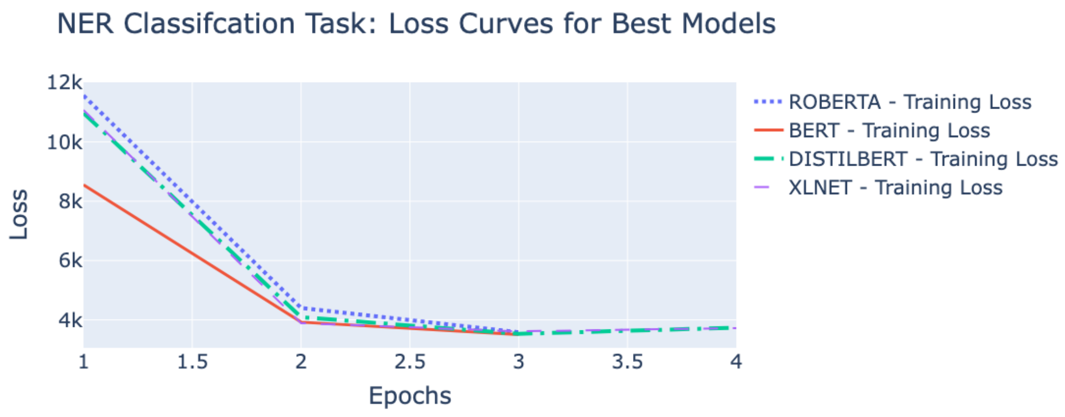


Figure 7.25: Loss curves for the various models trained for the NER task

It should be noted that due to the format of the produced labelled NER dataset, the Prodigy tool was utilised to train the transformer LLMs as well. This was naturally conducted with the same collection of hyperparameters utilised for the other models, however, the calculation of loss is done differently in Prodigy, meaning the values of loss are depicted as quite high with values reaching as high as 12000. Additionally, Prodigy only provides the training loss, disallowing the integration of the validation loss in the figure. Nonetheless, the loss curves prove that the loss plateaus around the second epoch across the board, indicating, that increasing the number of epochs, will not provide substantially better results.

Combining this with the insights of **Figure 7.24** no further attempts were made in increasing the labelled dataset, as it was unlikely to improve the performance of the model. Consequently, the Prodigy model was therefore chosen as the NER classifier for the pipeline. The three models were subsequently integrated to form one cohesive pipeline that, when given a segment of text, will firstly classify its intent, i.e. is it concerning a Sale, Request or simply a Discussion. Secondly, the category is determined by the second model, and lastly, the NER model extracts the prices found in the text. The code connecting these three models in sequence can be found in **Appendix D1**. The unstructured text could now be passed through this pipeline, thereby extracting the information it contains, and its analysis could commence.

8 Analysis

Having successfully scraped data from the chosen illicit forums, as well as built NLP models to process the data, the analysis of this data could commence. The datasets were separated by forum and were initially analysed in a thread segment and user segment. In the following subsections, the various analytical approaches and their reasoning will be elaborated. All graphics were generated in Python, and the code associated with them can be found in the accompanying zip file in the `DataAnalysis.ipynb`, `plotting.py` and `utils.py` files respectively.

8.1 Threads

Firstly the investigation will focus on the thread segment. Specifically, their size by site, the products they contain, their categorical distribution and their main topics. Each of these will be discussed in the following subsections.

8.1.1 Thread Distribution

While the forums investigated contain many discussion-related forums, the focus of this thesis is the **Sales** and **Request** of digital products being conducted on the forums. The focus was therefore as previously mentioned directed towards their *Marketplace* subforums. As the forums have emerged at different times, the selection of threads in these subforums is hence quite varied in size as given by **Figure 8.1**.

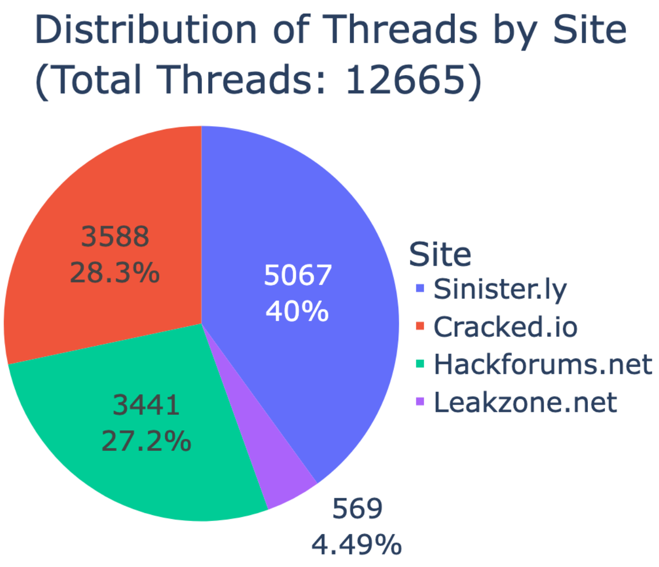


Figure 8.1: Distribution of threads in the *Marketplace* subforum across the investigated forums

From the figure it is evident that Sinister.ly holds majority over the other sites, with Cracked.io and Hackforums.net having 10% less than Sinister.ly. Lastly Leakzone.net is by far the minority, which when correlated with each of the forum's self-proclaimed number of threads as given in **Table 8.1**, matches with the fact that Leakzone.net is simply a smaller site.

Website	Total forum threads	Total marketplace threads 2023	% of Total
Sinister.ly	155058	5067	3.27
Hackforums.net	6300000	3441	0.05
Cracked.io	842000	3588	0.43
Leakzone.net	35655	569	1.60

Table 8.1: Proclaimed number of threads of investigated forums (including all subforums. Sources: [58, 61, 59, 62])

From the table, it is also evident that the number of threads, from 2023, that occur within the `Marketplace` subforum of each of the forums is rather modest compared to the overall sizes of the forums. This just proves that the forums are much more than illegal marketplaces for hackers to gather. This especially holds true for Hackforums, however seeing as it is a forum that has been around for longer, the total number of threads is naturally higher. This hence makes the number of threads in the marketplace section for 2023, quite small. That isn't to say that the subsequent analysis conducted in this section is not indicative. As the data is focused on a specific subforum and only concerns 2023, it naturally is a rather small subset of the total amount of threads on the forums. However, the analysis still grants valuable insights into the trends of **Sales** and **Requests** of the `Marketplace` subforum for 2023 for each of the investigated forums. In addition to **Table 8.1**, **Table 8.2** displays the number of collected threads being classified as a **Sale**, **Request** or **Discussion** threads respectively.

Site	Sale Threads	Request Threads	Discussion Threads
Cracked.io	246 (14.03%)	1437 (81.97%)	70 (3.99%)
Hackforums.net	3635 (52.74%)	3099 (44.97%)	158 (2.29%)
Leakzone.net	8 (0.71%)	1104 (98.31%)	11 (0.98%)
Sinister.ly	4505 (51.98%)	4081 (47.09%)	81 (0.93%)

Table 8.2: Distribution of threads classified as **Sale**, **Request**, or **Discussion** for each of the investigated forums

The table clearly displays a fairly even distribution of **Sale** and **Request** related threads for Hackforums.net and Sinister.ly. Cracked.io has a far more skewed distribution with just 14% **Sale** threads, and 82% **Request** threads, though not as badly as Leakzone.net, with only 0.7% of threads classified as **Sales**. Hence in the following subsections, the visualisations produced for Hackforums.net and Sinister.ly will paint a robust picture of their actual **Sale** and **Request** related threads. Conversely, the conclusions drawn for Cracked.io and Leakzone.net will be far more indicative of their threads related to **Requests**, as opposed to **Sales**.

8.1.2 Thread Categories

Having parsed all of the collected threads through the NLP pipeline their category as given in **Section 6.3**, have been assigned. Looking at their distribution in terms of sales in **Figure 8.2**, it is clear that each of the sites has its own unique focus. Specifically, Sinister.ly's sales mainly focuses on *Money-earning-guides* (46%), followed by *Cryptocurrency* (36%). Additionally, Sinister.ly has more **Sales** threads than all the other sites combined.

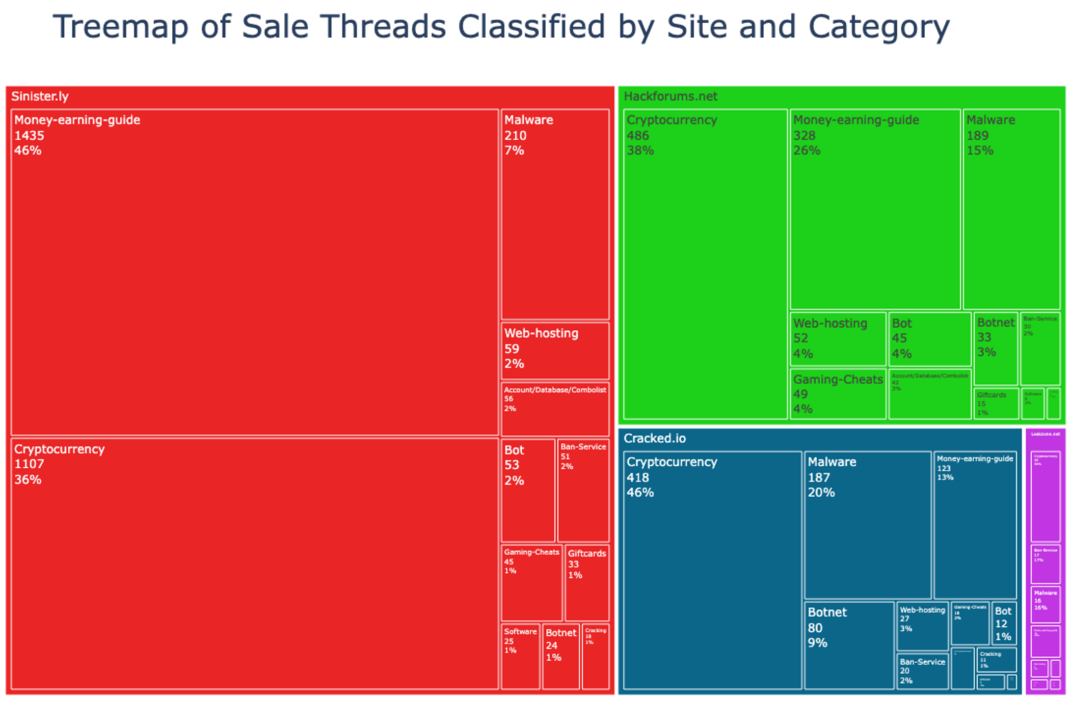


Figure 8.2: Treemap of the categories for the **Sale** threads on the investigated forums

Surprisingly, despite its name Cracked.io's main focus in **Sales** threads lies not on *Cracking*, but on *Cryptocurrency* (46%), *Malware* (20%), and *Botnets* (9%). Hackforums.net follow a similar pattern as the previous forums, having *Cryptocurrency* (38%), *Money-earning-guides* (26%) and *Malware* (15%) as the top digital products sold. Leakzone.net likewise follows the trend, mainly focusing on *Cryptocurrency* (39%) and *Malware* (16%). However, it differs from the others, by having 17% of the listings related to *Ban-services*. From the collected threads, it is clear that the main topics of 2023 across the forums seem to be *Malware*, *Cryptocurrency* and *Money-earning-guides*. In the case of *Cryptocurrency* this is no surprise as the world's largest cryptocurrency has risen almost 80% this year alone.[104] How to easily earn money is naturally a topic that never dies, and *Malware* occurrences have seen a 15% increase in Q1 2023 compared to Q1 2022.[105]

Turning to the main categories when it comes to **Requests**, paints an entirely different picture is given as illustrated in **Figure 8.3**.

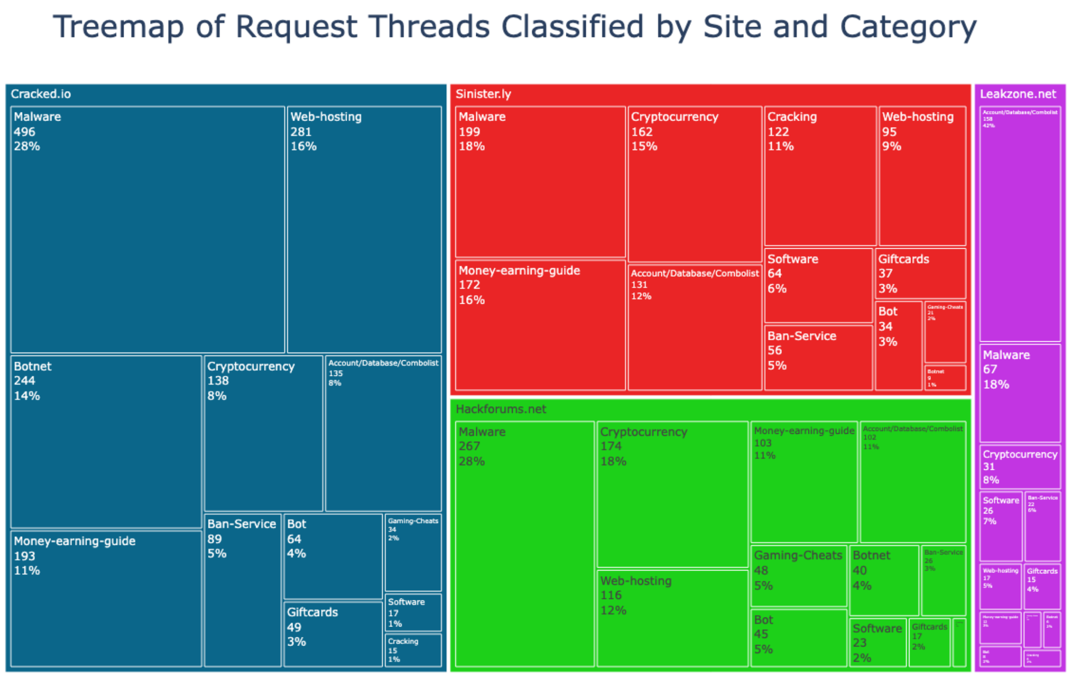


Figure 8.3: Treemap of the categories for the **Request** threads on the investigated forums

Cryptocurrency is no longer the main topic of interest and each class is more evenly distributed amongst the threads. In contrast to the **Sales** threads, where Sinister.ly had the most threads, Cracked.io, has the most **Request** threads, though the distribution between forums is much more even, apart from Leakzone.net. When it comes to **Requests**, Cracked.io's main focus has now shifted to *Malware* (28%), *Web-hosting* (16%) and *Botnets* (14%). A similar pattern is seen for Hackforums.net, however with *Cryptocurrency* (18%) instead of *Botnets*. Sinister.ly has a slightly more even distribution between its top 3 categories: *Malware*, *Cryptocurrency* and *Money-earning-guides* (15-18%), with Leakzone.net being the only site with a skewed distribution. Specifically, *Account/Database/Combolists* seem to be much more requested than other products.

Contrarily to the **Sales** threads, these statistics indicate **Request** threads are much more related to illicit products such as malware and botnets across most of the forums. One interpretation of this would be that for the sake of security and anonymity of the vendor, these products are not as heavily advertised as cryptocurrency and money-making methods are. To access these services, it would hence make sense for the vendor to simply monitor requests and directly contact potential customers.
To summarise the findings of this section:

- Sinister.ly has more **Sales** threads and Cracked.io more **Request** threads than the other sites.
- The top 3 categories found in **Sales** threads are *Cryptocurrency*, *Malware*, and *Money-earning-guides*

- The top category found in **Request** threads are *Malware*, followed by a split between *Cryptocurrency*, *Web-hosting* and *Money-earning-guides*

8.1.3 Thread Prices

From the distribution of the forums defined in the previous section, it would naturally also be of interest to investigate, the prices for items listed in the `Marketplace` subforums in terms of both **Request** and **Sale** threads. **Figure 8.4** displays *boxplots* of the distribution of the **Sales** and **Request** threads prices for each of the forums respectively. A boxplot determines the ***I****nter* ***Q****uartile* ***R****ange* (IQR) or the range of most common values. The *whiskers* at the end depict the minimum and maximum values respectively with the boxes' vertical lines displaying Q1, the median and Q3 respectively. Note the data has been filtered from outliers via IQR filtering. This effectively removes outliers from a dataset, allowing the analysis to continue with a focus on the central tendency of the data and reducing the influence of extreme values.

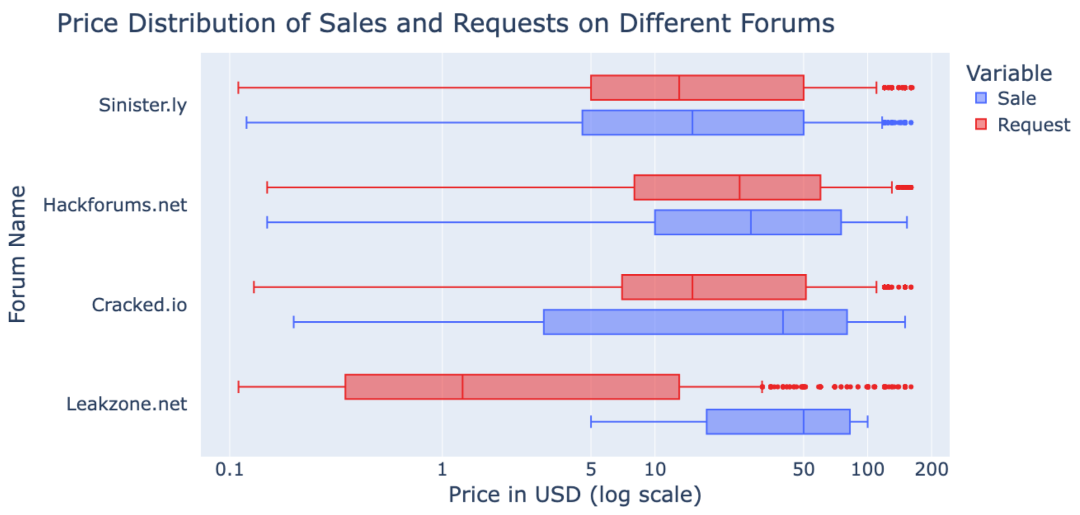


Figure 8.4: Box plot of the distribution of prices for **Sale** and **Request** threads across the investigated forums

The figure clearly displays that the vendors on Leakzone.net request products at much lower prices, than the other sites. From Q1 it is clear that 25% of the data falls below 0.5$, which is usually connected with subscription-based or token-based sales (e.g. price per SMS in SMS scamming). The distribution is also quite skewed, with a wide range between the Q3 at 11$, and the maximum whisker at 35$, followed by a selection of outliers. This indicates that the distribution of **Request** threads is quite skewed towards lower prices. Contrariwise the **Sales** are more skewed toward higher prices, with 25% of the sales being below 20$, and 75% of the data being above 80$. On average products on Leakzone.net are sold for around 50$.

In contrast, the IQR for both the **Sales** and **Request** threads of the other forums are skewed towards the higher end. Hackforums.net and Cracked.io have nearly identical distributions, apart from Cracked.io's Q1 **Sales** being at 4$ opposed to Hackforums.net's 11$. Sinister.ly has a slightly lower distribution of sale prices by about 3-4$ in the lower end compared to Hackforums.net.

Overall, apart from Leakzone.net, it can be concluded that:

- Most prices for **Sales** vary between 3-80$
- Most prices textbfRequests vary between 6-60$

This distinctively indicates that vendors on average expect a higher price for their items than customers believe it is worth. In the following subsection, the distribution of prices will be investigated based on the category, they have been classified as.

8.1.3.1 Prices Per Category: Sales

Taking the same approach as in **Figure 8.4**, the distribution of prices across forums as well as categories can be visualised. The following subsections will look at the price distribution by category for both **Sales** and **Request** threads respectively. In the generation of **Figure 8.5** it should be noted that there may be discrepancies between the distribution of the categories compared to those found in **Figure 8.2**. The NER model, as discussed in **Section 7.4.6**, achieves an average F1-score of 0.73, meaning it cannot be guaranteed that all prices in each of the threads were extracted successfully. Additionally, certain threads classified as a **Sale** or **Request** thread respectively may not mention any price in the thread, but link to external sources instead.

