MDPI

Article

Machine Learning-based Abuse Detection of Cloud-Based and Public Legitimate Services as C&C Infrastructure

Abstract: The proliferation of Cloud-based 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 have been inadequate at detecting malicious C&C traffic, as this traffic blends with legitimate traffic. This motivates the development of advanced detection techniques. Hence, this paper makes the following contributions to combat such attacks: Firstly, a novel approach is proposed that utilizes machine learning (ML) to detect the abuse of CLS as C&C channels. The approach analyzes static and derivative features extracted from Portable Executable (PE) files to identify the most effective algorithm for detecting the abuses of these services. Various ML classifiers are trained and evaluated, with the Random Forest classifier outperforming other classifiers and achieving a detection rate of 98%. Secondly, a robustness evaluation is undertaken to assess the resilience of the proposed detection model against white-box adversarial attacks. An adversarial 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%. Lastly, a new labeled dataset is introduced as a valuable resource for training and evaluating detection techniques aimed at identifying malicious bots that abuse CLS as C&C channels. The dataset enables researchers and practitioners to develop and test new approaches to counter this type of threat.

Keywords: cloud, malware, command and control, portable executable, dataset, malware detection, machine learning, feature selection, adversarial attack

Citation: Machine Learning-based Abuse Detection of Cloud-Based and Public Legitimate Services as C&C Infrastructure. *J. Cybersecur. Priv.* **2023**, *1*, 1–21. https://doi.org/

Received:
Revised:
Accepted:
Published:

1. Introduction

The increasing demand for cloud solutions across various industries, including healthcare, finance, education, and government, has led to the rapid growth of the global market for public cloud services. According to Forrester, this market is expected to surpass $1 trillion by 2026, more than doubling from $446.4 billion in 2022 [1]. However, with the widespread adoption of cloud services, new challenges for cybersecurity have emerged, including the abuse of CLS as a C&C communication channel.

Attackers can use the CLS to conceal their malicious activities and remotely control compromised systems, enabling them to operate covertly and efficiently while minimizing the chances of being detected or caught. To accomplish this, they exploit the trust between the CLS provider and the user, and abuse these services as C&C communication channels. The ability to conceal their actions and remotely control compromised systems is a key factor that enables adversaries to be more successful in cyber attacks. Examples of malware that have abused CLS as C&C servers include Hammertoss [2], RegDuke [3], SLUB [4], and DarkHydrus [5].

Hammertoss is a remote access tool that used third-party web servers such as LinkedIn, Twitter, and GitHub to avoid detection by security solutions and achieve full access to the victim's system. RegDuke is a piece of malware that abused Dropbox by hosting steganography images containing encrypted malicious commands for covert C&C operations. SLUB is a backdoor that was discovered and assessed by TrendMicro, which abused three legitimate platforms – Slack, GitHub, and File.io – for its C&C infrastructure. DarkHydrus is a

cyber threat group that has been known to abuse legitimate cloud services such as Google Drive for their 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 from 2013 to 2023. Second, Figure 2 indicates that the PE file format is the most commonly submitted among all file types on VirusTotal.

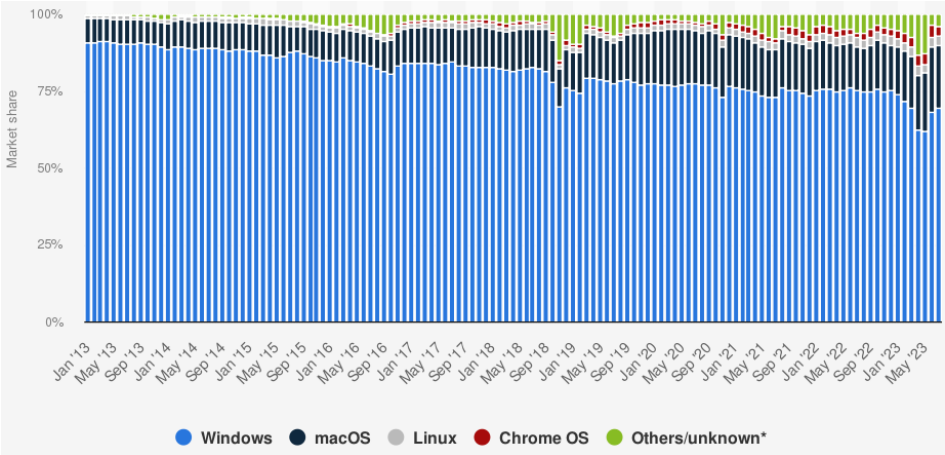

Figure 1. Global market share held by OS for desktop PCs [7].

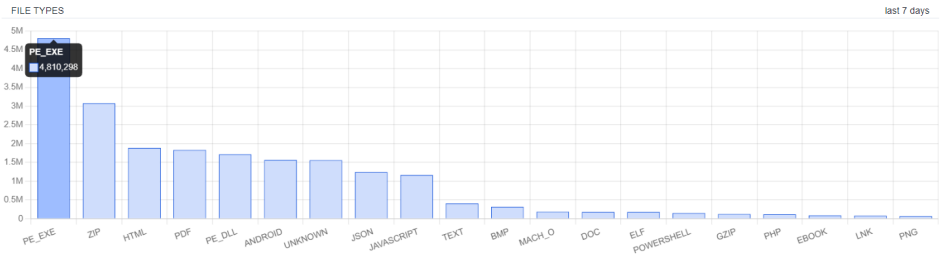

Figure 2. VirusTotal submission by file format [8].

