Journal of Cybersecurity and Privacy

MDPI

Article

Machine Learning Detection of Cloud Services Abuse as C&C Infrastructure

Turki Al lelah *, George Theodorakopoulos *, Amir Javed and Eirini Anthi

School of Computer Science and Informatics, Cardiff University, Cardiff CF24 4AG, UK; javeda7@cardiff.ac.uk (A.J.); anthies@cardiff.ac.uk (E.A.)

* Correspondence: allelaht@cardiff.ac.uk (T.A.l.); theodorakopoulosg@cardiff.ac.uk (G.T.)

Abstract: The proliferation of cloud and public legitimate services (CLS) on a global scale has resulted in increasingly sophisticated malware attacks that abuse these services as command-and-control (C&C) communication channels. Conventional security solutions are inadequate for detecting malicious C&C traffic because it blends with legitimate traffic. This motivates the development of advanced detection techniques. We make the following contributions: First, we introduce a novel labeled dataset. This dataset serves as a valuable resource for training and evaluating detection techniques aimed at identifying malicious bots that abuse CLS as C&C channels. Second, we tailor our feature engineering to behaviors indicative of CLS abuse, such as connections to known CLS domains and potential C&C API calls. Third, to identify the most relevant features, we introduced a custom feature elimination (CFE) method designed to determine the exact number of features needed for filter selection approaches. Fourth, our approach focuses on both static and derivative features of Portable Executable (PE) files. After evaluating various machine learning (ML) classifiers, the random forest emerges as the most effective classifier, achieving a 98.26% detection rate. Fifth, we introduce the "Replace Misclassified Parameter (RMCP)" adversarial attack. This white-box strategy is designed to evaluate our system's detection robustness. The RMCP attack modifies feature values in malicious samples to make them appear as benign samples, thereby bypassing the ML model's classification while maintaining the malware's malicious capabilities. The results of the robustness evaluation demonstrate that our proposed method successfully maintains a high accuracy level of 84%. In sum, our comprehensive approach offers a robust solution to the growing threat of malware abusing CLS as C&C infrastructure.

Keywords: cloud-based and public legitimate services; malware; command and control; Portable Executable; dataset; malware detection; machine learning; feature selection; adversarial attack

Citation: Al lelah, T.; Theodorakopoulos, G.; Javed, A.; Anthi, E. Machine Learning Detection of Cloud Services Abuse as C&C Infrastructure. *J. Cybersecur. Priv.* **2023**, *3*, 858–881. https://doi.org/10.3390/jcp3040039

Academic Editors: Martin Gilje Jaatun, Massimiliano Rak and Ferhat Ozgur Catak

Received: 6 September 2023
Revised: 27 October 2023
Accepted: 9 November 2023
Published: 1 December 2023

1. Introduction

The increasing demand for cloud solutions in various industries, such as healthcare, finance, education, and government, has driven the rapid growth of the global public cloud services market. According to Forrester, this market is projected to surpass USD 1 trillion by 2026, more than doubling its value of USD 446.4 billion in 2022 [1]. However, the widespread adoption of cloud services has also brought new cybersecurity challenges, including the abuse of cloud and public legitimate services (CLS) as a C&C communication channel.

Attackers can exploit the CLS to hide their malicious activities and remotely control compromised systems. This enables them to operate covertly and efficiently, reducing the likelihood of detection. They achieve this by leveraging the trust between the CLS provider and the user, using these services as C&C infrastructure. The capability to mask their actions and remotely manage compromised systems significantly enhances the success rate of adversaries in cyber attacks. Malware examples that have utilized CLS as C&C servers include Hammertoss [2], RegDuke [3], SLUB [4], and DarkHydrus [5].

Hammertoss is a remote access tool that leverages third-party web servers, including LinkedIn, Twitter, and GitHub, to evade detection by security solutions and gain full access to the victim's system. RegDuke is a malware variant that abuses Dropbox by hosting steganographic images containing encrypted malicious commands for covert C&C operations. SLUB is a backdoor identified and analyzed by TrendMicro, which abuses three legitimate platforms—Slack, GitHub, and File.io—for its C&C infrastructure. DarkHydrus is a cyber threat group known to abuse legitimate cloud services, such as Google Drive, for its infrastructure.

Traditional anti-malware solutions that rely on known malware signatures or behaviors may not be effective at detecting and preventing such abuses. As a result, ML classifiers have become an increasingly popular tool for detecting threats in the field of cybersecurity.

Our proposed work uses static and derived features from the PE file header as ML features to detect the abuse of CLS as C&C channels. The PE file is a standard format for executables, object files, and DLLs in the Windows operating system [6]. It contains information about the structure and layout of an executable file, including the entry point, the code and data sections, and the dependencies of the program.

The focus on the PE file format is driven by two primary factors. First, its widespread use in the Windows environment is highlighted in Figure 1, showing that Windows has maintained a dominant market share in desktop operating systems (OS) from 2013 to 2023. Second, Figure 2 indicates that the PE file format is the most commonly submitted among all file types on VirusTotal.

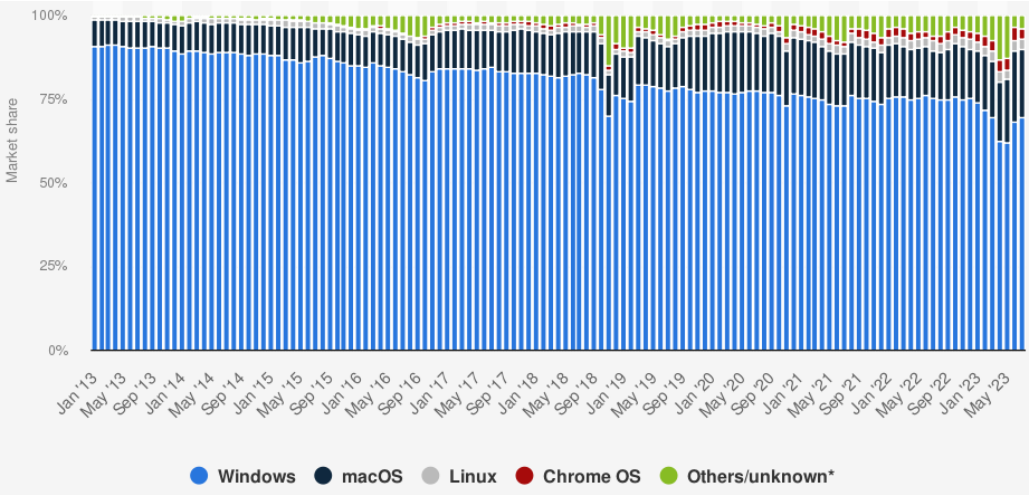

Figure 1. Global distribution of market share among different OS used in desktop PCs [7].

The contributions of this paper are as follows:

- Dataset creation: We have created a unique dataset that includes malware samples initiating network connections to CLS and benign samples making legitimate connections to the Internet. To our knowledge, no previous datasets have specifically addressed this emerging threat; they primarily covered generic malware samples. This dataset, the first of its kind, serves as a valuable resource for training and evaluating ML classifiers to detect the abuse of CLS.
- Feature engineering: We tailored informative feature engineering to behaviors that indicate CLS abuse, such as connecting to known CLS domains and making potential C&C API calls.
- Feature selection: To identify the most relevant features, we introduced a custom feature elimination (CFE) method designed to determine the exact number of features needed for filter selection approaches. For wrapper-based feature selection, we utilize various techniques, including sequential feature selector forward (SFSF), sequential feature selector backward (SFSB), and recursive feature elimination (RFE).

- Novel adversarial attack: We propose the Replace Misclassified Parameter (RMCP) as a novel white-box adversarial attack to evaluate the robustness of our proposed abuse detection system. Despite the adversarial attack, the approach retains a relatively high accuracy level. The detection accuracy rate drops from 98% to 83%, indicating that the proposed method maintains considerable effectiveness against adversarial attacks.

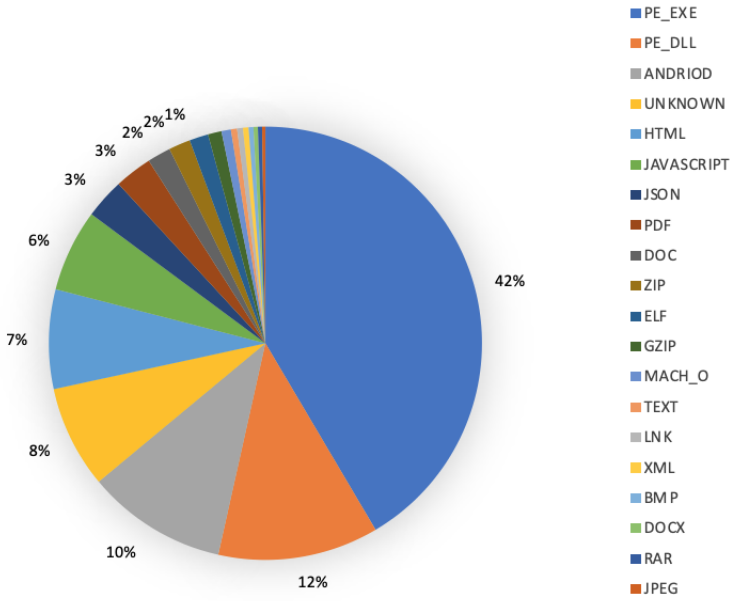

Figure 2. VirusTotal submission by file format [8].

The remainder of this paper is structured as follows. Section 2 is an overview of the PE files and gives brief descriptions of C&C channels communication channels and the threat model of abusing CLS as C&C channels. Section 3 reviews related work in the literature. The main steps of our methodology and experimental setup and results are presented in Sections 4 and 5. Comparison to related works and robustness evaluation and comparison are presented in Section 6. Finally, limitations, future work, and the conclusion are presented in Sections 7 and 8.

2. Background

In Section 2.1, we summarize the PE file format [6], including details on the structure and content of PE files, which are commonly used for running programs on Windows systems. In Section 3, we review previous research on identifying the use of legitimate cloud and service providers as C&C communication channels by malicious actors.

2.1. PE File Format

The PE file format is one of the most prevalent types of executable files used in malware. PE files have a certain structure and contain various fields that provide information about the file [9]. In the context of malware CLS abuse detection, specific fields within the PE file format can be crucial for distinguishing between malicious and benign files.

As depicted in Figure 3, the PE file format comprises several fields: COFF Header, Optional Header, Import Table, Export Table, Resource Directory, Relocation Table, Debug Information, and Section Table. Instead of offering an exhaustive description of every field, we focus on the ones most pertinent to abuse detection, as outlined in Table 1.

