Classifying Dark Web Executables Using Public Malware Tools

Brianna Stewart
Louisiana Tech University
briannaolivia1005@outlook.com

Brandon Vessel
Louisiana Tech University
brandon.vessel@outlook.com

William Bradley Glisson
Louisiana Tech University
glisson@latech.edu

Abstract

The proliferation of malware in today's society continues to impact industry, government, and academic organizations. The Dark Web provides cyber criminals with a venue to exchange and store malicious code and malware. Hence, this research develops a crawler to harvest source code, scripts, and executable files that are freely available on the Dark Web to investigate the proliferation of malware. Harvested executable files are analyzed with publicly accessible malware analysis tool services, including VirusTotal, Hybrid Analysis, and MetaDefender Cloud. The crawler crawls over 15 million web pages and collects over 20 thousand files consisting of code, scripts, and executable files. Analysis of the data examines the distribution of files collected from the Dark Web, the differences in the results between the analysis services, and the malicious classification of files. The results reveal that about 30% of the harvested executable files are considered malicious by the malware analysis tools.

Keywords: Dark Web, web crawler, malware analysis, hybrid analysis, public analysis tools.

1. Introduction

Cybersecurity is a major concern for the government, businesses, and the public (Biden, 2021). As society becomes increasingly dependent on technology and the Internet, the concern for data, systems, and network security continues to escalate. A decade ago, IBM announced that data would become a new natural resource for the 21st century (IBM, 2013).

Cybersecurity Ventures expressed that the attack surface for cyber threats has expanded extensively due to the increase in new technology (Cybersecurity Ventures, 2022). The attack surface includes everything from computer systems and networks to Internet of Things (IoT) devices like home automation systems and appliances, industry devices, and healthcare devices.

The Federal Bureau of Investigation's Internet Crime Report states that malware resulted in a total loss of 9,326,482 dollars in 2022 (Federal Bureau of Investigation, 2022). The 2023 Federal Bureau of Investigation's Internet Crime Report states that the Internet Crime complaint center received 880,418 complaints and estimates that the losses from these complaints will exceed $12.5 billion (Federal Bureau of Investigation, 2023). The report goes on to state that investment fraud, e-mail compromises, scams, and ransomware all contribute to the types of crimes contributing to the cost of cybercrime.

Over the years, illicit markets on the Dark Web have been an area of interest to researchers. These markets include hacking tools (Cherqi et al., 2018), attack services (Hyslip & Holt, 2019), firearm purchasing (Holt & Lee, 2023), stolen data (Holt et al., 2016), illicit drugs (Liggett et al., 2020) and illegal services (Campobasso & Allodi, 2023; Roddy & Holt, 2022). It is estimated that between 2011 and 2017 the revenues from anonymous online markets were around $15M (Van Wegberg et al., 2018).

The escalating impact of malware on organizations and the growth of the Dark Web prompts the hypothesis that crawlable executable files on the Dark Web have utility for malware analysis using VirusTotal, Hybrid Analysis, and MetaDefender Cloud. The hypothesis raises two questions: 1) Can executable files be collected from the Dark Web? 2) Can publicly accessible malware analysis tools characterize acquired executable files?

The contribution of this research paper is two-fold. First, the results show that about 30% of the executable files acquired from the Dark Web are portrayed as malicious by the malware analysis tools. Secondly, it provides necessary foundational research to perform additional malware analysis on files acquired from the Dark Web in the future.

The structure of the paper is as follows: Section II introduces relevant dark web crawling and hybrid analysis research. Section III describes the methodology used in this controlled experiment. Section IV presents the data, analyzes the results, and explores observations. Finally, Section V concludes and presents future work.

2. Literature review

While malware detection, identification, and analysis are of common interest in academics and to practitioners (Fitzgerald, 2023; Herron et al., 2021; Hightower et al., 2021; Luckett et al., 2018; Nguyen et al., 2017; Nguyen et al., 2020), research continues to investigate Dark Web crawlers (Aikhatib & Basheer, 2019; Boshmaf et al., 2023; Dalvi et al., 2021) and hybrid analysis tools (Ijaz et al., 2019; Jamalpur et al., 2018).

2.1 Dark Web crawlers

Previous work includes a variety of web crawlers that harvest data on Dark Web pages. Dalvi et al. (2021) implement a Dark Web crawler called SpyDark that utilizes The Onion Router (Tor) Browser to crawl a fixed set of pages based on keywords and site depth. The author states the crawler implements a text-filtering algorithm and eliminates irrelevant pages using Artificial Neural Network classifiers. The crawler analyzes the results through link activity and generates graphical representations of collected hyperlinks.

Aikhatib and Basheer (2019) create a web crawler called Darky that crawls the Dark Web markets for product and vendor information using the Tor Browser with Privoxy virtual private network software to provide security and anonymity. The authors state their web crawler lacks automatic generation of login credentials and login bypass. Aikhatib and Basheer manually create the credentials for the web crawler. The web crawler features bypass of the login CAPTCHA verification using XPath and optical character recognition analysis. If the bypass fails, the web crawler allows for manual verification. The web crawler iterates through each product category to collect individual product information and vendor information such as name, price, origin, reviews, and number of sales. Their results contained 179 total vendors and 6,387 total products in 16 hours. Digital items, drugs/chemicals, and guides/tutorials are the top three categories observed.