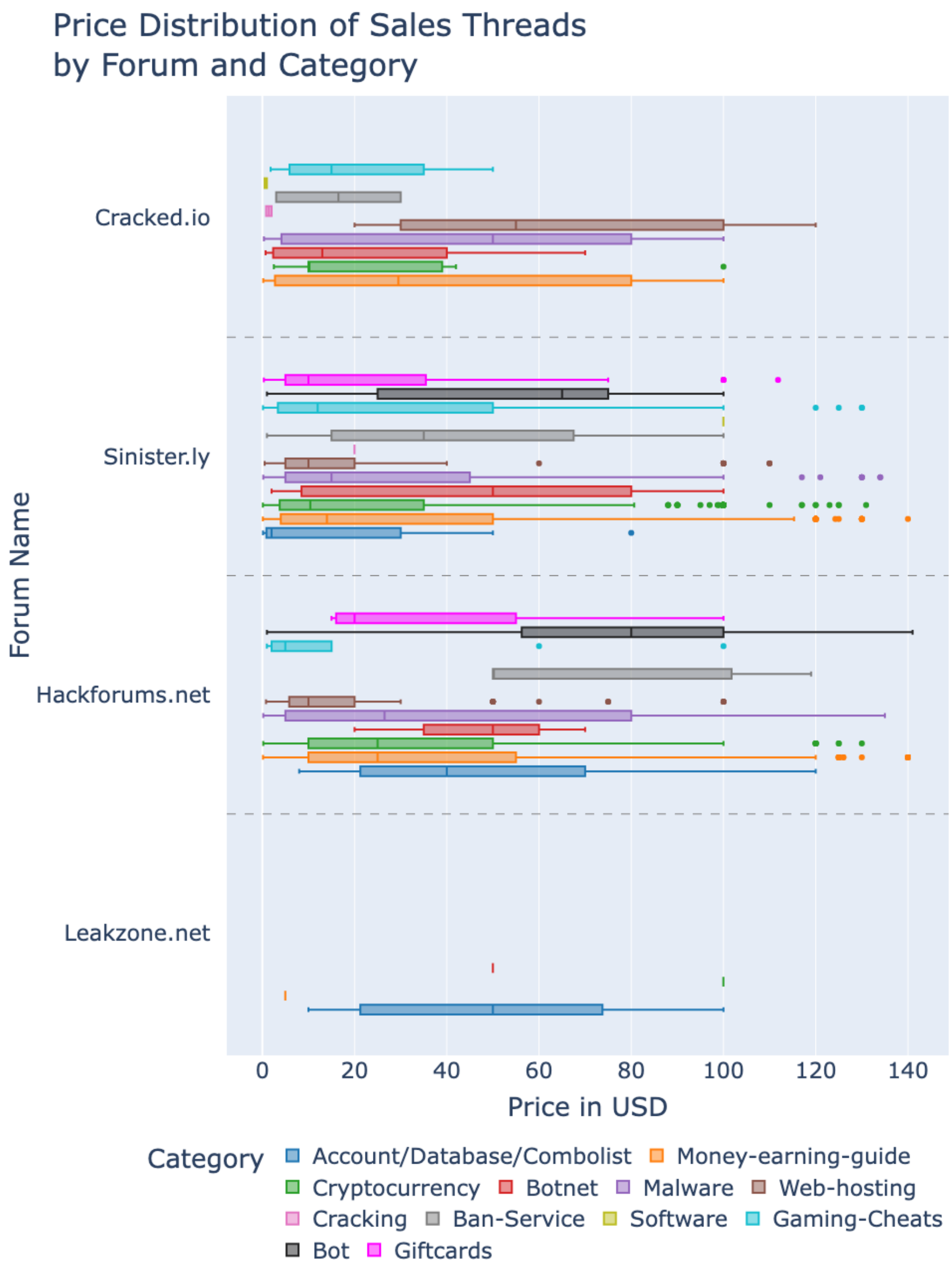


Figure 8.5: Price distribution by forum and category for **Sale** related threads

Starting with Cracked.io, from **Figure 8.5** it can be seen that *Malware* and *Botnet* prices are skewed toward the lower end. For the **Sales** of *Malware*, the IQR is quite broad, with prices ranging from 5$ up to around 80$, with an average of around 50$. *Botnets* are comparatively cheaper ranging between 2-40$ with a much lower median of 13$. Finally, *Cryptocurrency* related **Sales** quite closely resemble the *Botnet* sales, albeit with a slightly higher IQR from 10$ in Q1 to 40$ in Q3. *Gaming cheats* and *Ban-services* displays similar prices, while *Money-earning-guides* more closely resemble the prices found for *Malware* **Sales**. *Cracking* and *Software* is surprisingly low, however, this is most likely to be the cause of the NER model having extracted subscription-based prices (0.05$ per API request for instance). *Web-hosting* on the other hand has much higher prices than any of the categories. This is to be expected though as many of the hosting services involve cloud computing, where you purchase an entire cloud machine on a subscription basis, commonly found at 55-100$ a month.

Progressing to Sinister.ly, *Cryptocurrency* **Sales** are comparable to those found on Cracked.io, though slightly lower. Specifically ranging from 4-35$ and a median of 10$. *Botnets* however are priced much higher than on Cracked.io, with a median of 50$ (an increase of 280%). Contrarily, *Malware* is priced much lower with a median of 15$ (a decrease of 70%). A similar trend is seen in *Web-hosting* (80% decrease), *Money-earning-guides* and *Gaming-cheats* (50% decrease). Sinister.ly however also introduces some new categories, namely *Gift Cards*, *Bot*, and *Account/Database/Combolist*. Each of these, apart from *Bot*, lie in the cheaper end of the spectrum, with the *Gift Cards* category's IQR ranging from 5-35$, and *Account/Database/Combolist* ranging from 1-30$, similar to the pricing for *Cryptocurrency* on Sinister.ly. The reason accounts median lie at the cheap price of 2$ could be that *Account/Database/Combolist* **Sales** often list a price per account, despite selling dumps of several thousands of accounts at a time.

Next on Hackforums.net, the prices for *Web Hosting* and *Money-earning guides* are comparable to the prices found on Sinister.ly, while the prices for *Malware* more closely resembles those found on Cracked.io, though slightly more skewed towards lower prices. *Bots* and *Gift Cards* however are considerably more expensive than on Sinister.ly, with an increase in median of *Gift Cards* by 200% and an increase in the median price of *Bot* by 23%. Additionally, the median price of *Ban-services* is increased by 42% compared to Sinister.ly, and the price of *Cryptocurrency* sales by 92%. *Botnets* have nearly identical median pricing to Sinister.ly, though it is much more consistent, narrowing the IQR to be between 23-60$. Finally, compared to both of the previous sites the median price for *Gaming Cheats* is 55-65% cheaper.

Lastly, for Leakzone.net, the NER classifier was only able to extract prices from the **Sales** threads related to *Account/Database/Combolists*. Nonetheless, the prices it did extract show that **Sales** of *Account/Database/Combolists* on Leakzone.net have a significantly higher price (50$), than on Sinister.ly (2$ median price) and Hackforums.net (40$ median price) combined. Though the cause of the extremely low price on Sinister.ly is as mentioned most likely due to the price being per account vs a whole dump of accounts.

Table 8.3 grants a more comprehensible overview of the insights these price distributions provide, while clearly highlighting, for each category, where the highest and lowest price is found, as well as the average price across sites.

Category	Cheapest Sale on	Average	Most Expensive Sale on
Account/Database/ Combolist	Sinister.ly (2$)	30.66	Leakzone.net (50$)
Malware	Sinister.ly (15$)	30.5$	Cracked.io (50$)
Cryptocurrency	Cracked.io (10$)	15.13$	Hackforums.net (25$)
Money-earning-guide	Sinister.ly (14$)	34.25$	Cracked.io (29.5$)
Cracking	Cracked.io (2$)	2$	Cracked.io (2$)
Bot	Sinister.ly (65$)	72.5$	Hackforums.net (80$)
Web hosting	Sinister.ly (10$)	25$	Cracked.io (55$)
Software	Cracked.io (1$)	1$	Cracked.io (1$)
Gift cards	Sinister.ly (10$)	15$	Hackfroums (20$)
Gaming Cheats	Hackforums.net (5$)	10.66$	Cracked.io (15$)
Botnet	Cracked.io (13$)	37.66$	Hackforums.net (50$)
Ban Service	Cracked.io (16.5$)	33.83$	Hackforums.net (50$)

Table 8.3: Overview of the cheapest, most expensive and average price of sale threads for each category (Prices based on median values)

8.1.3.2 Prices Per Category: Requests

As mentioned in **Section 8.1.3.1** for **Figure 8.5**, it cannot be guaranteed that all categories are present. The same holds true for the categories present in **Figure 8.6**, versus the ones found in **Figure 8.3**. Though when looking at **Figure 8.6**, it is quite clear, that the prices for many more categories have been successfully parsed.

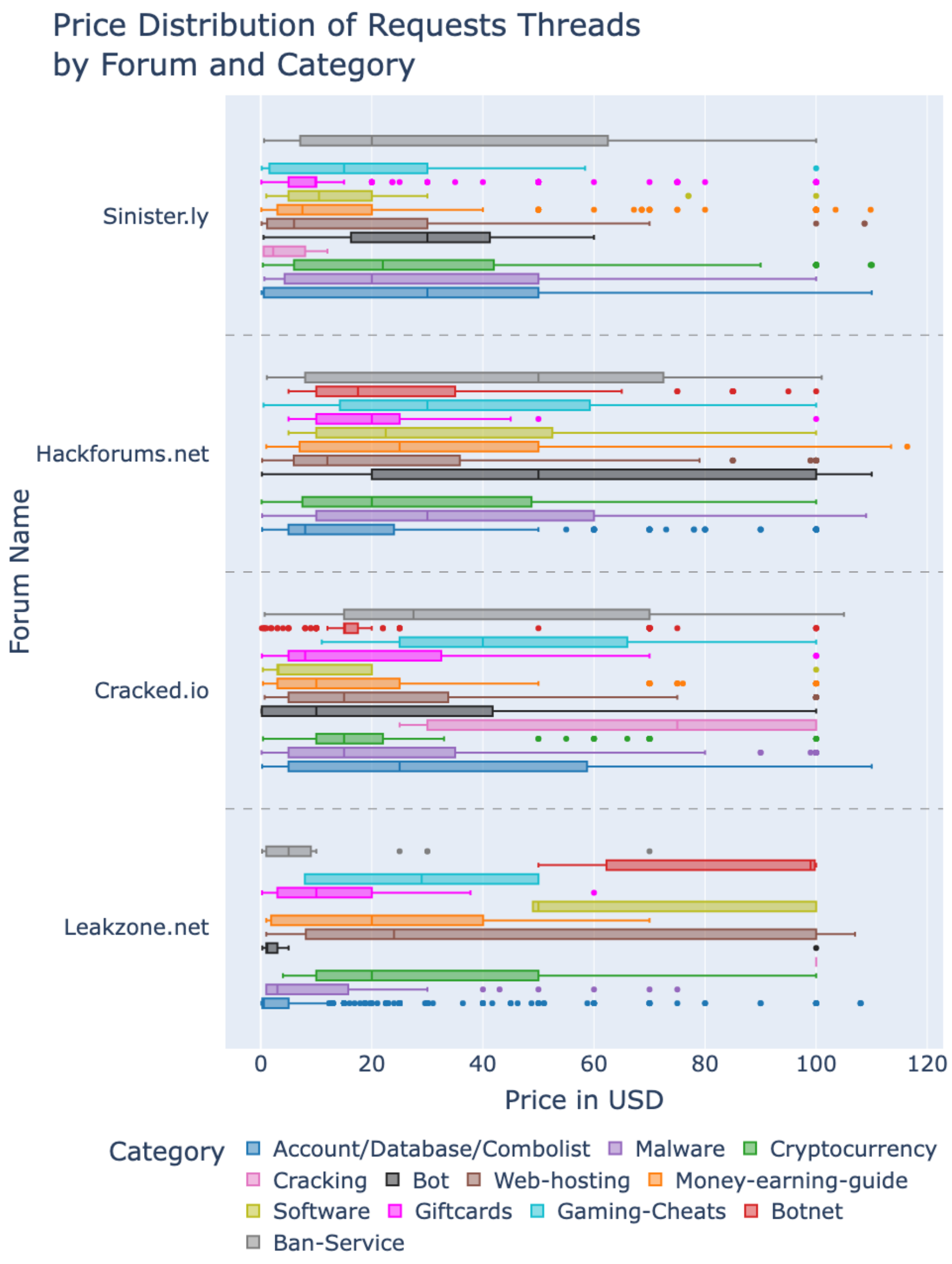


Figure 8.6: Price distribution by forum and category for **Request** related threads

Starting with Sinister.ly, the price of *Account/Database/Combolists* has increased considerably compared to the prices listed in **Sales** threads (previously 2$ vs 30$ for **Request** threads). Though this probably stems from customers requesting entire dumps of databases/accounts as opposed to a single specific account. The price of *Malware* increased marginally (15$ to 20$), and similar trends are found in *Gaming Cheats* (12$

to 15$) and *Cryptocurrency* (10.5$ to 22$). Contrarily, several categories had a reduction in price including *Ban-services* (35$ to 20$), *Money-earning-guides* (14$ to 7.5$), *Web-hosting* (10$ to 6$) and *Bots* (65$ to 30$). Additionally, the *Cracking* and *Software* categories are now present, with a median price of 2$ and 10.5$ respectively. Generalised, the users of the forum are willing to pay more on *Accounts/databases/combolists*, *Malware*, *Gaming Cheats* and *Cryptocurrency*, than what it is being sold for on Sinister.ly. In contrast, they are less willing to pay for *Ban-services*, *Money-earning-guides*, *Web-hosting* and *Bots* than what they are commonly sold for on the forum.

Taking a look at Hackforums.net, the reductions in median pricing are found in *Botnets* (50$ to 17.5$), *Bots* (80$ to 50$), and *Accounts/Databases/Combolists* (40$ to 8$), with minor changes in *Cryptocurrency* (25$ to 20$). Conversely, there are also considerable increases in *Gaming cheats* (5$ to 30$), while the categories *Money-earning-guide*, *Gift cards* and *Ban-services* remain the same. Summarised, the buyers on Hackforums.net are less willing to pay the market price (of the forum) for *Botnets*, *Bots* and *Accounts/Databases/Combolists*, while being willing to pay more for *Gaming cheats*. Additionally, the *Software* category is introduced, with a median price of 22.5$.

Progressing to Cracked.io, more categories are introduced compared to its **Sales** counterpart, namely *Gift Cards*, *Bots* and *Accounts/Databases/Combolists*, all three of which have their distribution skewed to a lower price range. As with the other forums, there has been an increase in prices in a selection of classes. Specifically, *Ban-services* (16.5$ to 27.5$), *Gaming cheats* (12$ to 40$), *Software* (0.75$ to 3$) and *Cracking* (1.5$ to 75$). The other categories only saw minor changes except for a couple of considerable decreases in *Money-earning-guides* (30$ to 10$) and *Web-hosting* (55$ to 15$). Abridged, the customers of Cracked.io, are willing to more for *Ban-services*, *Gaming cheats*, *Software* and *Cracking* while demanding cheaper prices for *Money-earning-guides* and *Web-hosting*.

Concluding with Leakzone.net, more categories than the puny one found for its **Sales** equivalent, are present. Upon further investigation users of the site requests a couple of services at the same price as the competition. Namely *Gaming Cheats* at around 30$, *Cryptocurrency* for around 20$ and G*ift cards* at around 10$. Contrariwise, a few of its services are requested at a substantially higher price than the competition. For instance *Botnets* (400% increase compared to the closest competition) as well as *Software*, with a median of 50$ vs Cracked.io's 3$ and Hackforums.net's 22.5$. Leakzone.net's userbase does however request *Account/Database/Combolist*, *Bots* and *Malware* at the cheapest prices of 0.55$, 1$ and 3$ respectively, when comparing to the other forums. In summary Leakzone.net requests *Account/Database/Combolists* at much lower prices than what is offered on the forum. Additionally, users on the site request *Botnets* and *Software* at much higher prices than the competition, while requesting *Bots* and *Malware* at much lower prices than the competition.

As with the sales-related threads, **Table 8.4** grants a more generalised overview of where categories are requested at the highest and lowest prices, as well as what price they are requested at on average.

Category	Cheapest Request On	Average	Most Expensive Request On
Account/Database/ Combolist	Leakzone.net (0.55$)	15.89$	Sinister.ly (30$)
Malware	Leakzone.net (3$)	17$	Hackforums.net (30$)
Cryptocurrency	Cracked.io (15$)	19.25$	Sinister.ly (22$)
Money-earning-guide	Sinister.ly (7.5$)	15.63$	Hackforums.net (25$)
Cracking	Sinister.ly (2.2$)	38.6$	Cracked.io (75$)
Bot	Leakzone.net (1$)	22.75$	Hackforums.net (50$)
Web hosting	Sinister.ly (6$)	14.25$	Leakzone.net (24$)
Software	Cracked.io (3$)	21.5$	Leakzone.net (50$)
Giftcards	Cracked.io (8$)	12$	Hackforums.net (20$)
Gaming Cheats	Sinister.ly (15$)	28.5$	Cracked.io (40$)
Botnet	Cracked.io (15$)	18.5$	Leakzone.net (99$)
Ban Service	Leakzone.net (5$)	25.63$	Hackforums.net (50$)

Table 8.4: Overview of the cheapest, most expensive and average price of request threads for each category (Prices based on median values)

Comparing **Table 8.3** and **Table 8.4**, combined with the observations made in **Section 8.1.3.1** as well as this section the following list of insights become apparent:

- *Accounts/Databases/Combolists*, *Malware*, *Money-earning-guides*, *Bot*, *Web-hosting*, *Gift cards*, *Botnets* and *Ban-services* are on average requested at a lower price, than what it is offered for.
- Conversely *Cryptocurrency*, *Cracking* and *Software* are on average requested at a higher price, than what they are offered for.
- Sinister.ly offers the **lowest** price across the highest number of categories
- Cracked.io offers the **highest** price across the highest number of categories
- The cheapest requests across the highest number of categories is made on Leakzone.net and Cracked.io
- The most expensive requests across the highest number of categories is made on Hackforums.net
- Users on Sinister.ly and Hackforums.net make requests at higher prices than the sales price on the forum across the most categories
- Users on Sinister.ly, cracked.io and Leakzone.net make requests at lower prices than the sales price on the forum across the most categories.
- Users on Cracked.io, make the request with the biggest difference from the sales price on the forum (55$ to 15$).

- The biggest increase from sales price versus requested price is found in the **Cracking** category on Cracked.io (1.5$ to 75$)
- The average sales price across all categories ranges between 1-72.5, with a mean of 25.69$
- The average request price across all categories ranges from 12-38.6 with a mean of 20.8$.

Given this list of insights, it is evident, that users across the sites on average request products and services, at a lower price, than what they are advertised for. Additionally, users on Hackforums.net are more inclined to request products or services at a higher price, than what it is actually worth. Contrariwise, users on Cracked.io and Leakzone.net are more predisposed to request products or services at a lower price, than what they are actually worth. Users on Sinister.ly are just as likely to request a service or product at a higher price, than the sales price on the forum, as they are to request it at a lower price.

8.1.4 Popular Thread Topics

To grant a better understanding of what the main topics of interest for each of the categories found in the threads of the forums are, *word clouds* were utilised. In this case, word clouds were created based on the titles of threads, with stopwords, special characters etc. filtered out. To easily generate word clouds from the data, the Python package `wordclouds`[106] was utilised. In the following subsections, the word clouds for each of the categories in the **Sale** and **Request** threads respectively will be investigated. Note that the categories have been broken into two groups for increased resolution of the word clouds.