For instance, the number of sections and the characteristics of the DLL have been shown to be useful in differentiating between malware and benign files. The optional header provides information such as the linker version and the sizes of code and data, which can also be valuable for classification purposes. Moreover, the section table contains

crucial data regarding the file's sections, including code, initialized data, imports, exports, and resources.

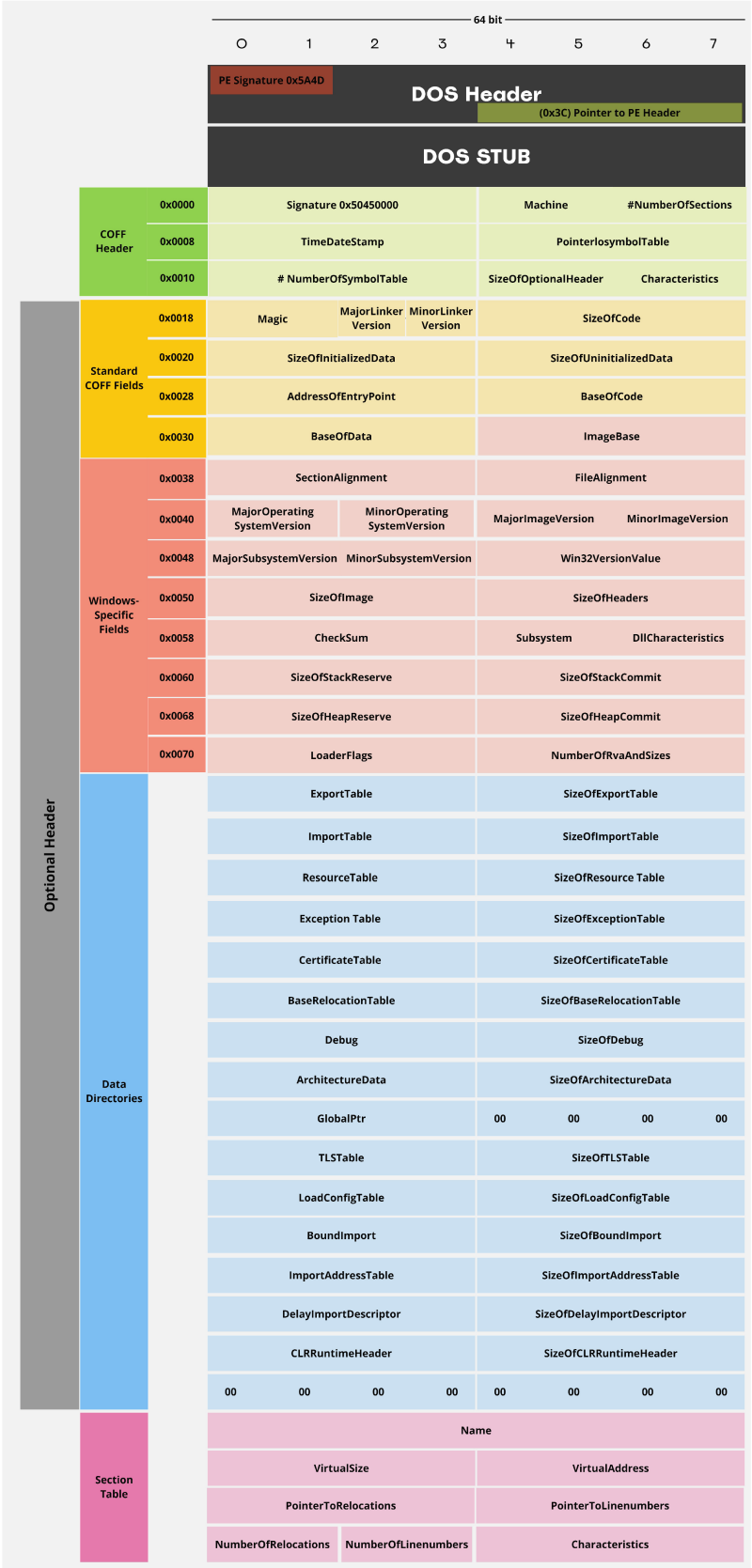

Figure 3. Detailed diagram of the structure of a Portable Executable (PE) file [10].

Table 1. PE file fields and derived features for malware classification.

Field	Description
COFF Header	• Machine: Type of machine that the object file is intended to run on. • NumberOfSections: Number of sections in the object file. • TimeDateStamp: Time and date that the object file was created.
Optional header	• Linker version: Version of the linker that created the object file. • Code and data sizes: Sizes of the code and initialized and uninitialized data in the object file. • Entry point address: Address of the entry point for the object file. • ImageBase: Address of the executable in memory. • CheckSum: Value used to validate the integrity of the image. • DllCharacteristics: DLL characteristics of the executable. • Import Table: List of DLLs and functions imported by the executable that can provide information about the functionality of the executable and indicate potential malicious behavior. • Export Table: List of functions exported by the executable that can provide information about the functionality of the executable and indicate potential malicious behavior. • Resource Directory: Resources used by the executable, such as icons, cursors, and bitmaps, that can provide information about the appearance and behavior of the executable and indicate potential malicious behavior. • Relocation Table: Information used by the linker to adjust addresses in the code when the executable is loaded into memory that can provide information about how the executable is organized and indicate potential malicious behavior. • Debug Information: Information used by debuggers to help debug the executable that can provide information about the internal structure of the executable and indicate potential malicious behavior.
Section Table	• Name: Name of the section. • VirtualSize: Size of the section in memory. • VirtualAddress: Address of the section in memory. • SizeOfRawData: Size of the section in the object file. • PointerToRawData: Location of the section in the object file.
Derived Features	• presence_of_CLS_domains: If any of the CLS domains appear in sections of the PE file, the value is 1; otherwise, it is 0. • potential_C&C_api_calls: If any of the potential C&C API calls appear as an import function in the PE file, the value is 1; otherwise, it is 0.

2.2. Command and Control Communication Channels

A malicious bot is a type of malware that infect computers via various means, such as phishing attacks, drive-by download attacks, and dropper attacks. Once the bot is executed on a victim's computer, it can be controlled remotely by the botmaster and added to the botnet.

The C&C communication channels, which are used by the botmaster to communicate with bots on the botnet, are typically concealed and often encrypted to evade detection. Common C&C channels include Internet Relay Chat (IRC), Domain Name System (DNS) Tunneling, Hypertext Transfer Protocol (HTTP) and HTTPS, and peer-to-peer (P2P) networks. Nevertheless, recent advancements in C&C communication channels have witnessed the abuse of CLS, wherein legitimate services such as cloud services are utilized as a means for C&C communication without detection.

Despite the advancement in the detection of IRC, DNS, HTTP, HTTPS, and P2P as C&C channels, there is a limited amount of studies, as stated in the following section, that focus on the detection of the abuse of legitimate services as C&C channels and none of them applied ML detection techniques. This is an active area of research and there are ongoing efforts to develop more effective techniques for detecting botnets that use legitimate services as C&C channels. Despite these efforts, malicious actors continue to exploit these legitimate services for their C&C purposes.

2.3. Threat Model

2.3.1. Post Exploitation

The threat model for the abuse of CLS as C&C communication channels starts with the initial compromise of a target device. This can occur through a variety of means, such as phishing attacks, malware infections, or the exploitation of software vulnerabilities. Once the device has been compromised, the attacker will typically install a bot or malware onto the device, which allows them to remotely control the device as part of a botnet.

2.3.2. Abuse of Cloud and Legitimate Services as C&C Channels

The next step in the threat model is the use of CLS as C&C communication channels as presented in Figure 4. This is achieved by the attacker using cloud services or other legitimate services to communicate with infected devices and issue commands. The C&C communication channels are typically hidden and encrypted, making them difficult to detect. The following steps outline the abuse of CLS as C&C channels

(a) The botmaster issues command to the bots in the botnet through the cloud or legitimate service;
(b) The bots continuously monitor the designated cloud or legitimate service for new commands from the botmaster;
(c) The bots then execute the command, which conducts a range of malicious activities, which can range from data leaks to denial of service attacks to the distribution of additional malware through the utilization of the cloud or legitimate services as a communication channel.

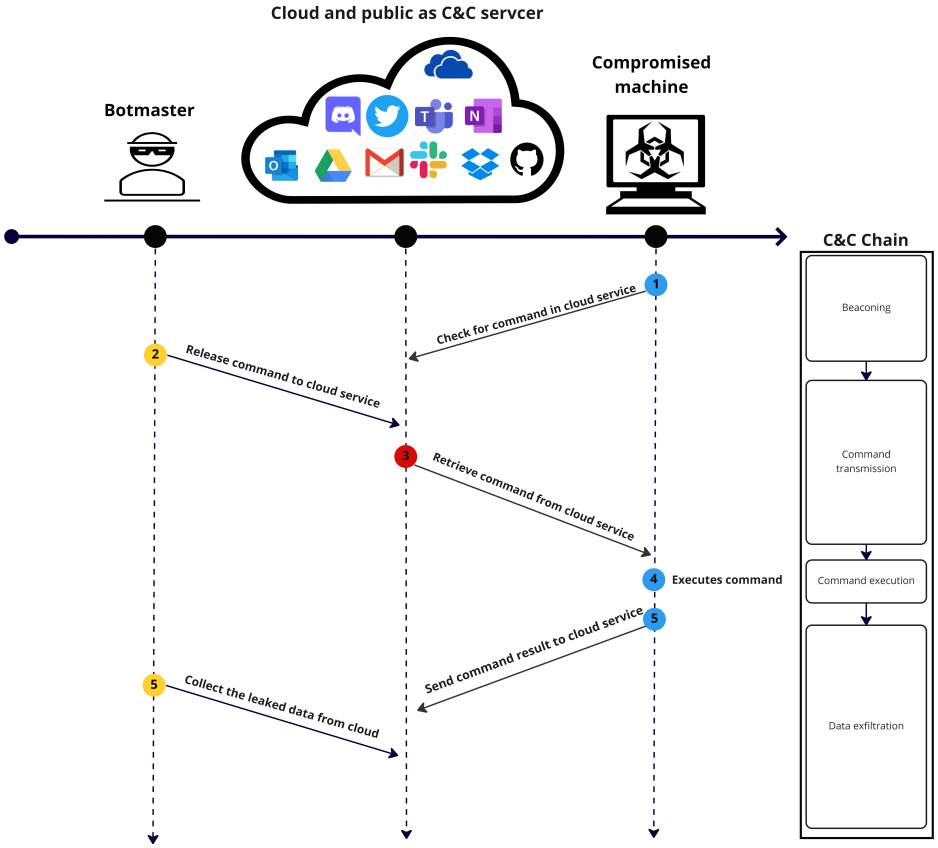

Figure 4. Abuse of CLS as C&C infrastructure.