Boshmaf et al. (2023) develops an open-source crawler for onion services called Dizzy. Dizzy crawled 63,267,542 web pages. The crawler hashes images to identify and group them, extracts public blockchain transactions, identifies wallets, and calculates cash flow for those wallets. Overall, the research provides data on domain operations, web content, cryptocurrency usage, and a web graph.

2.2 Hybrid analysis tools

Ijaz et al. (2019) apply static and dynamic analysis techniques on executable binary samples and then use the resulting reports for machine learning. For dynamic analysis, the authors use Cuckoo Sandbox, while for static analysis, they use the Python library PEFILE. From both analysis approaches, the authors utilize several machine learning methods and apply them to the analysis results. The highest percentage accuracy is the Gradient Classifier, with an area under the curve at 95%, where the false positive rate is 5.6% and the false negative rate is 14%. The authors state that the dynamic environment analysis failed to be efficient since several samples detected the environment. In addition, performing static analysis before the execution of malware is not efficient due to obfuscated samples.

Jamalpur et al. (2018) perform static and dynamic malware analysis on a malware sample. The authors use VirusTotal to inspect the properties of the sample and then use DependencyWalker and IDAPro to analyze the code of the sample. They create an isolated environment using Cuckoo Sandbox to perform a dynamic analysis of the sample. The environment consists of an Ubuntu Linux Cuckoo host, Windows XP and Windows 7 Cuckoo guests, and the Cuckoo agent to communicate between the host and guest systems. A distributed firewall with updated iptables is added as additional protection for the host. The Cuckoo guest systems use the VirtualBox hypervisor. After malware execution, they discover the sample was ransomware that created a message for the user on how to pay the ransom and decrypt their hard drive. It encrypts files on the machine with certain extensions, deletes all shadow volume files, and displays the instruction message again.

The aforementioned research spans hybrid analyses of previous experiments on malware using various techniques and tools. However, minimal research exists that implements a web crawler to specifically harvest executable files, source code, and scripts from the Dark Web and use publicly accessible hybrid malware analysis tools to analyze the harvested files.

3. Methodology

To investigate the hypothesis that crawlable executable files on the Dark Web have utility for malware analysis, the research approach was divided into two parts: (1) crawler development and data collection and (2) hybrid malware analysis. The first part explains how the Dark Web crawler functions. The crawler developer applied an incremental software development approach to program the Dark Web crawler (Karlsson, 2001). The incremental model includes four stages: requirements and analysis, design, coding and implementation, and testing. This part details how the crawler harvested executable files and recorded statistics. The second part outlines the application of malware analysis tools which include VirusTotal, Hybrid Analysis, and MetaDefender Cloud. The harvested data from the crawler provides the input for a controllable experiment as defined by (Shadish et al., 2002) for the hybrid malware analysis.

3.1 Crawler development

The objective of the Dark Web crawler is to identify any executable files in pages on the Dark Web. The crawler is designed to collect executable files, source code, and scripts. For this research endeavor, all harvested file types are considered for data collection, while only executable files are considered in scope for hybrid malware analysis. The crawler is designed in Go and uses supplementary Python and Bash scripts (Google; Python Software Foundation; Ramey, 2023). Go was chosen due to its lightweight nature with native concurrency and libraries that balance ease of use with performance (Rouse, 2017).

A virtual machine was created in VMware vSphere Client to deploy the crawler program (Broadcom). Throughout the development and testing of the crawler program, specific amounts of hardware were dedicated to the execution of the crawler. Tor was used to access the Dark Web (Tor Project). VSCodium was utilized as a code editor to develop the crawler. The tools and hardware used to implement the web crawler are presented in Table 1.

Table 1. Development toolset

Hardware & Software	Version
Virtual Machine	- VMware vSphere Client v7.0.3.00500 - Intel Xeon Gold 5318Y CPU @ 2.10GHz x70 processor - Ubuntu 20.04 LTS (64-bit) OS - 256 GB Memory - 2 TB Hard Disk
Go	1.18.2 linux/amd64
Python	3.8.13
GNU Bash	5.0.17(1)-release
Tor	0.4.2.7
VSCodium	1.75.1

The incremental software development model steps for developing the crawler are presented below.

For step one, the objective of the crawler project is to identify any files that can be executed, including executable files, source code, and scripts. Source code and scripts are included in the data collection since they can have executable forms. The files collected from the crawler do not bypass login restrictions. The definition of the objectives and scope includes the outline of phases to complete the project.

The primary logic of the crawler is developed from steps two to five. To efficiently crawl executable files without the concern of resource exhaustion, the crawling approach developed in step two revolves around MIME types found in the HTTP response body of web pages. This approach minimizes resource usage and optimizes the crawling process.

Next, the goal is to extract onion links from the page content for step three. Onion links are a specific domain that can only be reached using the Tor (Tor Project). Using Go's IO library to download a web page as a stream of bytes and a regular expression (regex) parser from the Regex module, onion links can be identified and extracted from the page content.