The contributions of this paper are as follows:

- We introduce a unique approach that utilizes ML to detect the abuse of CLS as C&C communication channels. To the best of our knowledge, no prior works have utilized ML-based detection techniques to detect the abuse of such services. Our approach achieves a detection rate of 98%, demonstrating its effectiveness in detecting the abuse of CLS as C&C communication channels.
- We evaluate the robustness of our proposed abuse detection system by performing a specifically designed white-box adversarial attack, which we call Replace Misclassified Parameter (RMCP). 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.

- We provide a labeled dataset of PE files, consisting of malware samples that initiate network connections to CLS and benign samples that initiate legitimate connections to the Internet. This dataset is the first of its kind and serves as a valuable resource for training and evaluating ML classifiers to detect the abuse of CLS.

The remainder of this paper is structured as follows. Section 2 is an overview of the PE files and 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 the proposed approach's methodology and experimental setup and results are presented in Section 4 and Section 5. Comparison to related works and robustness evaluation and comparison are presented in Section 6 and Section 8. Finally, limitations, future work, and the conclusion are presented in Section 9 and Section 10.

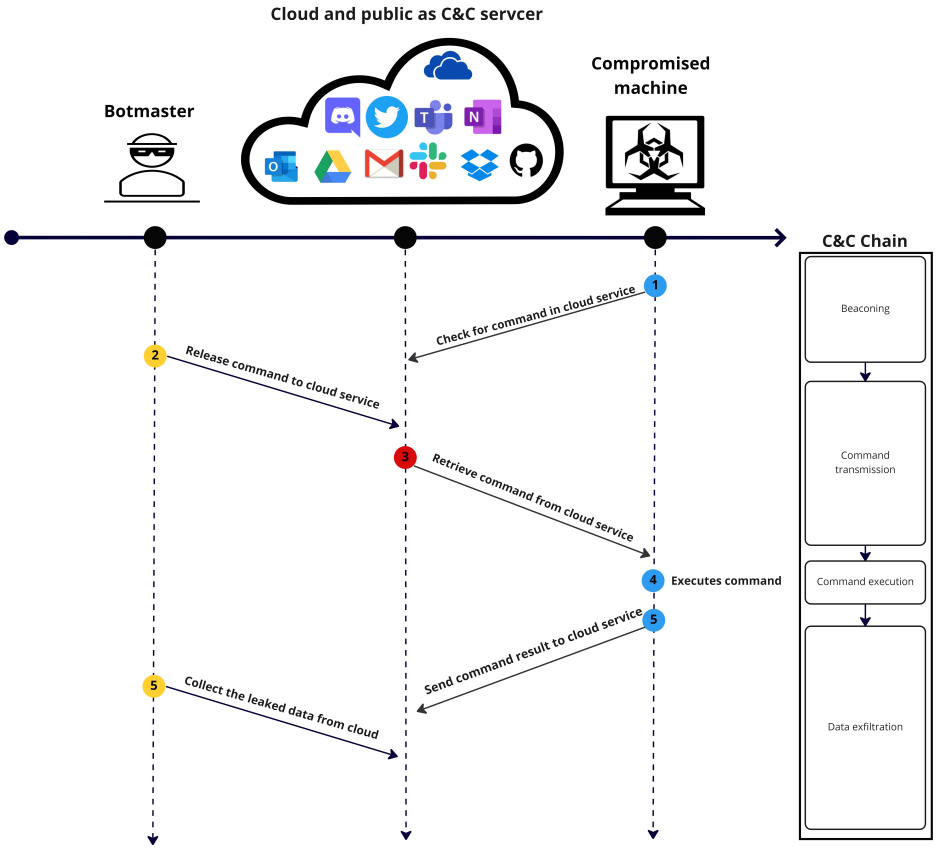

Figure 3. Abuse of CLS as C&C infrastructure

2. Background

In Section 2.1, we summarize the essential context of 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 B, 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 4, the PE file format comprises several fields: the 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 2.

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.

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 the bots in 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 3. This is achieved by the attacker using cloud services or other legitimate services to communicate with the 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 that 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 service as a communication channel.

