Part I:

Collection, Pre-processing, Visualization and Analysis of Data for Developing Machine Learning based Web Security Solutions

Chapter 2: Data Capturing and Pre-processing

In this chapter, we bring out details of how the data was collected and pre-processed for building ML models.

2.1 Capturing Data for Machine Learning (ML)

In ML, the prediction is as good as the quality of the data used to train the models. Since the thesis focuses on using Artificial Intelligence (AI) for cross-device web security solutions, there is a need for high-quality data. Mainly two datasets, as per the requirements specified against each, are required to fulfill the scope of this thesis.

- **Dataset of Benign and Malicious Webpages:** The data should be recently collected and should have an acceptable representation of the both the classes (at least 1.5-2% representation of the 'malicious' class), and should capture as many relevant attributes as possible.

- **Dataset of Android hybrid Apps:** This too should have been collected recently from the Google Play store and should have an acceptable representation of malicious Android hybrid apps.

From where could we source the datasets listed above? Were they readily available on the Internet? Were the datasets that were already available complete, credible, and met the quality requirements? While few similar datasets were hosted on the Internet, none could meet the requirements of data considered for the study. Thus, due to the unavailability of suitable datasets, data collection became an important aspect. Further, pre-processing of a dataset is as essential as other steps in a ML process. This chapter is devoted to the collection and pre-processing of data for this thesis. Analysis and visualization of the datasets prepared is given in Chapter 3. Details of the pre-processing steps for the two datasets, including code extracts, are given in Appendices A and B, respectively.

Data collection from the Internet requires a web crawler. Like Google bot, the best of the crawlers can reach only an estimated 4% of the total Internet for indexation [37]. Over the years, this percentage has increased, but the non-indexed Darknet has proliferated even further, keeping this proportion nearly constant. Even in the indexed web, the pro-rata share of malicious webpages is less than 1%, as per the Sitelock report [5]. So, if a generic crawler is used for data collection, the malicious webpages will comprise less than 1% of the dataset and represent less than 0.04% of the total malicious webpages present. Thus, a focused crawler was required that could seek more malicious websites to collect a better share of malicious webpages on the Internet for this research. Moreover, this focused crawler was expected to have the necessary capabilities of handling the complexities associated with crawling and downloading a malicious webpage. These requirements led us to develop 'MalCrawler', a crawler for seeking and crawling malicious webpages. MalCrawler was developed as part of the thesis and is the backbone of the data collection carried out. Section 2.2 will detail the design, working, and performance of MalCrawler.

The dataset on 'Malicious and Benign Webpages', which was collected using MalCrawler, is described in Section 2.3, Chapter 3 (preliminary analysis and visualization) and Appendix A (pre-processing code).

The dataset on 'Hybrid Android apps', which too was prepared for this thesis, is described in Section 2.4, Chapter 3 (preliminary analysis and visualization) and Appendix B (pre-processing code).

2.2 MalCrawler

These days, the Internet has become a source of information, entertainment, and e-commerce, resulting in its augmented utilization. With this increased use, the reprobaters have become quite active. The web crawlers, in this context, are making a fruitful contribution towards web security nowadays. Focused web crawlers are being used for a variety of purposes to maintain the safety of the Internet. They have been used extensively for scientific research, apart from being used for selective search indexing. As part of the work of the thesis, one such focused web crawler has been developed with the objective to search malicious web pages efficiently and accurately. While the need for MalCralwer was prompted by the absence of any credible datasets, the benefits of MalCrawler go beyond being a mere crawler for data collection.

Malicious websites are now a major concern in the field of cybersecurity. As per Annual Internet Threat Report published by Symantec, web attacks using malicious websites have increased [12]. Out of the billions of URLs present on the Internet, one in every 1,126 websites is infected with malware. Drive-by download web attacks by malicious websites, in which the attack malware gets downloaded as the user clicks and browses the web page, are a major threat to internet security [38]. Conventional security measures, such as antivirus based on signature detection, have limited success in today's scenario [39]. Drive-by download attacks from malicious web pages that compromise attacks through the browser route on computers/mobiles have become the primary means of spreading infected code [39]. With ever-increasing web traffic, the vulnerability from this type of attack is also on the rise. Generally, cybersecurity agencies and antivirus firms are on the lookout for such malicious sites. These agencies typically maintain a blacklist of infected websites and keep updating them by crawling new sites. These agencies also analyze the malwares to develop an antivirus signature or prevention mechanism for the type of attack detected. As it emerges from this discussion, web crawling is the first step towards seeking such sites. This section describes the focused crawler, named "MalCrawler", developed specifically to identify malicious websites efficiently and accurately. Comparing MalCrawler to a generic crawler used for search indexing, the following aspects are highlighted:

- MalCrawler finds more malicious webpages compared to the existing web crawlers. If the probability of encountering malicious webpages by a generic search crawler is 'x', then the probability of finding a malicious webpage by this crawler is '$a*x$', where $a>1$.