2.3.3. Threat Scenario

Once the C&C communication channels have been established, the attacker can use the botnet to carry out a range of malicious activities, such as data theft, denial of service

attacks, or distribution of additional malware. The attacker may also use the botnet to expand their network of compromised devices, increasing the size and scope of their botnet.

To detect and prevent the abuse of CLS as C&C communication channels, it is important to have a robust threat model that can accurately identify and block malicious activities.

3. Related Works

Several techniques have been proposed for detecting abuse of cloud-native platforms as C&C communication channels. Six of these detection strategies have been implemented for use in a computer environment, while only one has been specifically implemented for use on the Android OS. These techniques primarily focus on three approaches: rule-based, behavior tree-based, and ML-based detection methods.

3.1. Rule-Based Detection

Kartaltepe et al. [11] introduced a dual-level abuse detection system, comprising client-side and server-side mechanisms. On the client side, they defined three features to detect botnets: self-concealment, unusual network traffic, and questionable provenance. They posited that connections to social media platforms might be considered suspicious unless driven by human actions. To differentiate between legitimate users and bots, they employed behavioral biometrics, user input reactions, and graphical user interface (GUI) interactions as detection metrics. On the server side, they operated under the assumption that any communication with social media platforms that involves textually encoded messages or posts is suspicious. They utilized the J48 decision tree algorithm to categorize input messages, differentiating between Base64 or Hexadecimal-encoded text and regular language content. However, these detection methods come with certain constraints: (i) they do not offer real-time detection since the tests were conducted in a post-analysis lab environment, and (ii) crafty adversaries could potentially bypass detection by using image-steganography techniques to embed malicious commands within posts.

Vo et al. [12] developed the API Verifier, a tool that uses CAPTCHA verification to authenticate social media account access based on MAC addresses. This tool determines whether an API call is made by a human or a bot, adding a protective layer against automated bot activities. However, the API Verifier proposed by Vo et al. has several drawbacks. First, the CAPTCHA verification system can be vulnerable to relay attacks, potentially enabling botnets to circumvent the verification. Second, depending solely on MAC addresses for user identification might fall short in situations where users switch between multiple devices or when MAC addresses are easily spoofed.

Ghanadi et al. [13] delve into the study of stego-botnets that utilize steganographic images on online social networks for C&C operations. They proposed a system named SocialClymene, designed to detect covert botnets in social networks using stego-images. The system has a negative reputation subsystem that analyzes images shared by social network users and calculates a reputation score for each user based on their history of participating in suspicious activities. The goal is to recognize botnets by analyzing the behavior of the users and their history of involvement in suspicious activities. Nevertheless, these detection approaches have certain limitations: (i) the system might not detect new botnets lacking a history of suspicious behavior, and (ii) it can be challenging to accurately identify a user's reputation, especially in dynamic online settings where reputations can shift swiftly.

3.2. Behavior Tree-Based Detection

Yuede et al. [14] introduced a behavior tree-based detection framework designed to identify social bots through host activity monitoring. This framework is composed of three main components: a host behavior monitor, a host behavior analyzer, and a detection methodology. To construct and analyze a suspicious host behavior tree, they crafted a social botnet called wbbot. Their design incorporated sample collections from two distinct sources: real-world social bots [15–18] and social bot malware samples curated by researchers [19].

After running and evaluating this collection of social bots over a specified duration, they created a template library. This library was subsequently used to determine the highest similarity value when compared to the suspicious behavior tree. Upon the behavior tree's completion, the tree edit distance method was utilized to to calculate its similarity to the template, resulting in the final detection result. Nevertheless, this detection methodology presents a notable limitation: a substantial false positive rate of 29.6%. Additionally, this system could potentially be bypassed if attackers deploy a multi-process strategy or spread malicious behaviors over varied time intervals.

Burghouwt et al. [20] proposed a causality detection mechanism designed to pinpoint Twitter-based C&C channel communication. This is achieved by measuring the correlation between user activity and network traffic. The authors operate under the assumption that any network traffic directed to the OSN that is not a result of human actions like specific keystrokes or mouse movements, should be considered suspicious. The causality detection approach utilizes a time frame that begins immediately after a user event. This helps differentiate network activities triggered by genuine user actions from those initiated by bots. However, this detection approach has certain limitations. First, legitimate API calls, which are used for routine automated polling, might be mistakenly identified as suspicious. Second, the primary metrics used to determine the time gap between user activity and network requests might not be universally accurate. This is because different machines and operating systems can have varied delay times and performance attributes. Lastly, sophisticated bots could potentially circumvent this detection by observing user activities and executing commands in response to user-initiated events.

3.3. ML-Based Detection

Ji et al. [21] undertook a detailed assessment of several previously studied abusive social bots. The authors incorporate spatial and temporal correlations to identify patterns of the social bots. They gathered source code, builders, and execution patterns from established social botnets, including Twitterbot (Singh [22]), Twebot (Burghouwt et al. [23]), Yazanbot (Boshmaf et al. [24]), Nazbot (Kartaltepe et al. [11]), wbbot (Ji et al. [25]), and fbbot. Their objective was to scrutinize the techniques these bots employ to bypass current detection systems. Drawing from their analysis, they proposed a detection strategy using 18 features. This strategy uses spatial correlations to recognize patterns across child processes, like multiple bots on one IP, and temporal correlations to study event sequences for behavior patterns. However, their focus on just six bots could limit the study's applicability to other bots.

Ahmadi et al. [26] introduced a method designed to detect Android applications that abuse Google Cloud Messaging (GCM) for C&C communication. By adopting the Flowdroid tool [27] to extract GCM flows and identify GCM callbacks, they trained an ML model using features like GCM registration ID, sender ID, and a GCM type of message. Their findings indicate that GCM flow features can effectively distinguish malicious applications. Nonetheless, the method might be vulnerable to evasion tactics like obfuscation, which adversaries might use to mask GCM flows. Additionally, its applicability is constrained, as it only works for Android OS and is not applicable in Windows OS environments.

The existing literature predominantly focuses on the use of rule-based and behavior tree-based detection techniques within the realm of social networking platforms, while ML techniques for detecting abuse in CLS environments are often neglected. This creates a gap in the detection of C&C abuse within CLS environments. To address this limitation, we introduce a detection technique that employs ML and is specifically designed to identify C&C abuse across diverse CLS environments. Given the lack of prior research applying ML to detect abuse of cloud services as a C&C channel, we have undertaken a comparative analysis of our approach alongside existing studies that leverage ML techniques for general malware detection.

4. Methodology

4.1. Data Collection

In this study, we utilized a dataset obtained from VirusTotal [28] between 2017 and 2021, which encompassed various malware formats. Our research specifically focused on PE files that abuse CLS as a C&C infrastructure.

To collect malware samples, we leveraged the VirusTotal Intelligence Agent coupled with a custom Python script to extract samples exhibiting communication with known CLS-hosted domains listed in Table 2. The remaining corpus was executed in a controlled Cuckoo sandbox environment [29], retaining only samples demonstrating CLS domain connections for our final experimental dataset, as depicted in Figures 5 and 6. We excluded any samples not connecting to the CLS domain.

Table 2. CLS domains abused as C&C infrastructure.

Names of the CLS Domains		
api-content.dropbox.com	api.twitter.com	docs.google.com
mail.google.com	chat.google.com	classroom.googleapis.com
sheets.googleapis.com	slides.googleapis.com	storage.googleapis.com
mail.google.com	smtp.gmail.com	onedrive.com
dropbox.com	twitter.com	github.com
pastebin.com	raw.githubusercontent.com	api.twitter.com
dev.twitter.com	publish.twitter.com	apps.twitter.com
status.twitter.com	youtube.com	twitter.com
docs.google.com	script.google.com	translate.google.com
storage.googleapis.com	spreadsheets.google.com	api.slack.com
app.slack.com	slack.com	gmail.com
hotmail.com	outlook.com	amazonaws.com
azure.com	portal.office.com	discord.com
telegram.com	instagram.com	OneNote.com
Teams.com	Evernote.com	publish.twitter.com
apis.google.com	imap.gmail.com	m.youtube.com
aws.amazon.com		

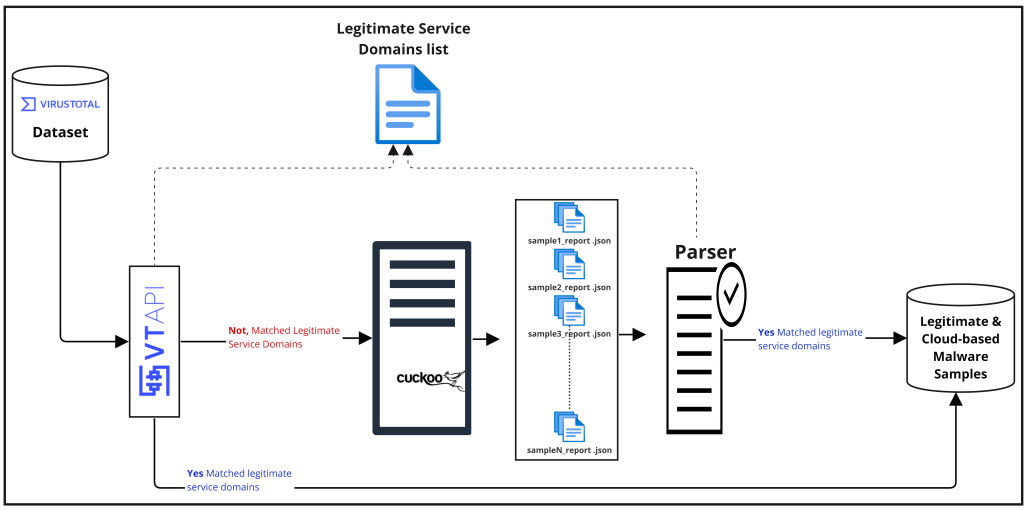

Figure 5. Detailed workflow for extracting a sub-dataset from the VirusTotal datasets.

Additionally, the benign samples included in the dataset were obtained from the sources of cnet [30] and sourceforge [31]. These benign programs were also verified by submitting them to VirusTotal to obtain anti-virus detection scores. If the detection score was found to be zero, we further executed it in a controlled sandbox environment to verify its internet connectivity. Only samples that were determined to have zero detection scores and internet connections were ultimately included in the dataset.