In step four, the logic for the classification of page content and executable files is defined. To classify the page content and any possible executable files, it needs to correspond to a parsable MIME type. Once the MIME type is parsed, the logic to determine if the MIME type is a valid executable file MIME type is implemented. Valid MIME types include the following:

- Executable files like Portable Executable (PE), Executable Linkable Format (ELF), Apple Disk Image files (DMG), Mach object files (Mach-O), etc.
- Source code like Python, Java, Go, Perl, etc.
- Scripts like Borne Again Shell (Bash), Z Shell scripts, etc.
- Additional file types like OS data and compressed files

Only source code, scripts, and executable files are downloaded for further analysis, while any other file types are considered out of scope for the purposes of this research. The SHA256 hashing function from the Crypto module is used to hash the onion links to avoid links that point to unwanted content. The module is also used to name the valid files as their respective hashes, resulting in a naming system without collisions. In step five, a proxy system is integrated using Tor daemons to ensure anonymous and secure connections. A SOCKS5 proxy is added to the Go HTTP clients. During this step, a queue system is created to manage link processing. New links are prioritized, while timed-out links are moved to a secondary queue. The timed-out links are processed at a later time with a longer timeout. The entire system functions as a complete priority queue, which optimizes system speed and prevents potential disruptions caused by dead links.

Additional features such as statistics and monitoring are added in step six. The statistics are displayed at specified intervals to the terminal and to a Discord channel via webhook to provide the live progress of the crawler over time. The Discord module relays crawler statistics to a predefined Discord webhook that provides automated updates. The webhook is attached to a Discord server where the data statistics are posted to a text channel in the server. The statistics are saved for future reference and analysis.

Steps seven to ten included various aspects of crawler optimization and improvements. In step seven, live tests and adjustments are conducted to optimize the queue system's structure. These adjustments ensure effective handling of excessive dead links from large web pages or quick responses. For step eight, to optimize the resource utilization and performance of the crawler, live code tests are performed, and adjustments to the number of Tor daemons and workers are made to determine the best parameters based on the CPU cores and bandwidth of the machine.

Two major improvements to the crawler are added in step nine. Recognizing that executable files are often stored in compressed or archived files, a decompression daemon system is implemented to harvest more executable files. This system enables immediate decompression or downloading of files into memory to determine valid executable contents without issues like zip bombs or irrelevant files. The Archiver module assists in extracting files from those archives. The Ranger module saves resources by partially parsing archives without downloading the entire archive using the HTTP request body. The second improvement to the crawler is a queue observing system that continuously monitors and manages the queue to optimize memory usage and maintain efficient crawling operations. The system removes links that reached maximum page limits, links that were unresponsive, or dead links.

In the final phase, the crawler program is deployed to a dedicated server. Specifications of the dedicated server are shown in Table 1. The storage of the dedicated server is adjusted to support the data harvested from the crawler. The overall process of the crawler program consists of the engine and workers. The core of the program is called the engine and contains the main execution loop, while the workers are individual processes. The engine employs the workers to utilize concurrent threads to scrape, enqueue, and process new links.

3.2 Data collection

The development toolset information is available in Table 1. The source links to start the crawler are gathered from a custom crawling script for onion and Tor-related subreddits. To start the data collection process, any existing Tor daemons are killed. A custom script to create new daemons is executed. Given the amount of Tor daemons, the script creates a Tor configuration file for every daemon and creates a bash script to start the daemons. After three to five minutes, the latest version of the source code is built and executed. The wait time allows the Tor daemons to start up completely.

The data collection consists of two executions. The first initial data collection occurred for about a week. The data collection ended earlier than expected due to Tor daemons reaching timeout in domains. To fix this issue, a domain timeout was added to prevent the crawler from getting stuck on pages with massive amounts of links. Each domain has a maximum number of pages crawled before it is purged from the queue. After resolving the issue, the crawler was executed again. The second data collection occurred for approximately three weeks. After there was no change in collected executable files for three days, data was extracted from the crawler.

The Discord application is used throughout data collection to monitor the crawler statistics from another device. The program outputs the statistics of the crawler every minute to the Discord channel in a private server via webhook. The statistics include information about the data collected and the crawler's efficiency.

3.3 Hybrid malware analysis

To determine the executable files for analysis, the collected files needed to be categorized. The Linux file, grep, sort, and uniq commands were used to categorize the collected files. The *file* command is used to determine the file type, and the output is piped into the *awk* command to parse the file type name. The resulting output is piped into the *sort* command to sort the file types, then the *uniq* command with the count option is used to count each file type. The output from the commands is used to determine the executable files in the data. They are copied from the entire dataset to make a new dataset for hybrid analysis. To determine malicious behavior about the executable files, the analysis services (VirusTotal), (Hybrid Analysis), and (MetaDefender Cloud) are used to analyze each executable file.

Source code or script files are not used since these files are not in their executable form. Therefore, they are out of scope for this analysis. VirusTotal (VT), Hybrid Analysis (HA), and MetaDefender Cloud (MD) were chosen due to their public accessibility, automation ability using Application Programming Interface (API) access, and request limits. To automate the process of analyzing each executable and its results, Python is used to create scripts utilizing each analysis service's API.