- MalCrawler is capable of handling cloaking (websites which cloak, show a different webpage to a web crawler and a different page to a user's browser). It uses various tricks to detect whether the HTTP request has come from a web crawler or a web browser. The commonly used trick is to read the User-Agent string of the HTTP request. MalCrawler sends multiple HTTP requests to a website, staggered in time, with different User-Agent strings, one being of a popular search bot and others of common browsers. Thus, it would detect cloaking by comparing the response to these HTTP requests.

- Generally, sites with extensive content, for example, social networking sites, have a very complex graph of hyperlinks. Such sites are also targets of cross-site scripting attacks by hackers. Thus, these sites may have malicious code implanted by such attacks. Many small and big loops exist in such sites, which can entangle a crawler and leave it stuck for a long time. MalCrawler is designed to avoid such entanglements while doing a detailed crawl in depth. Therefore, MalCrawler will be able to handle entanglements [40].

- Nowadays, AJAX has become a prevalent web technology (Asynchronous JavaScript and XML- AJAX is a web technology for showing dynamic webpages using JavaScript). Since we are looking for malicious webpages and malwares in them, we need to probe a little deep and hence cannot avoid AJAX content. With its ability to crawl deep while avoiding entanglement, MalCrawler is capable of managing webpages with AJAX content well.

To ensure that the crawler spends more time crawling malicious websites, MalCrawler has an advisory engine, which uses various information to advise the crawler. The advisory engine keeps track of the URLs being crawled currently, and if the crawler gets entangled in the site, it helps the crawler recover from it. The advisory engine also advises the crawler on the breadth and depth of crawl for the website.

Certain aspects of the crawler design were considered while designing this focused crawler. Firstly, the crawler needs to crawl and reach malicious web pages. Crawling the complete web is difficult. How to crawl to reach maximum malicious pages in minimum time? Secondly, most malicious content is hidden deep into websites and is only accessible through queries to their linked databases. How to crawl deep to catch malicious code? Thirdly, on many websites, mainly social media portals, malicious content is found in dynamic pages that use JavaScript extensively. Social media sites are generally not crawled by commercial search crawlers. These sites are nowadays used extensively for malicious drive-by-download attacks by injecting malicious JavaScripts. How to crawl social media sites with dynamic content? A generic web crawler cannot do these jobs, and thus, we require a specially designed crawler. MalCrawler meets all these requirements. Subsequent sub-sections (2.2.1 to 2.2.6) describe these capabilities of MalCrawler.

2.2.1 Maximizing Malicious Page Seek Rate

Malicious page seek rate (α) is defined as:

$$\left(\frac{Malicious\ Pages\ Crawled}{Total\ Pages\ Crawled}\right) \times 100 = \alpha \qquad (2.1)$$

The main objective is to maximize α for a crawl. As the Internet is vast, we might end up wasting time crawling benign sites when we start looking for malicious content. Crawling the entire Internet is infeasible, and the crawl cycle may take months. If we take a long time looking for malicious websites, the malicious content might get removed by that time. So, how do we maximize α? Few strategies used in this work to maximize α are given below:

- **Starting Crawl with Malicious Seed:** Usually, crawling begins with a seed of URLs. Using the initial seed of URLs, the crawler follows the hyperlinks and crawls further. We can start our crawl from known malicious sites. Since a malicious site is more likely to host hyperlinks to other malicious sites [41], this approach, as compared to random crawling, gives us a higher probability of encountering malicious sites.

- **Seeking Dynamic Content:** Dynamic content is more likely to contain malicious code. Dynamic content uses many server-side scripts and client-side scripts (JavaScripts) to provide an interactive intuitive web experience. While this is appealing to users, and the web is embracing it (also popularly called Web 2.0) , it is also vulnerable to hacks like cross-site scripting [40]; and thus, ends up hosting malicious drive-by-download JavaScript code. Therefore, if more malicious content is sought, such sites with dynamic content need to be crawled more.

- **Smart Filtering:** With smart filtering techniques, we reduce the number of pages that need to be examined in detail [42]. This fast filter uses a static heuristics algorithm to identify and discard pages likely to be benign quickly.

