GraphTunnel: Robust DNS Tunnel Detection Based on DNS Recursive Resolution Graph

Guangyuan Gao, Weina Niu, *Senior Member, IEEE*, Jiacheng Gong, *Graduate Student Member, IEEE*, Dujuan Gu, Song Li, *Member, IEEE*, Mingxue Zhang, and Xiaosong Zhang

***Abstract*— DNS tunnels, due to their versatility and concealment, have become a preferred method for attackers to execute Command and Control (C&C) attacks, posing a significant security threat to terminal devices. Therefore, the efficient and accurate detection of DNS tunnels is important in reducing the economic losses and privacy risks faced by both enterprises and individuals. Despite notable advancements in the research of intelligent detection of DNS tunnels, existing model-based approaches predominantly concentrate on the surface-level features of domain names or packet payloads. This narrow focus leads to low detection accuracy when dealing with unknown DNS tunnel attacks and traffic from wildcard DNS. Furthermore, these methods struggle with accurately identifying DNS tunneling tools, complicating the task of swiftly locating and mitigating malware for analysts. This paper proposes GraphTunnel, a framework based on graph neural networks for detecting DNS tunnels and identifying tunneling tools. It delves into the correlations among DNS resolutions to construct paths that represent the recursive resolution process of DNS. By using central nodes that denote the gateways, these paths are connected and transformed into graph structures. Concurrently, it employs GraphSage to aggregate the features of nodes and their edges in the graph, enabling effective detection of DNS tunnels. Additionally, GraphTunnel utilizes the G2M algorithm to capture the statistical features of nodes in the graph and maps them into grayscale images, which are then processed by a CNN for multi-class identification of DNS tunneling tools. Experimental results demonstrate that in non-wildcard DNS scenarios, GraphTunnel achieves a 100% accuracy in DNS tunnel detection, encompassing unknown DNS tunnels. Even in high false-positive environments caused by wildcard DNS, GraphTunnel maintains an F1-Score of 99.78%. Moreover, GraphTunnel can identify DNS tunneling tools with an accuracy rate exceeding 98.57%, enhancing the rapid mitigation capabilities of emergency responders in dealing with malicious DNS tunnels.**

***Index Terms*— DNS tunnel detection, unknown DNS tunnels, wildcard DNS, tunneling tool identification, graph neural networks.**

I. Introduction

DNS, as a fundamental internet infrastructure, facilitates the translation of domain names into IP addresses for user access to resources. However, the omnipresence and inherent stealth of DNS render it susceptible to exploitation by hackers for DNS tunneling attacks. A study conducted by the National Institute of Standards and Technology (NIST) [1] revealed that in 2021, an astounding 72% of organizations globally were subjected to DNS attacks. These attacks encompassed Distributed Denial of Service (DDoS) (46%), DNS tunneling (35%), and cache poisoning (33%). Furthermore, the "Global DNS Threat Report" disseminated by EfficientIP in 2022 [2] disclosed that 88% of companies encountered DNS attacks, involving tactics such as DNS phishing, DNS tunneling, and DNS-based malware. Among these DNS attacks, those utilizing DNS tunneling techniques constituted 28%, marking a 4% surge compared to the previous year. On average, these attacks precipitated a financial loss of $942,000, with 24% of enterprises experiencing data exfiltration, imposing severe consequences on both businesses and individuals.

In response to the security threats posed by DNS tunneling, numerous researchers are currently engaged in studies to detect DNS tunnels promptly [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21].

Current DNS tunnel detection methods are primarily categorized into rule-based and model-based approaches [3]. Rule-based methods rapidly identify tunnel traffic by matching packet signatures or comparing specific feature thresholds. However, these methods rely on rule sets generated from specific tunneling software and may fail to effectively detect software deliberately modified by attackers. In contrast, model-based approaches learn features from large-scale raw traffic data, capturing distinctions between benign DNS traffic and tunnel traffic. Leveraging the characteristics of binary

Manuscript received 11 January 2024; revised 2 June 2024 and 1 August 2024; accepted 6 August 2024. Date of publication 14 August 2024; date of current version 22 August 2024. This work was supported in part by CCF-NSFOCUS "Kunpeng" Research Fund under Grant CCF-NSFOCUS 2023013, in part by the National Science Foundation of China under Grant 62372086 and Grant U2336204, in part by Sichuan Natural Science Foundation under Grant 24ZNSFSC0038, and in part by the Financial Support for Outstanding Talents Training Fund in Shenzhen. The associate editor coordinating the review of this article and approving it for publication was Dr. Daisuke Mashima. *(Corresponding author: Weina Niu.)*

Guangyuan Gao and Jiacheng Gong are with the School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China (e-mail: ggyshuaige@gmail.com; gongjc.uestc@gmail.com).

Weina Niu and Xiaosong Zhang are with the School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China, and also with the Institute for Advanced Study, University of Electronic Science and Technology of China, Shenzhen 518110, China (e-mail: vinusniu@uestc.edu.cn; johnsonzxs@uestc.edu.cn).

Dujuan Gu is with NSFOCUS Technologies Group Company Ltd., Beijing 100089, China (e-mail: gudujuan@nsfocus.com).

Song Li and Mingxue Zhang are with the State Key Laboratory of Blockchain and Data Security and the School of Cyber Science and Technology, Zhejiang University, Hangzhou 310058, China (e-mail: songl@zju.edu.cn; mxzhang97@zju.edu.cn).

Digital Object Identifier 10.1109/TIFS.2024.3443596

classification algorithms, these methods train highly accurate detection models, offering flexibility to adapt to changes in different tunneling software. Nevertheless, existing model-based methods encounter the following challenges:

C1: Suboptimal Accuracy in Detecting Unknown DNS Tunnels and Wildcard DNS. Existing model-based detection methods typically focus solely on surface-level features of domain names or packet payloads, lacking attention to the behavioral structure characteristics of the establishment and attack processes of DNS tunnels. This approach exhibits reduced detection performance when confronted with unknown DNS tunnel attacks and wildcard DNS.