The analysis algorithm features three primary methods for all analysis services: hash querying, executable file uploading, and analysis retrieving. The SHA256 hash of the executable file is used to query the analysis service's database, and the status of the analysis is recorded as queried. If the file hash is found in their database, then the analysis of the file is saved. The analysis status is marked as completed, and the algorithm iterates to the next executable file. If the file hash is not identified in their database, it is recorded as unknown. The file is uploaded to the analysis service, and the analysis status is marked as uploaded. If the analysis of the file is completed, then the analysis of the file is saved, and the analysis status is recorded as completed. If the analysis of the file is not completed, then the analysis script is executed to retrieve the missing analysis information and complete the analysis for that file. Once the analysis is saved, the algorithm iterates to the next executable file.

To simplify analysis and data clustering, the results were saved as a dictionary in a YAML file. The format made it easier to index specific values of the analysis to identify and cluster data for the results of the experiment. Each analysis file's name is the executable file's name with the .yml file extension. SHA256 hash was used to hash the file. Since there were two data collections, there are two datasets from which the executable file originated. Hence, the *dataset_id* field can either be 1 or 2, with 1 representing the first data collection and 2 representing the second collection. The analysis from each analysis service is saved in separate sections of the file.

An *analysis_id* field is used to identify the results after file upload. Executable files that have been identified via hash analysis have a null *analysis_id* value since file upload is not required to analyze the file. Valid values for the status field of the analysis are queried, uploaded, or completed. The *is_hash_known* field is used to identify if the analysis services are aware of the file using its SHA256 hash. The data from the analysis service is saved in the *analysis_data* field.

For the hash query method, if the request limit is reached, the program sleeps or exits, depending on which limit is exceeded. The request limits for VT are 4 requests per minute, 240 per hour, and 500 per day. VT is the only analysis service that combines the file reputation and file submission request limits. With HA, the file reputation request limits are 200 scans per minute and 2,000 scans per hour with no daily limit. Meanwhile, HA allows 100 file submissions per day with no minute or hour limits. For MD, 1,000 scans for file reputation are allowed daily with no minute or hourly limits. However, 10 submissions are allowed per minute for file submission requests, while 100 file submissions are allowed daily.

If the request limit has not been reached, the program loads the dictionary from the analysis file and determines if the hash has already been queried to the analysis service. The hash will iterate to the next file if it has been queried. If it has not been queried, it queries the hash to the analysis service, changes the analysis status field to queried, and logs the results in a log file. Each entry in the log file contains the time and date of the action, the file hash, the performed action, and the HTTP code response of the performed action. If the hash is found in the analysis service's database, then the *is_hash_known* field is changed to true, status is changed to completed, and *analysis_data* is changed to the data saved from the query to the analysis service. The analysis file is overwritten with changes to the dictionary. This process repeats until all hashes are queried to the analysis service.

For the upload file method, the structure of the process is like the hash query method with some minor differences. First, the request limit is checked to ensure it has not been exceeded. Using the dictionary in the analysis file, the *is_hash_known* field is checked. If the *is_hash_known* field is true, then the program iterates to the next file. Otherwise, it uploads the file to the analysis service. Each analysis service has differences for uploading files. For example, when uploading a file to VT, if the file size was larger than 32 MB, then a large upload link was required to upload the file. HA requires an environment ID when uploading a file. Additionally, HA had a file size upload limit, so files larger than 100 MB were excluded from uploading. Once a file is uploaded, the *analysis_id* field is changed to the analysis ID returned in the response from the request, and the analysis status field is changed from queried to uploaded. The analysis file is updated with changes to the dictionary. The process repeats for all files until all unidentified files have been uploaded to each analysis vendor.

To gather the results from the file uploads, queries are made to retrieve the analysis data from each analysis service. The request limits are checked, and the program reads the contents of the analysis file. If the analysis status field is uploaded, then a request is made to the analysis service and the results are saved into the *analysis_data* field. The status field is updated to completed, and the updated dictionary is saved to the analysis file. The process repeats until all uploaded files have analysis results.

To cluster the data and analyze the results, several functions are written to identify data relations between analysis services such as the number of files found and not found, file classification (benign, malicious, suspicious, etc.), and analysis service comparison. The scripts index the dictionary in each analysis file to gather data for analysis.

The *count_known_hashes* function counts the number of hashes known or not known for each analysis service and writes the results to a dictionary. The function iterates through all the analysis files and sums the *is_hash_known* field in the resulting dictionary. The *get_file_classification_info_and_details* function iterates through all the analysis files and determines the classification of the file using the *analysis_data* field. The possible classifications for each file are benign, malicious, suspicious, error, and timeout. The *file_classification_count* dictionary records the sum of each classification per analysis service. Analysis services provide different levels of analysis, so the *file_classification_info* dictionary holds the additional details from each analysis service. When a file is classified, the count is increased for that classification. The SHA256 hash of the file and its classification is updated as a key-value pair in the *hash_to_classification_mapping* dictionary. The file hash to classification mapping is used to perform additional analysis for other functions. The file classification method varies for each analysis service.

VT reports the results of each scan engine's indications. There can be any number of engines, some well-known or some that are not. Since VT does not specifically determine maliciousness through cumulative engine results, a threshold value is implemented to determine maliciousness and avoid false positive classified malicious files. The function iterates through the engines and their results. If 15% or more engines report the file as malicious, then the file is classified as malicious.