2.2.2 Crawls Deep to Detect Hidden Pages

A vast amount of web pages are placed deep and are generally difficult to be handled by crawlers. It has been observed that malicious content is found deep in website structure, especially in dynamic pages linked with databases, as they are more vulnerable to cross-side scripting [40]. A traditional crawler avoids crawling in-depth and will not submit dynamic queries during the crawl. On the contrary, MalCrawler, designed as a deep web crawler, checks for dynamic content and submits queries [43]. It crawls deep while taking precautions to avoid getting stuck or entangled.

2.2.3 Uses Anti-cloaking Measures

With time, malicious sites have evolved and become smart. They sense the user agent trying to send them HTTP request and accordingly guide them to different pages on a site. For example, if the malicious site gets a HTTP request from a 'Mozilla Firefox Browser', it guides it to a malicious page, and if it receives a HTTP request from the Google bot, it shows a benign page. They show malicious pages to users (who are targeted victims) and different benign pages to crawlers (search engines and malware detection firms). MalCrawler is capable of detecting cloaking and, it does this by manipulating the user agent field of the HTTP request.

2.2.4 Emulates Different Browsers

Some malwares hosted on sites are targeted for a particular browser and environment [44]. For example, there might be a targeted malware for 'Internet Explorer 8.0'. The malicious site may not redirect us to the malware location until it is sure that we have 'Internet Explorer 8.0'. Thus, the focused crawler should be able to emulate all browser environments. We have implemented emulation for popular browsers like 'Internet Explorer', 'Mozilla Firefox', and 'Google Chrome'.

2.2.5 Emulates a User

Crawling speed and pattern should be similar to a user. Patterns available from web usage statistics could be emulated. To compensate for the slow pace of crawling, parallelization could be adopted, and multiple sites could be crawled simultaneously. However, it has been designed to keep browsing patterns similar to any user's browsing style (like opening breadth-first links, etc.).

2.2.6 Crawls AJAX Pages while Avoiding Entanglement

Web 2.0 AJAX concept has made browsing more interactive and integrated to backend databases. A link to a weblog is expected to point to a perennially changing page. The depth of such dynamic sites varies and increases as more links are added by user actions. How deep should a crawler crawl such sites? If the crawler goes too deep, then the crawler might get entangled. MalCrawler has used the crawler guidance engine to guide the crawler through such sites and prevent it from getting tangled.

The design of "MalCrawler" has been validated on the Internet by implementing it as a Java application. Subsequent sub-sections will provide the background and literature survey, methodology and design, testing methodology, results and analysis of the crawler.

2.2.7 MalCrawler: Background and Literature Survey

Web crawling for search indexing is an extensively researched field, which has now matured to a certain extent. The search engines like Google, Yahoo, Bing, etc., are a testimonial to this advancement.

Web crawling for web content mining of malicious webpages is somewhat similar to crawling for search indexing, barring a few differences. These significant differences are listed below:

- **Importance of Links to Pages:** Forward and back links from a website to specific suspicious sites indicate malicious activity.

- **The Seed for Crawling:** A good selection of malicious seeds (Seed refers to the initial set of URLs from where crawl starts) can help us find more malicious webpages, as pointed out by Invernizzi et al. in their paper on EVILSEED [41].

- **Crawling Dynamic Content:** Dynamic websites are more prone to cross-site scripting and SQL injection attacks [40],[45]. Thus, they become the primary source of the spread of malware. The depth of crawl in dynamic content also varies. At times, the crawler is expected to increase the depth, and at times it has to keep it shallow, programmed in a way that it avoids getting entangled. The crawler needs to avoid entanglements and still be effective.

- **Bot Detection by Malicious Web Sites:** Malicious sites nowadays have increased awareness of bots that crawl them for malware detection. They cloak and provide a different benign page when a bot visits [46]. The crawler needs to crawl in a manner to evade such detection.

The process of finding malicious web pages involves many steps [47]. In the first step, the crawler needs a point to start the search, i.e., seeds. In the second step, the initial URLs are crawled, downloaded, and parsed to extract the hyperlinks. These hyperlinks can be sorted out and queued in priority based on the crawler design. In the third step, the crawl is expanded to all those queued URLs. While the pages are being queued and crawled, fast filters can tell us whether the crawler is finding more malicious sites or not [42],[48]. For the fourth step, we need detection systems with high accuracy in predicting the maliciousness of web pages. For this, 'Honey Clients' can be used [49] [50]. 'Honey Client' based systems use browser emulation, client emulation, and various other methods to detect malicious code. Such 'Honey Clients' are very accurate but are slow by magnitudes. Resources for checking malicious websites are neither free nor infinite. Thus, the number of URLs given to such

'Honey Clients' should be reduced by making the first three steps more efficient. MalCrawler has been explicitly designed to meet this requirement.