- Multiple existing studies on DNS tunnel detection do not explicitly address the proble of traffic generated by unknown DNS tunneling tools. In studies that do focus on detecting unknown samples, such as [21] and [20], a "leave-one-out" method is employed to partition the dataset. However, these studies have not conducted detection experiments specifically on traffic generated by a completely unknown DNS tunneling tool.
- Wildcard DNS is a technique that permits subdomains to utilize the wildcard * for ambiguous matching. This characteristic endows subdomains with flexibility in terms of length and character arrangement, closely resembling the domain names employed by DNS tunnels for data transmission. When a DNS tunnel detection system employs surface-level domain features such as subdomain length and information entropy for detection, it can easily engender confusion between legitimate wildcard domains and potential DNS tunnels, thereby precipitating instances of false positives.

C2: Suboptimal Accuracy in Distinguishing DNS Tunneling Tools. Despite the utilization of diverse encryption techniques by recognized DNS tunneling tools, there is a notable overlap in certain features such as domain length, information entropy, and packet size. This similarity impedes the model's ability to discern the nuanced differential features among various tools, thereby leading to diminished accuracy in the multi-classification task pertaining to DNS tunneling tools.

To address the aforementioned challenges, we propose GraphTunnel, a framework based on graph neural networks designed for real-time detection of DNS tunnels and identification of tunneling tools. Specifically, to tackle **C1**, GraphTunnel filters DNS traffic from network traffic and constructs paths representing the DNS recursive resolution process. The nodes in each path symbolize authoritative domains, with edges mapping correlations among these domains. To enhance the modeling of inbound and outbound traffic dynamics during DNS resolution, a central node is used to denote the gateway. This central node effectively connects individual paths, thus obtaining a comprehensive graph representation. Subsequently, GraphTunnel employs the GraphSAGE [22] algorithm to aggregate node and edge features within the graph. In this way, GraphTunnel fits the DNS recursive resolution process well and extracts unique spatio-temporal features from it, thereby maintaining high robustness even in the scenarios of unknown DNS tunnels and wildcard DNS. To address **C2**, GraphTunnel utilizes the G2M algorithm to statistically learn the node features within the graph and convert them into grayscale images. A convolutional neural network (CNN) is then applied to process the image, resulting in a multi-classification model for DNS tunneling tools. By this method, GraphTunnel effectively utilizes the statistical information and the capability of CNN, and gradually highlights the differences in node features within the graph, thus achieving high accuracy in distinguishing DNS tunneling tools.

In summary, the primary contributions of this paper are as follows:

- We constructed DNS recursive resolution graphs and employed graph neural networks to detect DNS tunnels by analyzing the different behavioral graph patterns between normal DNS resolution and DNS tunnel resolution.
- We developed the G2M algorithm to improve the multi-classification of DNS tunneling tools by statistically analyzing node feature vectors and organizing them into a grayscale image matrix for effective convolutional aggregation.
- Experimental results demonstrate that GraphTunnel achieves an F1 Score exceeding 99.78% in DNS tunnel detection, maintaining high robustness even against unknown DNS tunnels and wildcard DNS scenarios. Furthermore, it achieves high accuracy exceeding 98% in the multi-classification of DNS tunneling tools.

The structure of this paper is as follows: Section II furnishes an overview of current endeavors in detecting DNS tunnels. Section III delves into the architecture of the GraphTunnel system, elucidating the composition and interrelationships of each module. In Section IV, we expound on the experimental environment settings and dataset collection, showcasing the results of the experimental evaluation. The discussion of research limitations is succinctly presented in Section V, culminating in the overall conclusion in Section VI.

II. Related Work

Wang et al. [3] conduct a comprehensive analysis of nearly all detection methods developed from 2006 to 2020, categorizing DNS tunnel detection methods into two types: rule-based detection and model-based detection. Building on this, we analyze some recent studies based on several indicators. For instance, methods with an accuracy rate exceeding 90% are classified as HA (High-Accuracy), those encompassing datasets from five or more tunneling tools are labeled as ETC (Extensive Tool Coverage), and those capable of detecting DNS tunnels on different platforms are categorized as CP (Cross-Platform). Additionally, we introduce specific metrics to assess the capabilities of these methods, including UDT (Unknown DNS Tunnel), WD (Wildcard DNS), TTI (Tunneling Tool Identification), and RM (Real-time Monitoring). The details are outlined in Table I.

TABLE I
COMPARISON OF EXISTING METHODS FOR DNS TUNNELING DETECTION

Typical Method	Method		HA	ETC	UDT	CP	WD	TTI	RM
rule-based	signature-based	Sheridan [4]	×	×	×	×	×	×	×
		Adival [5]	✓	×	×	×	×	×	×
		Salat [6]	×	×	×	×	×	×	×
		Ghosh [7]	×	×	×	×	×	×	×
	threshold-based	ozery [8]	✓	×	×	×	×	×	✓
		Sani [9]	×	×	×	×	×	×	×
		Ellens [10]	×	×	×	×	×	×	✓
		Paxson [11]	×	✓	×	×	×	×	×
		Ishikura [12]	✓	×	×	×	×	×	×
model-based	Ibraheemi [13]		✓	×	×	×	×	×	×
	Sakarkar [14]		✓	×	×	×	×	×	×
	Lal [15]		✓	×	×	×	×	×	×
	D'Angelo [16]		✓	×	×	×	×	×	×
	Bai [17]		✓	×	×	×	×	×	×
	Altuncu [18]		✓	×	×	×	×	×	✓
	Shafician [19]		✓	×	×	×	×	×	×
	Liang [20]		✓	✓	✓	×	×	×	×
	Wang [21]		✓	×	✓	×	×	×	×
	GraphTunnel		✓	✓	✓	✓	✓	✓	✓

A. Rule-Based Detection

Rule-based detection can be further divided into two categories: signature-based methods and threshold-based methods [3].

① *Signature-Based Method*: The signature-based method detects DNS tunnels through the matching of specific signatures. These signatures are typically derived from professionals analyzing and extracting static features from traffic packets. For instance, the traffic packet of dnscat2 [23] will contain the content "dnscat".

Sheridan and Keane [4] employ snort with Iodine feature rules to detect anomalous background traffic generated by Iodine through traffic signature analysis and baseline comparison. However, this method struggles to detect more complex covert channels.

Similarly, Adiwal et al. [5] propose a DNS intrusion detection system (DID) based on snort. This approach extracts corresponding features and generates signatures for ids rules by simulating DNS tunnel attacks, DNS amplification attacks, and DNS DoS attacks. Nevertheless, it is confined to known attack types and cannot effectively cope with new types of DNS attacks.