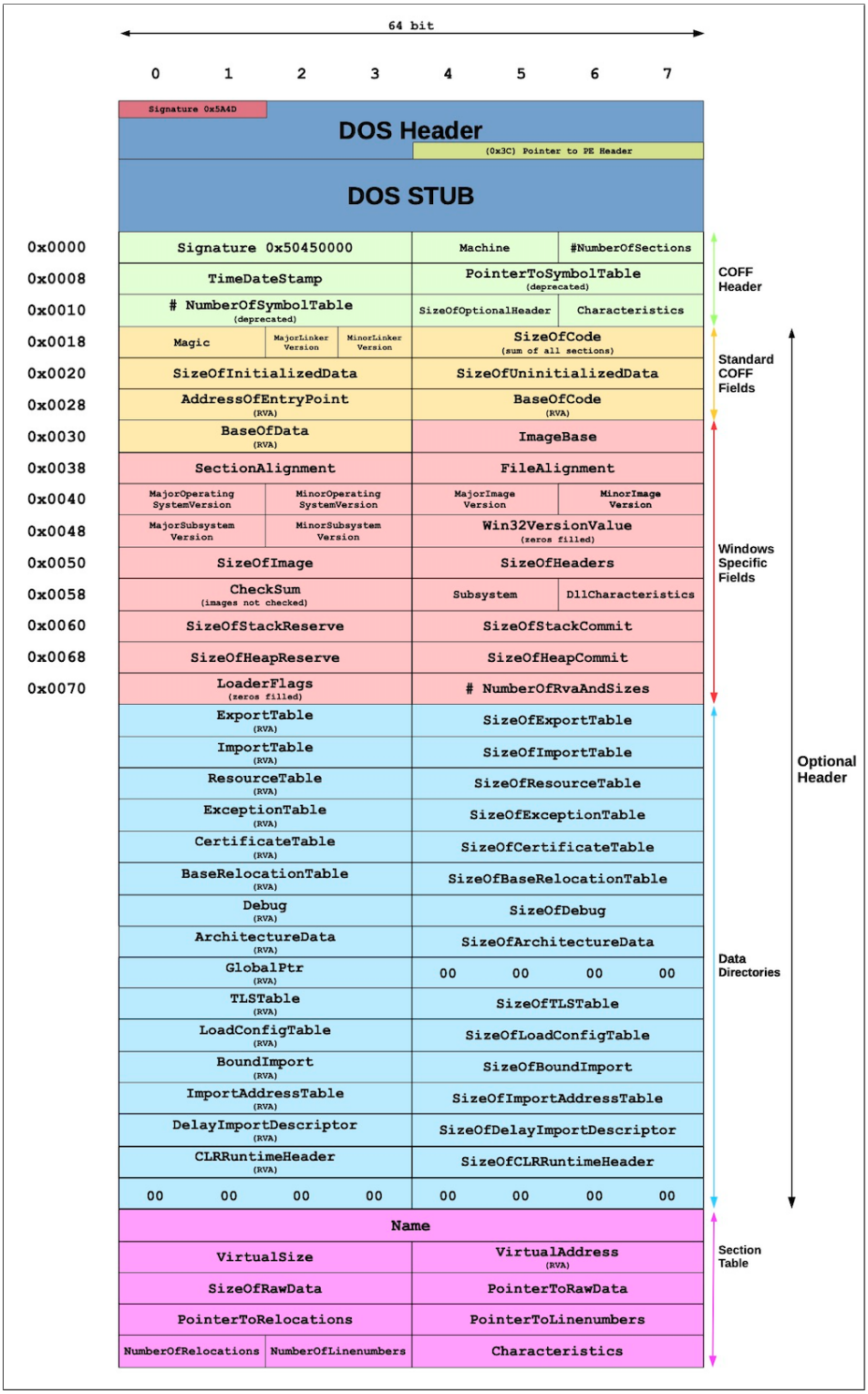

Figure 4. Structure of a Portable Executable [10]

2.3.3. Threat Scenario

Once the C&C communication channels have been established, the attacker can use the botnet to conduct a range of malicious activities, such as data theft, denial of service attacks, or the distribution of further 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 implemented specifically for use on 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] developed a two-level abuse detection system. The client side identified botnets using self-concealing behaviors, suspicious traffic, and unreliable origins, differentiating between legitimate users and bots through behavioral metrics and GUI interactions. The server side viewed encoded messages on social media as suspicious, using the J48 decision tree to classify content. However, the system wasn't real-time detection and could be bypassed with image-steganography by adept adversaries.

Vo et al. [12] introduced the API Verifier, which uses CAPTCHA to authenticate social media access via MAC addresses. It distinguishes between human and bot API calls, adding a layer of defense against bots' activities. However, it's vulnerable to relay attacks, where CAPTCHAs are solved by humans for bots, and MAC address spoofing, which can evade detection.

Ghanadi et al. [13] investigated stego-botnets using steganographic images on social networks for CC tasks. They introduced SocialClymene, a system assigning negative scores to users based on past activities. This system aggregates suspicious values in a graph, which are then weighted by adjacent nodes' reputation scores, highlighting potential malicious actions.

3.2. Behavior Tree-based Detection

Yuede et al. [14] introduced a behavior tree-based system to detect social bots by observing host activities. Using a designed social botnet, wbbot, they created a suspicious behavior tree from real-world and research-based bot samples. After analyzing these bots, a template library was made to measure similarity with the suspicious behavior tree. The final detection relied on tree edit distance calculations. Notably, this method had a high false positive rate of 29.6%, and crafty attackers could potentially bypass it with a multi-process tactic or spread malicious behaviors over varied time intervals.

Burghouwt et al. [15] introduced a causality detection mechanism that identifies Twitter-based C&C channel communication by measuring the correlation between user activity and network traffic. The authors made an assumption that any network traffic event to the OSN that is not caused by human events based on specific keystrokes or mouse actions can be deemed suspicious. The causality detection method employs a time frame starting immediately post-user event to distinguish network incidents instigated by user actions from those initiated by bots. However, this detection approach has certain limitations. Firstly, legitimate API calls used for periodically automated polling may be falsely flagged as suspicious. Secondly, the primary parameters employed to evaluate the time frame between user activity and network requests may lack accuracy, given that different machines and operating systems exhibit varying delay times and performance

characteristics. Lastly, advanced bots can potentially bypass this detection technique by monitoring user events and executing commands based on user-triggered events.