2.2.8 MalCrawler: Design and Methodology

The overall system architecture of MalCrawler is depicted in **Figure 2.1**.

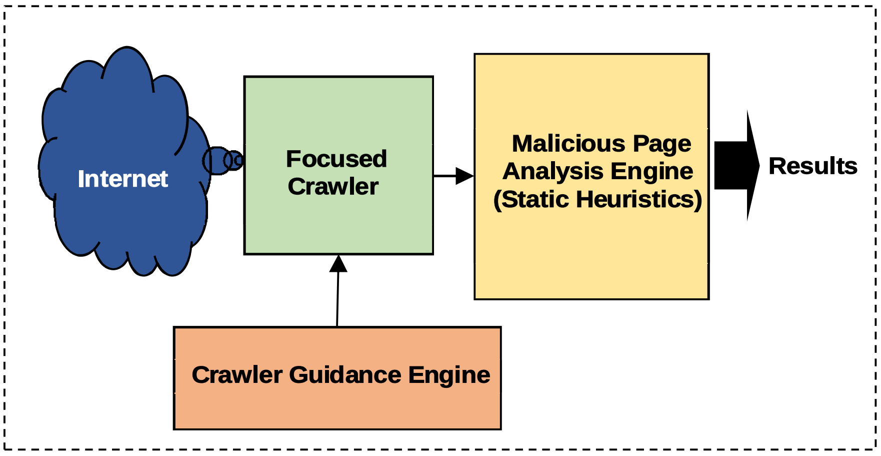

Figure 2.1: Architecture of MalCrawler

Focused Crawler Module. This module is capable of carrying out focused crawling (focused crawling refers to crawl covering specific category of webpages). It has the following sub-modules:

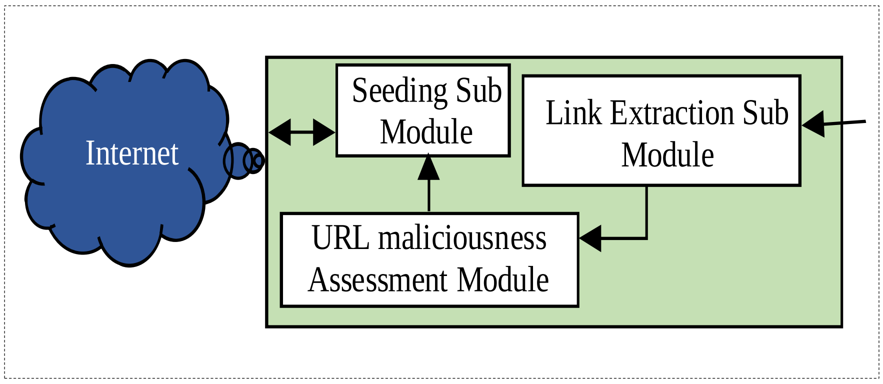
Internet
Seeding Sub Module
Link Extraction Sub Module
URL maliciousness Assessment Module

Figure 2.2: Focused Crawler Module

- *Seeding Sub Module:* The seeding module provides the initial seed for the crawl. In crawlers, the initial seed is the set of links and webpages from which a crawler starts crawling. As it crawls, it extracts links from all pages visited and provides them to a FIFO queue. The crawler, after that, keeps picking up URLs from the FIFO queue and keeps crawling. This is how the process of crawl starts and continues. Since we have made a focused crawler crawl malicious sites, the initial seed is chosen to be malicious as proposed in EVILSEED [41]. We have taken the initial seed from the website 'Malware Domain List' [51]. After that, the crawl proceeds by following the links extracted from the URLs provided in the initial seed. The links extracted are processed, as described in the next paragraph, and the crawled malicious URLs are added to the FIFO queue for the crawl.

- *Link Extraction Sub Module:* The link extraction sub-module is responsible for extracting URL links from the pages crawled by the crawler. It extracts all these links and stores them in a linked list for evaluation. After that, the URL maliciousness assessment module works on these links to select the malicious links. Links that are not likely to be malicious are dropped.

- *URL Maliciousness Assessment Module:* The URL maliciousness assessment module is responsible for picking up the URLs and assessing whether they are malicious or not using keyword analysis (keywords in URLs are compared with a list of keywords generally found in malicious websites).

The crawler was written with the help of JSoup [52] Java library. JSoup provides a very convenient API for extracting and manipulating data, using the best of DOM, CSS, and jquery-like methods. Though the crawler was written from scratch in Java, the JSoup library has been used extensively for parsing functions, extractions, etc.