Additionally, the threshold approach is implemented for errors and timeouts with a 25% value. Hence, if 25% or more engines report the file with an error or timeout, then the file is classified with an error or timeout, respectively. VT provides additional data such as the imported libraries and functions and exports for PE and ELF files. MD specifies the file classification in their analysis.

If a file is reported as infected and has a malware family, malware type, or threat name, then it is classified as malicious. If the file is reported as suspicious, then it is classified as suspicious. If the file is reported with no threat detected, then it is classified as benign. If the results are unknown, the scan is aborted, or the scan has failed, then it is classified as an error. MD reports additional data such as malware family, type, and threat. Like MD, HA reports file classification.

If the file is reported as malicious and has a malware family, reported as no specific threat and has a malware family, or reported as malicious and has no malware family, then it is classified as malicious. If the file is reported as suspicious, then it is classified as suspicious. If the file is reported as having no specific threat and does not have a malware family, reported as no verdict, or reported as whitelisted, then the file is classified as benign. If the file is reported with an error, then the file is reported as an error. HA specifies additional information such as malware families.

An error occurred when saving the exports for PE and ELF files from the analysis data. The export name was over 128 characters for the dictionary key in the YAML file. The implicit block mapping was converted to an explicit block mapping due to the maximum character length being reached. To resolve this issue, the maximum string length was changed from 128 to 1,024 characters in the *yaml* module configuration files.

The *count_agreements_between_analysis_services* function uses the *hash_to_classification_mapping* dictionary to compare the file classification results between analysis services. For the malicious, suspicious, and benign classifications, the valid comparisons are displayed below in Table 2.

Table 2. Agreements between analysis services

Agreement	Definition
None	None of the analysis services agreed on classification
Only VT Only HA Only MD	Only one analysis service classified differently
Only VT and HA	Only VT and HA agreed on classification
Only VT and MD	Only VT and MD agreed on classification
Only HA and MD	Only HA and MD agreed on classification
All	All the analysis services agreed on classification

The *sort_file_classification_info_by_frequency* function sorts the *file_classification_info* dictionary by value. This function provides better visualization of the additional information from each analysis service.

The *count_malicious_executables_per_os* function sums the number of malicious executables per operating system for each analysis service. The function uses the *hash_to_classification_mapping* dictionary to read the file classification. Then, it uses the *file* and *awk* commands to determine the type of executable. MS-DOS, PE32, and PE32+ are classified as Windows OS while ELF files are Linux OS and Mach-O files are Mac OS.

4. Results and analysis

In total, two crawls were conducted to collect executable files, source code, and scripts. In the first crawler execution, 8,717,948 pages on the Dark Web crawled and 68 target files are downloaded. Table 3 shows the results of the first crawler execution. As mentioned earlier, the first execution of the crawler was unresponsive due to Tor daemons reaching timeout in domains, resulting in a significantly smaller number of executable files.

In the second crawler execution, more files are discovered. 7,152,104 pages on the Dark Web are crawled and 184,657 executable files are downloaded. Table 4 shows the number of crawled resources for the second execution. The total number of crawled pages and downloaded files are shown in Table 5.

Table 3. First crawler execution results

First Crawl	
Crawled pages	8,717,948
Downloaded files	68
Unique files	68

Table 4. Second crawler execution results

Second Crawl	
Crawled pages	7,152,104
Downloaded files	184,657
Unique files	20,000

Table 5. Total crawler execution results

Combined Crawls	
Crawled pages	15,870,052
Downloaded files	184,725
Unique files	20,068

Figure 1 displays the distribution of file types crawled. Source code files are the biggest category of collected files at about 47%. The biggest source code language collected is Java at 8,823 files with Python following at 959 files. The smallest source code language collected is Assembly at one file. The smallest category of collected files is executable files at about 5%. The most collected executable file type is ELF files at 635 files while the smallest is Mach-O files at 24 files. Scripts and text files are the next biggest collected file categories at about 7% and 39%, respectively.

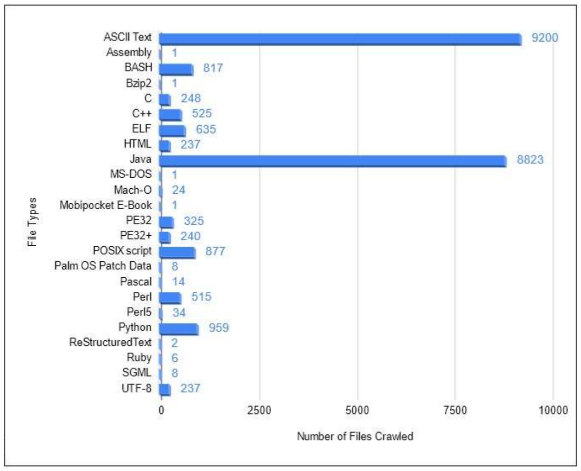

Figure 1. Total distribution of file types crawled

Executable files categorized from the crawled target files are analyzed using the publicly accessible analysis tools: VirusTotal (VT), Hybrid Analysis (HA), and MetaDefender Cloud (MD) Cloud. There are 1,225 total executable files in the entire dataset. Three duplicate files changed the total number of executable files to 1,222 files. The duplicate files were identified by SHA256 when combining both datasets into one directory. Custom scripts are made to cluster the analysis data collected from the analysis services. Linux has the most executable files at 635 files while Mac has the smallest executable files at 24 files. Windows is in the middle of both at 563 files.