Burghouwt et al. [15] proposed a causality detection method to identify suspicious Twitter-based C&C communications by correlating user activity with network traffic. They deemed non-human-initiated network traffic as suspect if not linked to specific human interactions. This method uses a post-user event time frame to differentiate between user-driven and bot-driven network requests. However, there are concerns with this approach. System-initiated legitimate API calls might be mistakenly labeled as suspicious, and sophisticated bots can potentially elude detection by imitating user behaviors.

3.3. ML-based Detection

Ji *et al.* [16] assessed several social bots, Twitterbot (Singh [17]), Twebot (Burghouwt *et al.* [18]), Yazanbot (Boshmaf *et al.* [19]), Nazbot (Kartaltepe *et al.* [11]), wbbot (Ji *et al.* [20]), and fbbot. Drawing from their analysis, they proposed a detection strategy using 18 features: nine new and nine from previous methods. 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.* [21] introduced a method to detect Android apps using Google Cloud Messaging (GCM) for CC. By modifying the Flowdroid tool [22] to extract GCM flows and identify GCM callbacks, they trained an ML model using features like GCM registration ID and message type. Their findings indicate that GCM flow features can effectively distinguish malicious applications.

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, 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 [23] 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 1. The remaining corpus was executed in a controlled Cuckoo sandbox environment ([24]), retaining only samples demonstrating CLS domain connections for our final experimental dataset, as depicted in Figure 5. We excluded any samples not connecting to the CLS domain.

Table 1. 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		

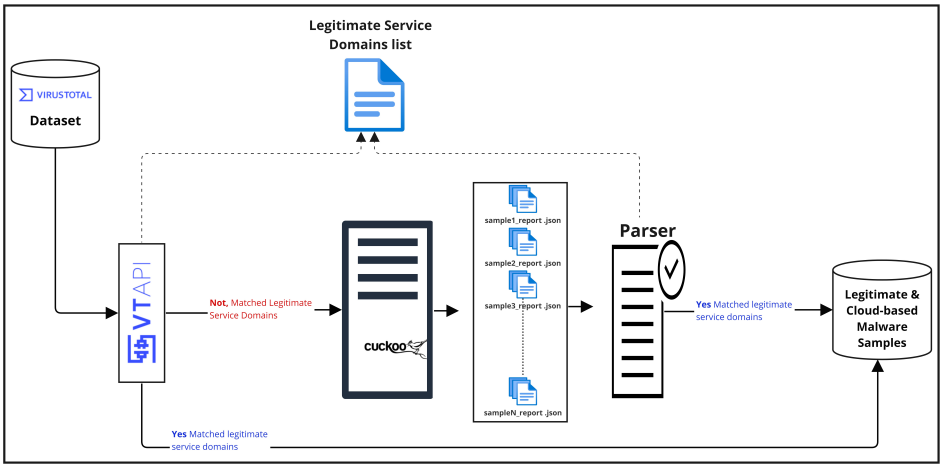

Figure 5. Detailed workflow for extracting a sub-dataset from the VirusTotal datasets

Additionally, the benign samples included in the dataset were obtained from the sources of cnet [25] and sourceforge [26]. 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.

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 0, 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 [24] were 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.