Salat et al. [6] propose a method for detecting DNS tunnel attacks in cloud environments based on elastic stack. This approach analyzes DNS traffic data in the cloud environment and formulates rules for detection using suricata. However, it tends to increase the false positive rate during cloud migration.

Ghosh et al. [7] propose a multi-stage DNS tunnel detection technique. In the second stage, the method detects SSH handshakes in DNS tunnel traffic, analyzes data streams to retrieve base64 encoded SSH signatures, and formulates rules for matching detection. However, the signatures in this method can only match known DNS tunnel traffic in specific scenarios.

Signature-based methodologies swiftly discern suspected tunnel traffic by matching distinct content within packets, providing effective detection of known DNS tunnels. However, they are contingent on signature rule sets generated for specific tunnel software, making them prone to false negatives when facing customized or modified tunneling tools. Furthermore, adversaries can deliberately alter or obfuscate signatures within packets, rendering the signature-based approach ineffective.

② *Threshold-Based Approach*: This approach detects DNS tunnels by scrutinizing specific features, such as the quantity of distinct hostnames and the cache hit rate. It hinges upon the comparative analysis of threshold values associated with these features, facilitating the differentiation between benign and tunneling DNS traffic.

Ozery et al. [8] propose an information-based real-time detection method called ibHH. It is deployed on DNS servers, capturing timestamps and domain names transmitted to registered domains to calculate information volume. When the volume exceeds a set threshold, it flags the domain as malicious. However, the method is still susceptible to false positives due to the impact of wildcard DNS resolution.

Sani and Setiawan [9] investigate a method for detecting DNS tunnels using elasticsearch. The approach leverages watcher to assess the diversity of hostnames, and flags the traffic as a tunnel if it exceeds 300. However, the method adopts an empirically derived threshold, which necessitates adaptation to different environments.

Ellens et al. [10] detect tunnels by analyzing traffic features, such as the byte count of each flow, and applying statistical detection methods. They set corresponding thresholds for different features, but the false positive rate is high.

Paxson et al. [11] devise a principled method. This method employs a configurable threshold to constrain the information transfer through DNS. The core concept involves utilizing lossless compression for estimating the information entropy of the entire DNS query flow to obtain an upper bound on the information quantity. However, the approach is susceptible to traffic splitting across multiple domains by the attacker.

Ishikura et al. [12] propose a method based on DNS cache properties. This approach leverages the characteristics of cache

hits or misses generated on cache servers, introducing features such as cache hit rate, access hit rate, and access miss count. It generates rule filters and LSTM filters for tunnel detection.

The threshold-based detection method of DNS tunnels offers the advantage of adjustable sensitivity by site-specific security policies, thereby providing configurable tunnel traffic detection. However, this method faces challenges in dealing with low-traffic tunnels, which can evade detection by maintaining traffic below the threshold. Furthermore, experiential judgment is necessary for setting the threshold. An excessively high threshold may lead to under-detection, while an overly low threshold may increase the false positive rate. The threshold method is also susceptible to evasion tactics employed by attackers, such as distributing traffic across multiple domains.

B. Model-Based Detection

Model-based detection involves learning crucial features extracted from extensive network data packets, including packet size, TTL (Time to Live), DNS query type, and DNS request domain name length. These features are then utilized in training machine learning or deep learning algorithms to construct robust detection models.

Ibraheemi et al. [13] propose a method of hybrid genetic algorithm and support vector machine. This method simulates the utilization of four protocols such as HTTPS and FTP. By capturing traffic during operation and applying a genetic algorithm for feature selection, it discerns the optimal subset of features and amalgamates it with an SVM classifier for detection.

Sakarkar et al. [14] propose a method grounded in natural language processing. They utilize wireshark to capture malicious network packets, extract features such as timestamps and message information, fit them through word embeddings, and employ LSTM and GRU algorithms for detection.

Lal et al. [15] propose a hybrid deep learning architecture named DNS-Tunnet. This approach transforms DNS queries into raw text, utilizing a CNN for automatic feature extraction. The extracted features are subsequently input into an SVM classifier for binary classification.

D'Angelo et al. [16] extract features from the DNS query payload, organize them into a 6×4 matrix to form a two-dimensional representation, and subsequently employ the CNN algorithm for binary classification.

Bai et al. [17] propose a method for identifying application behavior in DNS tunnels based on spatiotemporal information. They simulate different user application behaviors to capture DNS traffic, divide each DNS traffic into equal-length segments, and extract packet length and timing characteristics from them. Features are selected through the information gain rate index and then applied to three machine learning algorithms: bayes net, decision tree and random forest for classification.

Altuncu et al. [18] harness the Alexa top million websites and tools like Iodine for data collection. They developed a deep feed-forward neural network model for classification and conducted real-time testing in a network environment, achieving better real-time detection performance compared to the study by [24].

Shafieian and Zulkernine [19] launch various attacks on enterprise networks within the AWS cloud to capture traffic, and then utilize feature engineering and integrated learning methods to detect low-feature network intrusions.

Liang et al. [20] apply the "leave-one-out" method to generate multiple datasets. They design a FECC model that integrates CNN and k-means clustering, employing sliding windows to distill implicit features from the original DNS payload. Furthermore, they utilize k-means clustering to assess the homogeneity and exclusivity of the features, which are then applied in classification tasks and the detection of samples from unknown classes.

Wang et al. [21] propose KRTunnel, a pioneering method for capturing DNS tunnel traffic from the Android side for detection. This method employs a User-Agent check within traffic packets to filter out Android traffic. Subsequently, it extracts features such as subdomain average entropy and TTL from DNS requests and responses, using the isolation forest algorithm for binary classification.

In summary, model-based methods train on extracted features from datasets, showing great potential in DNS tunnel detection. They can effectively resist attackers' attempts to evade detection by modifying signatures and maintain scalability in complex network topologies. However, current model-based research primarily focuses on surface-level features of domain names or packet payloads, lacking attention to the behavioral structure characteristics of DNS tunnel establishment and attack processes. This often results in high false positive rates when dealing with unknown DNS tunnels and wildcard DNS resolution scenarios. Additionally, there is notable overlap in certain features such as domain length, information entropy, and packet size, making it challenging to identify the specific DNS tunneling tools being used.