8.1.4.1 Thread Topics by Category: Sales

Starting with the **Sales** threads, looking at **Figure 8.7**, the words associated with the *Account/Database/Combolist* category, unsurprisingly usually consist of `Account`, and their associated type (e.g. `Amazon`, `Skype` etc.). More unexpectedly, the word `Bump` is also quite prevalent, which in this case, simply refers to when a vendor tries to bring their threads back to the front page by `Bumping` it up, through replying to it.

Next, in the *Money-earning-guide* category, the words `Method` and `Guide` are naturally the most frequent. However, the other common words associated with this class seemingly have little to do with money-making guides (e.g. `Gift Card`, `Order ID`). Though when browsing the *Money-earning-guide* related threads, it does become evident that the selection of methods employed to `Make money fast` are quite broad, ranging from automatic gift card generation to refunding methods (given and `Order ID`).

Most shockingly, for the *Cryptocurrency* category, there does not seem to be a single crypto-related word. Everything concerns contracts, guides or variations of account and purchasing methods. As with the previous category, after browsing the *Cryptocurrency* related sales threads on several of the forums, there is a clear trend of these threads relating to crypto matters outside of the forum. For instance, referrals to crypto exchanges or guides to mining crypto, which explains the prevalence of the words such as `Guide` and `Contract`.

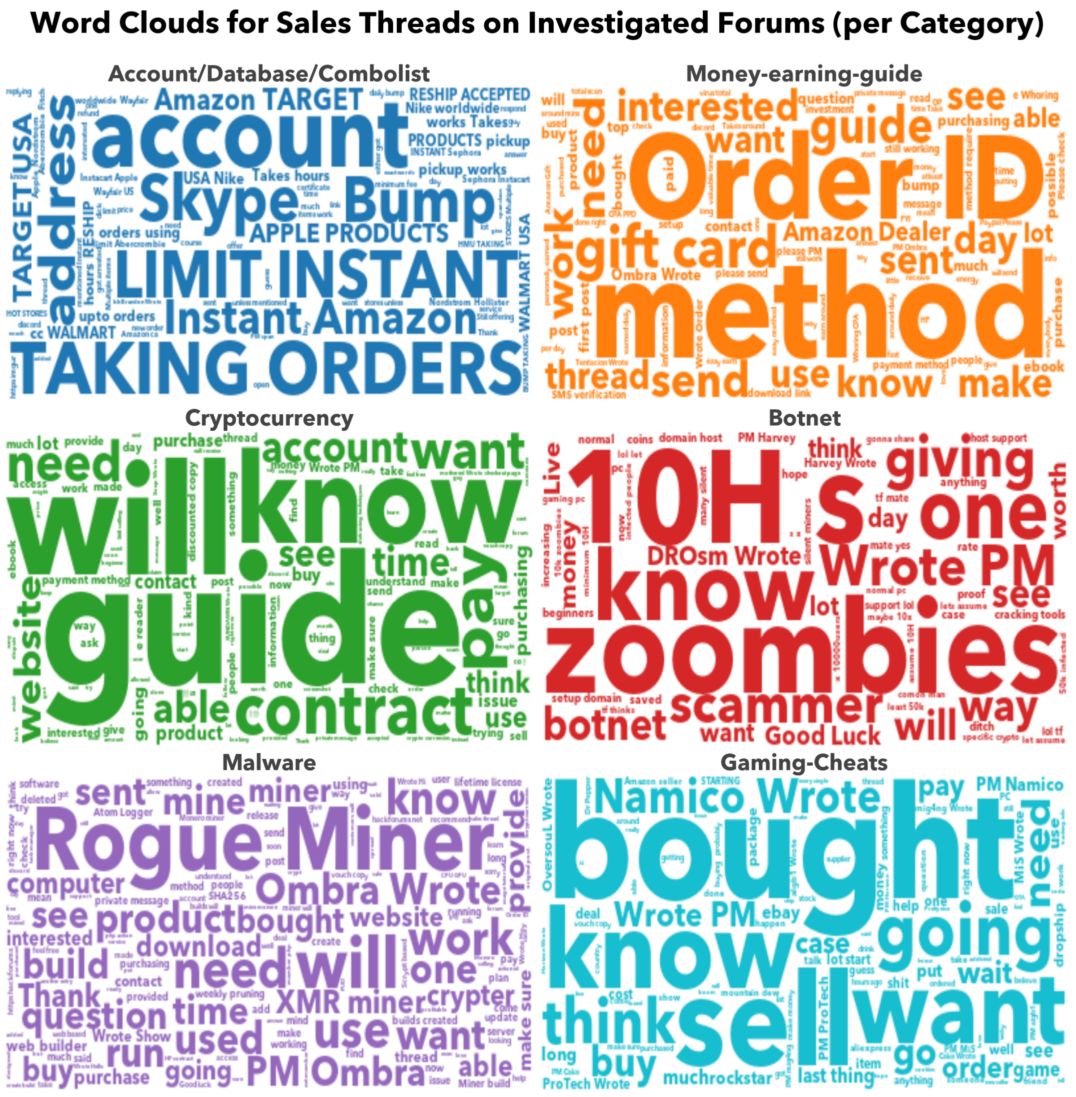


Figure 8.7: Word clouds for **Sales** threads in the categories: *Account/Database/Combolist*, *Money-earning-guide*, *Cryptocurrency*, *Botnet*, *Malware* and *Gaming Cheats*

The *Botnet* word cloud is much more predictable, with words such as `Zoombies` and `Botnet` being quite frequent. A `Zoombie` in this context refers to a so-called *Bot* in a botnet. Additionally, there appears to be a fair bit of dissatisfaction from the users browsing in this category, indicated by the sizeable word `Scammer`. The words for the *Malware* category are likewise quite anticipated. Words such as `Rogue` and `Miner` are usually used in the context of installing malware undetected and installing cryptocurrency mining software, respectively. Additionally, a considerable amount of words related to building, buying and running are present, indicating that this category is active in both sales, implementation and utilisation of malware. Finally, the *Gaming-Cheats* category, like the *Cryptocurrency* category, does not provide words indicative of its name. Threads in this category are apparently much more focused on the procurement of cheats games as given by the prevalence of purchasing-related words such as `Bought` and `Sell`. One thing to note is the common occurrence of `Wrote PM` clearly indicating that sales are made by directly writing to the vendor bringing the purchase off the forum.

Progressing to the *Web-hosting* category in **Figure 8.8** barely any words related to web hosting can be found. Though as with the *Gaming-Cheats* category, this may relate to the high occurrence of `Wrote PM`, meaning the deals and discussions for web hosting take place privately between individuals. Similarly in the *Cracking* category, seemingly no words related to cracking are present. Though given their overall small prevalence in the dataset as given by **Figure 8.2** and **Figure 8.3**, respectively, this may simply be because there are barely any threads to base the word cloud off of.

The *Ban-Service* category seems to be much more focused on reputability from vendors, denoted by words such as `Vouch`, `Guy` and `Rep`. Similarly to *Gaming-Cheats* and *Web-hosting* these services seem to also frequently occur through private messaging given the popularity of `Wrote PM`. The same holds true for the *Software* category, however, here there are a fair few discernible software-related terms such as `Settings` and `Config`. Contrarily for the *Bot* category, there are no words fitting the category name. The focus in this category is much more directed towards seemingly unhappy clients, designated by words such as `Refund`, `Dispute` and `Contract`. Finally the *Giftcards* category obviously mainly contains the word `Gift Card`, along with similar trend of `Wrote PM`.

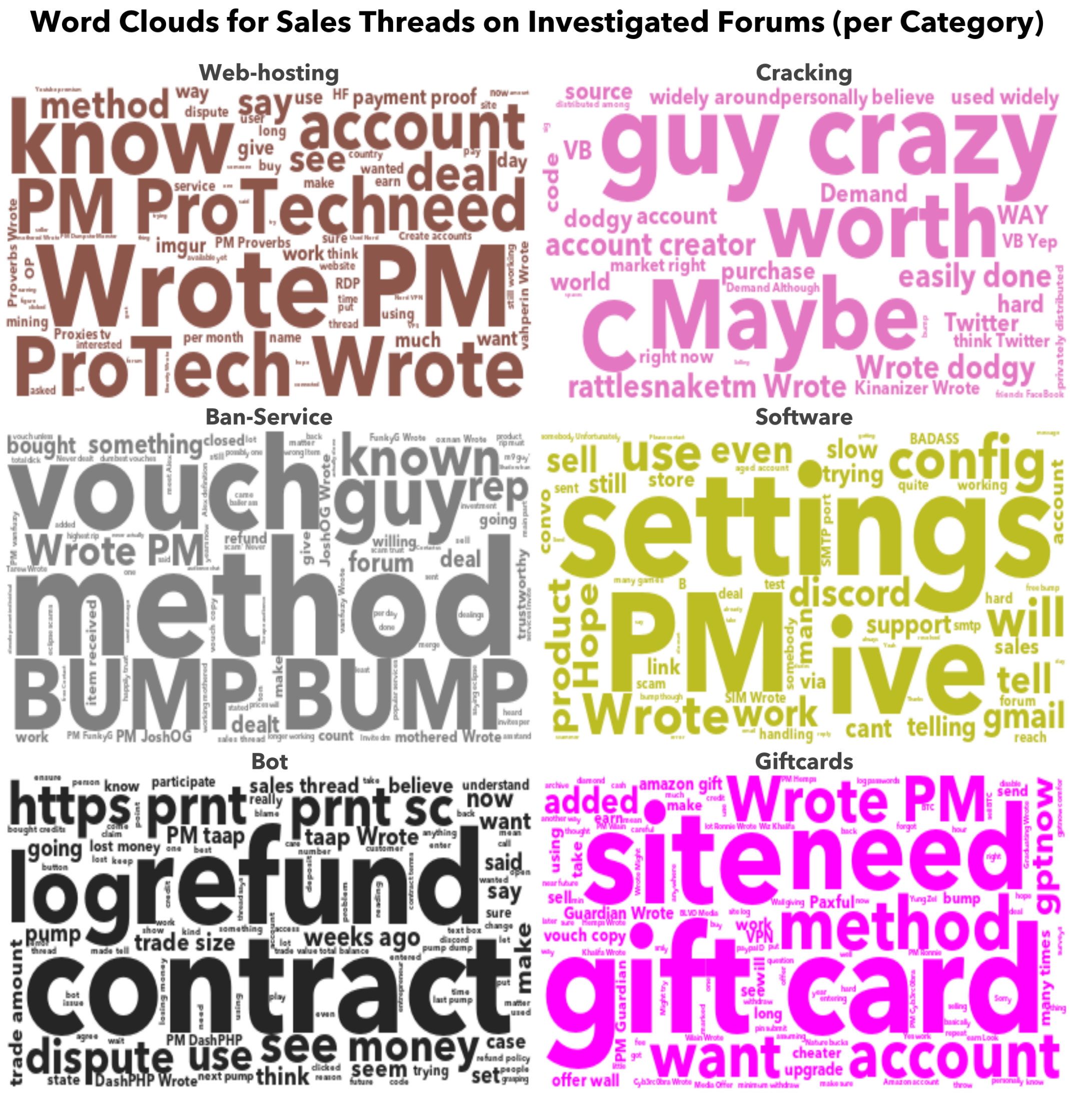


Figure 8.8: Word clouds for **Sales** threads in the categories: *Web-hosting*, *Cracking*, *Ban-Service*, *Software*, *Bot*, *Gift cards*

8.1.4.2 Thread Topics by Category: Requests

Already from the *Account/Database/Combolist* category's word cloud, in **Figure 8.9**, it becomes apparent that there is a lot of difference between **Sales** threads and **Request** threads, despite the overall category being the same. Specifically, where the **Sales** related threads for this category were more focused on words such as `Account` and different account type names, the **Request** related threads for this category are much more focused on `Tokens`. In this context, tokens refer to access tokens to various accounts. Additionally, `tokinaist` and `sellix.io` are noticeable, which are popular domains for purchasing social media accounts.[107]

The *Malware* category, similarly deviates from its **Sales** equivalent, with `Order ID` being the most frequent term, followed by words like `Need` and `Discord`.[108] Comparable to the **Sales** threads, the *Cryptocurrency* category has no discernible crypto-related words. Contrarily, however, it does have a fair amount of market-related words such as `Dipping` and `Limit`. This could indicate that the vendors requesting cryptocurrency-related products often mention the current state of the market in their threads.

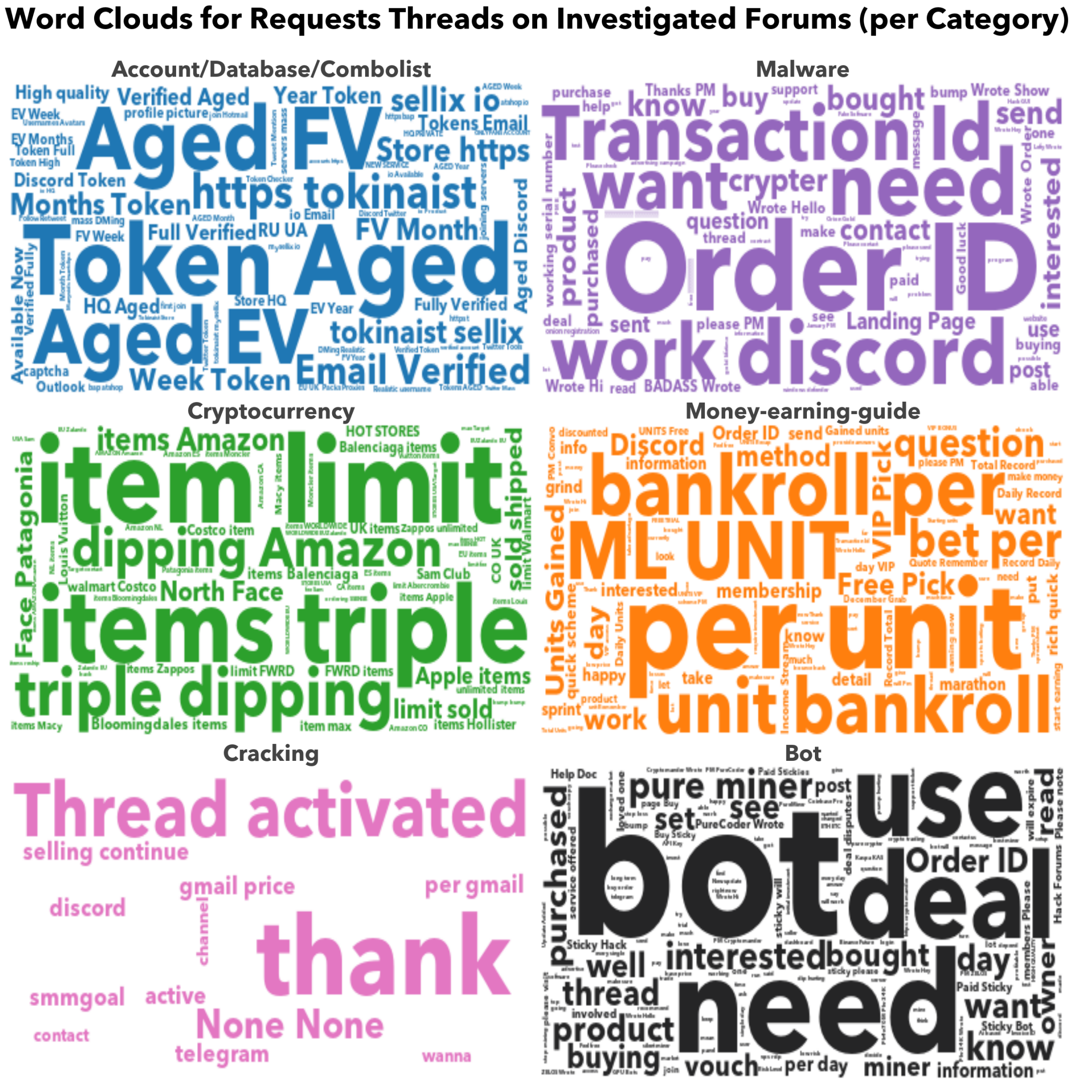


Figure 8.9: Word clouds for **Request** threads in the categories: *Account/Database/-Combolist*, *Money-earning-guide*, *Cryptocurrency*, *Botnet*, *Malware* and *Gaming-Cheats*

In contrast, the *Money-earning-guide* category for **Request** threads has quite a different suite of words compared to the **Sales** threads. Specifically, `Bankroll` and `Per Unit` are quite common, referring to a flow of money and money to be made per device/investment etc., respectively. As for the **Sales** threads there are barely any *Cracking* related threads in the dataset, which in **Figure 8.9** is confirmed by the lack of words, none of them being particularly indicative to cracking. Finally, the *Bot* related threads in the corpus of **Request** related threads are much more unsurprising, with the main word simply being `Bot`. Additionally, there seem to be fewer words indicating a negative relation to vendors, compared to the **Sales** related *Bot* threads.

Moving on to **Figure 8.10**, the *Web-hosting* category is entirely predictable, with nearly every single prominent word relating to `Proxies`, i.e. locations to pass web traffic through to avoid geo-restrictions or traceability. *Software* on the other hand, unexpectedly barely has any recognisable software names. Instead, simple words such as `Auto Bumped` are seen, signifying that software-related threads are less active, seeing as the forums automatic thread refresher `Auto Bumped` the threads. However, there are still a few software-related terms such as `Code` and `Xupgrader`, though they are not the most frequent.

As opposed to the *Money-earning-guide*, the *Gift cards* category shockingly does not host the word `Giftcards`. It does, however, contain a lot of brand names such as `Ralph Lauren`, assumedly to the brand the gift card is for. As opposed to the **Sales** threads, the words in the *Gaming-cheats* category in the **Request** threads are actually quite indicative of its title, with some of the most prevalent words simply being `Game` and `Cheat`. Additionally, there is a fair bit of `Bumps` meaning there is likely not much activity on the threads where a user requests game cheats. The *Botnet* category likewise contains a lot of `Bumps` and as opposed to its **Sales** counterpart it is not at all indicative of its title. The **Request** related *Botnet* threads appear to be much more focused on the botnet `Service`. This is quite expected as a user would usually request a botnet `Service`, as opposed to buying the botnet itself. Contrariwise to the *Botnet* category the *Ban-service* category contains a lot of very telling words. Particularly the things to ban, i.e. `Instagram`, `TikTok` etc, as well as the form it is requested in (e.g. `Package` or `Service`).