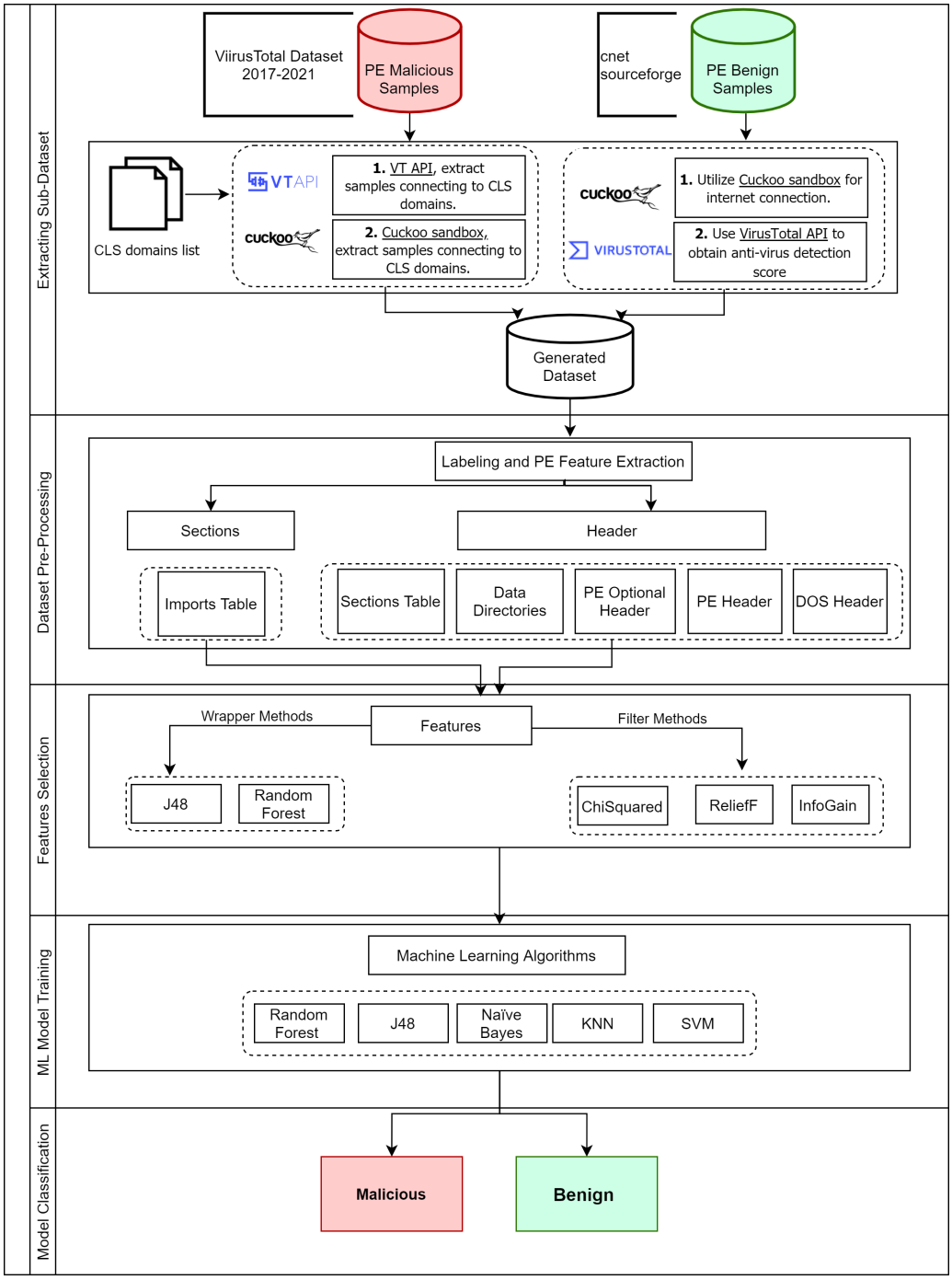

Figure 6. Illustrative overview of the proposed detection system.

The initial dataset was imbalanced, with 3067 malicious and 3652 benign samples. To ensure a balanced dataset and prevent classifier bias, the extra benign samples were removed, resulting in a balanced dataset that retained the same characteristics as the remaining benign samples. To ensure that the removal of extra benign samples did not compromise the dataset's quality, we retained the original characteristics of the remaining benign samples. Since the collection of benign samples was based solely on connections to the internet and VirusTotal's detection score of zero, removing the extra benign samples did not alter the dataset's properties. Therefore, we were able to balance the dataset without compromising its quality, ensuring that the evaluation of classifier accuracy was based on a reliable and representative dataset.

The experiment, involving the extraction of a sub-dataset from the VirusTotal dataset, the parsing of PEs, and the execution of malware and benign samples in a Cuckoo sandbox [29], was carried out on a machine powered by an Intel Xeon (Skylake IBRS) CPU running at 2.2 GHz, equipped with 64GB RAM, and using Ubuntu 20.04.1 LTS amd64 as its operating system.

Our dataset is unique and valuable to the field of cybersecurity as it includes examples of malware abusing CLS as a C&C channel, a type of threat that has not been well-represented in previous datasets.

4.2. Feature Extraction

Feature engineering involves extracting particular attributes from PE files to objectively determine whether they are benign or malicious. Our study focused on feature engineering using PE header features. We analyzed the header and sections of each file in our sample and identified a total of 38 relevant features, comprising 36 raw features and 2 derivative features (Table 1). Raw features can be directly extracted from the PE file with no further processing. Such features are Characteristics, DllCharacteristics, SizeOfImage, AddressOfEntryPoint, and ResourceSize.

To generate derivative features, we need to process the PE file. The first feature, called presence_of_CLS_domains, is generated by examining each section in the PE file to check if it contains any CLS domains (Table 2). The feature value is set to one if a CLS domain is found; otherwise, it is set to zero. The second feature, called potential_C&C_api_calls, is generated by examining the Import Address Table (IAT) in the PE file for any API function calls that could be used for C&C activities. The names of the potential C&C API calls [32] are listed in Table 3. The feature value is set to one if a potential C&C API call is identified; otherwise, it is set to zero.

Table 3. Potential API calls for abusing CLS.

API Call	Description	Potential Abuse Case
InternetOpenA	Open an Internet session	Establish a connection to CLS
InternetConnectA	Connect to a remote server	Connect to the servers of CLS
HttpOpenRequestA	Open an HTTP request handle	Open HTTP requests to CLS
InternetReadFile	Read data from an open Internet file	Read data from a file on CLS
InternetWriteFile	Write data to an open Internet file	Write data to a file on CLS
WinHttpOpen	Open an HTTP session	Open HTTP sessions with CLS
WinHttpConnect	Connect to a remote server	Connect to the servers of CLS
WinHttpOpenRequest	Open an HTTP request handle	Open HTTP requests to CLS
WinHttpSendRequest	Send an HTTP request	Send HTTP requests to CLS
WinHttpReceiveResponse	Receive an HTTP response	Receive HTTP responses from CLS
WinHttpReadData	Read data from an HTTP request	Read data from an HTTP request to CLS
WinHttpWriteData	Write data to an HTTP request	Write data to an HTTP request to CLS
URLDownloadToFileA	Download a file from the Internet	Download files from CLS
HttpSendRequestA	Send an HTTP request	Send HTTP requests to CLS
InternetOpenUrlA	Open an HTTP or FTP session	Open HTTP or FTP sessions with CLS
InternetReadFileExA	Read data from an open Internet file	Read data from a file on CLS
InternetWriteFileExA	Write data to an open Internet file	Write data to a file on CLS

Importance of Derived Features

In our study, we evaluate a total of 38 pertinent features, of which 36 were raw PE features and 2 were derived features. Utilizing various feature selectors, as discussed in Section 4.3, our results identified a combination of 21 raw features and the derived feature "potential_C&C_api_calls" to yield the highest classification accuracy. Detailed results and

comparisons before and after inclusion of the derived features can be perused in Tables 4 and 5. Our final model that delivered the highest accuracy employed the RF classifier in conjunction with the aforementioned feature selector. We also charted the importance of each feature. This plot offers a visual representation of the relative significance of each feature as depicted in Figure 7.

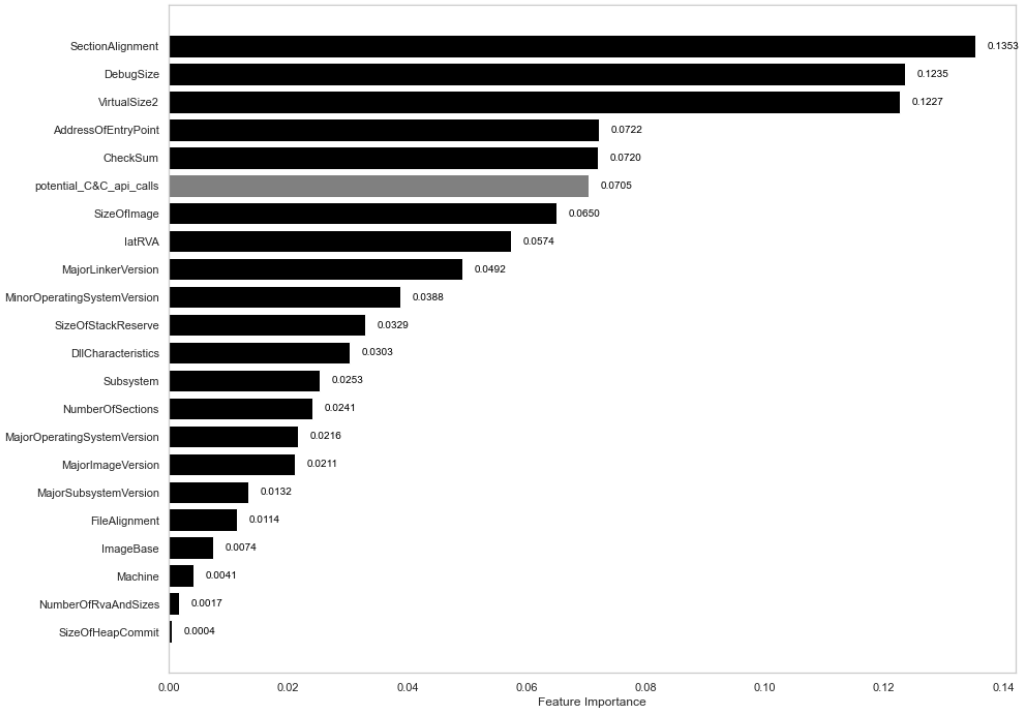

Figure 7. Feature importance highlighting the prominence of "potential_C&C_api_calls" among the top influential features in the model.