Therefore, we propose GraphTunnel, a model-based approach that constructs DNS recursive resolution graphs. This method accurately simulates the DNS resolution process and captures the differing behavioral patterns between normal DNS resolution and DNS tunnel resolution. This enhances the differentiation in feature space, ensuring our method remains robust even when faced with unknown DNS tunnels and wildcard DNS resolution scenarios. Moreover, GraphTunnel effectively utilizes statistical information and the capabilities of CNNs, gradually highlighting the differences in node features within the graph, achieving high accuracy in distinguishing between various DNS tunneling tools.

III. Methodology

To address the issues mentioned in Section I, we propose GraphTunnel. This framework is engineered for the real-time detection of DNS tunnels and the identification of tunneling tools. As illustrated in Figure 1, it encompasses modules for traffic input, traffic parsing, DNS tunnel detection, and tunneling tool Identification. In the subsequent sections, we provide a detailed description of each module.

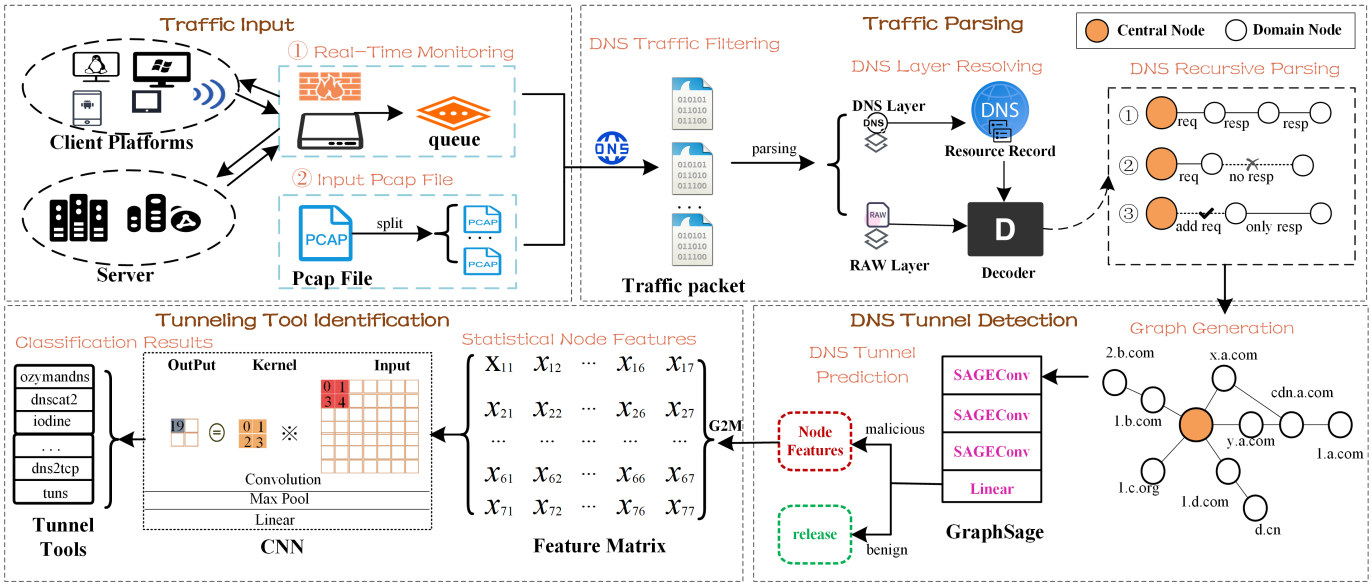

Fig. 1. The framework of GraphTunnel.

A. Traffic Input Module

In the traffic input module, GraphTunnel can accept two modes of input. The first mode, real-time monitoring, employs scapy [25] to oversee the specified network card interface and utilizes multi-threading for traffic processing. One thread is dedicated to traffic capture and storage in the message queue, while another retrieves packets from it for parsing. This multi-threaded design circumvents the issue of traffic capture speed being constrained by parsing and detection, thereby enhancing overall monitoring and detection efficiency. The second mode inputs traffic packets through pcap files. To expedite data analysis, GraphTunnel partitions pcap packets into smaller segments. Subsequently, multi-threading is employed to concurrently process these smaller pcap packets.

B. Traffic Parsing Module

In the traffic parsing module, GraphTunnel receive the packets transmitted from the traffic input module and filter out DNS traffic based on protocols and port numbers. Subsequently, we uniformly parse DNS traffic from different platforms to obtain data at the DNS layer or RAW layer. We then analyze DNS recursive resolution on this data, generating corresponding recursive resolution paths.

1) DNS Traffic Filtering: We classify network packets as DNS traffic based on an analysis of the UDP protocol and port numbers. Initially, we ascertain the presence of a UDP layer within the packet. Subsequently, header information is extracted from this layer, and an examination is conducted on both the source and destination ports. If either port is identified as 53, the packet is categorized as DNS traffic. Otherwise, it is classified as non-DNS traffic.

2) DNS Layer Resolving: Upon obtaining DNS traffic, further packet analysis is necessary to extract the DNS layer information. However, these DNS flows originate from different platforms, and due to variances in the kernel and protocol stack implementations, certain encapsulation formats such as the cooked format may vary across platforms. For instance, traffic from Android devices may exhibit the Linux cooked capture v2 identifier, while Linux traffic utilizes Linux cooked capture v1, and Windows traffic is presented as Ethernet. This discrepancy may arise from specifying different values with the -i parameter during tcpdump traffic capture, directly impacting the format of captured data and the included information. Therefore, We utilize scapy version 2.5.0 [25] to address cross-platform compatibility issues, ensuring uniformity in parsing results for traffic captured across different platforms, such as Windows, Linux, and Android. Concurrently, when dealing with DNS layer data, it is necessary to consider the varying impacts of different resource records on the parsing process, which may otherwise lead to parsing errors and the inability to retrieve information corresponding to the intended layer. Moreover, due to disparate implementation approaches among various tunneling tools, certain response traffic may lack DNS layer headers. In such instances, GraphTunnel directly extracts and analyzes the raw data from the final layer of the packet.