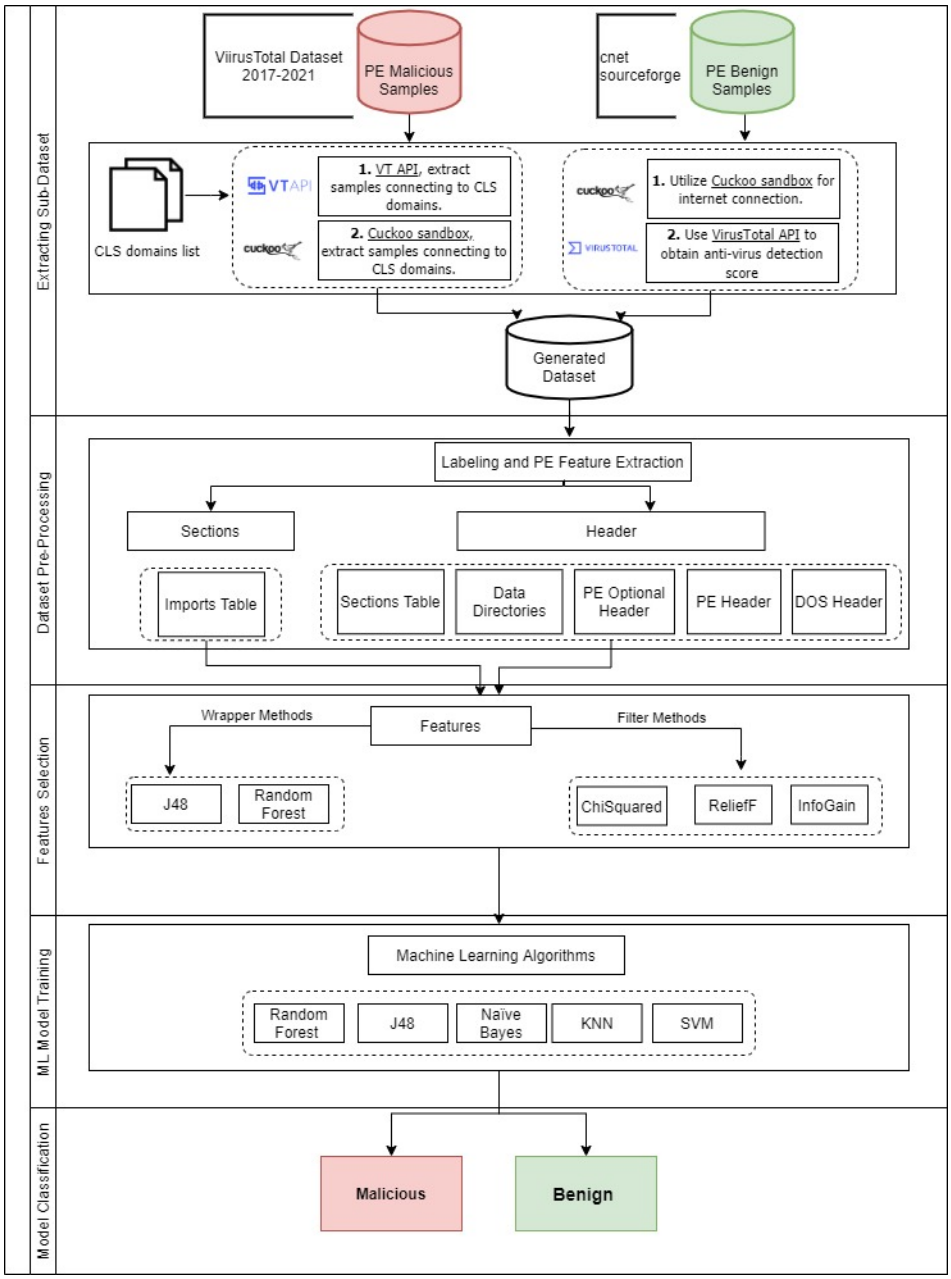

Figure 6. Illustrative overview of the proposed detection system.

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 2). Raw features can be directly extracted from the PE file with no further processing. Such features are Characteristics, DllCharacteristics, SizeOfImage, AddressOfEntryPoint, and ResourceSize.

Table 2. 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.

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 1). The feature value is set to 1 if a CLS domain is found; otherwise, it is set to 0. 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 [27] are listed in Table 3. The feature value is set to 1 if a potential C&C API call is identified; otherwise, it is set to 0.

Table 3. Potential API calls for abusing CLS

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

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) [28–30], Chi-Squared, and ReliefF [31].

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 implement 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 Figures 7, 8, and 9, respectively.