Although the "presence_of_CLS_domains" may not have prominently boosted the classification accuracy, as indicated in Table 4, its relevance stems from our data collection approach, detailed in Section 4.1. In particular, the dynamic analysis phase of our data collection was designed to extract a subset from the VirusTotal dataset that predominantly connects CLS domains.

Table 4. Comparison of detection accuracy: evaluating the impact of including derived features.

Features Used	Validation Approach	
	70:30 Split	10-Fold CV
Optimal feature using RF feature selector (excluded potential C&C calls)	0.977186	0.981416
+ presence_of_CLS_domains	0.977729	0.980600
+ potential_C&C_api_calls	0.981532	0.982557
Included derived features	0.978273	0.986601

4.3. Feature Selection

After extracting or creating features from the malware samples, we move to feature selection. This step is crucial to pinpoint the most relevant and informative features, enabling the model to deliver accurate predictions. In this section, we outline the feature selection methodology we adopted, encompassing three filter-based and six wrapper-based methods. Each of these nine techniques identifies a feature subset. From these subsets, we select the one that delivers the highest detection accuracy rate.

4.3.1. Filter-Based Feature Selector

We employ three well-established filter-based feature selection methods commonly employed and proven effective in the literature: information gain (InfoGain) [33–35], chi-squared, and ReliefF [36].

Filter-based methods rank features based on their relevance to the label class, but they do not specify an exact number of features to use. To address this, we develop and implement our a custom feature elimination technique (CFE) on each of the filter-based methods to determine the number of features to use. The CFE technique leverages the feature importance rankings provided by the aforementioned filter-based methods: InfoGain, chi-squared, and ReliefF. It operates by progressively examining the features, starting from those with the highest importance rankings. For each iteration, it appends the current feature to the selected features to form a temporary feature set. This temporary set is then used to compute the accuracy of a random forest classifier using 10-fold stratified cross-validation. If the accuracy improves, the current feature is added to the selected features, and the accuracy progress is recorded. If there is no improvement in accuracy, the process is terminated, and the selected feature set is returned. The accuracy progression of the CFE technique for each filter-based method, InfoGain, chi-squared, and ReliefF, is depicted in Figure 8, Figure 9, and Figure 10, respectively.

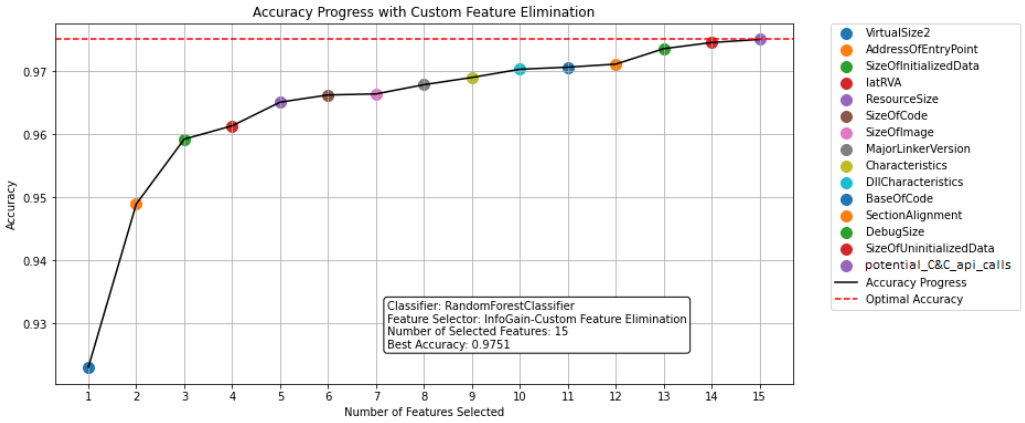

Figure 8. Accuracy and subset of features using InfoGain.

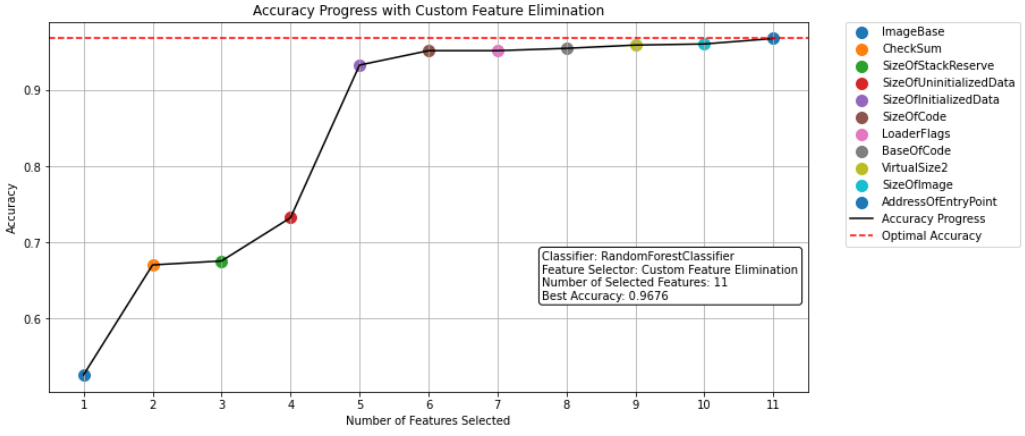

Figure 9. Accuracy and subset of features using chi-squared.

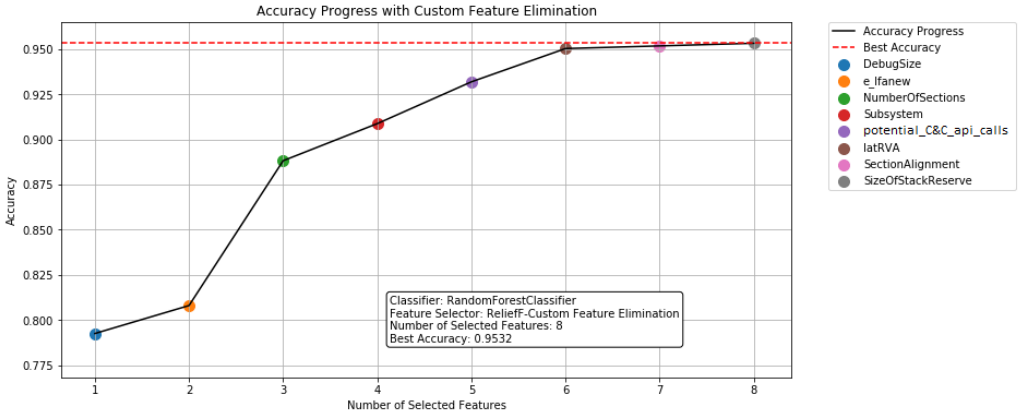

Figure 10. Accuracy and subset of features using ReliefF.

4.3.2. Wrapper-Based Feature Selector

In addition to the filter-based methods, we employed wrapper-based techniques using random forest (RF) and decision tree (DT) as the foundational models for feature selection. We paired each of these algorithms with the following three distinct strategies: sequential feature selector forward (SFSF), sequential feature selector backward (SFSB), and recursive feature elimination (RFE).

This led to a total of six combinations: RF-SFSF, RF-SFSB, RF-RFE, DT-SFSF, DT-SFSB, and DT-RFE. Each method evaluates feature subsets by training and testing models on varying feature subsets, ultimately choosing the subset with the best performance. For the wrapper-based feature selection methods, we utilized the Mlxtend Python library [37].

Figures 11 and 12 display the highest accuracy rates achieved by each combination, using a different number of selected features ranging from 1 to the maximum number of 38 features available in the dataset.

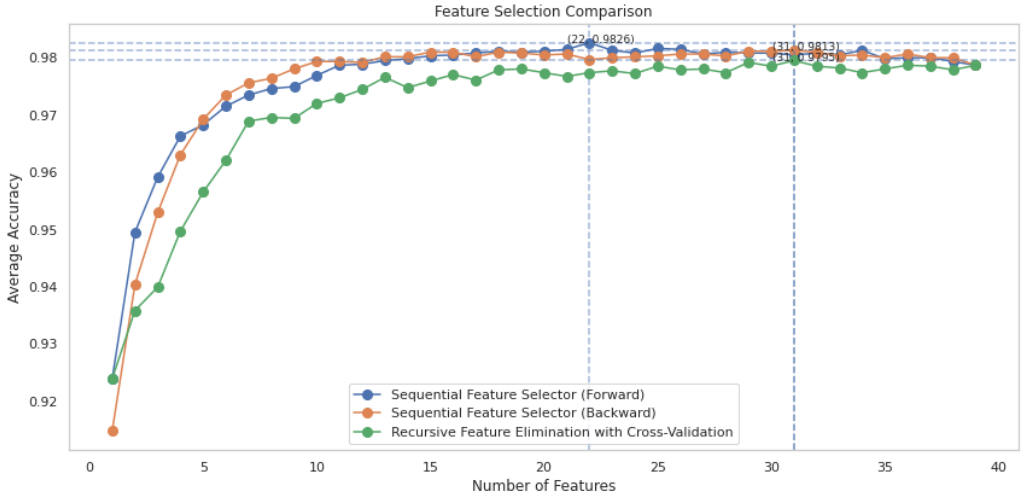

Figure 11. Comparative analysis of wrapper feature selection: RF-SFSF vs. RF-SFSB vs. RF-RFE, highlighting the optimal feature count for maximum accuracy as indicated by dotted lines.

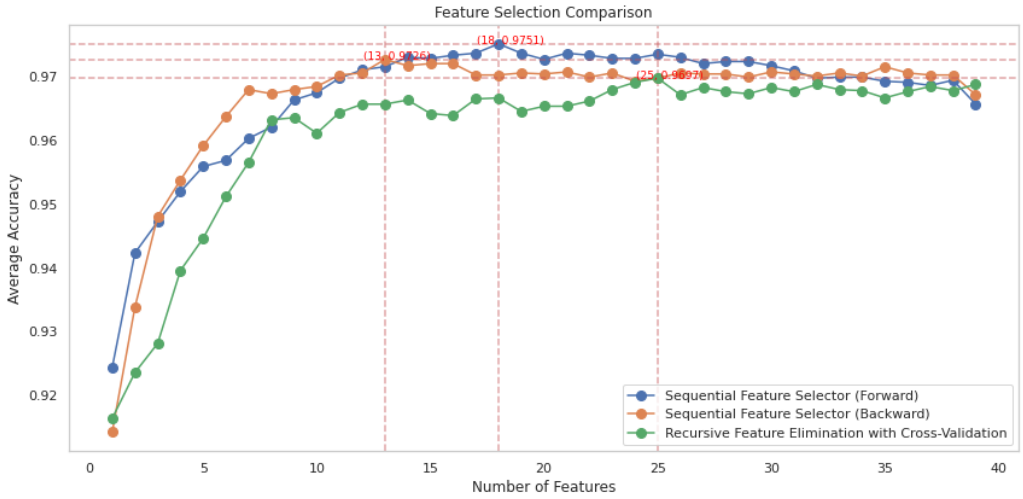

Figure 12. Comparative analysis of wrapper feature selection: DT-SFSF vs. DT-SFSB vs. DT-RFE, highlighting the optimal feature count for maximum accuracy as indicated by dotted lines.