Crawler Guidance Engine. The crawler guidance engine is the one that controls the crawler. It does the following:

- Keeps track of the URLs crawled and being crawled.
- Keeps track of crawl time.
- Recovers crawler when it gets entangled or stuck.
- It decides the depth and breadth of the crawl.

This module works closely with the crawler module. The crawler module updates its state in the guidance engine by using various local and global variables. Every time the crawler sends an HTTP GET request, it updates the variable. The crawler module keeps a close watch on the time spent on a domain and its URL. It keeps checking the depth and the breadth of the crawl using the URL separators. If the crawl-depth goes beyond the laid down limit or the crawler is taking too much time, the guidance engine resets the crawler and resumes crawling of a different domain. We have not used parallelization while designing this crawler to avoid complexity. However, the application is multithreaded, with each module running on a separate thread.

Malicious Page Analysis Engine. The malicious page analysis engine analyzes the pages crawled to identify maliciousness. Here, primarily an analysis of JavaScript-based malwares is done. The HTML Unit Browser [53] and Rhino Emulation library [54] have been used in this. Rhino engine and HTML Unit emulator check and analyze the following aspects (as listed below) of JavaScript-based malwares. Features have been extracted from these aspects and classified using the WEKA library [55] (using C4.5 [56] Decision Tree classifier) to make predictions for URLs being crawled.

Redirection and cloaking: Most malicious websites use redirections and cloaking. Redirection is used to guide the browser to a page containing the exploit code. Cloaking is generally used to avoid showing malicious pages to search engine crawlers or to serve different pages for different vulnerable browser environments (based on browser & OS fingerprinting results). We used the two features described below to detect these activities:

- *Feature 1- Redirections.* There are two types of redirections. Firstly, HTTP response status 302 redirection, and secondly, redirection using JavaScript 'document.location' property. The number of these redirections and the URLs where these redirections landed were recorded. The HTTP status 302 redirection can be checked by the response received by the browser (in this case, the HTML Unit [53] emulated the browser). And, 'document.location' based redirect was picked up by running the script using Rhino JavaScript Engine [54].

Thus, this feature was recorded as described above with nominal values {Yes, No}.

- *Feature 2- Cloaking.* We detect cloaking by manipulating the user agent field of the HTTP request. We change user agent strings to emulate 'Mozilla Firefox', 'Google Chrome', 'Internet Explorer', 'Google Bot', etc. If we receive a different response every time, then it indicates cloaking. This feature will be limited to the values {Yes, No}. HTML Unit has been used for detecting cloaking. It has been used to taper the User-Agent field in the HTTP request header for depicting various browsers.

De-obfuscation: Generally, malicious JavaScript is found to be obfuscated. In obfuscation, JavaScript is encoded/encrypted to avoid detection. This obfuscated code is de-obfuscated only at the run time [57] [58]. Varieties of obfuscation techniques are used, e.g., base64; also, encryption is often used. No matter how they are encoded/encrypted, JavaScripts have to be decoded/decrypted at runtime for execution. Thus, runtime analysis has been used to detect de-obfuscated code. Following five features have been extracted for this task:

- *Feature 3- String Definitions and their Use in Code.* The JavaScript code is de-obfuscated at runtime using the eval() function in the Rhino emulator and is thereafter analyzed. The number of JavaScript functions used to define new strings, e.g., substring(), fromCharCode(), etc., and number of their uses, e.g., document.write() and eval() were counted. The significant presence of such functions indicates maliciousness. The feature has a numeric value from 0 to 5.

- *Feature 4- Number of Dynamic Code Executions.* The number of function call runs for dynamic interpretation of JavaScript code were measured (e.g., DOM changes using document.createElement(), document.write(), eval() and setTimeout()) [59]. This was checked by running the code in

Rhino sandbox [54]. This feature has a numeric value depicting the number of dynamic code executions.

- *Feature 5- Length of Dynamically Evaluated Code.* The eval() function is used for dynamic execution in JavaScript. Thus, the string length passed in eval() function as an argument was captured. This value depicts the length of dynamically evaluated code. The length of dynamic code passed is a good indicator of the maliciousness of obfuscated JavaScript (malicious scripts generally have high lengths). This value is picked up while running JavaScript code in Rhino sandbox. This feature has a numeric value depicting the code length.

- *Feature 6- Bytes Allocated in Memory Space.* Memory space allocated to string functions like concat() is monitored at runtime. Heap exploitation techniques generally allocate large memory [58]. For example, heap spraying attacks may allocate up to 100MB of space. This feature is captured as a numeric value using the Rhino sandbox.