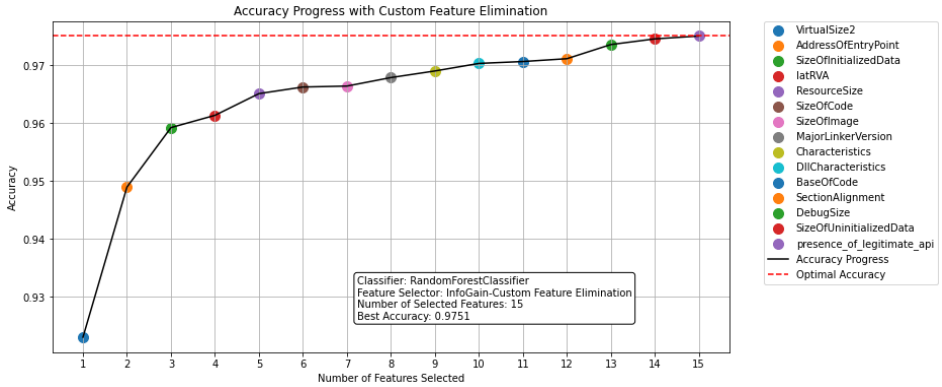

Figure 7. Accuracy and subset of features using InfoGain

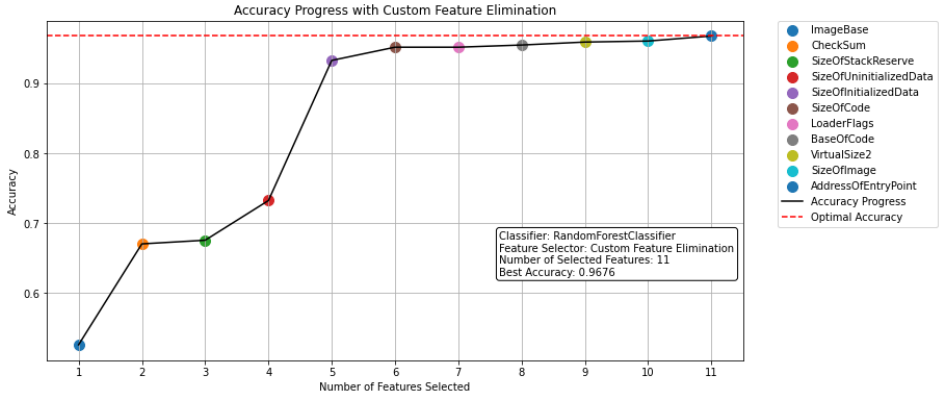

Figure 8. Accuracy and subset of features using Chi-Squared

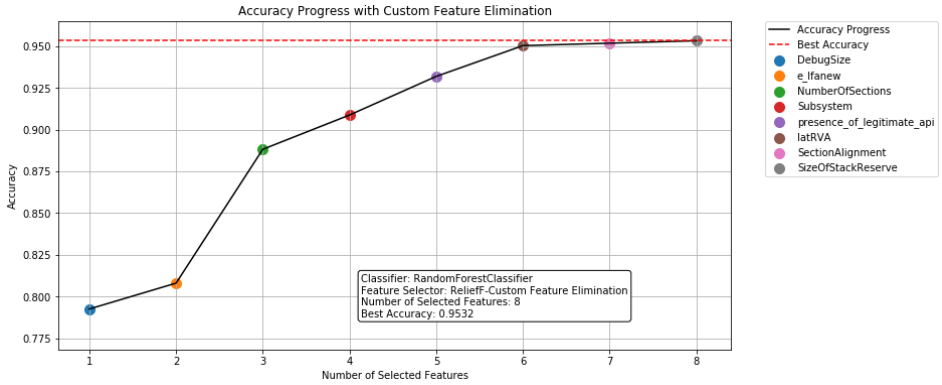

Figure 9. Accuracy and subset of features using Chi-Squared

4.3.2. Wrapper-based Feature Selector

Besides 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 [32].

Figures 10 and 11 display the highest accuracy rates achieved by each combination, using a different number of selected features ranging from 1 to the maximum number of features (38) in the dataset.

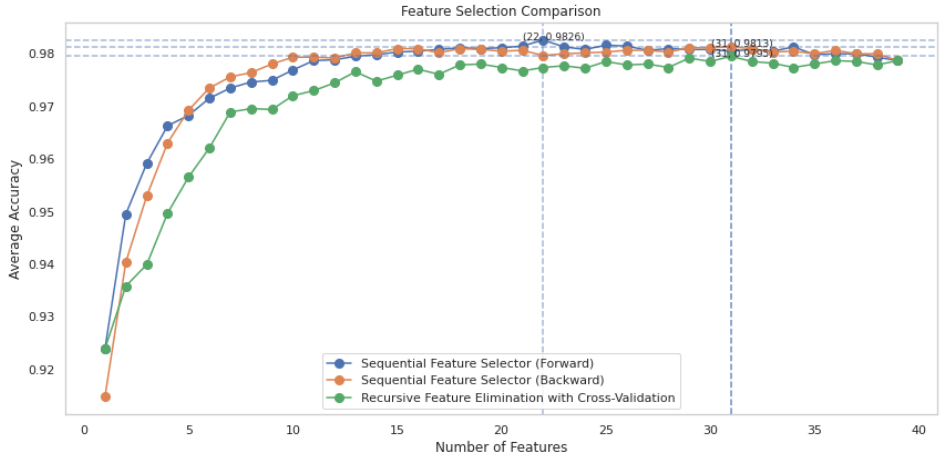

Figure 10. Comparative analysis of wrapper feature selection: RF-SFSF vs. RF-SFSB vs. RF-RFE

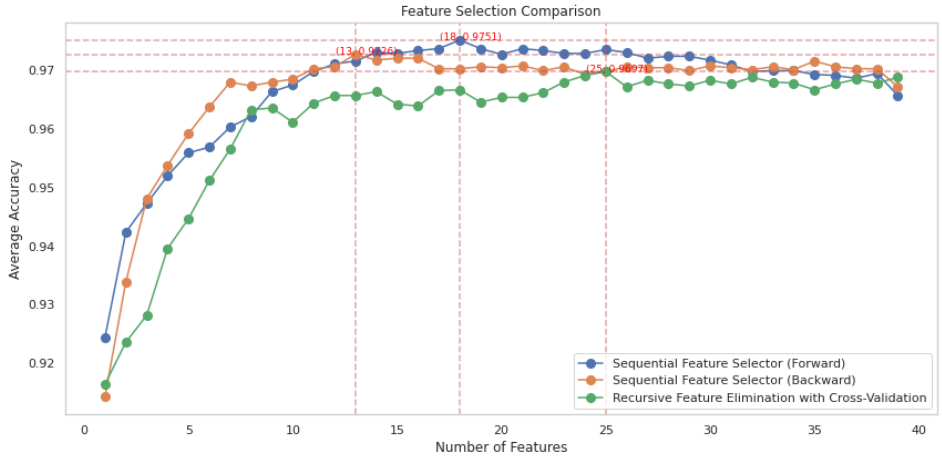

Figure 11. Comparative analysis of wrapper feature selection: T-SFSF vs. DT-SFSB vs. DT-RFE

4.4. Classification

Our evaluation applies five ML-based classifiers, which are implemented in Scikit-learn [33]: 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 [28,31,34]. By employing a range of classifiers, we aim to comprehensively assess the performance of various ML classifiers on our dataset, which is detailed in Subsection 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 [33] for implementing these ML algorithms. All evaluations took place on a computing environment powered by an Intel i7-9700K processor clocked at 3.60GHz, complemented with 32GB of RAM, and running the Windows 10 operating system.

5. Experimental Results and Analysis

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 4. 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.