3) DNS Recursive Parsing: After extracting information from the DNS layer, we conduct a recursive parsing analysis of this data to generate corresponding recursive parsing paths. To preserve the integrity of the DNS resolution process, we consider both DNS request and response traffic, doing a one-to-one mapping based on the request domain. However, mismatches may occur during DNS domain resolution, where it is not always possible to establish a one-to-one correspondence between requests and responses. Specifically, if only request data is presented, and subsequent response traffic is absent, GraphTunnel parses the request traffic, treating it as an isolated node. Conversely, if only response traffic is

TABLE II
USER COVERAGE SCORES FOR DIFFERENT QUERY TYPES

Type	Score	Type	Score
A	99.948	NS	0.906
AAAA	82.082	TKEY	0.026
PTR	50.078	NAPTR	0.363
SRV	69.653	SPF	0.026
MX	1.165	CNAME	0.233
ANY	39.513	AXFR	0.026
SOA	9.089	NULL	0.026
TXT	17.607	Others	0

presented, and preceding request traffic is missing, we observe that the response traffic data will contain the content of the corresponding request. GraphTunnel prioritizes extracting and parsing the request part before parsing the response part. In some instances of real-world traffic, we have encountered peculiar cases where the status code is marked as a response packet, yet the data only encompasses the request portion without any response data. GraphTunnel can still handle this normally according to the aforementioned rules.

4) Feature Extraction: We extract eight features from the traffic data packets, which effectively capture the distinctions between benign DNS traffic and DNS tunnel traffic.

a) Record type corresponding scores: Herrmann et al. [26] conducted a classification and statistical analysis of massive DNS logs, generating datasets of different query types, including the query volume, domain number, and other indicators for each type. The findings reveal that the user coverage rates of A and AAAA type queries are the highest, reaching 99.948% and 82.082%. This indicates that the majority of users employ these two query types, while other types commonly seen in DNS tunnels, such as TXT and NULL, are used by very few users. Consequently, we incorporate this information as a feature by assigning a score to the user coverage for each record type. Record types that are not encompassed in the paper's results suggest an absence of user queries, thus we assign their corresponding scores as zero. Detailed data is presented in Table II.

b) Length of subdomain: The stipulated maximum length for a fully qualified domain name (FQDN) is confined to 255 bytes, as delineated in the DNS protocol RFC 1035 [27]. The second-level domain name, a relatively immutable component of the domain name structure, serves as the principal identifier and brand of the domain name. Conventional DNS queries are primarily utilized for website access or server resolution, hence the subdomain name length is generally not extensive. However, to obscure transmission, DNS tunnels frequently construct elongated subdomain names subsequent to the second-level domain name to accommodate an increased data volume.

$$F_2 = |D_{\text{sub}}| = |D_{\text{FQDN}} - D_{\text{second}}|, \quad 0 < |D_{\text{FQDN}}| \leq 255$$

where D_{FQDN} represents the fully qualified domain name, D_{second} represents the second-level domain, and D_{sub} represents the subdomain, which is the part before the second-level domain.

c) Maximum length of subdomain: The DNS protocol stipulates that the length of each subdomain should not exceed 63 bytes. DNS tunnels, aiming to transmit concealed data, often employ a multi-level subdomain structure, with each level having a considerable length, yet still confined within the 63-byte limit. In contrast, conventional DNS queries, devoid of the need for data concealment, exhibit a relatively straightforward and shallow hierarchy in their subdomain structure, with shorter lengths at each level. Consequently, for a given DNS traffic, we can statistically analyze the depth of the domain name layers, extract the length of each subdomain, and determine the maximum length among them, thereby effectively distinguishing between DNS tunnel and benign traffic.

$$F_3 = |D_{\max}| = \max_{i=1}^{n} |D_{\text{sub}_i}|, \quad 0 < |D_{\text{sub}_i}| \leq 63$$

where $D_{\max}$ represents the maximum length domain name in subdomains, and D_{sub_i} represents the i-th layer of the subdomain.

d) Count of prolonged consecutive consonant strings: Subdomains within DNS tunnels often comprise a plethora of random letter combinations to maximize the transmission of information, resulting in an abundance of elongated consonant strings. In contrast, conventional DNS subdomains typically employ meaningful word groups, making the generation of excessive elongated consonant strings unlikely. Experimental results indicate that in the longest subdomain, noticeable differences in features emerge when the count of consecutive consonant characters exceeds 4. For benign subdomain, the count of continuous consonant strings with lengths greater than 4 is typically limited to the range of 0-1, whereas DNS tunnel subdomain exhibits multiple instances of such strings. Hence, the tally of continuous consonant strings in the longest subdomain can be an effective feature for distinguishing DNS tunnels from benign traffic. It is important to note that the hyphen "-" is a frequently occurring character in domain names and should be considered a delimiter during enumeration rather than being completely disregarded.

$$F_4 = \sum_{i=1}^{m} I\left(|C_i| > 4\right), \quad C_i \in \{D_{\max}\}$$

where C_i represents the i-th consecutive consonant string in the subdomain, m denotes the number of consecutive consonant strings, and I stands for the indicator function, determining whether the condition within the brackets is true.

e) Entropy of the longest subdomain: Information entropy can quantify the randomness and uncertainty of a string. A higher entropy indicates stronger randomness and greater uncertainty, while a lower entropy suggests weaker randomness and more regularity. In DNS tunnels, encoding techniques are often employed to conceal transmitted data, resulting in more random letter combinations and, consequently, higher entropy values for subdomains. Conversely, conventional DNS subdomains employ semantically coherent word groups, culminating in a comparatively diminished information entropy. Therefore, the information entropy of the longest subdomain can be harnessed as a distinguishing feature

between DNS tunnel and benign traffic.

$$F_5 = -\sum_{i=1}^{n}(p(x_i)logp(x_i)).$$

Here, $p(x_i)$ represents the frequency of x_i appearing in the entire string.

f) Time-to-live (TTL) value: The TTL value in DNS query results indicates the lifespan of the corresponding resolution record, representing the maximum time the resolution result can be cached by DNS servers. Under regular circumstances, TTL values are typically set to longer durations to minimize the necessity for repeated queries to nameservers. However, in DNS tunnels, TTL values are deliberately set to be short to ensure the rapid expiration of cached resolution records, facilitating the timely acquisition of the latest results for real-time data transmission. In essence, the TTL value distribution in tunnel traffic tends to be concentrated within a shorter time frame, whereas the TTL values in benign traffic are relatively longer and exhibit a broader distribution. Consequently, by statistically analyzing and comparing the TTL values in the returned packets of DNS traffic, we can identify an effective characteristic for detecting DNS tunnels.