- *Feature 7- Number of Likely Shell Code Strings.* Exploits those target memory violation vulnerabilities which attempt to execute shellcode. Shellcode can be embedded in the JavaScript or be dynamically created. For identifying shellcode, the script was parsed, and Unicode encoded non-printable character strings longer than a certain threshold were extracted (we took 128 bytes as the threshold). Rhino [54] with custom Java code was used for this task. The feature has numeric values 0, 1, 2…., etc., based on the number of shellcode strings found.

Exploitation: The last step of any malware attack is exploitation. The following three features were analyzed for checking exploitation:

- *Feature 8- Number of Instantiated Browser Components.* Browser components (plug-ins) are checked for instantiation. Exploits generally use vulnerabilities in such components.

HTML Unit [53] was used to emulate the browser environment and the exploitation actions. This feature has a numeric value depicting the number of instantiated components.

- *Feature 9- Attribute and Parameter Values in Method Calls.* Exploits generally use long strings to cause a buffer overflow. Thus, values passed in method calls were tracked, as large string values may indicate buffer flow. This has been analyzed using Rhino sandbox and custom Java code encapsulating Rhino. The feature is nominal with values {Yes, No} describing the presence or absence of such attributes or parameters in method calls.

- *Feature 10- Sequence of Execution of Method Calls.* The sequence of execution of method calls is a strong indicator of the good or ill intent of the code. Many sequences (like file download followed by execution) are known to indicate malicious intent. This feature was analyzed using both Rhino and HTML Unit. The feature is nominal with values {Yes, No} describing the presence of such a malicious sequence of calls.

2.2.9 MalCrawler: Software Design

The software design of MalCrawler is based on the modular architecture described above. The complete application has been designed as a standalone application on Java SE 7. The various modules are running on multiple threads to improve responsiveness. Postgres SQL has been used as the database for this application. The Java application connects to the Postgres SQL server using the JDBC connector. The thread-level architecture of the application is given in **Figure 2.3**.

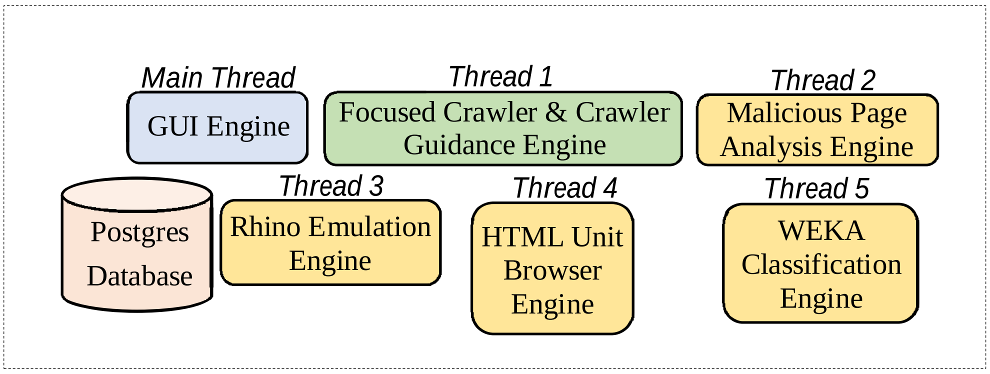

Figure 2.3: Thread Wise Software Design of MalCrawler

The crawler has been developed on the Java SE platform. The source code of the crawler is hosted online on GitHub to support further research [60]. The complete architecture is modular, and the following libraries have been used in the design (more details on these software and libraries is given in Appendix C):

- *JSoup Library:* The JSoup [52] library is a Java-based library with web page parsing capability. This library has been used for parsing web pages and extracting - hyperlinks, document content, and JavaScript tags.

- *Rhino JavaScript Emulation Library:* Rhino [54] is also a Java based library that can run JavaScript. It has been used to run JavaScript in a sandboxed environment for analyzing runtime behavior.

- *HTML Unit Browser Emulation Library:* The HTML Unit [53] is a browser emulation library based in Java. It has been used to emulate a browser session. Certain features can be tested only by emulating a Browser session, e.g., redirection, cloaking, etc. It has been used for such testing.

- *WEKA Data Mining Library:* The WEKA [55] Data Mining library has been used to classify URLs being visited. A C4.5 [56] Decision Tree model is trained using a known list of malicious webpages. This trained model is then used to predict the maliciousness of the URLs being crawled by the MalCrawler.