Table 4. Comparative analysis of abuse detection accuracy: all features vs. selected features

Classifier	No Features Selector		Features Selector									
			Wrapper Methods				Filter Methods					
			Random Forest as Feature Selector		J48 as Feature Selector		InfoGain		ChiSquared		ReliefF	
	70.0% train, remainder test	10-fold cross-validation	70.0% train, remainder test	10-fold cross-validation	70.0% train, remainder test	10-fold cross-validation	70.0% train, remainder test	10-fold cross-validation	70.0% train, remainder test	10-fold cross-validation	70.0% train, remainder test	10-fold cross-validation
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%**	**98.25%**	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 presence_of_legitimate_api	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

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.

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. Hence, this section discusses the result of an accuracy detection rate achieved with selected features.

Table 4 compares the detection accuracy of various classifiers after feature selection, employing both wrapper and filter techniques.

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 ChiSquared. For example, the detection accuracy with InfoGain as a feature selector is 97.12% for 70:30 split and 97.47% for 10-fold cross-validation, while the accuracy using ChiSquared as a feature selector is 96.47% for 70:30 split and 96.59% for 10-fold cross-validation.

Figures 10 and 11 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% 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 4.

6. Comparison to related works

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.* [35] and Raman *et al.* [36]. It's essential to highlight that the evaluations for both our work and the other two studies were based on our unique dataset.

Table 5 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 5. 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.* [35]	94.46	94.90	93.63	94.26	94.68	94.93	94.08	94.50
Raman *et al.* [36]	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

7. Adversarial Attack

To compare the robustness of our ML models with related works, we conducted a white-box adversarial attack to evaluate their performance against adversarial attacks that manipulate 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 6.

- 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 where the model incorrectly predicted malware (1) when the sample was actually benign (0), which means the model was misclassifying 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 the model identified a software as malware and it was indeed malware.

Table 6. Confusion matrix

		Predicted	
		Benign	**Malware**
Actual	**Benign**	True Positive (TP)	False Positive (FP)
	Malware	False Negative (FN)	True Negative (TN)

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 7 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 7. Features whose value modification corrupts PE and causes of corruption

Features	**Reasons for Potential Corruption**
AddressOfEntryPoint	Start 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	Lead to incorrect memory allocation.
SizeOfImage	Lead to incorrect memory allocation.
IatRVA	Break 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 are no modifications, as shown in Table 4.
   After each substitution, we evaluate 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 7, as tampering with them can corrupt the file.

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.

Algorithm 1: RMCP Adversarial Attack

```
Require: Data as features X and targets y
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
```

The results of our robustness evaluation against adversarial attacks are summarized in Table 8, 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 8, it's 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 8. Evaluating the impact of adversarial attacks on detection accuracy: proposed work

Features	Benign Feature Parameter	Original Accuracy	Accuracy After Replaced Feature Parameter	Original Confusion Matrix	Confusion Matrix After Replacing Feature Parameter
MajorLinkerVersion	8	0.981532	0.977729	TP:935 FP:11 FN:23 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	65536		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	32467821		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
presence_of_legitimate_api	0		0.965779		TP:935 FP:11 **FN:52** TN:843

8. Robustness Comparison to related works

In comparison, Table 9 presents the robustness results for Kumar et al. [35], 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.

Table 9. Evaluating the impact of adversarial attacks on detection accuracy: Kumar *et al.* [35]

Features	Benign Feature Parameter	Original Accuracy	Accuracy After Replaced Feature Parameter	Original Confusion Matrix	Confusion Matrix After Replacing Feature Parameter
MajorOperatingSystemVersion	1	0.946768	0.923411	TP: 901 FP: 45 FN: 53 TN: 842	TP: 901 FP: 45 **FN: 96** TN: 799
DllCharacteristics	34112		0.813688		TP: 901 FP: 45 **FN: 298** TN: 597
SizeOfStackReserve	16777216		0.811515		TP: 901 FP: 45 **FN: 302** TN: 593
MajorSubsystemVersion	6		0.937534		TP: 901 FP: 45 **FN: 70** TN: 825
MinorSubsystemVersion	0		0.941879		TP: 901 FP: 45 **FN: 62** TN: 833
Characteristics	263		0.690386		TP: 901 FP: 45 **FN: 525** TN: 370
e_lfanew	304		0.697990		TP: 901 FP: 45 **FN: 511** TN: 384
CreationYear	1		0.827811		TP: 901 FP: 45 **FN: 272** TN: 623

Similarly, Table 10 illustrates the impact of the RMCP technique on Raman *et al.* [36], 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. Evaluating the impact of adversarial attacks on detection accuracy: Raman *et al.* [36]

Features	Benign Feature Parameter	Original Accuracy	Accuracy After Replaced Feature Parameter	Original Confusion Matrix	Confusion Matrix After Replacing Feature Parameter
MajorImageVersion	10	0.956545	0.697447	TP: 914 FP: 32 FN: 48 TN: 847	TP: 914 FP: 32 **FN: 525** TN: 370
DebugSize	84		0.543726		TP: 914 FP: 32 **FN: 808** TN: 87
ExportSize	393079		0.951113		TP: 914 FP: 32 **FN: 58** TN: 837
ResourceSize	5296		0.762629		TP: 914 FP: 32 **FN: 405** TN: 490

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 8, 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. [35] and Raman et al. [36], respectively.