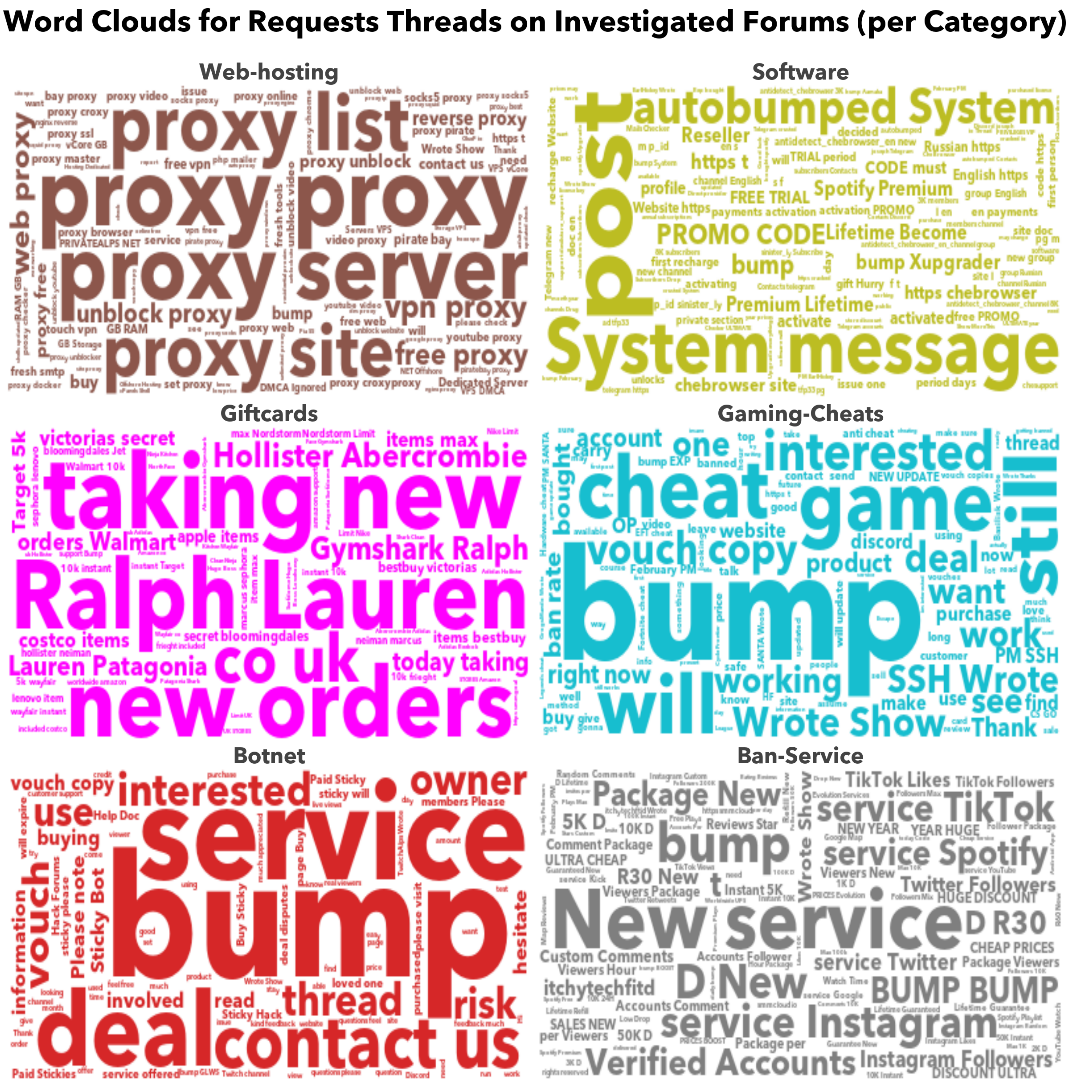


Figure 8.10: Word clouds for **Request** threads in the categories: *Web-hosting*, *Cracking*, *Ban-Service*, *Software*, *Bot*, *Gift cards*

To summarise the comprehensive exploration of the topics of each of the categories across **Sales** and **Request** related threads given in the preceding and current subsections, the following insight was gathered:

- The *Web-hosting* category was more easily recognised in the **Request** threads, mainly consisting of the word `Proxy`
- *Botnets* were more directly discussed in **Sales** threads, directly mentioning `Botnet` and `Zoombies`
- No words in the *Cracking* category were indicative of its content, due to its low prevalence in the dataset

- Users in **Sales** related threads in the *Bot* category, seemingly display dissatisfaction with the services indicated by words like `Refund`
- The *Gift card* category was easily recognisable in both **Sales** and **Request** related threads through brand names and the word `Gift Card`.
- The **Sales** threads *Software* category was more identifiable, through words such as `Settings` and `Config`, compared to the **Request** threads `System`.
- The **Request** threads *Gaming-cheats* category naturally contained frequent mentions of `Game` and `Cheat`, while its **Sales** counterpart mostly featured words such as `Bought` and `Sell`
- The *Ban-service* category in the **Sales** threads were commonly focused on credibility, while the **Request** related threads were focused on specific account types.
- Users more commonly requested tokens in the *Account/Database/Combolist* category compared to **Sales** related threads that more specifically focus on the type of account
- The *Money-Earning-Guides* often concerned the use of `Order IDs` (refunding methods)
- The *Cryptocurrency* category was not seemingly identifiable in either of the thread types
- The **Request** threads of the *Malware category*, were more focused on purchasing, than the type of malware requested as opposed to the **Sales** related threads that mostly discuss the technique used in the malware.

Collectively the **Sales** threads naturally differ from the **Request** given that how a product is advertised is usually quite different, than what a user might request. Given that these forums are hacking forums, there is a surprisingly small amount of slang prevalent in any of the categories. However, this could be from the fact that when requesting and advertising products, more professional/clear language is often used, compared to the language used in discussions.

8.1.4.3 Overall Topics by Forum

Investigating what topics are usually discussed in each of the various thread categories, while interesting in itself, regrettably does not depict what the main overall topics are per forum. **Figure 8.2** and **Figure 8.3**, provide a general overview of the distribution of topics per site for the **Sale** and **Request** thread groups. However to paint a clear picture of the difference in words used for various categories across forums and thread groups, one has to look at the words used on the forums individually. Consequently, the word clouds presented in the previous sections are now generated on a forum basis, on all of their threads (including discussions), across all their categories. The following subsections will provide an investigation into the overall topics for each of the investigated forums.

Sinister.ly
As discovered in **Section 8.1.3.1** and **Section 8.1.3.2**, Sinister.ly contained threads within all of the categories specified in **Section 6.3**. Consequently, when looking at **Figure 8.11**, one would expect a quite varied distribution of words.

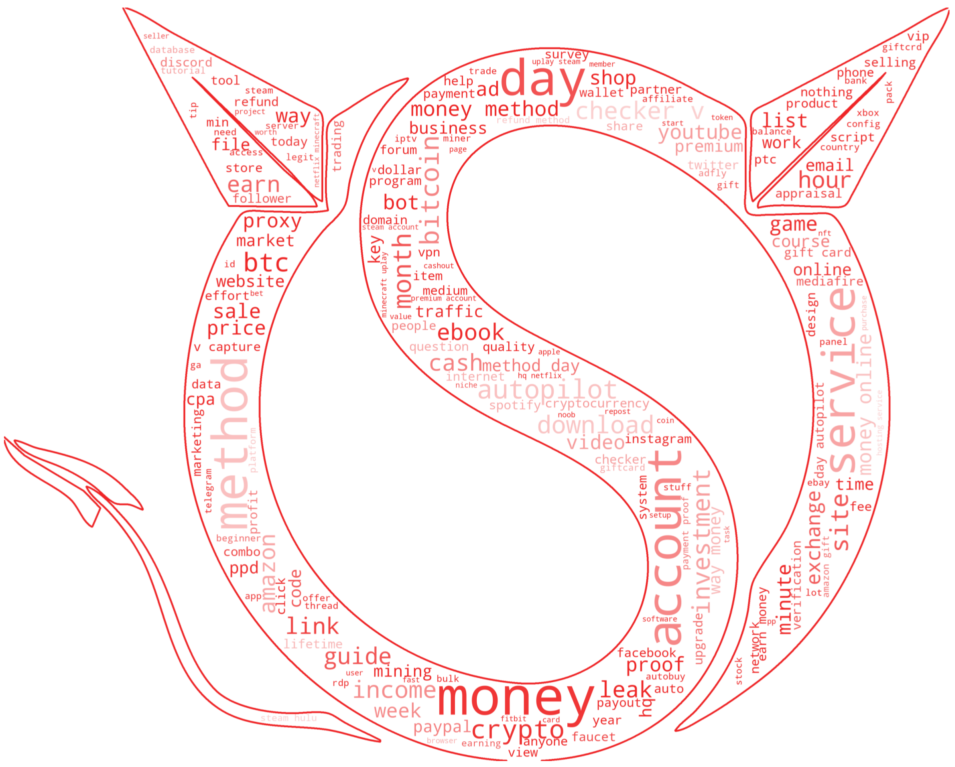


Figure 8.11: Word cloud of threads for Sinister.ly's *Marketplace* section

Figure 8.11 clearly displays that the more commonly discussed topics relate to `Money`, `Accounts`, `Services` or `Methods`. Combined with the knowledge gathered from browsing of the site, these keywords are often seen together in forms such as `Money making methods`, or `Account selling services` as well as `Ban services`. Conversely, terms such as `Database` (upper left horn) or `Script` (middle right horn) are less frequent. These terms usually appear in connection with database leaks or malware-related activities respectively, which one would expect to be frequent on a site like Sinister.ly. The minimal presence of `Database` could stem from the fact that the names of accounts are quite varied. The same holds true for `Scripts`, where their name might frequently be used (e.g. K2LL33D).

A couple of references to *Cryptocurrency* such as `BTC` (below left horn) and `Crypto` (bottom middle), are fairly prevalent alongside *Web-Hosting* terms such as `Proxy`, `Website`. Apart from these the other categories of **Section 6.3**, are quite spread out, with the seemingly only non-prevalent categories being *Gaming Cheats* and *Botnets* (not to be confused with the *Bot* category). Combining these observations with those of **Figure 8.2** and **Figure 8.3** respectively confirms that the threads of Sinister.ly mostly fall within the categories, *Money-earning-guides*, *Account/Database/Combolists* and *Ban-services*.

Additionally, each of these categories mostly seems to be identified by words like `Money`, `Account` and `Service` respectively.

Hackforums.net

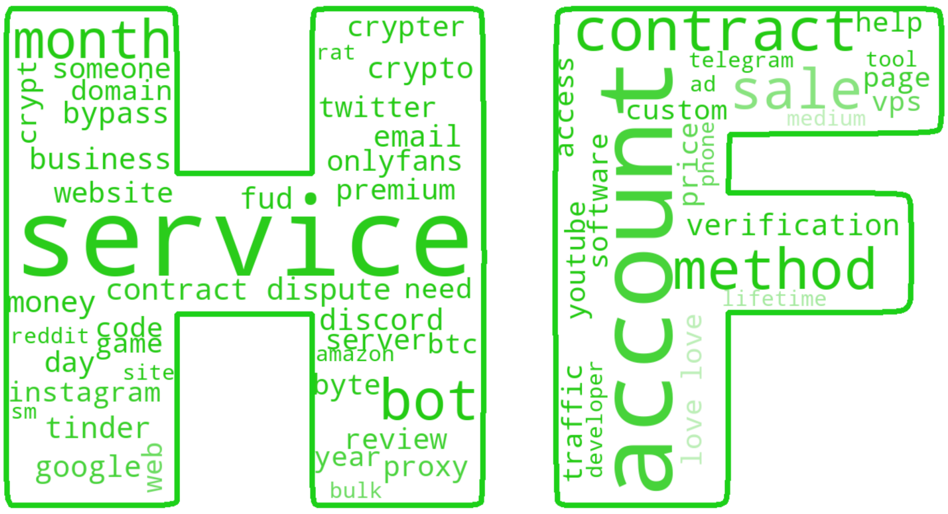


Figure 8.12: Word cloud of threads for Hackforums.net's *Marketplace* section

As with Sinister.ly, Hackforums.net's threads mainly seem to discuss `Services`, `Accounts` and `Methods` as depicted in **Figure 8.12**. However, a slight difference is seen in the amount of hacking terms such as `Domain Bypass`, `Bot` and `Access`. This could indicate that Hackforums.net is more concerned with hacking, while Sinister.ly is more concerned with account dumps and money-making methods. Additionally, a few words related to *Cryptocurrency* are present, such as `Crypt` and `Crypto`. This is confirmed by the prevalence of threads categorised as `Cryptocurreny` and `Malware` in **Figure 8.2** and **Figure 8.3** respectively.

Cracked.io

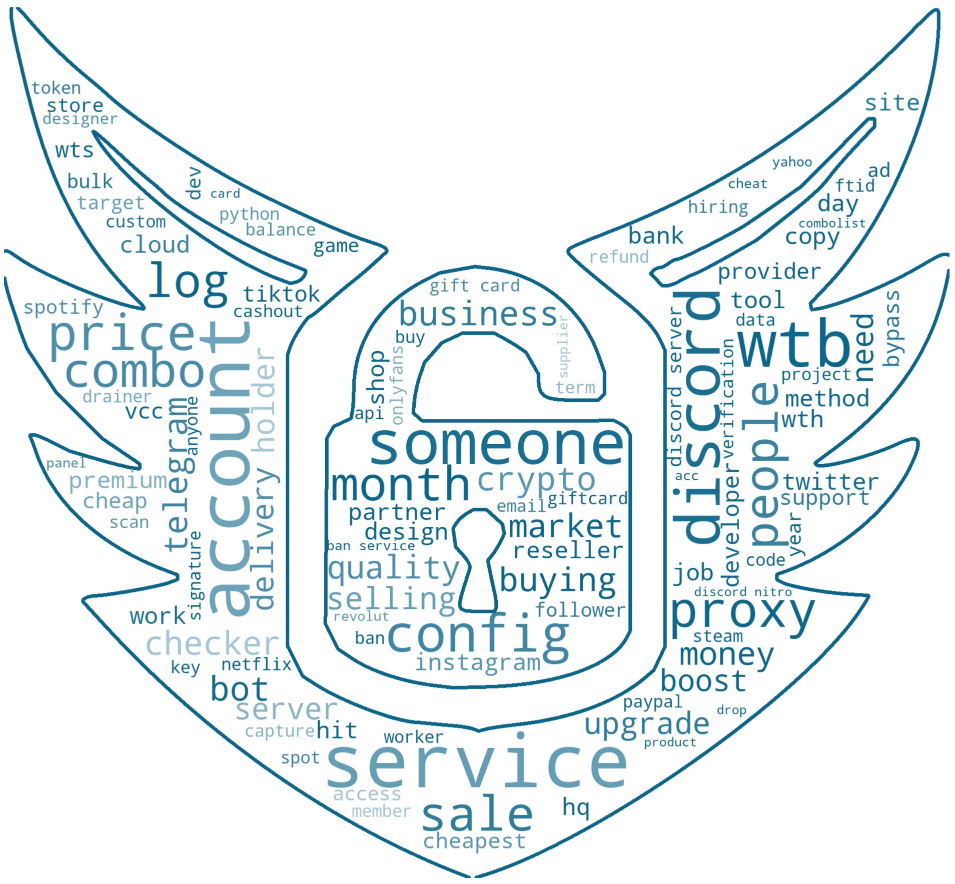


Figure 8.13: Word cloud of threads for Cracked.io's *Marketplace* section

Moving on the Cracked.io, it similarly has its main focus on `Accounts` and `Services` as portrayed in **Figure 8.13**. As Cracked.io is a place for leaks,[61] it is not surprising to see words such as `Combo` or `Log` (upper left wing) occur frequently, `Combo` here meaning a so-called *Combolist*. Interestingly there are a lot of words related to purchasing (`Buying`, `WTB` (want to buy), `Selling`). Combining this with the skewed distribution of **Table 8.2**, these words should in fact not be connected to sales, but requests, which is confirmed by Cracked.io's major presence in **Figure 8.3**

Leakzone.net
Finally, Leakzone.net seems to be no exception as the most frequent terms occurring on its marketplace forum are `Account` and `Service` as illustrated in **Figure 8.14**.

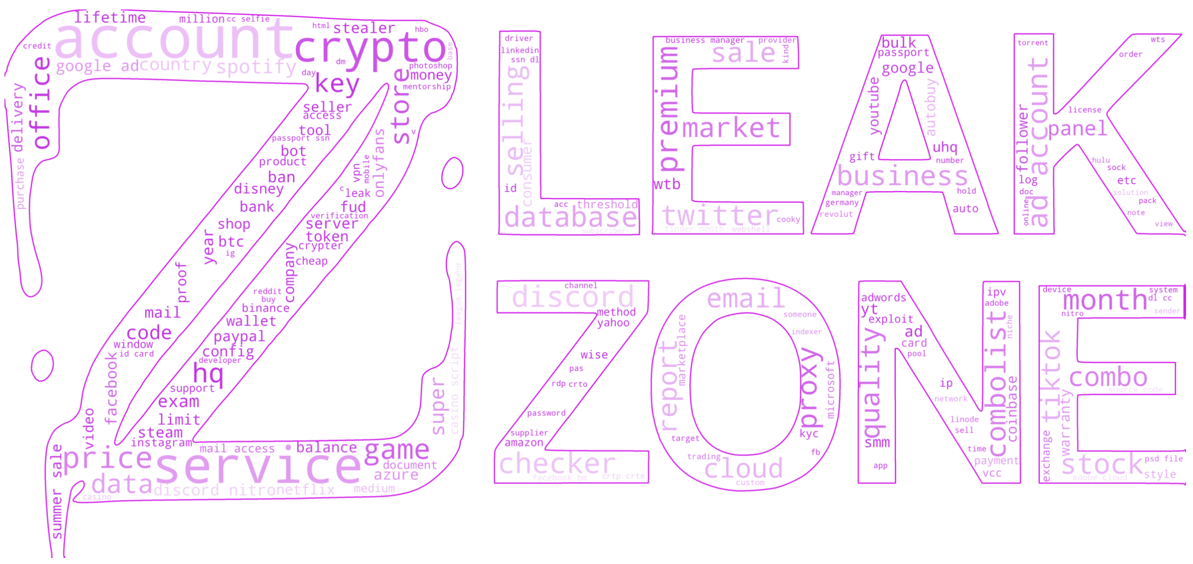


Figure 8.14: Word cloud of threads for Leakzone.net's *Marketplace* section

Apart from this, Leakzone.net's most popular terms seem to be a combination of some of the other sites, having both several purchasing-related terms (`Market`, `Price`), leaking related terms (`Database`, `Combolist`) and hacking related terms (`Stealer`, `Key`) occurring regularly. Where it sets itself apart is in that `Crypto` related subjects seem to be a common topic of its marketplace threads.

In summary from the word clouds, it can be concluded that topics related to `Accounts` and `Services` were the most frequent across the board. This paints an entirely different picture than that given in **Figure 8.2** and **Figure 8.3** respectively, as neither of these figures, indicate *Accounts/Database/Services* or *Ban-services* as being the main topics for any of the sites except for Leakzone.net. This may be an indication of misclassification by the intent classifier in the NLP pipeline. However, it may also be that the more prevalent words such as `Account`, `Service` may often be used in the context of several of the different categories (e.g. `Bank Account` in Money-earning-guide).

8.1.5 External Links

As the investigated forums are on the clear web, many of the transactions don't occur directly on the site for the sake of anonymity as indicated by the prevalence of words such as `Wrote PM` as stated in **Section 8.1.4.1**. Therefore many of the threads concerning **Sales** or **Requests** link to external sites such as Telegram[109], Discord[108] or .onion addresses. By the simple use of regular expressions as explained in **Section 2.1.2.1**, the links found in the forums threads can be extracted and displayed on a Sankey diagram as portrayed in **Figure 8.15**.