g) Packet size in bytes: DNS tunnels necessitate the concealment of supplementary data within DNS packets, resulting in considerably larger sizes for both corresponding request and response messages compared to benign DNS traffic. In contrast, conventional DNS queries merely require the inclusion of fundamental query information, resulting in a byte size distribution that falls within a relatively confined range. Therefore, the byte size of resolved DNS data packets can be statistically analyzed and utilized as a distinguishing feature between tunneling traffic and benign traffic.

h) Average response time: For each DNS request, we can document the timestamp at which the request message is dispatched, as well as the reception time of the corresponding response message. The difference between these two values represents the total response time for the query. Conventional recursive DNS resolution necessitates interaction with multiple authoritative domain name servers, with each query requiring a certain amount of time, thereby influencing the overall response time. In contrast, DNS tunneling, designed for real-time data transmission, necessitates prompt responses. Typically, attackers employ DNS tunneling tools on the server side to facilitate DNS responses, resulting in shorter total query response times. By dividing the total response time by the number of recursive layers traversed by the query, we can obtain the average response time per layer. The average response time for tunneling traffic is noticeably shorter than that for regular recursive queries. Therefore, this can serve as a characteristic feature of the edges in the query path.

$$F8 = \frac{T_{response} - T_{request}}{L_n}$$

Here, T_{response} represents the response time, T_{request} represents the request time, and L_n represents the number of recursive layers traversed during the query.

C. DNS Tunnel Detection Module

In this module, we map the parsing path through the central node into a graph form, and apply GraphSage to aggregate neighbor node features to detect DNS tunnels.

1) Graph Generation: To promptly detect DNS tunnel traffic, it is imperative to minimize the volume of traffic processed during each detection cycle, while maintaining a high degree of detection accuracy. Consequently, we have empirically set the graph size (K) to a small value of 20, determining the number of recursive query paths to be mapped and constructed. We establish a graph structure by connecting individual query paths through a central node, with each path representing a complete DNS recursive resolution, and each node denoting a unique domain. Commencing from the initially queried domain, whenever a recursive query is forwarded to a new authoritative domain server, we ascertain whether this server node preexists in the graph. If it does, we establish a direct connection. Otherwise, we instantiate a new node to symbolize the server and subsequently connect. Each node encapsulates seven attributes, including information entropy and TTL values, which are detailed in the section III-B. The edges within the path delineate the recursive relationship among domain name resolutions, with the edge attribute corresponding to the average response time of the domain name resolution process. Ultimately, we obtain a comprehensive recursive query path extending from the root node to the leaf node. We replicate the aforementioned procedure, mapping and constructing K independent recursive query paths into a relational graph enriched with features associated with nodes and edges.

2) DNS Tunnel Prediction: GraphSAGE [22], as an effective algorithm for aggregating neighbor features and compatible with various types of graphs, is particularly suitable for DNS traffic analysis applied to the construction of recursive query relationship graphs. Specifically, we first set the recursive depth n for the central node to aggregate neighbor features. Then, we aggregate the features of neighbor nodes from the 1st to the nth order for the central node in stages. After mapping through multiple graph convolution layers and activation functions, we obtain the fused expression of node features at different orders. Consequently, the central node aggregates and encodes the topological structure information of the entire DNS query relationship graph and the feature information of each node. Finally, we use the comprehensive feature vector of the central node as the embedded expression of the entire graph, and input it into the linear layer for computation, thereby achieving binary classification of traffic.

D. Tunneling Tool Identification Module

In the tunneling tool identification module, we propose a G2M algorithm as depicted in Algorithm 1.

Upon detecting DNS tunnels, we apply statistical methods to analyze each category of features in the incoming graph embedding vector, extracting seven statistical attributes: variance, mean, standard deviation, range, median, skewness, and kurtosis. These features belong to the categories of central tendency, dispersion, and distribution shape, and are commonly used in statistical analysis across various fields [28],

Algorithm 1 G2M Algorithm

Input: Graph G
Output: Matrix M

$V \leftarrow \text{GraphEmbedding}(G)$
$M \leftarrow \text{InitializeEmptyMatrix}(7 \times 7)$
for each feature F in V **do**
  $M[i] \leftarrow [\text{var}(F), \text{mean}(F), \text{std}(F), \text{range}(F),$
    $\text{median}(F), \text{skewness}(F), \text{kurtosis}(F)]$
end for
return M

[29], [30]. They are relatively simple and efficient to compute, sufficiently describing the statistical characteristics of the data while avoiding computational complexity. This process results in a 7×7 two-dimensional matrix. Then we arrange this matrix to form a grayscale image and input it into a convolutional neural network. The convolutional layers in the network capture correlations between statistical features, while the pooling layers aggregate feature information, progressively abstracting and compressing statistical features. Through this methodology, the differences in node features within the graph are amplified, enabling accurate identification of various DNS tunneling tools.

IV. Evaluation

We have conducted an evaluation of GraphTunnel's detection performance and generalization capabilities. This section presents the results of these experiments.

A. Experiment Settings

Environmental Setup: The traffic data is collected on four distinct platforms: Windows 11 AMD64, Kali Linux x86_64, Centos7 $\times$ 86_64, and Thunder Simulator 9. The evaluation of GraphTunnel is conducted on a 16-node GPU cluster. Each node in this cluster is equipped with an Intel (R) Core (TM) i9-10920X CPU operating at 3.50 GHz, 256GB of RAM, and two NVIDIA RTX 3080 GPUs. The system runs on Ubuntu 20.04 LTS with Linux kernel v.5.4.0. We deploy GraphTunnel in Python 3.10.

B. Datasets Description

Existing model-based detection methodologies predominantly derive their datasets from two categories: public accessible datasets and self-collected datasets. These datasets, however, have limits on the diversity, magnitude, and accessibility of DNS tunnel traffic.

Studies such as [8], [13], and [16] use public available datasets, encompassing both benign DNS traffic and DNS tunnel traffic. Despite this, these datasets incorporate a limited variety of DNS tunneling tools, typically including just three to five. Furthermore, Ozery et al. [8] used the publicly available ZIZA dataset [35], which only provides csv files containing information such as user_ip, domain, timestamp, and entropy, without including the original pcap files. This limitation prevents the application of other methods that require extracting more contextual information from raw traffic.