Each file hash is queried to each analysis service. Either the file hash is known or not known in each analysis service's database. If the file hash is known, then the analysis results are retrieved. Otherwise, the file is uploaded to the service for further analysis. Figure 2 shows the number of hashes known and not known to each analysis service.

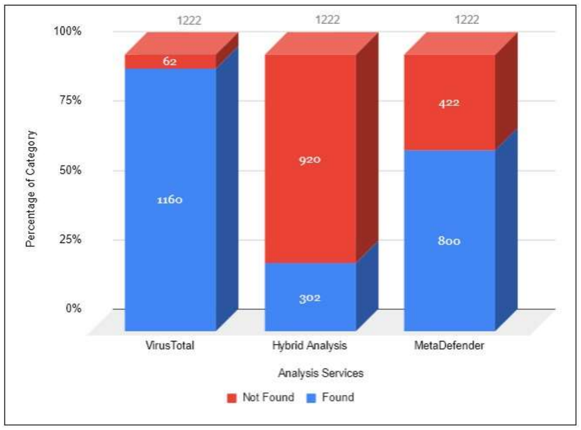

Figure 2. Hashes from each analysis service

As seen in Figure 2, VT has the most identified file hashes out of the three analysis services. MD has the second most identified file hashes while HA has the least identified file hashes. VT did not identify approximately 5% of the file hashes submitted while HA and MD did not identify approximately 75% and 34% of the file hashes, respectively. Figure 3 presents the file classification counts from each analysis service.

Each analysis service reports different classifications for the entire dataset. VT reports approximately 24% of the files as malicious, while HA and MD report approximately 35% and 32% of the files as malicious. Only HA and MD report files as suspicious, with 12% from HA and 2% from MD. HA encounters the most errors with 30% of the files during classification. The results of the classification are interesting since only 30% of the executable files are classified as malicious and malware contributes to the Dark Web's bad reputation. This result could be because not all of the Dark Web contains illegal content. (Kaur & Randhawa, 2020; Nazah et al., 2020)

Upon further investigation of the large error amount, it is determined that it is due to inconclusive analysis. HA already knows about the files, but it could not conclude the classification of the files due to insufficient analysis. VT classifies 75% of the files as benign, while HA and MD classify 21% and 61% of the files as benign, respectively. VT is the only analysis service to report timeout errors at less than 1% of the entire dataset.

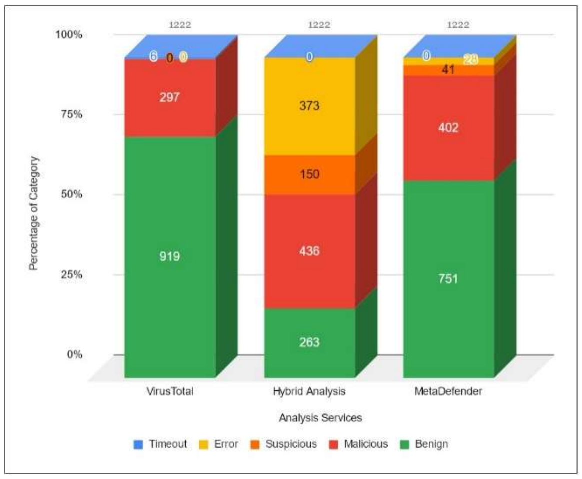

Figure 3. Total distribution of file classification

Figure 4 displays the breakdown of malicious classified files per operating system. Out of the entire dataset, Windows is classified as the operating system with the most malicious files. Linux follows Windows with a smaller number of malicious files, while Mac has no malicious files. Only VT reports malicious files for Windows, while HA and MD report malicious files for Windows and Linux. None of the analysis services report malicious files for Mac.

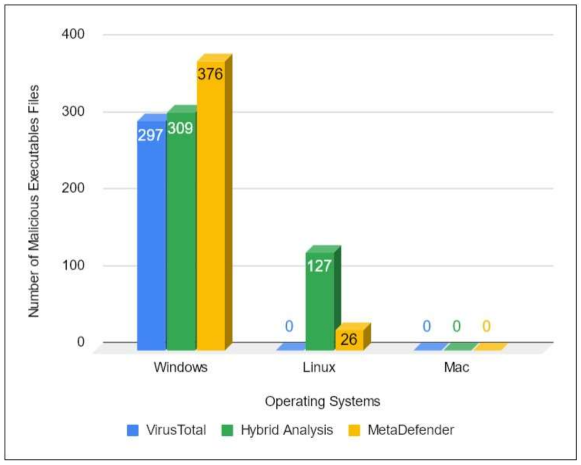

Figure 4. Distribution of malicious files

Although all vendors provide classification details about each executable file, not all vendors provide the same level of detail. Table 6 shows the top 5 malware families that were identified from HA. Most of these malware families are classified as cryptocurrency miners. Table 7 shows the top 5 malware families identified from MD. Most of the identified malware families are risk tools, crypto-miners, general malware, and risk ware.