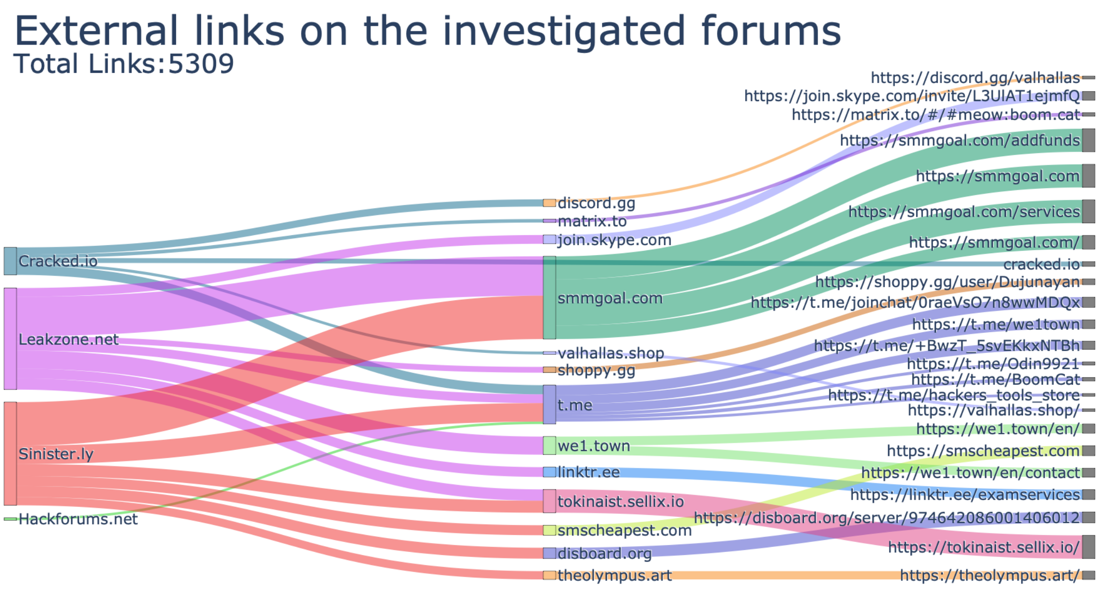


Figure 8.15: Sankey diagram over top 10 external links in the investigated illicit forums threads

In the Sankey diagram the width of the connecting bars displays the occurrence of a given link. The figure clearly illustrates that all four sites have a significant number of external links pointing to Telegram[109] (`t.me`). This observation is not unexpected, considering that Telegram, an anonymous messaging platform, can provide anonymity for the users on these forums. Additionally, Sinister.ly and Leakzone.net exhibit a considerable portion of references to `smmgoal.com`, a social media management platform. `smmgoal.com` offers likes, views etc., for various social media platforms, which, while not sounding inherently illegal, can often be obtained via dubious methods, that may be in violation of the social media platforms terms of service. Sinister.ly and Leakzone.net also commonly mention `tokinaist.sellix.io` and `we1.town`, which also provide social media services, however in this case, sales of social media accounts and account creation software respectively.

Analysing the distribution of links, it becomes apparent that a significantly higher percentage of links are posted on Sinister.ly (43% of total) and Leakzone.net (43% of total), than on Cracked.io (12% of total) and Hackforums.net (1% of total) combined. In the case of Sinister.ly, this could be attributed to a larger number of scraped threads compared to the other sites, as discussed in **Section 8.1.1**. For leakzone, as it is a smaller site than the others, as portrayed in **Table 8.1**, its small userbase is primarily dedicated to redirecting users to other sites for various purposes, such as external sales. The scarcity of links on Hackforums.net is quite surprising, as no obvious explanation for its minimal contribution of external links can be determined. Finally, regarding Cracked.io, since many of the cracking/leaking services offered are meant to be shared anonymously, it is reasonable to expect a significant number of links to Telegram and Discord[108] (`discord.gg`).

Examining the `.onion` (TOR) links mentioned in threads, as depicted in **Figure 8.16**, reveals a similar narrative. Of the forums investigated, only Sinister.ly featured references to `.onion` addresses, with a significant majority (75%) of these links directing to the TOR address associated with the **Just Kill** marketplace. **Just Kill** is a Russian marketplace offering a wide range of hacking-related services such as SMS flooding and email flooding. Due to the ephemeral nature of `.onion` links, there is no guarantee that all of these links lead to active destinations. After testing each of the links, only the **Just Kill** forum returned a response.

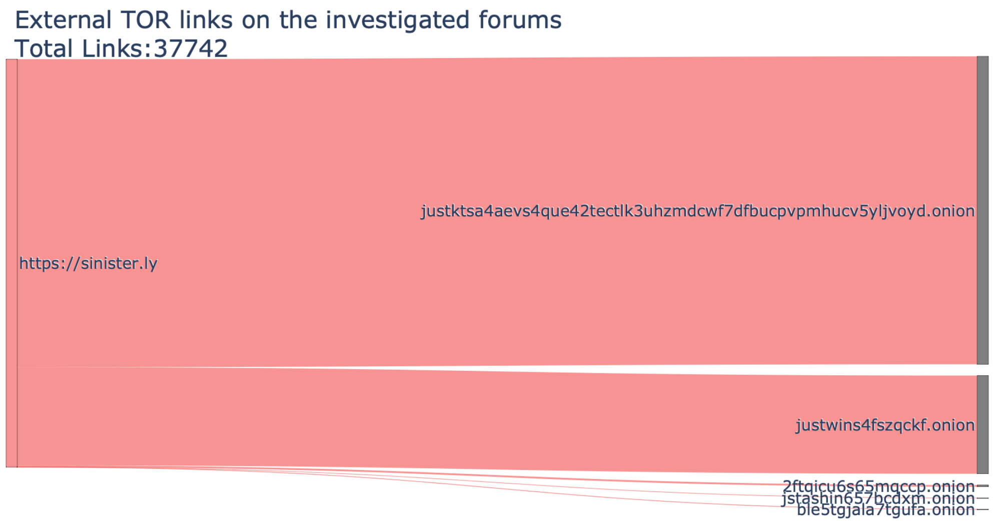


Figure 8.16: Sankey diagram over top 10 external TOR links from the investigated illicit forums

Given the prevalence of the aforementioned Russian hacking marketplace as the most commonly linked website on Sinister.ly, it could suggest that a considerable portion of the illicit activities carried out on the platform are associated with Russian hackers. In summary, this section discovered:

- There were considerably more external links on Sinister.ly and Leakzone.net than Cracked.io and Hackforums.net combined
- Hackforums.net only linked to Telegram
- Sinister.ly and Leakzone.net both mainly link to `smmgoal.com`, a social media management platform
- The most popular external sites listed are `smmgoal.com`, `t.me` and `tokinaist.mysellix.io`
- From the collected dataset only Sinister.ly hosted `.onion` links, the only active one leading to the Russian hacking marketplace **Just Kill**

8.2 User Analysis

Each forum has different ranks for its users depending on their popularity, number of posts or other similar metrics. In some cases, the ranks can be directly purchased for a relatively small fee (usually 20-60€). In others, however, they are earned through reputation or certain actions. Users that have a higher rank generally have a higher presence (number of posts written) on the forum, which is rather intuitive as they have either invested money or obtained credibility on the forum to gain their high rank. In the following subsections, the distribution of user ranks for Sinister.ly is investigated. For the other scraped forums, regrettably, only 1-2% of their total user base was scraped, meaning any subsequent analysis on these, would not be indicative of their true nature.

8.2.1 Users: Sinister.ly

The user ranks Sinister.ly fall within one of the following categories:

- Normal ranks
  - Senior Member
  - Junior Member
  - Member
  - Newbie
  - Registered
  - Account not Activated
  - Spiders/bots
  - Banned
- Premium ranks
  - Administrator
  - Super Moderator
  - Moderator
  - Registered (Deceit)
  - Registered (Platinum)
  - Registered (Diamond)
  - Registered (Gold)
  - Registered (Silver)
  - Registered (Bronze)
  - Registered (Legends)
  - Otaku

Hereof only the Registered Bronze, Silver, Gold, Diamond, and Platinum are purchasable ranks. As displayed on **Figure 8.17** the distribution of these ranks is quite skewed with just over 0.5% of the users, being premium users and the majority of these (54%) being the lower tier premium users (Bronze).

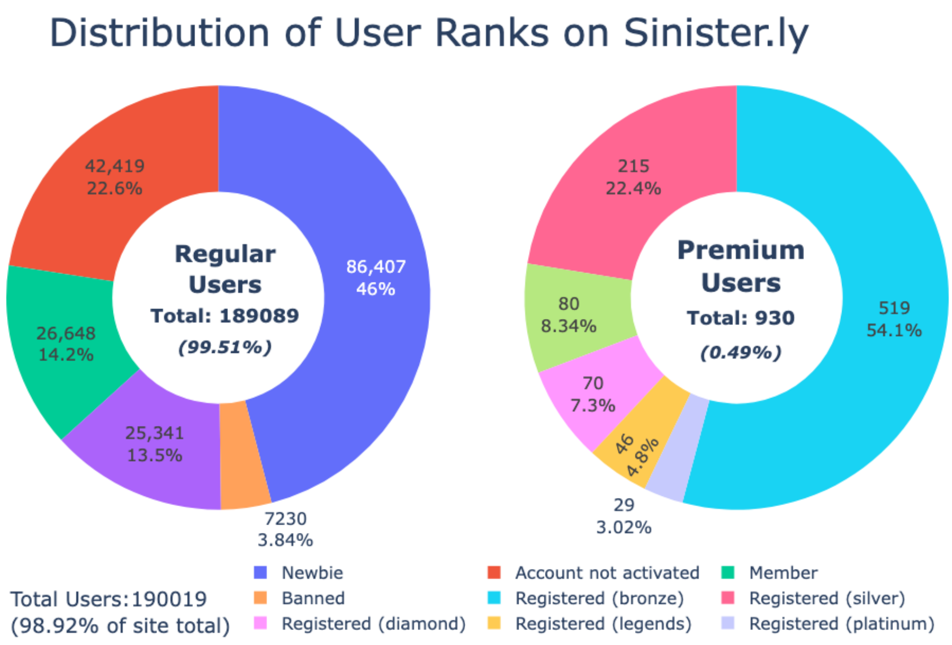


Figure 8.17: Distribution of User ranks on Sinister.ly (before 23/04/2023)

This indicates that very few users are willing to spend money on their accounts for the site itself. Of the few that do upgrade they are most likely to simply go for the bronze tier as this grants them access to the *Upgraded Lounge*, *Upgraded Giveaways & Freebies* and *Advanced Security* sections. Any users purchasing ranks above this are usually sellers, to get *Gold*, *Platinum* etc. prefixes to their threads, making them stand out more than regular threads.

On the left-hand doughnut chart, it can also be seen that the majority of regular users are *Account not Activated* and *Newbie* (69%). This is clearly an indication that the majority of users of the site are new users, or simply users that made a temporary account and never bothered activating it. The high number of *Account not Activated* users may be due to bots creating basic accounts to avoid detection while scraping the site. The same could be the case for *Newbie* accounts, as these are simply accounts recently joined and verified. Though this could also be users simply testing out the site.

Deducting the *Banned* users leaves just over 27% (52945) of users that are to some extent more active to the site along with just over 900 actually investing money in their account. Given that the amount of scraped users amounts to 99% of what the forum itself claims to have (as of 14/05/2022) it can be assumed this illustration is quite reliable.

8.2.2 Comparing Users on Different Forums

As the forums investigated in this project have a fair similarity in what their purpose is, it is safe to assume that a given user, is also likely to have an account on one of the other forums. As depicted in **Figure 8.18**, users on Hackforums.net commonly also have an account on Sinister.ly.

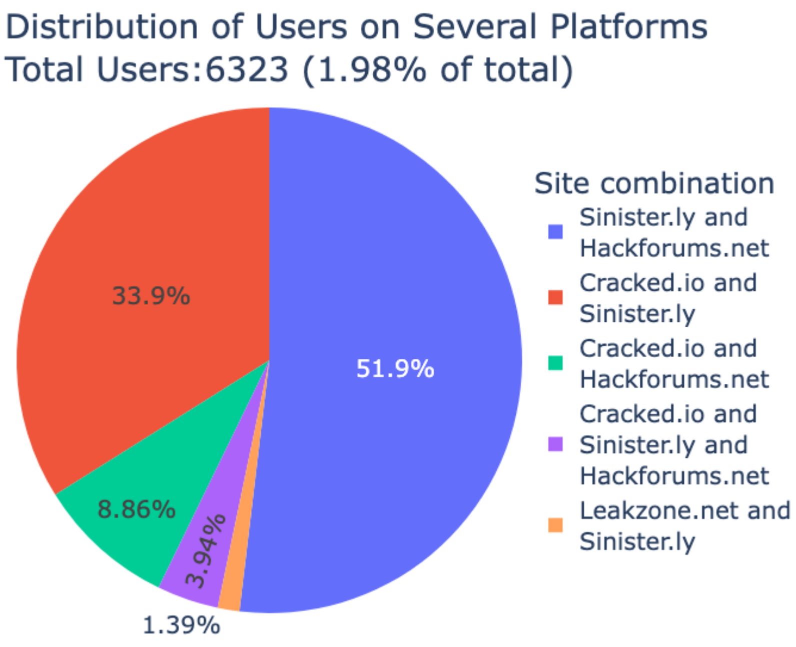


Figure 8.18: Distribution of users that have accounts on several sites (combinations below 0.1% filtered out)

It should be noted that the figure is generated on the basis of the usernames used on the various sites. While these do not guarantee to identify an individual it is common for users to reuse their username similarly to how they might reuse their password.[110] Moreover, the figure also displays that users on Sinister.ly additionally often have an account on Cracked.io. Subsequent combinations of forums are more rare, though still present. Though as indicated by the title of the figure, only 2% of the total amount of users scraped had identical usernames on other sites. Additionally as mentioned in the introduction of **Section 8.2**, only Sinister.ly had a substantial amount of users scraped, compared to the number of users available, explaining why **Figure 8.18**'s most prevalent Forum is Sinister.ly.

Summarizing the preceding chapters it can be concluded that:

- Only 2% of users present on one site, have an account on another
- Only 0.5% of users on Sinister.ly are premium users

- Of the 0.5% half are users at the low tier (Bronze)
- 68% of users on Sinister.ly are new users

The rank distribution is highly indicative given that nearly all users on Sinister.ly were scraped. However the distribution of users having accounts on several sites, mostly regard Sinister.ly, given that less than 2% of the user base of the other sites were collected.

9 Discussion & Future Works

This chapter aims to provide an assessment of the goals for the thesis, outline in **Chapter 1**. Subsequently, **Section 9.1** offers a concise summary evaluating the requirements introduced in **Chapter 5**. **Section 9.2** grants an overview of the limitations of the project, while **Section 9.3** explores directions of potential future research.

9.1 Evaluation of Requirements

This section includes an individual evaluation of the requirements specified in **Chapter 5**. The section is divided into two main subsections, one for each of the main objectives of **Chapter 5**, namely the *Web Scraper* requirements and *NLP and LLM* requirements.

9.1.1 Web Scraper Evaluation

Chapter 5 clearly stated several functional as well as non-functional requirements. Each of these will now be evaluated, divided into the ones that were implemented and the ones that were not. For each of the requirements, a short description of what they entail is given along with how they were implemented. Should they not be implemented the reasoning behind this will likewise be described.

Implemented

1. **Robustness**: This requirement states the the web scraper should be flexible in terms of sites and obstructions encountered. The web scraper was to be able to handle several site structures and layouts as well as common obstructions such as request timeouts. The web scraper easily handles the site structure of all the different forums, despite their differences in layout, while correctly catching request timeouts and waiting a few minutes before retrying.
2. **Data Quality**: The data that the web scraper collects should be cleansed and validated during the scraping process. As given in **Section 7.2.3**, the web scraper actively removes special Unicode characters to ensure consistency and quality in the database.

3. **Data Parsing**: As a web scraper, it must naturally be able to parse the DOM structure of the site in question. As given in **Section 7.1.1.1**, **Section 7.1.1.2** and **Section 7.1.1.3** respectively, the scraper accurately navigates to the most interesting parts of the investigated forums, while extracting relevant information.

4. **Data Storage**: To effectively analyse the data, the web scraper should store the data in an efficient and structured format. **Section 6.2** and **Figure 6.3** clearly outlined the structure of the data stored by the web scraper. Additionally as given in **Section 2.2** the web scraper stores the data in a database in the common key-value pairs format, allowing for easy manipulation of the data during the analysis.

5. **IP Rotation**: Due to the IP banning policies of the investigated forums, to operate effectively the web scraper had to implement ways to change IP addresses programmatically. Specifically, **Section 7.3.2.1** outlined how the scraper was set up with the capabilities of changing IP address via NordVPN with a specific time interval, effectively bypassing IP banning restrictions

6. **Bypassing Anti-scraping techniques**: Similar to the IP banning policies, many of the sites employ DDoS protection and subsequent CAPTCHA blockades, both of which the web scraper should handle. As given in **Section 7.3.3** the scraper utilised the `undetected_chromedriver` package to appear like a normal user, effectively bypassing the anti-scraping techniques employed by the forums. Additionally, the `hcaptcha-challenger` package was utilised, enabling the scraper to bypass hCAPTCHAs.

As seen above, each of the functional requirements was implemented for the designed web scraper. Moving on to the non-functional requirements most of these were met as well, as seen below.

1. **Scalability**: The scraper should be able to handle any volume of data found on the forums dexterously. During the scraping process, the scraper had no issue extracting the data of interest from the site, provided the data was passed through the cleansing process described in **Section 7.2.3**. The only limitation on this front was the database itself, of which the backend RPC calls could handle limited load sizes. Though as given in **Section 7.2.4**, this was handled by manually generating a query, to send to the database, as opposed to using the SurrealDB provided CRUD functions.

2. **Rate Limiting**: This requirement stated that in order to avoid stressing the forums investigated a rate-limiting feature should be incorporated into the scraper. **Section 3.8.1** clearly outlined that the scraper was set up to make no more than 0.2 requests per second, hence inducing fairness over the network.

Not implemented

1. **Efficiency**: The scraper should have a minimal computational footprint, while still operating at a reasonable efficiency. Running for a single site, the scraper could amount to up to 3-5GB of RAM, which while achievable on consumer hardware, is not particularly effective. Additionally, the scraper was not set up to run for several forums at once, however simply duplicating the instance (Jupyter Notebook) running the code, allowed for several instances of the web scraper to run concurrently. This however does not allow the concurrent instances to communicate, thereby not guaranteeing that unnecessary duplicate data is not scraped.

From the iteration of requirements above it is apparent that all the requirements generated in **Section 5.1** were met, with the slight exception of the non-functional **Efficiency** requirement.

9.1.2 NLP and LLM Requirements

The NLP and LLM requirements defined a much more modest decorum of requirements, each of which is assessed below.

Implemented

1. **High Performance**: The weighted average F1-score for each of the fine-tuned LLM models in the NLP pipeline should be relatively high. The performance of the first classifier as given in **Table 7.3** was very satisfactory achieving an F1-score of 0.92. **Table 7.4**, gave a bit more modest, though still fair score of 0.84. Finally, the NER classifier offered the lowest score of 0.73 given in **Table 7.5**. Though as discussed in **Section 7.4.6**, adding more labelled data to train this model would not necessarily promise any noticeable increase in performance.

2. **Efficiency**: This requirement stated that the fine-tuned LLM models should be trained in an efficient manner. The training time in all cases stated in **Tables 7.3** to **7.5** did not exceed 8 minutes, with the exception of the XLNet model trained at a higher batch size for the intent classifier. From this, it can be concluded that the models trained are quite effective. Granted they are run on a high-performing NVIDIA A10 GPU, though they were able to run on the author's (M1 MacBook) laptop as well, with a runtime closer to 20-40 min. Hence even on consumer hardware the fine-tuning of LLMs was quite efficient at batch sizes of 16.

Not Implemented

1. **Input Flexibility**: For the requirement it was given that the model should be able to handle any length of input. Regrettably, the base BERT model used for the intent classifier is limited to 512 tokens, thereafter simply truncating the input.

The same holds true for the RoBERTa and DistilBERT models, though not for the XLnet model. Hence the category classifier meets this requirement, while the intent classifier does not. The NER classifier has no input limit and hence also meets this requirement.

From the enumeration of the requirements above it can be confirmed that all the non-functional requirements were implemented for all the models. In terms of the functional requirement, while a general text classification model should effectively handle any length of the input, the requirement did not seem to matter much for the intent classifier, which despite truncating its input at times, still achieved an F1-score of 0.92. The category-classifier utilised XLNet and hence did not have this restriction, though achieved lower performance. In other words in the context of text classification, truncation of the input text can be considered acceptable, provided the model still provides high performance.

9.2 Limitations

The project had a high entry requirement when developing the web scraper from scratch for the first site. As there was no pre-built backend for handling request errors, and navigating the specific investigated forums, the project had a slightly slow start. However, as soon as the web scraper was developed for the first forums, the functionality for handling subsequent forums could be more efficiently implemented. Automated scraping tools such as *Octoparse*[111] could have been integrated as well, for faster iteration, though the abstraction level of such tools is quite high. They thereby do not specifically clarify how anti-scraping measures are effectively bypassed, nor do they provide the custom verification and cleansing of the data utilised in this thesis. Additionally, these services are often more focused on the speed of extraction, thereby not considering fairness over the network as done with the web scraper developed in this thesis.