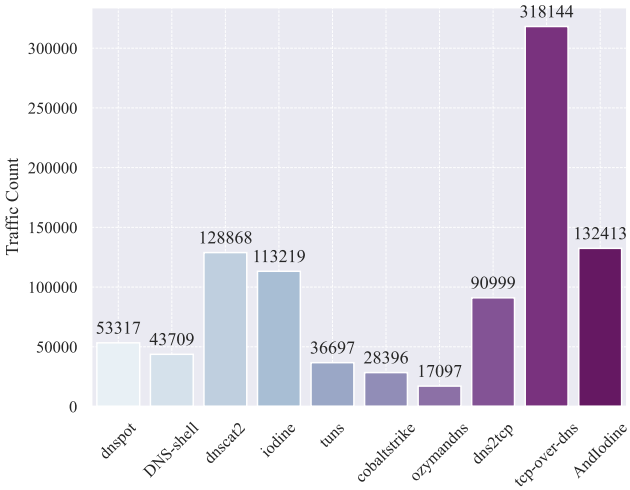

Fig. 2. Traffic data collected by various tunneling tools.

Research such as [14], [17], [19], and [20] independently gather benign DNS traffic and DNS tunnel traffic by emulating real-world attack-defense environments. Nevertheless, the spectrum of DNS tunneling tools incorporated in these studies is restricted to merely three to five types, and a substantial fraction of the datasets remains publicly inaccessible.

To assess the overall performance of GraphTunnel, we conduct experiments using four distinct datasets. The first, denoted as $Dataset_{our}$, comprises a substantial amount of benign traffic alongside DNS tunneling traffic generated by ten different DNS tunneling tools. The second, labeled as $Dataset_{wildcard}$, encompasses traffic data associated with wildcard domains. The third dataset, referred to as $Dataset_{CIC}$, is sourced from publicly available data [36]. The final dataset, denoted as $Dataset_{korving}$, is created by [37].

1) $Dataset_{our}$: We obtained the top 1,000,000 domain names from the Cloudfare [38] website. Utilizing a distributed approach, we invoked browser instances to simulate genuine user interactions with the assigned sublists of domain names. Concurrently, wireshark is employed to monitor and collect the generated standard DNS traffic data. In addition, we replicated a real intranet environment and utilized ten DNS tunneling tools, including iodine [39], dnscat2 [23] and dns2tcp [40], to establish DNS tunnels between the local intranet and public servers. Subsequently, we masqueraded as attackers and operated various intranet information collection tools, such as Ladon [41], linEnum [42], and gather [43], on different operating systems to acquire a variety of sensitive information from the target system. This process allowed us to capture the DNS tunnel traffic during the interaction. Detailed tunneling traffic data is illustrated in Figure 2. It's worth noting that due to the varying data lengths transmitted by different tunneling tools, the volume of traffic collected also varied.

To evaluate the generalization ability of GraphTunnel in detecting unknown DNS tunnel traffic, we partition the collected data. Table III comprises 2,012,494 instances of benign DNS traffic and 375,810 instances of DNS tunneling traffic generated by five DNS tunneling tools such as Iodine [39] and dnscat2 [23]. This dataset is utilized for training the detection

TABLE III
TRAFFIC DATA FOR DIFFERENT DNS RECORD TYPES BY VARIOUS TUNNELING TOOLS

Traffic Type	Traffic Count	Total
normal	2012494	2012494
dnspot [31]	53317	53317
DNS-shell [32]	43709	43709
dnscat2-CNAME	39547	128868
dnscat2-MX	51371	
dnscat2-TXT	37950	
iodine-A	23749	113219
iodine-CNAME	17449	
iodine-MX	17527	
iodine-NULL	12557	
iodine-PRIVATE	12544	
iodine-SRV	16814	
iodine-TXT	12579	
tuns [33]	36697	36697

TABLE IV
TRAFFIC DATA FOR DIFFERENT DNS RECORD TYPES BY UNKNOWN TUNNELING TOOLS

Traffic Type	Traffic Count	Total
cobaltstrike	28396	28396
ozymandns	17097	17097
dns2tcp-KEY	66012	90999
dns2tcp-TXT	24987	
tcp-over-dns-TXT [34]	112233	318144
tcp-over-dns-CNAME	205911	
AndIodine-CNAME	37625	132413
AndIodine-MX	30273	
AndIodine-NULL	14652	
AndIodine-SRV	29508	
AndIodine-TXT	20355	

model. Table IV contains an additional 587,049 instances of traffic generated by five other DNS tunneling tools, employed to evaluate the model's performance on unknown samples. It is important to note that in the real world, the balance between regular DNS resolution traffic and DNS tunnel traffic is skewed, with DNS tunnel traffic constituting only a small portion. Consequently, our dataset does not adhere to a one-to-one balanced ratio. In an effort to facilitate further research in this domain, we are making our dataset publicly available on https://github.com/ggyggy666/DNS-Tunnel-Datasets.

2) $Dataset_{wildcard}$: We collected the Top 1000 domains from Cloudflare [38] and adjusted the relevant scripts of the subdomain enumeration tool OneForAll [44] to verify whether these domains have adopted wildcard resolution. Figure 3a depicts the distribution of domains that enable wildcard resolution among the top 1,000 domains. Specifically, we successfully detected 913 accessible live domains within the Top 1000 domains. Subsequently, utilizing OneForAll, we obtained subdomains for each live domain and conducted a check for enabled wildcard resolution on these subdomains, revealing 31,305 subdomains supporting wildcard resolution. Figure 3b presents a word cloud generated from a subset of domains with enabled wildcard resolution, including well-known international corporations like google.com, amazon.com, and facebook.com, highlighting the widespread application of wildcard domains in the network ecosystem.

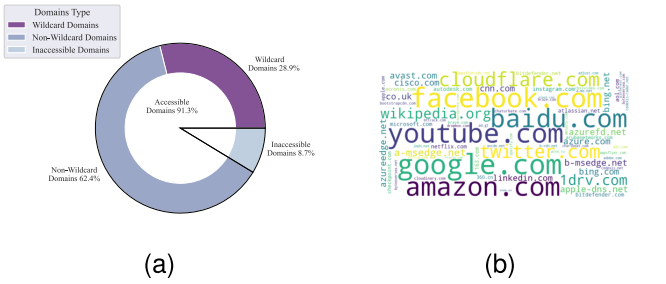

Fig. 3. The proportion of Accessible, Inaccessible, Wildcard and Non-Wildcard Domains on Cloudfare TOP 1000.