Table 6. Top 10 Hybrid Analysis malware families

Top 10 Malware Families	Amount
CoinMiner.KA potentially unwanted application	112
Win64/CoinMiner.GG potentially unwanted application	52
Win64/CoinMiner.MW potentially unwanted application	24
Win64/CoinMiner.JI potentially unwanted application	21
RiskTool.CryptoMiner	17

Table 7. Top 10 Metadefender malware families

Top 10 Malware Families	Amount
Risktool	95
Coinminer	68
Monero	68
Malware	65
Riskware	60

VT did not report a category for malware families but included data about imported functions and libraries for PEs. VT reports that the top 287 imported functions utilized by malicious files have been used 297 times. As observed from Figure 3, VT reports 297 of the files as malicious. So, all the malicious files from VT use the same top 287 imported functions. Those malicious files use a variety of functions. PE functions are used for common tasks but could be used by malware for malicious purposes such as privilege escalation, recording keystrokes and displays, data exfiltration, process injection, and persistence. All the malicious PE files from VT use ADVAPI32, IPHLPPAPI, KERNEL32, SHELL32, USER32, WS2_32, and MSVCRT Dynamic-Link Libraries (DLLs). Most of the malicious files use CRYPT32, SETUPAPI32, and GDI32 libraries.

To investigate the agreements between each analysis service on malicious files, the results of each analysis service were compared. All the analysis services disagree on 285 malicious files, while all the analysis services agree on 148 malicious files. HA and MD agree the most on malicious files with 72 files. VT and MD agree on the second most common, with 59 malicious files, while VT and HA agree on the third most common, with 29 malicious files. For each analysis service, only HA classifies 169 files as malicious differently compared to the other analysis services. MD and VT classify differently 57 and three malicious files, respectively.

All the analysis services disagree the most for 645 suspicious files. Only HA and MD agree on four suspicious files. Only HA classifies 146 files differently and only MD classifies 27 files differently.

All the analysis services agree that 176 files are benign, while all of them disagree that 161 are benign. VT and MD agreed the most on benign files at 261 files. VT and HA agree the second most at 16 benign files while HA and MD agree the third most at one benign file. Only VT classifies the most files differently at 129 benign files, HA classifies differently the second most at 60 benign files, and MD classifies differently the third most at 17 benign files.

5. Conclusion and future work

The proliferation of malware in today’s society continues to impact industry, government, and academic organizations. The web crawler successfully harvested executable files from the Dark Web. The results from the experiment indicate that public malware analysis tools can agree on classifying executable files as malicious and benign. All three analysis services detect an approximate average of 30% of the entire dataset as malicious files. Only Hybrid Analysis and MetaDefender Cloud classified an approximate average of 7% of the dataset as suspicious files. All three analysis services identified an approximate average of 52% of the dataset as benign files. The most identified malware family from Hybrid Analysis and MetaDefender Cloud is the crypto miner family. The results from VT reveal that common DLLs are used to perform malicious actions. The Windows operating system had the most malicious files with an average of 327 files between all analysis services. Data collected from this research supports the hypothesis that crawlable executable files on the Dark Web have utility for malware analysis.

The research conducted provides the foundation for future research on Dark Web crawlers and malware analysis. Future research will investigate the graphical representation of the pages crawled. This could depict the percentage of the Dark Web crawled compared to the actual surface of the Dark Web. Additional work will investigate expanding the source links for the crawler to provide a comprehensive dataset. The research will also investigate the application of classes of malware seen in cyber attacks and on the surface web in conjunction with the malware seen on the Dark Web. The idea is to identify and investigate trends to determine if the Dark Web is ahead or behind in malware development and distribution.

Future research will investigate acquired source code, scripts, and their executable forms for malicious functionality using static analysis and publicly accessible analysis tools such as VirusTotal, Hybrid Analysis, and MetaDefender Cloud. In addition to these research activities, future work will explore extending the crawler to bypass login restrictions to collect more data and tailoring the crawler to acquire data from the Dark Web in the areas of drug trafficking, human trafficking, sale of stolen weapons, and identity theft. More work will explore extending the scope of the crawler to non-Dark Web pages to analyze the impact of malicious executable files on regular users.

12. References

Aikhatib, B., & Basheer, R. (2019). Crawling the Dark Web: A Conceptual Perspective, Challenges and Implementation. *Journal of Digital Information Management*.

Biden, J. (2021). *Executive Order on Improving the Nation’s Cybersecurity* https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/

Boshmaf, Y., Perera, I., Kumarasinghe, U., Liyanage, S., & Jawaheri, H. A. (2023). *Dizzy: Large-Scale Crawling and Analysis of Onion Services* Proceedings of the 18th International Conference on Availability, Reliability and Security, Benevento, Italy.