Concerning the issues faced as the project progressed, as given by **Section 3.2.1.1**, the scraper was initially developed for Breaced.vc, which justifiably but regrettably was seized by the FBI during the project. As a result, the overall findings of the thesis were reduced and a sizeable portion of work was wasted customising the web scraper to that forum. Furthermore, **Section 7.4.4.2** described how fine-tuning the LLM models at batch sizes higher than 16 commonly resulted in the GPU running out on memory, even with the 24GB of the NVIDIA A10. Consequently, the training time for these models are not truly indicative of their effectiveness in training. The models could still run on consumer CPU hardware, albeit remarkably slower.

This project is also limited to investigating the categories described in **Section 6.3**. Hence despite the selection of categories being quite high, it fails to discern between products within a category, such as ransomware, and viruses under the malware category. Having an expanded palette of categories would improve the granularity of the results, thereby providing more refined analysis results. Finally, the data scraped was limited to 2023. This was chosen simply to make it achievable within the time frame, but while painting a picture of the most recent state of these forums, fails to describe their evolution.

9.3 Future Works

Numerous features could be added to improve both the web scraper and the NLP pipeline. Naturally, due to time constraints not all of the features could be implemented and in the list below a selection of possible improvements is discussed.

- **Reduced resource usage of the scraper**: As given in **Section 9.1.1**, the web scraper had a relatively high RAM usage, which while not ideal, still allowed it to be run on a basic laptop. The increased RAM usage may stem from caching when iterating over the extensive forum site structure, and hence future efforts could be conducted to clear out unused cached data while scraping.
- **Parallelisation**: The code for the web scraper is currently only set to run for one forum at a time. Future efforts could be directed towards running subprocesses for each forum at the same time. This would allow for faster scraping, without overloading the individual forum with requests,
- **Increased LLM performance**: With the classifiers developed in this thesis having weighted average F1-scores ranging from 0.73-0.92, there is definitely room for improvement, especially for the NER classifier. As given in **Figure 7.24**, adding more data is not necessarily the solution to improving the performance of this classifier, necessitating the investigation of other approaches.
- **Increased range of forums**: As the first investigated forum was seized by the FBI the output of the thesis became reduced. Additionally, the investigated forums are just a few of the many available. Expanding the collection of investigated forums will grant a more definitive overview of the true nature of these forums as a whole, and hence future efforts could be engaged in investigating more forums
- **Refining product categories**: Despite the number of categories classified in this thesis being quite high, there is still room for improved granularity as mentioned for the *Malware* category for example. Having more refined categories could allow for investigating much more specific topics, or even specific malware.
- **Further Analysis of the dataset**: The approaches taken for the analysis in this thesis are a subset of the many ways the dataset could be explored. For instance, a stronger focus on vendor reputation could be taken.

Given the dynamic and evolving nature of illicit forums, there remains constant room for improvement in accurately portraying their intricate nature and underlying mechanisms. As such the above suggested future works are merely suggestions and only comprise a subset of the possible expansions to this project.

10 Conclusion

The dynamic nature of online illicit forums necessitates continuous research into their characteristics. This thesis aimed to explore the extent and nature of the illegitimate digital products found on illicit forums operating on the clear web. To address this research objective, three sub-research objectives were formulated: developing a web scraper to extract data from these forums, applying NLP techniques to parse the extracted data, and gathering insights from the subsequent analysis of this data.

Firstly, a robust web scraper was developed, capable of extracting data from the illicit forums: `Breached.vc`, `Sinister.ly`, `Cracked.io`, `Hackforums.net` and `Leakzone.net`. The scraper effectively bypassed anti-scraping techniques employed by each of the forums, including IP banning, DDoS protection and CAPTCHA blockades. Additionally, it instilled fairness over the network, and validated and cleansed the scraped data. Secondly, NLP models were created, based on state-of-the-art LLMs to extract information from the collected textual data. This produced an NLP pipeline consisting of three classifiers: a BERT-based intent classifier, an XLNet-based category classifier and a spaCy-based NER classifier. The two text classifiers allowed for categorising the data into **Sales** and **Request** related threads, along with which topic category they belonged to. Lastly, the NER classifier determined the prices given in the threads. The extracted information allowed for subsequent analysis to determine valuable insights into the inner workings of the investigated sites, adding to the overall contribution of the thesis.

The analysis focused on metrics such as *Prices*, *Size* and common *Topics*, and showed that across all the categories investigated, products and services were sold at just over 25$ on average. In contrast, products were on average requested at a lower price of 20$, denoting users were less willing to pay the amount a certain product or service was offered at. Of the investigated forums Sinister.ly was the cheapest while Cracked.io was the most expensive across the most categories. Furthermore, Sinister.ly had the most **Sales** threads, while Cracked.io had the most **Request** threads.

Additionally, across all the forums, the most noticeable topics were *Accounts* and *Services*. Individually however each site had its own discernible topics with a slightly different focus. Specifically Sinister.ly had a high prevalence of topics related to *Money Making Methods*, while Hackforums.net had a fair selection of *Hacking* related threads. Contrariwise, Cracked.io contained common occurrences of *Combolists*, i.e. leaked lists of username and password combinations, whereas Leakzone.net consisted of an evenly distributed combination of the other forums' main topics. However, the project still leaves space for additional research, specifically in improving the NLP models, expanding the selection of forums and increasing the efficiency of the web scraper.

References

Articles in Journals

[3] Sergio Pastrana et al. “CrimeBB: Enabling Cybercrime Research on Underground Forums at Scale.” In: (2018). DOI: 10.1145/3178876.3186178. URL: https://doi.org/10.1145/3178876.3186178.

[4] Anh V Vu et al. “Turning Up the Dial: the Evolution of a Cybercrime Market Through Set-up, Stable, and Covid-19 Eras.” In: *IMC '20* (October 2020). DOI: 10.1145/3419394.3423636. URL: https://doi.org/10.1145/3419394.3423636.

[5] Alice Hutchings and Thomas J Holt. “A crime script analysis of the online stolen data market.” In: *The British Journal of Criminology* ().

[7] Samuel Couture and Philippe Lebrun. “The Stealer Malware Ecosystem: A Detailed Analysis of How Infected Devices Are Sold and Exploited on the Dark & Clear Web.” In: *Flare Report* ().

[8] Renushka Madarie et al. “Journal of Crime and Justice Stolen account credentials: an empirical comparison of online dissemination on different platforms.” In: (2019). ISSN: 2158-9119. DOI: 10.1080/0735648X.2019.1692418. URL: https://www.tandfonline.com/action/journalInformation?journalCode=rjcj20.

[9] Agus Nugroho and An An Chandrawulan. “Research synthesis of cybercrime laws and COVID-19 in Indonesia: lessons for developed and developing countries.” In: *Security Journal* (August 2022). DOI: 10.1057/s41284-022-00357-y. URL: https://doi.org/10.1057/s41284-022-00357-y.

[12] Authors J Broséus et al. “Title Studying illicit drug trafficking on Darknet markets: structure and organisation from a Canadian perspective.” In: (2016). DOI: 10.1016/j.forsciint.2016.02.045. URL: http://dx.doi.org/10.1016/j.forsciint.2016.02.045.

[13] Diego Zambiasi. “Drugs on the web, crime in the streets: The impact of Dark Web marketplaces on street crime.” In: (2020). URL: http://hdl.handle.net/10419/228205.

[14] Roderic G Broadhurst, Donald Maxim, and Hannah Woodford-Smith. “Malware Trends on 'Darknet' Crypto-markets: Research Review Vulnerability to social engineering View project Malware in Spam Email View project.” In: (July 2018). DOI: DOI:10.13140/RG.2.2.36312.60168. URL: https://www.researchgate.net/publication/326436666.

[18] Irfan Darmawan et al. “Evaluating Web Scraping Performance Using XPath, CSS Selector, Regular Expression, and HTML DOM With Multiprocessing Technical Applications.” In: *JOIV : International Journal on Informatics Visualization* 6 (March 2022), page 904. DOI: 10.30630/joiv.6.4.1525.

[29] Mallikarjun B.C. et al. “Intelligent Automated Text Processing System - An NLP Based Approach.” In: *2020 5th International Conference on Communication and Electronics Systems (ICCES)* (2020), pages 1026–1030.

[30] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.” In: *Google AI Language* (). URL: https://github.com/tensorflow/tensor2tensor.

[31] Necva Bölücü and Burcu Can. “Joint PoS Tagging and Stemming for Agglutinative Languages.” In: *Department of Computer Engineering, Hacettepe University* ().

[33] Krishna Juluru et al. “Bag-of-Words Technique in Natural Language Processing: A Primer for Radiologists.” In: *RadioGraphics* 41 (2021), pages 1420–1426. DOI: 10.1148/rg.2021210025. URL: https://doi.org/10.1148/rg.2021210025.

[34] Qi Liu, Matt J Kusner, and Phil Blunsom. “A Survey on Contextual Embeddings.” In: (2020).

[35] Yuqi Si et al. “Enhancing clinical concept extraction with contextual embeddings.” In: *Journal of the American Medical Informatics Association* (). DOI: 10.1093/jamia/ocz096.

[36] Ashish Vaswani et al. “Attention Is All You Need.” In: ().

[40] Zhilin Yang et al. “XLNet: Generalized Autoregressive Pretraining for Language Understanding.” In: *Google AI Brain Team* (). URL: https://github.com/zihangdai/xlnet.

[41] Yinhan Liu et al. “RoBERTa: A Robustly Optimized BERT Pretraining Approach.” In: (2019). URL: https://github.com/pytorch/fairseq.

[42] Victor Sanh et al. “DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter.” In: *Energy Efficient Machine Learning and Cognitive Computing* (). URL: https://github.com/huggingface/transformers.

[45] Harleen K Flora and Swati V Chande. “A Systematic Study on Agile Software Development Methodologies and Practices.” In: *(IJCSIT) International Journal of Computer Science and Information Technologies* 5.3 (2014), pages 3626–3637. URL: www.ijcsit.com.

[63] James Martin and Nicolas Christin. “Ethics in Cryptomarket Research 1.” In: (2016). DOI: 10.1016/j.drugpo.2016.05.006. URL: http://dx.doi.org/10.1016/j.drugpo.2016.05.006.

[66] "ALFRED HOSPITAL ETHICS COMMITTEE GUIDELINES ALFRED HOSPITAL ETHICS COMMITTEE GUIDELINES: RESEARCH THAT POTENTIALLY INVOLVES LEGAL RISKS FOR PARTICIPANTS AND RESEARCHERS." In: (2023).

[69] Thomas J Holt. "Identifying gaps in the research literature on illicit markets on-line, Global Crime." In: 18.1 (2017), pages 1–10. ISSN: 1744-0572. DOI: 10.1080/17440572.2016.1235821. URL: https://www.tandfonline.com/action/journalInformation?journalCode=fglc20.

[70] Rui Cai et al. "iRobot: An Intelligent Crawler for Web Forums." In: (April 2008). URL: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=a7b776c9301cd616f0cc7887059b96dc9beee93e.

[71] Jingtian Jiang et al. "FoCUS: Learning to Crawl Web Forums." In: *IEEE Transactions on Knowledge and Data Engineering* 25.6 (2013), pages 1293–1306. DOI: 10.1109/TKDE.2012.56.

[72] Bissan Audeh et al. "Vigi4Med Scraper: A Framework for Web Forum Structured Data Extraction and Semantic Representation." In: (December 2017). DOI: 10.1371/journal.pone.0169658. URL: https://github.com/.

[73] Bassel Alkhatib and Randa Basheer. "Crawling the Dark Web: A Conceptual Perspective, Challenges and Implementation." In: *Journal of Digital Information Management* 17 (2019). DOI: 10.6025/jdim/2019/17/2/51-60.

[75] Joe Buskirk et al. "The closure of the Silk Road: What has this meant for online drug trading?" In: *Addiction (Abingdon, England)* 109 (February 2014). DOI: 10.1111/add.12422.

[77] Stuart E Middleton et al. "Information Extraction from the Long Tail: A Socio-Technical AI Approach for Criminology Investigations into the Online Illegal Plant Trade." In: 4 (July 2020). DOI: 10.1145/3394332.3402838. URL: https://doi.org/10.1145/3394332.3402838.

[79] Sergio Pastrana et al. "Characterizing Eve: Analysing Cybercrime Actors in a Large Underground Forum." In: *RAID 2018: Research in Attacks, Intrusions, and Defenses* (). URL: https://www.repository.cam.ac.uk/bitstream/handle/1810/284436/RAID_2018.pdf?sequence=1.

[80] Mhd Wesam Al-Nabki, Eduardo Fidalgo, and Javier Velasco Mata. "DarkNER: A Platform for Named Entity Recognition in Tor Darknet." In: (1995). URL: https://www.clips.uantwerpen.be/conll2003/ner.

[81] Dogu Tan Araci. "FinBERT: Financial Sentiment Analysis with Pre-trained Language Models." In: (June 2019).

[82] Martin Glinz. "On Non-Functional Requirements." In: (2007). URL: https://www.ptidej.net/courses/log3410/summer11/Lectures/Article_6.pdf.

[94] Diego Perino, Matteo Varvello, and Claudio Soriente. “ProxyTorrent: Untangling the Free HTTP(S) Proxy Ecosystem.” In: (April 2018). DOI: 10.1145/3178876.3186086. URL: https://doi.org/10.1145/3178876.3186086.

Web pages

[1] Netacea Ltd. *What Is The Clear Web? | Clear Web vs Dark Web | Netacea.* June 2020. URL: https://netacea.com/glossary/clear-web/.

[2] Crowdstrike. *Deep Web vs Dark Web: What’s the Difference? | CrowdStrike.* October 2022. URL: https://www.crowdstrike.com/cybersecurity-101/the-dark-web-explained/deep-web-vs-dark-web/.

[6] Eastern District of Wisconsin United States Attorney’s Office. *Eastern District of Wisconsin | Genesis Market Disrupted in International Cyber Operation | United States Department of Justice.* April 2023. URL: https://www.justice.gov/usao-edwi/pr/genesis-market-disrupted-international-cyber-operation.

[11] Drew Todd and SecureWorld News. *New Record: Darknet Markets Are Booming.* February 2021. URL: https://www.secureworld.io/industry-news/record-profits-for-darknet-markets.

[15] Inc. Twitter. *Twitter API Documentation | Docs | Twitter Developer Platform.* URL: https://developer.twitter.com/en/docs/twitter-api.

[16] *What is a REST API? | IBM.* URL: https://www.ibm.com/topics/rest-apis.

[17] Leonard Richardson. *Beautiful Soup: We called him Tortoise because he taught us.* URL: https://www.crummy.com/software/BeautifulSoup/.

[19] Software Freedom Conservancy. *Selenium.* URL: https://www.selenium.dev/.

[20] Twilio Inc. *segmentio/nightmare: A high-level browser automation library.* URL: https://github.com/segmentio/nightmare.

[21] QIN2DIM. *hcaptcha-challenger · PyPI.* URL: https://pypi.org/project/hcaptcha-challenger/.

[22] ZenRows. *How to Avoid Bot Detection with Selenium - ZenRows.* April 2023. URL: https://www.zenrows.com/blog/selenium-avoid-bot-detection#how-anti-bots-work.

[23] UltrafunkAmsterdam. *undetected-chromedriver · PyPI.* URL: https://pypi.org/project/undetected-chromedriver/.

[24] Vitally Ilyukha. *Difference Between Relational vs. Non-Relational Database.* URL: https://jelvix.com/blog/relational-vs-non-relational-database.

[25] *Figma Organization.* URL: https://www.figma.com/organization/.

[26] Inc. Dgraph Labs. *Dgraph | GraphQL Cloud Platform – GraphQL . Javascript . Distributed Graph Engine | Deploy a Production Ready GraphQL Backend in Minutes.* URL: https://dgraph.io/.

[27] SurrealDB Ltd. *SurrealDB | The ultimate serverless cloud database.* URL: https://surrealdb.com/.

[28] SurrealDB Ltd. *SurrealDB | Features | The ultimate serverless cloud database.* URL: https://surrealdb.com/features.

[32] Explosion Inc. *spaCy · Industrial-strength Natural Language Processing in Python.* URL: https://spacy.io/.

[37] Google Inc. *Google Research.* URL: https://research.google/.

[39] Nicolo Cosimo Albanese. *Fine-Tuning BERT for Text Classification | by Nicolo Cosimo Albanese | Towards Data Science.* May 2022. URL: https://towardsdatascience.com/fine-tuning-bert-for-text-classification-54e7df642894.

[43] *Is there an ideal ratio between a training set and validation set? Which trade-off would you suggest? | ResearchGate.* URL: https://www.researchgate.net/post/Is-there-an-ideal-ratio-between-a-training-set-and-validation-set-Which-trade-off-would-you-suggest.

[44] Krusche & Company GmbH. *What is Agile software development?* URL: https://kruschecompany.com/agile-software-development/.

[46] *Project Jupyter | Home.* URL: https://jupyter.org/.

[47] Inc. Reddit. *hacking: security in practice.* URL: https://www.reddit.com/r/hacking/.

[48] Inc. Reddit. *Black Hat Hackers.* URL: https://www.reddit.com/r/BlackHatHackers/.

[49] Inc Brave Software. *What's the most secure Web browser? | Brave Browser.* February 2023. URL: https://brave.com/learn/most-secure-browser/.

[50] Yuval Shini. *Top 5 Hacker Forums on the Deep and Dark Web in 2022 | Webz.io.* June 2022. URL: https://webz.io/dwp/top-5-hacker-forums-on-the-deep-and-dark-web-in-2022/.

[51] Feedspot Media Database Team. *Top 30 Cracking Forums, Discussions, and Message Boards in 2023.* April 2023. URL: https://blog.feedspot.com/cracking_forums/.

[52] *xss.is.* URL: https://xss.is/.

[53] *mpgh.net.* URL: https://www.mpgh.net/.

[54] Breached Forums. *BreachForums.* https://breached.vc/. URL: https://web.archive.org/web/20230319045616/https://breached.vc/.

[55] *Six months into Breached: The legacy of RaidForums? • Kela.* September 2022. URL: https://ke-la.com/six-months-into-breached-the-legacy-of-raidforums/.

[56] *Download Tenable Nessus Vulnerability Assessment | Tenable®.* URL: https://www.tenable.com/products/nessus.

[57] The United States Department Of Justice. *Justice Department Announces Arrest of the Founder of One of the World's Largest Hacker Forums and Disruption of Forum's Operation | OPA | Department of Justice.* March 2023. URL: https://www.justice.gov/opa/pr/justice-department-announces-arrest-founder-one-world-s-largest-hacker-forums-and-disruption.

[58] *Sinister.ly.* URL: https://sinister.ly/.

[59] *Hackforums.net.* URL: https://hackforums.net.

[60] *cracked.io Website Traffic, Ranking, Analytics [March 2023] | Semrush.* URL: https://www.semrush.com/website/cracked.io/overview/.

[61] *Cracked.io.* URL: https://cracked.io/.

[62] *Leakzone.net.* URL: https://leakzone.net/.

[64] *How to Estimate How Many Website Visitors Your Hosting Can Deal With.* May 2022. URL: https://servebolt.com/articles/calculate-how-many-simultaneous-website-visitors/.

[65] *Cracked.io | Netflix Gift Card Codes.* URL: https://cracked.io/Thread-For-Everyone-Netflix-Gift-Card-Generator-MalwareByte-Code-Generator-3-MODULE.

[68] Armando Hernandez. *What Does it Mean to Obstruct Justice?* January 2022. URL: https://www.armandohernandezlaw.com/blog/2022/january/what-does-it-mean-to-obstruct-justice-/.

[76] Us Department of Justice. *AlphaBay-Cazes Forfeiture Complaint and Exhibits.* Washington DC, July 2017. URL: https://www.justice.gov/opa/press-release/file/982821/download.