2.2.10 MalCrawler: Result and Analysis

To test the MalCrawler, as per the scope defined in sub-sections 2.2.1 to 2.2.6, it must be validated for maximizing malicious page seek rate α, crawling deep, using anti-cloaking measures, emulating browsers and crawling AJAX content. To validate 'α', the Google Safe Browsing API was used [61]. Pages crawled by MalCrawler were cross-checked with the Safe Browsing API to determine the number of malicious webpages downloaded. Other metrics, viz., crawling depth, number of cloaking incidents, number of AJAX instances, were measured using custom Java code. The MalCrawler application was hosted on the Internet for crawling. The application log (collected in log file) and webpages downloaded (in Postgres database) were analyzed. Summary of the results obtained is given in **Table 2.1**.

Table 2.1: Summary of Results- MalCrawler

Parameter	Result
Results with Focused Crawling Off	
Websites Crawled with focused crawling off.	567,898*
Malicious sites visited with focused crawling off.	702
Malicious Page Seek Rate (α) with focused crawling off.	0.123 %
Results with Focused Crawling Turned On	
Websites Crawled with focused crawling on.	567,544*
Malicious sites visited with focused crawling on.	11,388
Malicious Page Seek Rate (α) with focused crawling on.	2.01 %
Results of Entanglement Avoidance	
Entanglement with crawl depth set to 3	4
Entanglement with crawl depth set to 5	17
Entanglement with crawl depth set to 8	56
Results of Anti-cloaking	
Number of websites where cloaking detected	93
Results of User Emulation	
Browsers Emulated	Mozilla, Chrome, and Internet Explorer
Regulating crawling speed	Speed regulated by guidance engine
Results of Handling AJAX Sites	
AJAX sites handled	7866
Depth to which handled with 0% entanglement	3
Depth to which handled with 50% entanglement	8
*Note: Due to limited computing resources available for this test, crawl had to be stopped at about 0.57 million records.	

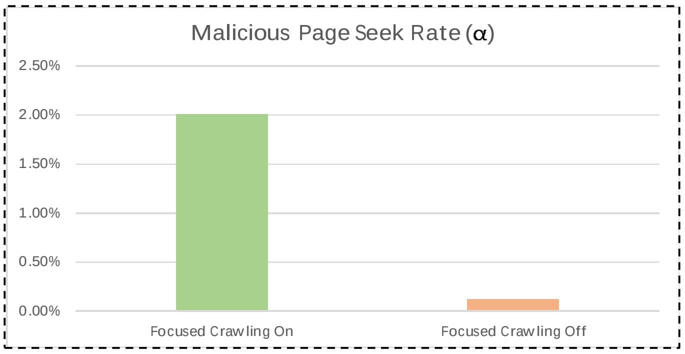

Figure 2.4: Malicious Page Seek Rate ( )

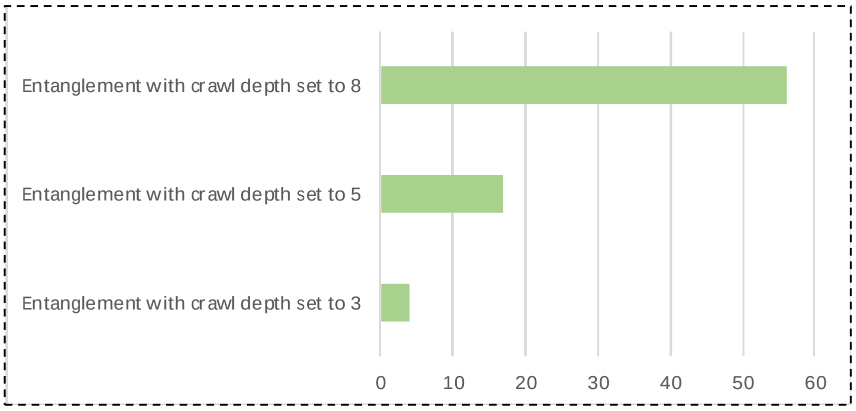

Figure 2.5: Entanglement Avoidance

The malicious page seek rate was better with focused crawling turned on. Thus, this focused crawler is able to seek more malicious pages than benign pages. The focused crawler exhibited good entanglement avoidance. The Crawler Guidance Engine was able to pull it out successfully in most situations. The logs showed that whenever the entanglement happened, it lasted only a few milliseconds, and after that, it extracted itself successfully and continued crawling. The crawler detected cloaking successfully on many sites. The crawler could emulate both the user environment and its browsing behavior successfully. Various browsers like Mozilla, Chrome, and Internet Explorer were emulated. AJAX content was handled well by the crawler as very few entanglements were logged while crawling AJAX webpages. Further, the crawler could send and handle XMLHttpRequest (XHR) the way it is done while opening an AJAX site in a browser, thereby emulating human behavior while opening an AJAX site.