These findings show the crucial role of white-box adversarial attacks. As depicted in Table 5, 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 decline. This demonstrates that our work is more resilient to adversarial attacks, thus offering a more robust and reliable solution for abuse detection.

9. Limitations and Future Work

In our proposed work, we utilize static analysis features of PE files to, enabling ML classifiers to identify the 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.

10. Conclusion

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 4, 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.

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

Funding: This research was funded by the Jubail Industrial College (JIC) at Royal Commission for Jubail and Yanbu (RCJY), Saudi Arabia.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

References

1. Announcing The Public Cloud Market Outlook, 2022 To 2026 Public cloud's stormy path to growth. https://www.forrester.com/blogs/announcing-the-public-cloud-market-outlook-2022-to-2026/.
2. HAMMERTOSS: Stealthy Tactics Define a Russian Cyber Threat Group | FireEye. https://www.fireeye.com/current-threats/apt-groups/rpt-apt29.html, 2017.

3. Operation Ghost: The Dukes aren't back – they never left | WeLiveSecurity. https://www.welivesecurity.com/2019/10/17/operation-ghost-dukes-never-left/, 2019.
4. Pernet, C.; Cao, E.; Horejsi, J.; Chen, J.C.; Sanchez, W.G. New SLUB Backdoor Uses GitHub, Communicates via Slack. https://www.trendmicro.com/en_gb/research/19/c/new-slub-backdoor-uses-github-communicates-via-slack.html, 2019.
5. Robert Falcone, B.L. DarkHydrus delivers new Trojan that can use Google Drive for C2 communications. https://unit42.paloaltonetworks.com/darkhydrus-delivers-new-trojan-that-can-use-google-drive-for-c2-communications/, 2019.
6. PE Format - Win32 apps | Microsoft Learn. https://learn.microsoft.com/en-us/windows/win32/debug/pe-format.
7. Desktop operating system market share 2013-2023 | Statista. https://www.statista.com/statistics/218089/global-market-share-of-windows-7/. (Accessed on 08/31/2023).
8. VirusTotal - Stats. https://www.virustotal.com/gui/stats. (Accessed on 08/31/2023).
9. Inside Windows: An In-Depth Look into the Win32 Portable Executable File Format, Part 2 | Microsoft Learn. 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 03/07/2023).
10. Portable Executable - Wikipedia. https://en.wikipedia.org/wiki/Portable_Executable. (Accessed on 09/04/2023).
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. Springer, 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. IEEE, 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). IEEE, 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). IEEE, 2014, pp. 320–327.
15. 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. Springer, 2011, pp. 131–143.
16. Ji, Y.; He, Y.; Jiang, X.; Cao, J.; Li, Q. Combating the evasion mechanisms of social bots. *computers & security* **2016**, *58*, 230–249.
17. Singh, A. Social networking for botnet command and control **2012**.
18. 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. Springer, 2013, pp. 117–131.
19. Boshmaf, Y.; Muslukhov, I.; Beznosov, K.; Ripeanu, M. Design and analysis of a social botnet. *Computer Networks* **2013**, *57*, 556–578.
20. 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. Springer, 2014, pp. 75–89.
21. Ahmadi, M.; Biggio, B.; Arzt, S.; Ariu, D.; Giacinto, G. Detecting misuse of google cloud messaging in android badware. In Proceedings of the Proceedings of the 6th Workshop on Security and Privacy in Smartphones and Mobile Devices, 2016, pp. 103–112.
22. Arzt, S. Static data flow analysis for android applications **2017**.
23. VirusTotal - Home. https://www.virustotal.com/gui/home/upload.
24. Cuckoo Sandbox - Automated Malware Analysis. https://cuckoosandbox.org/. (Accessed on 05/18/2023).
25. Free software downloads and reviews for Windows, Android, Mac, and IOS – Cnet Download.
26. SourceForge.Net. https://sourceforge.net/projects/sourceforge/.
27. Windows Internet - Win32 apps | Microsoft Learn. https://learn.microsoft.com/en-us/windows/win32/api/_wininet/.
28. 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. Springer, 2013, pp. 271–280.
29. Shalaginov, A.; Banin, S.; Dehghantanha, A.; Franke, K. Machine learning aided static malware analysis: A survey and tutorial. *Cyber threat intelligence* **2018**, pp. 7–45.
30. Baldangombo, U.; Jambaljav, N.; Horng, S.J. A static malware detection system using data mining methods. *arXiv preprint arXiv:1308.2831* **2013**.
31. 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, July 18-19, 2013. Proceedings 10. Springer, 2013, pp. 41–61.
32. mlxtend. https://rasbt.github.io/mlxtend/.
33. scikit-learn: machine learning in Python — scikit-learn 1.2.2 documentation. https://scikit-learn.org/stable/.
34. 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). IEEE, 2019, pp. 1–6.
35. Kumar, A.; Kuppusamy, K.; Aghila, G. A learning model to detect maliciousness of portable executable using integrated feature set. *Journal of King Saud University-Computer and Information Sciences* **2019**, *31*, 252–265.
36. Raman, K.; et al. Selecting features to classify malware. *InfoSec Southwest* **2012**, *2012*, 1–5.