[78] *2021 Top Malware Strains | CISA.* URL: https://www.cisa.gov/news-events/cybersecurity-advisories/aa22-216a.

[83] *Open-source tool that uses simple textual descriptions to draw beautiful UML diagrams.* URL: https://plantuml.com/.

[84] *Cybersixgill - Threat Intelligence Solutions | Cybersixgill.* URL: https://cybersixgill.com/news/articles/how-gift-cards-are-used-to-launder-money.

[85] Lucid Software Inc. *Homepage | Lucid.* URL: https://lucid.co/.

[86] The Noun Project by ratubilqis1986. *Document icon.* URL: https://thenounproject.com/icon/document-5698656/.

[87] papers by leo-graph.com The Noun Project. *Documents icon.* URL: https://thenounproject.com/icon/papers-974712/.

[88] Freepik. *NLP chip icon.* URL: https://www.flaticon.com/free-icon/natural-language-processing_9831371.

[89] The Noun Project by David McDonald. *Dataset icon.* URL: https://thenounproject.com/icon/valid-data-3512792/.

[90] Semantic Evolution Ltd. *Named Entity Recognition (NER) — Semantic Evolution - Intelligence Built In.* URL: https://semantic-evolution.com/named-entity-recognition-ner.

[91] *MyBB - Free and Open Source Forum Software.* URL: https://mybb.com/.

[92] Baiju Muthukadan. *7. WebDriver API — Selenium Python Bindings 2 documentation.* 2018. URL: https://selenium-python.readthedocs.io/api.html.

[93] SurrealDB. *surrealdb · PyPI.* October 2022. URL: https://pypi.org/project/surrealdb/.

[95] Nord Security. *The best online VPN service for speed and security | NordVPN.* URL: https://nordvpn.com/.

[96] Aliza Vigderman and Gabe Turner. *NordVPN Review 2023 | Security.org.* March 2023. URL: https://www.security.org/vpn/nordvpn/review/.

[97] KEPIOS. *Global top websites by time per visit 2022 | Statista.* November 2022. URL: https://www.statista.com/statistics/1201901/most-visited-websites-worldwide-time-visit/.

[98] DDoS-Guard. *DDoS-Guard | Reliable DDoS Protection & Mitigation.* URL: https://ddos-guard.net/en.

[99] Proton AG. *Proton — Privacy by default.* URL: https://proton.me/.

[100] *koaning/bulk: A Simple Bulk Labelling Tool.* URL: https://github.com/koaning/bulk.

[101] Adam Pearce Andy Coenen. *Understanding UMAP.* URL: https://pair-code.github.io/understanding-umap/.

[102] Explosion Inc. *Prodigy · Prodigy · An annotation tool for AI, Machine Learning & NLP.* URL: https://prodi.gy/.

[103] Explosion Inc. *Training Pipelines & Models · spaCy Usage Documentation.* URL: https://spacy.io/usage/training.

[104] Forbes. *Why Is Crypto Going Up Today: May 2023 – Forbes Advisor INDIA.* April 2023. URL: https://www.forbes.com/advisor/in/investing/cryptocurrency/why-is-crypto-going-up/.

[105] Forbes. *Cybersecurity Trends & Statistics; More Sophisticated And Persistent Threats So Far In 2023.* May 2023. URL: https://www.forbes.com/sites/chuckbrooks/2023/05/05/cybersecurity-trends--statistics-more-sophisticated-and-persistent-threats-so-far-in-2023/.

[106] *wordcloud · PyPI.* URL: https://pypi.org/project/wordcloud/.

[107] Tokinaist For Social Media Services. *Tokinaist Shop | Buy High-Quality Accounts for Your Business.* URL: https://tokinaist.mysellix.io/.

[108] *Discord | Your Place to Talk and Hang Out.* URL: https://discord.com/.

[109] Telegram. *Telegram FAQ*. URL: https://telegram.org/faq#q-how-secure-is-telegram.

[110] *25+ Password Statistics that may change your password habits*. URL: https://www.comparitech.com/blog/information-security/password-statistics/.

[111] *Web Scraping Tool & Free Web Crawlers | Octoparse*. URL: https://www.octoparse.com/.

Appendix

This thesis is accompanied by an attached .zip file with an extended appendix. In this zipped appendix the following is included:

1. Main scraper code (Found in `siteScraping.ipynb` `user_extraction.py`, `forum_extraction.py`, `thread_extraction.py` and `scraping.py` respectively)
2. Database functions (Found in `db_func.py` and `utils.py` respectively)
3. Code used to train NLP models (Found in `NLP_pipeline_creation.ipynb`, `nlp_func.py` and `Train_NLP_model_with_params.ipynb` respectively)
4. Jupyter notebooks and python files used to generate figures (Found in `DataAnalysis.ipynb`, `plotting.py` and `utils.py` respectively)
5. Another zip archive containing the latex code comprising the thesis

Note that the code found in this appendix is only a subset of the code found in the zip file, as the functions in this appendix are only mainly meant to supplement the text of the thesis where appropriate.

Appendix A: Scraper Code

A1: Main Scraping Function

Listing 1: Main scraping function `get_pages` to scrape illicit forums

```
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.common.by import By
from selenium.common.exceptions import TimeoutException
from db_func import query
import datetime
from random import randint
from utils import internet_on
import time
import requests
from bs4 import BeautifulSoup
from scraping import (
    solve_h_capthca,
    extract_links_from_responses,
    get_youngest_thread_date,
)
import re
from urllib.parse import urlparse
from db_func import (
    create_database_entry,
    relate_user_to_post,
    relate_user_to_thread,
    relate_post_to_thread,
)
from user_extraction import extract_user_data
from thread_extraction import extract_thread_data
from nordvpn_switcher import initialize_VPN, rotate_VPN
from surrealdb import Surreal

vpn_settings = initialize_VPN(stored_settings=1)
last_vpn_switch_time = datetime.datetime.now()
rotate_VPN(vpn_settings)

async def get_pages(
    url_main: str,
    headers: dict,
    pages: list = None,
    is_recursive: bool = False,
    is_retry: bool = False,
    table: str = None,
    update: bool = True,
    minimum_date: datetime = None,
    db: Surreal = None,
):
    global vpn_settings
    global driver
    global last_vpn_switch_time

    if pages is None:
        pages = [url_main]
    responses = []
    i = 0
    for page in pages:
        request_url = page
        if url_main not in page:
            request_url = url_main + request_url
        try:
            if table is not None and update == False:
                db_entry = await query(
                    "SELECT * FROM " + table + " WHERE url = '" +
   request_url + "'",
                    db=db,
                )
                if db_entry:
                    if type(db_entry) == list:
                        db_entry = db_entry[0]
                    if table == "forums":
                        if db_entry["threads"] != []:
                            responses.append(db_entry)
                            i = i + 1
                            continue
                    else:
                        responses.append(db_entry)
                        i = i + 1
                        continue
            datetime_now = datetime.datetime.now()
            if (
                (datetime_now - last_vpn_switch_time).total_seconds() / 60.0
            ) > 5 and not is_retry:
                rotate_VPN(vpn_settings)
                last_vpn_switch_time = datetime.datetime.now()
            if internet_on():
                scrape_time = datetime_now.strftime("%Y-%m-%d %H:%M:%S")
                driver.get(request_url)
                WebDriverWait(driver, 300).until_not(EC.
title_contains("DDoS-Guard"))
                WebDriverWait(driver, 300).until_not(
                    EC.title_contains("Cracked.to Under Attack")
                )
                if "HACKFORUMS" in request_url.upper():
                    try:
                        WebDriverWait(driver, 30).until_not(
                            EC.title_contains("Hack Forums - Challenge")
                        )
                    except TimeoutException:
                        print("CloudFlare blocked by site. Trying again in
10 min")
                        time.sleep(600)
                        await get_pages(
                            url_main=url_main,
                            headers=headers,
                            pages=[page],
                            is_recursive=is_recursive,
                            is_retry=is_retry,
                            table=table,
                            update=update,
                            db=db,
                        )
                        continue
                time.sleep(randint(1, 5))
            else:
                print("Not connected to the internet. Waiting 10 sec to
    reattempt")
                time.sleep(10)
                responses.extend(
                    await get_pages(
                        url_main=url_main,
                        headers=headers,
                        pages=[page],
                        is_recursive=is_recursive,
                        is_retry=is_retry,
                        table=table,
                        update=update,
                        db=db,
                    )
                )
                continue
        except requests.exceptions.ConnectionError as e:
            if "Max retries exceeded" in str(e):
                print(
                    "Connection Error: Max retries exceeded after fetching "
                    + str(
                        len(responses)
                        + sum(
                            [
                                len(
                                    response.get("pagination_pages", [])
                                    for response in responses
                                )
                            ]
                        )
                    )
                )
                if is_recursive:
                    return {
                        "scrape_time": scrape_time,
                        "url": request_url,
                        "response": response,
                    }
                else:
                    return responses
            elif "Remote end closed connection" in str(e):
                print(
                    "Connection Error: Remote host closed connection.
Waiting to retry..."
                )
                rotate_VPN(vpn_settings)
                time.sleep(20)
                response = await get_pages(
                    url_main,
                    headers,
                    pages=[page],
                    table=table,
                    is_recursive=False,
                    is_retry=True,
                    db=db,
                )
            else:
                if is_recursive:
                    return {
                        "scrape_time": scrape_time,
                        "url": request_url,
                        "response": driver.page_source,
                    }
                else:
                    return responses
        except requests.exceptions.Timeout as e:
            if "Can't fetch current ip" in str(e):
                # Wait 5 minutes and try to reconnect
                time.sleep(300)
                response = await get_pages(
                    url_main,
                    headers,
                    pages=[page],
                    table=table,
                    is_recursive=False,
                    is_retry=False,
                    db=db,
                )
                if is_recursive:
                    return response[0]
                else:
                    responses.extend(response)
                continue
            print(e)
            if is_recursive:
                return {
                    "scrape_time": scrape_time,
                    "url": request_url,
                    "response": driver.page_source,
                }
            else:
                return responses
        except Exception as e:
            try:
                h_captcha = driver.find_element(By.ID, "h-captcha")
                if h_captcha:
                    await solve_h_capthca(site=request_url, ctx=driver)
            except Exception as e2:  # no Captcha found on site
                print("Unknown exception occurres: " + str(e))
                try:
                    # see if scrape_time is defined
                    temp = scrape_time
                except:
                    scrape_time = datetime_now.strftime("%Y-%m-%d %H:%M:%S")
                html = BeautifulSoup(driver.page_source)
                title = html("title")[0].text
                if is_recursive:
                    return {
                        "scrape_time": scrape_time,
                        "title": title,
                        "url": request_url,
                        "response": driver.page_source,
                    }
                else:
                    return responses
        html = BeautifulSoup(driver.page_source)
        error_div = html.find_all("div", {"class": "error-code"})
        if error_div:
            if error_div[0].text == "ERR_NETWORK_CHANGED":
                print("Network changed. Reattempting in 15 sec")
                time.sleep(15)
                responses.extend(
                    await get_pages(
                        url_main=url_main,
                        headers=headers,
                        pages=[page],
                        is_recursive=is_recursive,
                        is_retry=is_retry,
                        table=table,
                        update=update,
                        db=db,
                    )
                )
                continue
            else:
                print("Unknown network error occurred. Reattempting in 15
sec")
                time.sleep(15)
                responses.extend(
                    await get_pages(
                        url_main=url_main,
                        headers=headers,
                        pages=[page],
                        is_recursive=is_recursive,
                        is_retry=is_retry,
                        table=table,
                        update=update,
                        db=db,
                    )
                )
                continue
        else:
            error_div = html.find_all("div", {"class": "error_message"})
            if error_div:
                if "The member you specified is either invalid" in
error_div[0].text:
                    return responses
        title = html("title")[0].text
        if "DDoS-Guard" in title:
            print("DDoS Guard blocked for site" + request_url)
            with open("ddos-guard-blocked.txt", "a") as f:
                f.write(request_url + "\n")
            return {
                "scrape_time": scrape_time,
                "title": title,
                "url": request_url,
                "html": html,
                "is_DDoS_Guard_blocked": True,
            }
        elif (
            "Forbidden" in title
            and not "getting" in title
            and not "banned" in title
            and any(char.isdigit() for char in title)
        ):
            # rotate_VPN(vpn_settings)
            response = await get_pages(
                url_main,
                headers,
                pages=[page],
                table=table,
                is_recursive=False,
                is_retry=True,
                db=db,
            )
            responses.extend(response)
            continue
        result = {
            "scrape_time": scrape_time,
            "title": title,
            "url": request_url,
            "html": html,
        }
        if is_recursive:
            return result
        else:
            responses.append(result)
        if not is_recursive:
            if "LEAKZONE.NET" in request_url.upper():
                tag = "span"
            else:
                tag = "div"
            pagination_divs = html.find_all(tag, {"class": "pagination"})
            if pagination_divs:
                pagination_divs = [
                    pagination_div
                    for pagination_div in pagination_divs
                    if pagination_div.get("style") != "display: none;"
                ]
                pagination_divs = list(set(pagination_divs))
                if pagination_divs:
                    is_recursive = True
                    for pagination_div in pagination_divs:
                        if pagination_div.get("style"):
                            continue
                        else:
                            break
                    # get max page:
                    max_page = max(
                        [int(n) for n in re.findall("\d+", pagination_div.
text)]
                    )
                    responses[i]["pagination_pages"] = []
                    request_url = driver.current_url
                    for j in range(2, max_page + 1):
                        if "USER" in request_url.upper():
                            page_url = [request_url + "?view=awards&page=" +
str(j)]
                        elif (
                            "HACKFORUMS" in request_url.upper()
                            or "CRACKED" in request_url.upper()
                        ):
                            page_url = [request_url + "&page=" + str(j)]
                        else:
                            page_url = [request_url + "?page=" + str(j)]
                        pagination_response = await get_pages(
                            url_main=url_main,
                            headers=headers,
                            table=table,
                            pages=page_url,
                            is_recursive=is_recursive,
                            db=db,
                        )
                        if type(pagination_response) == list:
                            pagination_response = pagination_response[0]
                        responses[i]["pagination_pages"].
append(pagination_response)
                        if minimum_date is not None:
                            if (
                                get_youngest_thread_date(
                                    pagination_response["html"],
site=request_url
                                )
                                <= minimum_date
                            ):
                                break
                    is_recursive = False
            parsed_url = urlparse(request_url)
            site = parsed_url.scheme + "://" + parsed_url.netloc
            if (
                not "is_DDoS_Guard_blocked" in responses[i]
                and title.upper() != "404 NOT FOUND"
            ):
                if (
                    ("/User-" in request_url)
                    or ("?action=profile&uid=" in request_url)
                    or ("CRACKED" in request_url.upper())
                    and not any([s in request_url for s in ["/Thread", "/
Forum"]])
                ):
                    responses[i] = await extract_user_data([responses[i]])
                    if type(responses[i]) == list:
                        responses[i] = responses[i][0]
                        if "user_info" in responses[i]:
                            user_database_id = await create_database_entry(
                                "users", responses[i]["user_info"],
site=site, db=db
                            )
                            responses[i]["user_info"]["id"] =
user_database_id
                if "/Thread-" in request_url or "showthread.php?tid=" in
request_url:
                    thread_data = await extract_thread_data(
                        responses[i], site=request_url
                    )
                    if "id" in thread_data:
                        responses.append(thread_data)
                        i = i + 1
                        continue
                    if thread_data == {}:
                        i = i + 1
                        continue
                    if not "url" in thread_data:
                        if "?page=" in request_url:
                            thread_url = request_url.split("?page=")[0]
                        else:
                            thread_url = request_url
                        thread_data["url"] = thread_url

                    responses[i]["thread_data"] = thread_data
                    posts_database_ids = []
                    # create post entries
                    for idx, thread_page in enumerate(thread_data["pages"]):
                        if thread_page != {}:
                            for post in thread_page["posts"]:
                                db_entry_post = await query(
                                    "SELECT * FROM posts WHERE site = '"
                                    + site
                                    + "' AND pid = '"
                                    + post["pid"]
                                    + "'"
                                )
                                if (db_entry_post == []) or update:
                                    if not "scrape_time" in post:
                                        post["scrape_time"] = scrape_time
                                    post_database_id = await
create_database_entry(
                                        "posts", post, site=site, db=db
                                    )
                                    if not "posts:" in post_database_id:
                                        post_database_id = "posts:" +
    post_database_id
                                    posts_database_ids.
    append(post_database_id)
                                    result = await relate_user_to_post(
                                        post,
                                        post_database_id,
                                        site,
                                        url_main,
                                        headers,
                                        db=db,
                                    )
                                else:
                                    posts_database_ids.
    append(db_entry_post[0]["id"])
                    del thread_data["pages"]
                    if posts_database_ids != []:
                        thread_data["posts"] = posts_database_ids
                    # create thread entries
                    if not "scrape_time" in thread_data:
                        thread_data["scrape_time"] = scrape_time
                    thread_database_id = await create_database_entry(
                        "threads", thread_data, site=site, db=db
                    )
                    thread_data["thread_database_id"] = thread_database_id
                    if thread_data["author_uid"] != 0:
                        result = await relate_user_to_thread(
                            thread_data,
                            thread_database_id,
                            site,
                            url_main,
                            headers,
                            db=db,
                        )
                    if "posts" in thread_data:
                        for post in thread_data["posts"]:
                            result = await relate_post_to_thread(
                                post_id=post, thread_id=thread_database_id,
    db=db
                            )
                if "/Forum-" in request_url or "forumdisplay.php?fid=" in
    request_url:
                    thread_links = []
```

```
                    responses[i] = extract_links_from_responses(
                        responses=[responses[i]], url_main=url_main
                    )
                    responses[i] = responses[i][0]
                    if "pagination_pages" in responses[i]:
                        for idx, response in enumerate(
                            responses[i]["pagination_pages"]
                        ):
                            if "links" in response:
                                thread_links.extend(

  responses[i]["pagination_pages"][idx]["links"][
                                        "thread_links"
                                    ]
                                )
                    if "links" in responses[i]:
                        thread_links.
  extend(responses[i]["links"]["thread_links"])
                    forum_data = {
                        "scrape_time": responses[i]["scrape_time"],
                        "title": responses[i]["title"],
                        "url": responses[i]["url"],
                        "nr_of_pagination_pages":
  len(responses[i]["pagination_pages"])
                        + 1
                        if "pagination_pages" in responses[i]
                        else 0,
                        "threads": list(set(thread_links)),
                    }
                    if "?FID=" in request_url.upper():
                        forum_data["fid"] = re.search(r"fid=(\d+)",
  request_url).group(
                            1
                        )
                    forum_database_id = await create_database_entry(
                        "forums", forum_data, site=site, db=db
                    )
                    responses[i]["forum_database_id"] = forum_database_id
        i = i + 1
    return responses
```