4.4. Classification

Our evaluation applies five ML-based classifiers, which are implemented in Scikit-learn [38]: namely decision tree (J48), random forest (RF), naïve Bayes (NB), k-nearest neighbors (K-NN), and support vector machine (SVM). These classifiers were selected for their diverse underlying algorithms and their widespread usage in malware detection literature [33,36,39]. By employing a range of classifiers, we aim to comprehensively assess the performance of various ML classifiers on our dataset, which is detailed in Section 4.1.

To measure the accuracy of these classifiers, we utilized two distinct evaluation techniques: 10-fold cross-validation and a training-to-testing split ratio of 70:30. We leveraged the Scikit-learn library [38] for implementing these ML algorithms.

The equipment utilized for this experiment was sourced from Cardiff University, United Kingdom. Specifically, the evaluations were conducted on an ASUS computer, equipped with an Intel i7-9700K processor with a clock speed of 3.60 GHz, supported by 32 GB of RAM, and running the Windows 10 operating system.

5. Discussion

5.1. Experimental Results of All Features

The evaluations were analyzed and compared with regard to detection accuracy. We present the detection accuracy of five classifiers, all of which utilize the extracted features for classification, as demonstrated in Table 5. The results show that the random forest classifier outperforms the other classifiers, achieving a high detection accuracy of 97.77% in the 70:30 split scenario and 97.80% in the 10-fold cross-validation scenario. The J48 and K-NN classifiers also demonstrate high accuracy, with detection rates of 96.41% and 93.12%, respectively, in the 70:30 split scenario, and 96.84% and 93.80%, respectively, in the 10-fold cross-validation scenario.

However, the NB and SVM classifiers show significantly lower accuracy compared to the other classifiers, indicating that they may not be suitable for this dataset. The NB classifier has detection accuracies of 65.13% and 63.76% in the 70:30 split and 10-fold cross-validation techniques, respectively. The SVM classifier has even lower detection accuracies of 52.47% and 52.54% in the 70:30 split and 10-fold cross-validation techniques, respectively.

Ultimately, when considering all extracted features, the results demonstrate that the RF, J48, and K-NN classifiers prove suitable for this dataset. Conversely, the NB and SVM classifiers are not recommended.

Table 5. Comparative analysis of abuse detection accuracy: all features vs. selected features. Here, "70:30" denotes a split of 70% training data and 30% testing data, while "10-fold" stands for 10-fold cross-validation.

Classifier	All Features Included		Feature Selector									
			Wrapper Methods				Filter Methods					
			Random Forest as Feature Selector		J48 as Feature Selector		InfoGain		ChiSquared		ReliefF	
	70:30	10-fold	70:30	10-fold	70:30	10-fold	70:30	10-fold	70:30	10-fold	70:30	10-fold
J48	96.41%	96.84%	96.80%	96.77%	96.20%	96.90%	95.87%	95.83%	94.19%	95.04%	94.51%	94.93%
Random Forest	97.77%	97.80%	98.15%	**98.26%**	97.45%	97.90%	97.12%	97.47%	96.47%	96.59%	95.93%	95.32%
Naïve Bayes	65.13%	63.76%	63.50%	63.06%	54.16%	52.87%	52.74%	52.38%	65.02%	65.57%	59.32%	64.18%
KNN	93.21%	93.80%	92.67%	93.63%	92.07%	92.70%	91.85%	92.96%	93.16%	93.77%	93.75%	93.81%
SVM	52.47%	52.54%	51.11%	52.48%	51.11%	52.53%	53.72%	53.52%	52.47%	52.54%	61.11%	60.68%
Selected Features	AddressOfEntryPoint SizeOfCode SizeOfInitializedData SizeOfUninitializedData BaseOfCode MajorLinkerVersion MajorImageVersion MajorOperatingSystemVersion DllCharacteristics SizeOfStackReserve NumberOfSections ImageBase SectionAlignment FileAlignment MinorOperatingSystemVersion MinorImageVersion MajorSubsystemVersion MinorSubsystemVersion SizeOfImage SizeOfHeaders CheckSum Subsystem SizeOfStackCommit SizeOfHeapReserve SizeOfHeapCommit LoaderFlags NumberOfRvaAndSizes SizeOfOptionalHeader Characteristics Machine e_lfanew DebugSize ExportSize VirtualSize2 ResourceSize IatRVA presence_of_CLS_domains potential_C&C_api_calls	AddressOfEntryPoint MajorLinkerVersion MajorImageVersion MajorOperatingSystemVersion DllCharacteristics SizeOfStackReserve NumberOfSections ImageBase SectionAlignment FileAlignment MinorOperatingSystemVersion MajorSubsystemVersion SizeOfImage CheckSum Subsystem SizeOfHeapCommit NumberOfRvaAndSizes Machine DebugSize VirtualSize2 IatRVA potential_C&C_api_calls	AddressOfEntryPoint SizeOfUninitializedData MajorLinkerVersion MajorOperatingSystemVersion DllCharacteristics SizeOfStackReserve NumberOfSections ImageBase SectionAlignment FileAlignment MinorOperatingSystemVersion MinorImageVersion MajorSubsystemVersion MinorSubsystemVersion SizeOfImage CheckSum SizeOfStackCommit SizeOfHeapCommit NumberOfRvaAndSizes SizeOfOptionalHeader DebugSize VirtualSize2 IatRVA presence_of_CLS_domains	VirtualSize2 AddressOfEntryPoint SizeOfInitializedData IatRVA ResourceSize SizeOfCode SizeOfImage MajorLinkerVersion Characteristics DllCharacteristics BaseOfCode SectionAlignment DebugSize SizeOfUninitializedData potential_C&C_api_calls	ImageBase CheckSum SizeOfStackReserve SizeOfUninitializedData SizeOfInitializedData SizeOfCode LoaderFlags BaseOfCode VirtualSize2 SizeOfImage AddressOfEntryPoint	DebugSize e_lfanew NumberOfSections Subsystem potential_C&C_api_calls IatRVA SectionAlignment SizeOfStackReserve	

5.2. Experimental Results on Selected Features

It is important to note that the use of all extracted features may not always be optimal, and feature selection techniques may be necessary to improve classification accuracy. Therefore, this section discusses the result of an accuracy detection rate achieved with selected features.

Table 5 compares the detection accuracy of various classifiers after feature selection, employing both wrapper and filter techniques.

5.3. Detection Evaluation

In terms of detection accuracy, the RF wrapper technique consistently outperforms other feature selection methods, whether the data is split into a 70:30 training-to-testing ratio or subjected to 10-fold cross-validation. For instance, the detection accuracy with RF as a feature selector is 98.15% for the 70:30 split and 98.04% for 10-fold cross-validation, whereas the detection accuracy with J48 as a feature selector is 96.80% for the 70:30 split and 96.77% for 10-fold cross-validation.

Among the filter methods, InfoGain and ReliefF perform better than chi-squared. For example, the detection accuracy with InfoGain as a feature selector is 97.12% for the 70:30 split and 97.47% for 10-fold cross-validation, while the accuracy using chi-squared as a feature selector is 96.47% for the 70:30 split and 96.59% for 10-fold cross-validation.

Figures 11 and 12 show the mean accuracy rate obtained by each combination using a different number of selected features ranging from 1 to the maximum number of features 38 in the dataset. The RF-SFSF approach outperformed the other five wrapper-based approaches, achieving the highest accuracy rate of 98.26% with 22 out of 38 features.

Upon comparing the results of the filter-based and wrapper-based methods, we concluded that the RF-SFSF approach is the most effective feature selection method for this dataset. The final subset of features employed in our study is detailed in Table 5.

6. Comparison to Related Works

6.1. Detection Accuracy Comparison

Given the limited research on using ML techniques to detect abuse of the CLS as a C&C infrastructure, we conduct a comparative analysis with existing studies that identify general malware using PE file properties as ML features, similar to the studies by Kumar et al. [40] and Raman et al. [41]. It is essential to highlight that the evaluations for both our work and the other two studies were based on our unique dataset.

Table 6 presents the results of the comparative analysis. With a detection accuracy of 98%, our proposed work outperforms the other two studies, which posted rates of 94% and 95%, respectively. This improved performance is attributed to our unique approach of leveraging both raw and derived features from PE files. Particularly, the derived features, presence_of_CLS_domains and potential_C&C_api_calls, played a significant role in enhancing the detection efficacy.

Table 6. Comparison of abuse detection accuracy between proposed and existing works.

Reference	J48				Random Forest			
	Accuracy	Precision	Recall	F1-Score	Accuracy	Precision	Recall	F1-Score
Kumar et al. [40]	94.46	94.90	93.63	94.26	94.68	94.93	94.08	94.50
Raman et al. [41]	94.75	93.83	95.08	94.45	95.65	96.36	94.64	95.49
Proposed work	96.80	96.04	97.43	96.73	98.15	98.75	97.43	98.09

6.2. Adversarial Attack

To compare the robustness of our ML models with related works, we propose the Replace Misclassified Parameter (RMCP) as a novel white-box adversarial attack to evaluate the robustness of our proposed abuse detection system by manipulating feature values to make malicious samples appear benign. In a white-box attack, the attacker has full knowledge of the ML model being used, including its features.

To elucidate the impact of adversarial attacks on model accuracy, we have detailed the components of the confusion matrix as follows, which are also presented in Table 7.

Table 7. Confusion matrix: delineating Predicted vs. Actual outcomes for Benign and Malware classifications

		Predicted	
		Benign	Malware
Actual	Benign	True positive (TP)	False positive (FP)
	Malware	False negative (FN)	True negative (TN)

- True positive (TP): The number of instances where the model correctly predicted benign software (0). This means the model identified a sample as benign and it was indeed benign.
- False positive (FP): The number of instances in which the model incorrectly predicted malware (1) when the sample was actually benign (0), which means the model misclassified benign software as malware.