Evasion: To escape detection by MalCrawler, the intruder can manipulate some of its features by detecting the crawler's visits. Also, the IP address of the crawler can be tracked by cyber attackers. To mitigate the risk of evasion from detection by the attackers, the crawl can be done from various IP addresses or using multiple proxies.

Comparison with EVILSEED Crawler: EVILSEED, an approach to improve the effectiveness of searching malicious web pages, was proposed by Invernizzi et al. [41]; where the researchers use a set of malicious seeds. Using this information, the researchers utilized the infrastructure of a search engine and Google's blacklist URLs to retrieve new malicious URLs. Comparing it to MalCrawler, it is seen that MalCrawler is not dependent on an external search engine or a blacklist. Further, MalCrawler has specific capabilities not seen in EVILSEED, viz., its ability to detect and handle cloaking, handle AJAX content, crawl deep, and emulate various browsers.

2.3 Pre-processing of the Webpages Dataset

MalCrawler, which was described in the previous section, was developed to crawl the Internet for preparing the webpages dataset. This dataset was required for the web security research carried out in the thesis (Chapter 4, 5, 6 and 8 utilize this dataset). The webpages downloaded using MalCrawler over a period of one month were pre-processed using custom Python code. The pre-processing code, description, and analysis of the dataset are described in detail in Chapter 3 and Appendix A. The metadata of this dataset is presented in **Table 2.2**.

Table 2.2: Summary of Webpages Dataset

No of Samples:	1.564 million
No of Classes:	2 ('benign', 'malicious')
Sample Distribution:	‘benign’- 1,172,747 (97.73%) ‘malicious’- 27,253 (2.271%)
Number of Attributes (excluding class label):	10/25*
Format:	CSV
***Note: -** A different version of this dataset was also compiled in ARFF (Attribute Relational File Format). This has 25 attributes and has been utilized in Chapter 4 for carrying out a comparative analysis of attributes. Details of this dataset are available in Chapter 4. The dataset with ten attributes which is listed in this table, is a subset of that larger dataset, from which the ten most relevant attributes (as evaluated in Chapter 4) have been selected.	

2.4 Pre-processing of the Hybrid Android Apps Dataset

Chapter 7 of the thesis analyses web security on mobile platforms, particularly the hybrid apps that run on the Android platform. The hybrid apps dataset was prepared for carrying out this analysis. Custom Python code was used to collect the APKs from the Internet. APK (Android Package Kit) is the format in which Android apps are distributed. This dataset was prepared from the following sources- Android Malware dataset 2017 (CICAndMal2017) [62], Android Application dataset for Malware Application [63], and Android Anti Malware dataset [64]. As these data sources did not have a common set of attributes, these datasets were processed using customized Python code, and some relevant attributes were added and irrelevant attributes deleted. Attributes not available in these datasets were extracted after downloading the APKs of these apps from a mirror of Google Play named 'APK Combo' [65] as Google Play does not permit downloading of APKs by bots. For this reason, a mirror site that allows downloading using bots was used. For disassembling the hybrid apps downloaded from the 'APK Combo' mirror, JADX Disassembler [66] was used. Further details on pre-processing (including code) and analysis of the dataset are given in Chapter 3 and Appendix B. Metadata about the dataset is given in **Table 2.3**.

Table 2.3: Summary of Hybrid Apps Dataset

No of Samples:	78,767
No of Classes:	2 ('benign', 'malicious')
Sample Distribution:	'benign'- 76955 (97.7%) 'malicious'- 1,812 (2.3%)
Number of Attributes (excluding class label):	12
Format:	CSV

2.5 Conclusion

This chapter covered aspects of capturing and pre-processing data. Further details of datasets are given in Chapter 3 (Preliminary analysis and visualization), Appendix A (pre-processing details of webpages dataset along with code) and Appendix B (pre-processing details of hybrid apps dataset along with code). This was a significant facet of the research, as the results of any AI-based analysis depend entirely on quality, comprehensiveness, and correctness of data. A unique focused crawler, named MalCrawler, was developed to collect web content. As evident from this chapter, the utility of MalCrawler goes beyond a mere data collection agent. MalCrawler, which seeks more malicious websites and handles them efficiently compared to generic crawlers, has immense potential in the field of web security. It can be used for requirements where malicious sites are either to be searched or avoided. Also, it can be used by search engines to make the search experience safer, and can be used by Internet security firms to discover new malwares.

The datasets that have been prepared for use in later chapters, can facilitate further research in the field of web security. Hence, they have been hosted online in the public domain.

Regarding future scope, MalCrawler may be improved further to incorporate the following features: parallelization of crawler instances, adaptive crawl speed to emulate various user responses, web response for CAPTCHA and login modules, and support for cloud deployment and scalability.