Broadcom. *VMware Docs: The vSphere Client*. https://docs.vmware.com/en/VMware-vSphere/7.0/com.vmware.vsphere.vm_admin.doc/GUID-588861BB-3A62-4A01-82FD-F9FB42763242.html

Campobasso, M., & Allodi, L. (2023). Know your cybercriminal: Evaluating attacker preferences by measuring profile sales on an active, leading criminal market for user impersonation at scale. 32nd USENIX Security Symposium (USENIX Security 23),

Cherqi, O., Mezzour, G., Ghogho, M., & Koutbi, M. E. (2018, 9-11 Nov. 2018). Analysis of Hacking Related Trade in the Darkweb. 2018 IEEE International Conference on Intelligence and Security Informatics (ISI),

Cybersecurity Ventures. (2022). *2022 Official Cybercrime Report*. Esentire.

Dalvi, A., Paranjpe, S., Amale, R., Kurumkar, S., Kazi, F., & Bhirud, S. G. (2021). SpyDark: Surface and Dark Web Crawler.

Federal Bureau of Investigation. (2022). *Internet Crime Report 2022*.

Federal Bureau of Investigation. (2023). *Federal Bureau of Investigation Internet Crime Report*. https://www.ic3.gov/Media/PDF/AnnualReport/2023_IC3Report.pdf

Fitzgerald, J., Mason, T., Mulhair, B., Glisson, W. B. (2023). Exploiting a Contact Tracing App to Attack Neighboring Devices.

Google. *The Go Project*. https://go.dev/

Herron, N., Glisson, W. B., McDonald, J. T., & Benton, R. K. (2021). Machine learning-based android malware detection using manifest permissions.

Hightower, J., Glisson, W. B., Benton, R., & McDonald, J. T. (2021). Classifying Android Applications Via System Stats. 2021 IEEE International Conference on Big Data (Big Data),

Holt, T. J., & Lee, J. R. (2023). A crime script model of dark web firearms purchasing. *American journal of criminal justice*, *48*(2), 509-529.

Holt, T. J., Smirnova, O., & Chua, Y. T. (2016). Exploring and estimating the revenues and profits of participants in stolen data markets. *Deviant Behavior*, *37*(4), 353-367.

Hybrid Analysis. https://www.hybrid-analysis.com/

Hyslip, T. S., & Holt, T. J. (2019). Assessing the capacity of DRDoS-for-hire services in cybercrime markets. *Deviant Behavior*, *40*(12), 1609-1625.

IBM. (2013). *What will we make of this moment? 2013 IBM Annual Report*. IBM.

Ijaz, M., Durad, M. H., & Ismail, M. (2019). Static and Dynamic Malware Analysis Using Machine Learning.

Jamalpur, S., Navya, Y. S., Raja, P., Tagore, G., & Rao, G. R. K. (2018). Dynamic Malware Analysis Using Cuckoo Sandbox.

Karlsson, E.-A. (2001). Incremental Development — Terminology and Guidelines. In *Handbook of Software Engineering and Knowledge Engineering* (pp. 381-400).

Kaur, S., & Randhawa, S. (2020). Dark Web: A Web of Crimes. *Wireless Personal Communications*, *112*, 2131-2158.

Liggett, R., Lee, J. R., Roddy, A. L., & Wallin, M. A. (2020). The dark web as a platform for crime: An exploration of illicit drug, firearm, CSAM, and cybercrime markets. *The Palgrave handbook of international cybercrime and cyberdeviance*, 91-116.

Luckett, P., McDonald, J. T., Glisson, W. B., Benton, R., Dawson, J., & Doyle, B. A. (2018). Identifying stealth malware using CPU power consumption and learning algorithms. *Journal of Computer Security*, 1-25.

MetaDefender Cloud. https://metadefender.opswat.com/

Nazah, S., Huda, S., Abawajy, J., & Hassan, M. M. (2020). Evolution of Dark Web Threat Analysis and Detection: A Systematic Approach. *IEEE Access*, *8*, 171796-171819.

Nguyen, T., McDonald, J. T., & Glisson, W. B. (2017). Exploitation and detection of a malicious mobile application.

Nguyen, T., McDonald, J. T., Glisson, W. B., & Andel, T. R. (2020). Detecting repackaged android applications using perceptual hashing.

Python Software Foundation. *Python*. https://www.python.org/

Ramey, C. (2023). *The GNU Bourne-Again SHell*. https://tiswww.case.edu/php/chet/bash/bashtop.html

Roddy, A. L., & Holt, T. J. (2022). An assessment of hitmen and contracted violence providers operating online. *Deviant Behavior*, *43*(2), 139-151.

Rouse, J. (2017). *Why Go is skyrocketing in popularity*. Opensource. https://opensource.com/article/17/11/why-go-grows

Shadish, W. R., Cook, T. D., & Campbell, D. T. (2002). *Experimental and quasi-experimental designs for generalized causal inference*. Houghton Mifflin and Company.

Tor Project. *Tor Project*. https://www.torproject.org/

Tor Project. *What is a .onion or what are onion services?* . https://support.torproject.org/onionservices/onionservices-2/

Van Wegberg, R., Tajalizadehkhoob, S., Soska, K., Akyazi, U., Ganan, C. H., Klievink, B., Christin, N., & Van Eeten, M. (2018). Plug and prey? measuring the commoditization of cybercrime via online anonymous markets. 27th USENIX security symposium (USENIX security 18),

VirusTotal. https://www.virustotal.com/gui/home/upload