- False negative (FN): The number of instances where the model incorrectly predicted benign software (0) when the software was actually malware (1), which means the model was misclassifying the malware as benign software.
- True negative (TN): The number of instances where the model correctly predicted malware (1). This means that the model identified the software as malware and it was indeed malware.

The adversarial attack experimental procedure consists of two stages, each with unique objectives and methodologies:

1. Identifying Modifiable Features:The first step aims to identify which features within an executable file can be altered without affecting the functionality of the executable file, specifically evaluating whether it retained its ability to execute or became corrupted. This was a crucial step, reflecting the real-world tactics of threat actors who strive to maintain an executable's malicious capabilities while modifying its attributes.
   As Table 8 shows, for 9 out of 38 features, modifying their value in a malicious file results in file corruption, rendering the malware non-executable. For instance, replacing the 'NumberOfSections' value of a malicious sample with other corresponding values from a benign one resulted in a corrupted executable file.

Table 8. Features whose value modification corrupts PE and causes of corruption.

Features	Reasons for Potential Corruption
AddressOfEntryPoint	Starts execution from an incorrect location.
NumberOfSections	OS misinterpreting the structure of the PE file.
ImageBase	New base address conflicts with other programs or system components.
SectionAlignment	OS may not properly load the sections into memory.
Subsystem	Program being run in an inappropriate environment.
Machine	OS attempting to run the code on an incompatible architecture.
VirtualSize2	Leads to incorrect memory allocation.
SizeOfImage	Leads to incorrect memory allocation.
IatRVA	Breaks the linking of imported functions.

2. Identifying Feature Values Leading to Maximum Misclassification: In the second step of our approach, we perturb the significant features of malicious samples with values from benign samples, aiming to make our proposed model misclassify the malicious samples as benign. We exploit the modifiable features identified earlier to deceive the model. For each of these features, we identify the value that maximizes the number of false negatives when applied to malicious samples. To guide the model towards such misclassification, we adopted a targeted white-box adversarial attack using the 'Replace Misclassified Parameter' (RMCP) technique, detailed in Algorithm 1.
   This method works by iteratively substituting each feature value of the malicious samples within the fixed test set with each benign feature value for that particular feature. Out of the 38 total features, we specifically targeted the 22 selected by the RF-SFSF feature selector, which achieved an accuracy rate of 98% when there were no modifications, as shown in Table 5.
   After each substitution, we evaluated the original model on a fixed test set without retraining. We note the replacement value that yields the most false negatives, where malicious samples are misclassified as benign. We intentionally avoided features listed in Table 8, as tampering with them can corrupt the file.

Algorithm 1: RMCP Adversarial Attack

```
Ensure: Worst benign value per feature, accuracy, confusion matrix
Initialize:
X_selected ← Features selected from X
Split data into X_train, X_test, y_train, y_test
Train model on X_train, y_train
Evaluate model on X_test, y_test
Record accuracy and confusion matrix as baseline
for each feature f in X_selected do
    Get unique benign values for f as B
    for each b in B do
        Substitute malicious X_test[f] with b
        Predict on modified X_test
        Compare to baseline accuracy and confusion matrix
    end
end
return Worst benign value per feature, accuracy, confusion matrix
```

Our evaluation was systematic. We began by establishing the baseline accuracy and confusion matrix. After each feature replacement, we recalculated the model's performance metrics.

The results of our robustness evaluation against adversarial attacks are summarized in Table 9, presenting the model's accuracy and confusion matrices, both pre and post-substitution.

During this experiment, our primary metric was the change in false negative (FN) values; an increase would signify a successful adversarial attack.

Analyzing results from Table 9, it is clear that most features maintain high true positive (TP) and true negative (TN) rates, indicating resilient defense against the RMCP attack. However, the 'DebugSize' feature displays a lower accuracy rate of 83.54% and an increased number of false negatives (FNs) from 23 to 292. This suggests that the 'DebugSize' feature may struggle to be effective in detecting the abuse of CLS as C&C.

Table 9. Comparison of detection accuracy before and after the RMCP adversarial attack, highlighting changes in FN values in the confusion matrix (CM): proposed work.

Features	Benign Parameter	Original Detection Accuracy	Post-RMCP Detection Accuracy	Original CM		Post-RMCP CM	
MajorLinkerVersion	8	0.981532	0.977729	TP:935 FN:23	FP:11 TN:872	TP:935	FP:11
						FN:30	TN:865
MajorImageVersion	10		0.975557			TP:935	FP:11
						FN:34	TN:861
MajorOperatingSystemVersion	1		0.976099			TP:935	FP:11
						FN:33	TN:862
DllCharacteristics	1024		0.971211			TP:935	FP:11
						FN:42	TN:853
SizeOfStackReserve	65,536		0.978815			TP:935	FP:11
						FN:28	TN:867
FileAlignment	4096		0.976099			TP:935	FP:11
						FN:33	TN:862
MinorOperatingSystemVersion	0		0.980445			TP:935	FP:11
						FN:25	TN:870
MajorSubsystemVersion	4		0.979902			TP:935	FP:11
						FN:26	TN:869
CheckSum	32,467,821		0.953829			TP:935	FP:11
						FN:74	TN:821
SizeOfHeapCommit	4096		0.981532			TP:935	FP:11
						FN:23	TN:872
NumberOfRvaAndSizes	16		0.980989			TP:935	FP:11
						FN:24	TN:871
DebugSize	28		0.835416			TP:935	FP:11
						FN:292	TN:603
potential_C&C_api_calls	0		0.965779			TP:935	FP:11
						FN:52	TN:843

6.3. Robustness Comparison

In comparison, Table 10 presents the robustness results for Kumar et al. [40], demonstrating that the replacement of certain feature values has a significant impact on the accuracy and confusion matrix of the model. For instance, when substituting the 'Characteristics' feature value, the model's accuracy drops from 94.67% to 69%, and the number of misclassified malicious samples as benign increases significantly from 53 to 525. Overall, the table reveals the vulnerability of their approach to the RMCP technique.

Similarly, Table 11 illustrates the impact of the RMCP technique on Raman et al. [41], resulting in a significant reduction in model accuracy. For instance, when substituting the 'DebugSize' feature value, the model accuracy dropped from 95.84% to 54.37% and the number of false negatives significantly increased from 48 to 808. Similarly, for the 'MajorImageVersion' feature, the model accuracy decreased from 95.84% to 69.74%, while the false negatives increased from 48 to 525.

Table 10. Comparison of detection accuracy before and after the RMCP adversarial attack, highlighting changes in FN values in the confusion matrix (CM): Kumar et al. [40].

Features	Benign Parameter	Original Detection Accuracy	Post-RMCP Detection Accuracy	Original CM		Post-RMCP CM	
MajorOperatingSystemVersion	1	0.946768	0.923411	TP: 901 FN: 53	FP:45 TN:842	TP: 901 FN: 96	FP:45 TN:799
DllCharacteristics	34,112		0.813688			TP: 901 FN: 298	FP:45 TN:597
SizeOfStackReserve	16,777,216		0.811515			TP: 901 FN: 302	FP:45 TN:593
MajorSubsystemVersion	6		0.937534			TP: 901 FN: 70	FP:45 TN:825
MinorSubsystemVersion	0		0.941879			TP: 901 FN: 62	FP:45 TN:833
Characteristics	263		0.690386			TP: 901 FN: 525	FP:45 TN:370
e_lfanew	304		0.697990			TP: 901 FN: 511	FP:45 TN:384

Table 11. Comparison of detection accuracy before and after the RMCP adversarial attack, highlighting changes in FN values in the confusion matrix (CM): Raman et al. [41].

Features	Benign Parameter	Original Detection Accuracy	Post-RMCP Detection Accuracy	Original CM		Post-RMCP CM	
MajorImageVersion	10	0.956545	0.697447	TP: 914 FN: 48	FP:32 TN:847	TP: 914 FN: 525	FP:32 TN:370
DebugSize	84		0.543726			TP: 914 FN: 808	FP:32 TN:87
ExportSize	393,079		0.951113			TP: 914 FN: 58	FP:32 TN:837
ResourceSize	5296		0.762629			TP: 914 FN: 405	FP:32 TN:490

6.4. Robustness Evaluation

Ultimately, in both Kumar et al. and Raman et al., we observe that substituting certain feature values can significantly impact a model's accuracy and confusion matrix. In Table 9, our proposed model demonstrates a robustness rate of 83.54% against adversarial attacks using the RMCP technique, outperforming the 69.03% and 54.37% achieved in the related works by Kumar et al. [40] and Raman et al. [41], respectively.

These findings show the crucial role of white-box adversarial attacks. As depicted in Table 6, the initial abuse detection accuracy rates of our model and related works were quite comparable. However, after conducting the robustness evaluation, a significant drop in the detection rates of the related works was observed, whereas our proposed work experienced only a slight decrease. This demonstrates that our work is more resilient to adversarial attacks, thus offering a more robust and reliable solution for abuse detection.

7. Conclusions

In this paper, we presented a novel approach that utilized ML-based techniques to detect the abuse of the CLS as a C&C infrastructure. Our approach focused on analyzing static and derivative features extracted from PE files, which are widely used in the Windows operating system. By leveraging these features, we were able to train and evaluate various ML classifiers to effectively identify the malicious samples that abuse the CLS for C&C activities.

Through our experiments, as illustrated in Table 5, the RF classifier combined with wrapper-based selected features, emerged as the most effective algorithm, achieving a

high detection rate of 98.15%. This demonstrated the potential of our proposed approach to detect the abuse of CLS as a C&C channel. Furthermore, we conducted a robustness evaluation to examine our approach's resilience against white-box adversarial attacks. Our findings indicated that the proposed method maintained high levels of accuracy of 83% even in the presence of such an attack.

To facilitate further research and development in this area, we introduced a new labeled dataset containing malicious associated with CLS usage and benign PE files. We believe that this dataset, being the first of its kind, will serve as a valuable resource for researchers and practitioners working on the development and evaluation of detection techniques aimed at identifying and countering malware that abuse CLS as a C&C channel.

In essence, our proposed approach highlights the potential of ML-based techniques in effectively detecting the abuse of CLS as a C&C infrastructure. By delivering a solution that is both robust and highly accurate, we contribute to the ongoing efforts in combating such sophisticated cyber threats.

8. Limitations and Future Work

In our proposed work, we utilize static analysis features of PE files to enable ML classifiers to identify the abuse of CLS as a C&C infrastructure. However, the encryption of a PE can pose significant challenges for our technique, especially when attempting to extract pivotal features like presence_of_CLS_domains and potential_C&C_api_calls. As a direction for future research, dynamic analysis could be integrated to provide additional features suitable as input for both ML and deep learning (DL) models. Dynamic analysis focuses on observing a program's real-time execution behavior. This can yield in-depth insights into its functionality and interactions with other systems. Integrating dynamic analysis into our detection methodology would reveal features that static analysis alone might miss.