A2: Extracting Forum, Thread and User Links From HTML

Listing 2: extract_links_from_html function for gathering forum, thread and user links in HTML

```
from urllib.parse import urlparse

def extract_links_from_html(url_main: str, html_dict: dict):
    dict_copy = html_dict.copy()
    for key, value in dict_copy.items():
        if not type(value) == BeautifulSoup:
            continue
        soup = value
        anchors = soup.find_all("a")
        page_links = []
        thread_links = []
        user_links = []
        for anchor in anchors:
            if not "href" in anchor.attrs:
                continue
            href = anchor["href"]
            if url_main == href:
                continue
            if ".onion" in href:
                continue
            if "https://" not in href:
                href = url_main + href
            if "?" in href:
                if "tid" in href and not "?action=whoposted" in href:
                    thread_links.append(href)
            elif "USER" in href.upper() or "member.php?action=profile" in
  href:
                user_links.append(href)
            elif "THREAD" in href.upper():
                if "https://" not in href:
                    href = url_main + href
                parsed_url = urlparse(href)
                href = parsed_url.scheme + "://" + parsed_url.netloc +
  parsed_url.path
                thread_links.append(href)
            elif "FORUM-" in href.upper() or "?FID=" in href.upper():
                page_links.append(href)
        if "SINISTER" in url_main.upper() or "HACKFORUMS" in url_main.
  upper():
            thread_links = []
            rows = soup.find_all("tr", {"class": "inline_row"})
            for row in rows:
                anchors = row.find_all("a")
                for anchor in anchors:
                    href = anchor["href"]
                    if url_main == href or ".onion" in href:
                        continue
                    if (
                        "THREAD-" in href.upper()
                        or "SHOWTHREAD.PHP?TID=" in href.upper()
                    ):
                        if "https://" not in href:
                            href = url_main + href
                        parsed_url = urlparse(href)
                        href = (
                            parsed_url.scheme
                            + "://"
                            + parsed_url.netloc
                            + parsed_url.path
                            + "?tid="
                            + href.split("?tid=")[-1]
                        )
                        if " " in href:
                            href = href.split(" ")[0]
                        if "&page=" in href:
                            href = href.split("&page=")[0]
                        href = href.replace("&action=lastpost", "").replace(
                            "&action=newpost", ""
                        )
                        thread_links.append(href)

        html_dict["links"] = {
            "page_links": list(set(page_links)),
            "thread_links": list(set(thread_links)),
            "user_links": list(set(user_links)),
        }
    return html_dict
```

Appendix B: Database Functions

B1: Database Creation Function

Listing 3: Functions used to create entries in the SurrealDB database

```python
import hashlib
import re
from typing import Union
import demoji
from surrealdb import Surreal
from surrealdb.ws import SurrealException, SurrealPermissionException
import datetime
from utils import wrong_input_error, convert_tuples_to_lists
import json


def get_unique_id(original_id: int, username: str, site: str):
    concatenated_str = str(original_id) + "-" + username + "-" + site
    hash_value = hashlib.sha256(concatenated_str.encode()).hexdigest()
    return hash_value


async def query(query: str, db: Surreal):
    """Execute a custom query."""
    try:
        data = await db.query(query)
    except SurrealException as e:
        if "already exists" in str(e):
            query = query.replace("CREATE", "UPDATE")
        data = await db.query(query)
    if data:
        data = data[0]["result"]
    return data


def remove_emojies(
    obj: Union[str, list, tuple, set, dict]
):  # Source: https://stackoverflow.com/questions/51217909/
   removing-all-emojis-from-text
    """Removes all emojies from a string"""
    emoji_pattern = re.compile(
        "["
        "\U0001F600-\U0001F64F"  # emoticons
        "\U0001F300-\U0001F5FF"  # symbols & pictographs
        "\U0001F680-\U0001F6FF"  # transport & map symbols
        "\U0001F1E0-\U0001F1FF"  # flags (iOS)
        "\U0001F1F2-\U0001F1F4"  # Macau flag
        "\U0001F1E6-\U0001F1FF"  # flags
        "\U0001F600-\U0001F64F"
        "\U00002702-\U000027B0"
        "\U000024C2-\U0001F251"
        "\U0001f926-\U0001f937"
        "\U0001F1F2"
        "\U0001F1F4"
        "\U0001F620"
        "\u200d"
        "\u2640-\u2642"
        "]+",
        flags=re.UNICODE,
    )
    if isinstance(obj, str):
        return (
            demoji.replace(emoji_pattern.sub(r"", obj), "")
            .replace("--", " ")
            .replace('"', "'")
            .replace("’", "'")
            .replace("\\", "\\\\")
        )
    elif isinstance(obj, (list, tuple, set)):
        return type(obj)(remove_emojies(elem) for elem in obj)
    elif isinstance(obj, dict):
        return {
            remove_emojies(key): remove_emojies(value) for key, value in
  obj.items()
        }
    else:
        return obj


async def create_with_id(table: str, custom_id: str, data: dict, db:
  Surreal):
    """
    Create a record with a specified id.
    This will raise an exception if the record already exists.
    """
    try:
        data = remove_emojies(data)
        if "url" in data:
            if " " in data["url"]:
                data["url"] = data["url"].replace(" ", "--")
        response = await db.create(table + ":" + custom_id, data)
    except KeyError as e:
        query_string = (
            "CREATE "
            + table
            + ":"
            + custom_id
            + " SET "
            + ", ".join(
                [
                    str(k)
                    + "="
                    + (
                        '"' + str(v) + '"'
                        if type(v) != list
                        else "[" + ", ".join(['"' + str(i) + '"' for i in
  v]) + "]"
                    )
                    for k, v in data.items()
                ]
            )
        )
        result = await query(query_string, db)
        return result


async def create_database_entry(
    table: str = None, values: dict = None, site: str = None, db: Surreal =
  None
):
    if table is None or values is None:
        wrong_input_error(table, str(values))
    if "pid" in values:
        original_id = str(values["pid"])
        username = str(values["uid"])
    elif "tid" in values:
        original_id = str(values["tid"])
        username = str(values["author_uid"])
        posts = values["posts"]
        values = {key: value for key, value in values.items() if key !=
  "posts"}
    elif "uid" in values:
        original_id = str(values["uid"])
        username = values["username"]
    elif "nr_of_pagination_pages" in values:
        original_id = values["url"]
        username = values["title"]
    elif "stats" in values:
        original_id = values["site"]
        username = values["title"]
    else:
        dict_str = json.dumps(values)
        with open("failed_parse.txt", "a") as f:
            f.write(dict_str + "\n")
        return
    if "site" in values:
        site = values["site"]
    elif site is None:
        raise ValueError("You have to specify the site the data is taken
  from")
    else:
        values["site"] = site
    hash_val = get_unique_id(original_id, username, site)
    values = convert_tuples_to_lists(values)
    try:
        await create_with_id(table=table, custom_id=hash_val, data=values,
  db=db)
    except SurrealPermissionException as e:
        if "already exists" in str(e):
            database_record = await db.select(table + ":" + hash_val)
            if database_record:
                database_record = database_record[0]
                hash_val = database_record["id"]
            old_date = datetime.datetime.
  fromisoformat(database_record["scrape_time"])
            new_date = datetime.datetime.
  fromisoformat(values["scrape_time"])
            if new_date > old_date:
                await db.update(table + ":" + hash_val, values)
        else:
            raise e
    # to correctly store linked records, they must be manually added not as
  strings
    if "tid" in values:
        if posts != [] and posts != " ":
            query_string = (
                "UPDATE "
                + ("threads:" + hash_val if not "threads:" in hash_val else
  hash_val)
                + " SET posts = ["
                + ", ".join(posts)
            )
```

```
            if len(posts) == 1:
                result = await query(query_string.replace("[", "") + ";")
            else:
                result = await query(query_string + "];")
        values["posts"] = posts
    return hash_val
```

Appendix C: Machine Learning Code

C1: BERT

Listing 4: Code used to train and find the optimal hyperparameters for a BERT text classification model

```
from nlp_func import get_data
from transformers import BertTokenizer
from nlp_func import get_train_test_val_split
import torch
from transformers import BertForSequenceClassification
from torch.optim import AdamW
from transformers import get_linear_schedule_with_warmup
from torch.nn.utils import clip_grad_norm_
from tqdm.notebook import tqdm
import numpy as np
import math
import pandas as pd
from torch.utils.data import TensorDataset, DataLoader, RandomSampler,
   SequentialSampler


def encode(docs, tokenizer):
    """
    This function takes list of texts and returns input_ids and
   attention_mask of texts
    """
    encoded_dict = tokenizer.batch_encode_plus(
        docs,
        add_special_tokens=True,
        max_length=512,
        padding="max_length",
        return_attention_mask=True,
        truncation=True,
        return_tensors="pt",
    )
    input_ids = encoded_dict["input_ids"]
    attention_masks = encoded_dict["attention_mask"]
    return input_ids, attention_masks


dataset_file = "./data/prodigy_datasets/categories.jsonl"
text, labels, label_names = get_data(dataset_file)

PRETRAINED_LM = "bert-base-uncased"
tokenizer = BertTokenizer.from_pretrained(PRETRAINED_LM, do_lower_case=True)
EPOCHS = 4
BATCH_SIZE = 16
LEARNING_RATE = 2e-05

df = pd.DataFrame({"text": text, "label": labels})
train_split = 0.8
validate_split = 0.1
train_df, valid_df, test_df = get_train_test_val_split(df, train_split,
   validate_split)
train_input_ids, train_att_masks = encode(train_df["text"].values.tolist())
valid_input_ids, valid_att_masks = encode(valid_df["text"].values.tolist())
test_input_ids, test_att_masks = encode(test_df["text"].values.tolist())

train_y = torch.LongTensor(train_df["label"].values.tolist())
valid_y = torch.LongTensor(valid_df["label"].values.tolist())
test_y = torch.LongTensor(test_df["label"].values.tolist())
train_y.size(), valid_y.size(), test_y.size()

train_dataset = TensorDataset(train_input_ids, train_att_masks, train_y)
train_sampler = RandomSampler(train_dataset)
train_dataloader = DataLoader(
    train_dataset, sampler=train_sampler, batch_size=BATCH_SIZE
)

valid_dataset = TensorDataset(valid_input_ids, valid_att_masks, valid_y)
valid_sampler = SequentialSampler(valid_dataset)
valid_dataloader = DataLoader(
    valid_dataset, sampler=valid_sampler, batch_size=BATCH_SIZE
)

test_dataset = TensorDataset(test_input_ids, test_att_masks, test_y)
test_sampler = SequentialSampler(test_dataset)
test_dataloader = DataLoader(test_dataset, sampler=test_sampler,
   batch_size=BATCH_SIZE)

N_labels = len(train_df.label.unique())
```

```
model = BertForSequenceClassification.from_pretrained(
    PRETRAINED_LM,
    num_labels=N_labels,
    output_attentions=False,
    output_hidden_states=False,
)
device = "cuda" if torch.cuda.is_available() else "cpu"
model.to(device)


optimizer = AdamW(model.parameters(), lr=LEARNING_RATE)
scheduler = get_linear_schedule_with_warmup(
    optimizer, num_warmup_steps=0, num_training_steps=len(train_dataloader)
  * EPOCHS
)

train_loss_per_epoch = []
val_loss_per_epoch = []


for epoch_num in range(EPOCHS):
    print("Epoch: ", epoch_num + 1)
    """
    Training
    """
    model.train()
    train_loss = 0
    for step_num, batch_data in enumerate(tqdm(train_dataloader,
  desc="Training")):
        input_ids, att_mask, labels = [data.to(device) for data in
  batch_data]
        output = model(input_ids=input_ids, attention_mask=att_mask,
  labels=labels)

        loss = output.loss
        train_loss += loss.item()

        model.zero_grad()
        loss.backward()
        del loss

        clip_grad_norm_(parameters=model.parameters(), max_norm=1.0)
        optimizer.step()
        scheduler.step()

    train_loss_per_epoch.append(train_loss / (step_num + 1))
    """
    Validation
    """
    model.eval()
    valid_loss = 0
    valid_pred = []
    with torch.no_grad():
        for step_num_e, batch_data in enumerate(
            tqdm(valid_dataloader, desc="Validation")
        ):
            input_ids, att_mask, labels = [data.to(device) for data in
  batch_data]
            output = model(input_ids=input_ids, attention_mask=att_mask,
  labels=labels)

            loss = output.loss
            valid_loss += loss.item()

            valid_pred.append(np.argmax(output.logits.cpu().detach().
  numpy(), axis=-1))

    val_loss_per_epoch.append(valid_loss / (step_num_e + 1))
    valid_pred = np.concatenate(valid_pred)

    """
    Loss message
    """
    print(
        "{0}/{1} train loss: {2} ".format(
            step_num + 1,
            math.ceil(len(train_df) / BATCH_SIZE),
            train_loss / (step_num + 1),
        )
    )
    print(
        "{0}/{1} val loss: {2} ".format(
            step_num_e + 1,
            math.ceil(len(valid_df) / BATCH_SIZE),
            valid_loss / (step_num_e + 1),
        )
    )
    model.eval()
test_pred = []
test_loss = 0
with torch.no_grad():
    for step_num, batch_data in tqdm(
        enumerate(test_dataloader), desc="Testing model...",
   total=len(test_dataloader)
    ):
        # input_ids, att_mask, labels = [data.to(device) for data in
   batch_data]
        input_ids, att_mask, labels = [data for data in batch_data]
        output = model(input_ids=input_ids, attention_mask=att_mask,
   labels=labels)

        loss = output.loss
        test_loss += loss.item()

        test_pred.append(np.argmax(output.logits.cpu().detach().numpy(),
   axis=-1))
test_pred = np.concatenate(test_pred)
```

Appendix D: NLP Pipeline

D1: Code For Running NLP Pipeline

Listing 5: Code used to run the NLP pipeline

```
from nlp_func import run_nlp_pipeline
import pandas as pd
import spacy
import torch
from nlp_func import get_tokenizer, get_torch_device, get_data
from flair.nn import Classifier
from flair.data import Sentence
import flair


def decode_numeric_text_classifier(numeric_label, label_map):
    return label_map[numeric_label]


def decode_dict_text_classifier(d):
    return max(d, key=d.get)


def get_nlp_pipeline(
    best_intent_classifier, best_category_classifier, best_ner_classifier,
   use_gpu=True
```

```
):
    device = get_torch_device(use_gpu)
    if "prodigy" in best_intent_classifier:
        intent_classifier = spacy.load(best_intent_classifier)
        intent_classifier.max_length = 2000000
    else:
        intent_classifier = torch.load(best_intent_classifier)
        intent_classifier.eval()
        intent_classifier.to(device)
    if "prodigy" in best_category_classifier:
        category_classifier = spacy.load(best_category_classifier)
        category_classifier.max_length = 2000000
    else:
        category_classifier = torch.load(best_category_classifier)
        category_classifier.eval()
        intent_classifier.to(device)
    if best_ner_classifier is None:
        ner_classifier = Classifier.load("ner-ontonotes-fast")
        ner_classifier.to(device)
    elif "PRODIGY" in best_ner_classifier.upper():
        ner_classifier = spacy.load(best_ner_classifier)
        ner_classifier.max_length = 2000000
    return intent_classifier, category_classifier, ner_classifier


def format_classification(
    text_df, intent_classifier_dataset_file,
  category_classifier_dataset_file
):
    if all(
        [
            type(el) == int or type(el) == float
            for el in list(text_df["intent_classification"])
        ]
    ):
        label_map = get_data(intent_classifier_dataset_file,
  return_label_map=True)
        label_map = {v: k for k, v in label_map.items()}
        text_df["intent_classification"] = text_df["intent_classification"].
  apply(
            lambda x: decode_numeric_text_classifier(x, label_map)
        )
    elif all([type(el) == dict for el in
  list(text_df["intent_classification"])]):
        text_df["intent_classification"] = text_df["intent_classification"].
  apply(
            decode_dict_text_classifier
        )
    if all(
        [
            type(el) == int or type(el) == float
            for el in list(text_df["category_classification"])
        ]
    ):
        label_map = get_data(intent_classifier_dataset_file,
  return_label_map=True)
        label_map = {v: k for k, v in label_map.items()}
        text_df["category_classification"] =
  text_df["category_classification"].apply(
            lambda x: decode_numeric_text_classifier(x, label_map)
        )
    elif all([type(el) == dict for el in
  list(text_df["category_classification"])]):
        text_df["category_classification"] =
  text_df["category_classification"].apply(
            decode_dict_text_classifier
        )
    return text_df


def get_text_classification_prediction(
    text, model, pretrained_lm_name=None, tokenizer=None, use_gpu=True
):
    if isinstance(model, spacy.lang.en.English):
        doc = model(text)
        return doc.cats
    else:
        if tokenizer is None:
            tokenizer = get_tokenizer(
                model.base_model_prefix.upper(), pretrained_lm_name
            )
        device = get_torch_device(use_gpu, print_message=False)
        encoded_text = tokenizer(
            text, padding=True, truncation=True, return_tensors="pt"
        ).to(device)
        with torch.no_grad():
            output = model(**encoded_text)
        prediction = torch.argmax(output.logits, dim=1).item()
        del encoded_text
        if use_gpu:
            torch.cuda.empty_cache()
    return prediction
```

```python
def get_NER_prediction(text, model, use_gpu=True):
    if isinstance(model, flair.models.sequence_tagger_model.SequenceTagger):
        sentence = Sentence(text)
        device = get_torch_device(use_gpu, print_message=False)
        sentence.to(device)
        model.predict(sentence)
        spans = sentence.get_spans("ner")
        tags = []
        for span in spans:
            span_text = span.tokens[0].form
            span_tag = span.tag
            tags.append({span_text: span_tag})
        del sentence
        torch.cuda.empty_cache()
    elif isinstance(model, spacy.lang.en.English):
        doc = model(text)
        tags = []
        for tag in doc.ents:
            span_text = tag.text
            span_tag = tag.label_
            tags.append({span_text: span_tag})
    return tags


def run_nlp_pipeline_on_row(
    row,
    intent_classifier,
    category_classifier,
    ner_classifier,
    pretrained_lm_name=None,
    tokenizer=None,
    use_gpu=True,
):
    if pd.isna(row["combined_post_body"]):
        text = row["body"]
    else:
        text = row["combined_post_body"]
    if pd.isna(text):
        if not pd.isna(row["title"]):
            text = row["title"]
        else:
            text = ""
    try:
        intent_prediction = get_text_classification_prediction(
            text, intent_classifier, pretrained_lm_name, tokenizer, use_gpu
        )
        category_prediction = get_text_classification_prediction(
            text, category_classifier, pretrained_lm_name, tokenizer,
   use_gpu
        )
        ner_prediction = get_NER_prediction(text, ner_classifier, use_gpu)
    except:
        return row
    row["intent_classification"] = intent_prediction
    row["category_classification"] = category_prediction
    row["ner_tags"] = ner_prediction
    return row


def run_nlp_pipeline(
    df,
    best_intent_classifier,
    best_category_classifier,
    best_ner_classifier,
    intent_classifier_dataset_file,
    category_classifier_dataset_file,
    model_type,
    pretrained_lm_name,
    use_gpu=True,
):
    intent_classifier, category_classifier, ner_classifier =
   get_nlp_pipeline(
        best_intent_classifier, best_category_classifier,
   best_ner_classifier, use_gpu
    )
    tokenizer = get_tokenizer(model_type, pretrained_lm_name)
    raw_classified = df.progress_apply(
        lambda row: run_nlp_pipeline_on_row(
            row,
            intent_classifier=intent_classifier,
            category_classifier=category_classifier,
            ner_classifier=ner_classifier,
            tokenizer=tokenizer,
            use_gpu=use_gpu,
        ),
        axis=1,
    )
    formatted_classified = raw_classified.copy()
    formatted_classified = format_classification(
        formatted_classified,
        intent_classifier_dataset_file,
        category_classifier_dataset_file,
    )
    return raw_classified, formatted_classified


MODEL_TYPE = "BERT"
PRETRAINED_LM_NAME = "bert-base-uncased"
best_intent_classifier = "./models/10_percent/textcat/categories/BERT/
  bert-base-uncased_4_epochs_2e-05_lr_16_batch_size.pt"
intent_classifier_dataset_file = "./data/prodigy_datasets/categories.jsonl"
best_category_classifier = (
    "./models/10_percent/textcat/content_categories/prodigy/model-best"
)
category_classifier_dataset_file = "./data/prodigy_datasets/
  content_categories.jsonl"
best_ner_classifier = "./models/NER/10_percent/PRODIGY/model-best/"
USE_GPU = True
raw_classified, formatted_classified = run_nlp_pipeline(
    df,
    best_intent_classifier,
    best_category_classifier,
    best_ner_classifier,
    intent_classifier_dataset_file,
    category_classifier_dataset_file,
    MODEL_TYPE,
    PRETRAINED_LM_NAME,
    USE_GPU,
)
```