Overall, future endeavors could focus on the incorporation of dynamic analysis, DL models, and network-based detection techniques to further enhance the accuracy and efficiency of our proposed approach for detecting the abuse of CLS as C&C channels.

Author Contributions: Conceptualization, T.A.l., G.T.; methodology, T.A.l.; validation, T.A.l., G.T.; formal analysis, T.A.l.; investigation, T.A.l.; resources, T.A.l.; writing—original draft preparation, T.A.l.; writing—review and editing, G.T., A.J., E.A.; supervision, G.T.; funding acquisition, T.A.l. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data used to support the findings of this study are available from the corresponding author upon request.

Acknowledgments: The first author thanks the Royal Commission for Jubail and Yanbu (RCJY) and Jubail Industrial College (JIC) for their generous PhD program sponsorship.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Announcing the Public Cloud Market Outlook, 2022 to 2026 Public Cloud's Stormy Path to Growth. Available online: https://www.forrester.com/blogs/announcing-the-public-cloud-market-outlook-2022-to-2026/ (accessed on 14 February 2023).
2. HAMMERTOSS: Stealthy Tactics Define a Russian Cyber Threat Group | FireEye. 2017. Available online: https://www.fireeye.com/current-threats/apt-groups/rpt-apt29.html (accessed on 14 February 2023).
3. Operation Ghost: The Dukes Aren't Back—They Never Left | WeLiveSecurity. 2019. Available online: https://www.welivesecurity.com/2019/10/17/operation-ghost-dukes-never-left/ (accessed on 14 February 2023).
4. Pernet, C.; Cao, E.; Horejsi, J.; Chen, J.C.; Sanchez, W.G. New SLUB Backdoor Uses GitHub, Communicates via Slack. 2019. Available online: https://www.trendmicro.com/en_gb/research/19/c/new-slub-backdoor-uses-github-communicates-via-slack.html (accessed on 14 February 2023).
5. Robert Falcone, B.L. DarkHydrus Delivers New Trojan That Can Use Google Drive for C2 Communications. 2019. Available online: https://unit42.paloaltonetworks.com/darkhydrus-delivers-new-trojan-that-can-use-google-drive-for-c2-communications/ (accessed on 14 February 2023).

6. PE Format—Win32 Apps | Microsoft Learn. Available online: https://learn.microsoft.com/en-us/windows/win32/debug/pe-format (accessed on 14 February 2023).
7. Desktop Operating System Market Share 2013–2023 | Statista. Available online: https://www.statista.com/statistics/218089/global-market-share-of-windows-7/ (accessed on 31 August 2023).
8. VirusTotal—Stats. Available online: https://www.virustotal.com/gui/stats (accessed on 31 August 2023).
9. Inside Windows: An In-Depth Look into the Win32 Portable Executable File Format, Part 2 | Microsoft Learn. Available online: https://learn.microsoft.com/en-us/archive/msdn-magazine/2002/march/inside-windows-an-in-depth-look-into-the-win32-portable-executable-file-format-part-2 (accessed on 3 July 2023).
10. Portable Executable—Wikipedia. Available online: https://en.wikipedia.org/wiki/Portable_Executable (accessed on 9 April 2023).
11. Kartaltepe, E.J.; Morales, J.A.; Xu, S.; Sandhu, R. Social network-based botnet command-and-control: Emerging threats and countermeasures. In Proceedings of the International Conference on Applied Cryptography and Network Security, Beijing, China, 22–25 June 2010; Springer: Berlin/Heidelberg, Germany, 2010; pp. 511–528.
12. Vo, N.H.; Pieprzyk, J. Protecting web 2.0 services from botnet exploitations. In Proceedings of the 2010 Second Cybercrime and Trustworthy Computing Workshop, Ballarat, Australia, 19–20 July 2010; pp. 18–28.
13. Ghanadi, M.; Abadi, M. Socialclymene: A negative reputation system for covert botnet detection in social networks. In Proceedings of the 7'th International Symposium on Telecommunications (IST'2014), Tehran, Iran, 9–11 September 2014; pp. 954–960.
14. Ji, Y.; He, Y.; Jiang, X.; Li, Q. Towards social botnet behavior detecting in the end host. In Proceedings of the 2014 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS), Hsinchu, Taiwan, 16–19 December 2014; pp. 320–327.
15. Thomas, K.; Nicol, D.M. The Koobface botnet and the rise of social malware. In Proceedings of the 2010 5th International Conference on Malicious and Unwanted Software, Nancy, France, 19–20 October 2010; pp. 63–70.
16. Ivanov, A.; Sinitsyn, F. The First Cryptor to Exploit Telegram | Securelist. 2016. Available online: https://securelist.com/the-first-cryptor-to-exploit-telegram/76558/ (accessed on 14 February 2023).
17. Singel, R. Hackers Use Twitter to Control Botnet | WIRED. 2009. Available online: https://www.wired.com/2009/08/botnet-tweets/(accessed on 14 February 2023).
18. Singh, A.; Toderici, A.H.; Ross, K.; Stamp, M. Social Networking for Botnet Command and Control. *Int. J. Comput. Netw. Inf. Secur.* **2013**, *5*. [CrossRef]
19. Nagaraja, S.; Houmansadr, A.; Piyawongwisal, P.; Singh, V.; Agarwal, P.; Borisov, N. Stegobot: A covert social network botnet. In Proceedings of the International Workshop on Information Hiding, Prague, Czech Republic, 18–20 May 2011; Springer: Berlin/Heidelberg, Germany, 2011; pp. 299–313.
20. Burghouwt, P.; Spruit, M.; Sips, H. Towards detection of botnet communication through social media by monitoring user activity. In Proceedings of the International Conference on Information Systems Security, Kolkata, India, 15–19 December 2011; Springer: Berlin/Heidelberg, Germany, 2011; pp. 131–143.
21. Ji, Y.; He, Y.; Jiang, X.; Cao, J.; Li, Q. Combating the evasion mechanisms of social bots. *Comput. Secur.* **2016**, *58*, 230–249. [CrossRef]
22. Singh, A. Social Networking for Botnet Command and Control. 2012. Master's Thesis, San Jose State University, San Jose, CA, USA, 2012. [CrossRef]
23. Burghouwt, P.; Spruit, M.; Sips, H. Detection of covert botnet command and control channels by causal analysis of traffic flows. In Proceedings of the International Symposium on Cyberspace Safety and Security, Zhangjiajie, China, 13–15 November 2013; Springer: Berlin/Heidelberg, Germany, 2013; pp. 117–131.
24. Boshmaf, Y.; Muslukhov, I.; Beznosov, K.; Ripeanu, M. Design and analysis of a social botnet. *Comput. Networks* **2013**, *57*, 556–578. [CrossRef]
25. Ji, Y.; He, Y.; Zhu, D.; Li, Q.; Guo, D. A mulitiprocess mechanism of evading behavior-based bot detection approaches. In Proceedings of the International Conference on Information Security Practice and Experience, Fuzhou, China, 5–8 May 2014; Springer: Berlin/Heidelberg, Germany, 2014; pp. 75–89.
26. Ahmadi, M.; Biggio, B.; Arzt, S.; Ariu, D.; Giacinto, G. Detecting misuse of google cloud messaging in android badware. In Proceedings of the 6th Workshop on Security and Privacy in Smartphones and Mobile Devices, Vienna, Austria, 24 October 2016; pp. 103–112.
27. Arzt, S. Static Data Flow Analysis for Android Applications. 2017. Ph.D. Thesis, Darmstadt University of Technology, Darmstadt, Germany, 2017.
28. VirusTotal—Home. Available online: https://www.virustotal.com/gui/home/upload (accessed on 18 May 2023).
29. Cuckoo Sandbox—Automated Malware Analysis. Available online: https://cuckoosandbox.org/ (accessed on 18 May 2023).
30. Free Software Downloads and Reviews for Windows, Android, Mac, and IOS—Cnet Download. Available online: https://download.cnet.com/windows/ (accessed on 18 May 2023).
31. SourceForge.Net. Available online: https://sourceforge.net/projects/sourceforge/ (accessed on 18 May 2023).
32. Windows Internet—Win32 Apps | Microsoft Learn. Available online: https://learn.microsoft.com/en-us/windows/win32/api/_wininet/ (accessed on 14 February 2023).

33. Santos, I.; Devesa, J.; Brezo, F.; Nieves, J.; Bringas, P.G. Opem: A static-dynamic approach for machine-learning-based malware detection. In Proceedings of the International Joint Conference CISIS'12-ICEUTE' 12-SOCO' 12 Special Sessions, Ostrava, Czech Republic, 5–7 September 2012; Springer: Berlin/Heidelberg, Germany, 2013, pp. 271–280.
34. Shalaginov, A.; Banin, S.; Dehghantanha, A.; Franke, K. Machine learning aided static malware analysis: A survey and tutorial. In *Cyber Threat Intelligence*; Springer: Cham, Switzerland, 2018; pp. 7–45.
35. Baldangombo, U.; Jambaljav, N.; Horng, S.J. A static malware detection system using data mining methods. *arXiv* **2013**, arXiv:1308.2831.
36. Yan, G.; Brown, N.; Kong, D. Exploring discriminatory features for automated malware classification. In Proceedings of the Detection of Intrusions and Malware, and Vulnerability Assessment: 10th International Conference, DIMVA 2013, Berlin, Germany, 18–19 July 2013; Proceedings 10; Springer: Berlin/Heidelberg, Germany, 2013; pp. 41–61.
37. Mlxtend. Available online: https://rasbt.github.io/mlxtend/ (accessed on 18 May 2023).
38. scikit-Learn: Machine Learning in Python—Scikit-Learn 1.2.2 Documentation. Available online: https://scikit-learn.org/stable/ (accessed on 18 May 2023).
39. Naz, S.; Singh, D.K. Review of machine learning methods for windows malware detection. In Proceedings of the 2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT), Kanpur, India, 6–8 July 2019; pp. 1–6.
40. Kumar, A.; Kuppusamy, K.; Aghila, G. A learning model to detect maliciousness of portable executable using integrated feature set. *J. King Saud Univ.-Comput. Inf. Sci.* **2019**, *31*, 252–265. [CrossRef]
41. Raman, K. Selecting features to classify malware. *Infosec Southwest* **2012**, *2012*, 1–5.