Finally, to simulate the pattern of wildcard resolution, we employed a character variable comprising numerals, letters, and permissible domain characters such as "−". We iterated through the input domain name list, randomly selecting strings of lengths ranging from 1 to 64 for each domain, and appended these to the original domain name to generate subdomains. Subsequently, we simulate browser access to capture the DNS resolution and response traffic generated during the access process, totaling 639,208 instances.

3) $Dataset_{CIC}$: $Dataset_{CIC}$ is a publicly available security dataset known as CIC-Bell-DNS-EXF-2021 [36]. This dataset encompasses 270.8 MB of DNS traffic, comprising various file types such as audio, compressed files, exe, images, text, and video. To simulate real attack scenarios, researchers conduct a five-day experiment involving both mild and severe attacks. Each day comprises a mixture of benign traffic and attack traffic generated by various types of file transfer attacks. In the case of severe attacks, the benign traffic to attack traffic ratio is 6:4, while in mild attacks, this ratio reaches 9:1. The final dataset encompasses 323,698 samples from severe attacks, 53,978 samples from mild attacks, and 641,642 distinct benign samples.

4) $Dataset_{korving}$: The dataset is created by Korving and Vaarandi [37]. They develop a configuration tool named DACA that executes end-to-end automated attack scenarios and extracts security datasets from the analyzed systems. Specifically, they deploy DNS servers implemented by different tools to respond to DNS requests, including BIND9, CoreDNS, Dnsmasq, and PowerDNS. They also simulate real scenarios and execute three DNS tunneling tools for command and control (C2) and file transfer, which include iodine, dns2tcp, and dnscat. Finally, they generate a total of 12,789 C2 traffic and 3,034,833 file transfer traffic.

C. Evaluation Metrics

In this study, we employ a variety of binary classification evaluation metrics to assess the performance of our DNS tunnel detection model.

$$Acc = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Recall = \frac{TP}{TP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$
$$F1 = \frac{2 * Precision * Recall}{Precision + Recall}$$

where TP (True Positives) represents the count of correctly labeled DNS tunnel traffic, TN (True Negatives) denotes the correct identification of benign DNS traffic. FN (False Negatives) signifies the misclassification of DNS tunnel traffic as benign, while FP (False Positives) indicates benign traffic incorrectly identified as DNS tunnel traffic.

D. Comparison Methods

To evaluate our approach, we select three categories of methods for comparison, including a set of baseline methods, multiple GNNs, and ensemble learning methods. These selected methods are either well-recognized benchmarks within the industry for multi-method comparisons or have demonstrated superior detection performance in recent years. Specifically, the methods considered are as follows:

1) Baseline Methods: D'Angelo et al. [16]: This method extracts 22 features from DNS query payloads, arranges them into 6×4 two-dimensional images through padding with two arbitrary constants, and utilizes the CNN algorithm for DNS traffic classification.

Mahdavifar et al. [45]: This method extracts a total of 30 stateful and stateless features, employing five machine learning algorithms such as GNB and RF for DNS tunnel detection.

Suman et al. [46]: This method categorizes DNS traffic features into lexicon-based features, DNS statistics-based features, and third-party-based features. It applies five machine learning algorithms such as SVM and KNN to train the DNS tunnel detection model.

Filippo et al. [47]: This method proposes a prototype of a protocol tunnel detector that combines machine learning and deep learning. It identifies anomalous connections deviating from the typically established network connections to detect DNS tunnels.

2) Multiple GNNs: We integrate our approach with multiple Graph Neural Networks (GNNs) for detection and classification, including GraphSAGE [22], GCN [48], GAT [49], and GIN [50]. These GNNs are implemented using PyTorch Geometric (PyG).

3) Ensemble Learning Methods: Chowdhary et al. [51]: This method employs query length and entropy as two primary features and integrates Gaussian Naive Bayes, Random Forest, Decision Tree, and K Nearest Neighbours algorithms to detect DNS tunnels.

E. Evaluation Results

To evaluate the effectiveness of GraphTunnel, we systematically address the following questions and design corresponding experiments for validation.

RQ1: How well does GraphTunnel perform in detecting DNS tunnels?

To ascertain the detection capabilities of GraphTunnel, we conduct relevant experiments using the dataset presented in Table III. We partition the dataset in a 6:4 ratio, allocating 60% for training and the remaining 40% for testing. Through cross-validation experiments, we determine that a batch_size of 64 and a learning rate of 0.005 yielded optimal parameters. We select four comprehensive graph neural network algorithms for comparison, including GraphSage, GCN, GAT, and GIN. Ensuring equivalent computational resources, we apply each algorithm for training on the same dataset and performed predictions on the test set. The specific outcomes are illustrated in Table V.

TABLE V
COMPARATIVE RESULTS OF GNNS ON THE DATASET

GNN	Accuracy	Precision	Recall	F1 Score
GraphSage	1	1	1	1
GCN	1	1	0.9998	0.9999
GAT	1	1	1	1
GIN	1	1	1	1

The experimental results table reveals that all four GNNs exhibit robust performance on the given binary classification task. This exemplary performance underscores the effectiveness of our proposed detection method based on GNN.

GraphSage employs a random neighbor sampling technique, aggregating information from neighboring nodes to infer the label of each node. In the presence of imbalances within the dataset, GraphSage adeptly captures the characteristics of tunneling traffic samples, demonstrating superior feature extraction capabilities. GCN employs analogous neighborhood aggregation strategies, effectively utilizing information from adjacent nodes for node classification.

GAT introduces an attention mechanism, enabling the model to focus on nodes crucial for the classification task. This mechanism enhances the discriminative power of the model by directing attention to key nodes in the graph. GIN, recognized for its high flexibility as a graph neural network algorithm, exhibits stable performance robust to imbalanced data interference. It accurately discerns disparities between benign and tunneling traffic, thereby achieving outstanding overall performance.

RQ2: How does the generalization capability of GraphTunnel perform against traffic from unknown DNS tunneling tools?

In the real world, unknown DNS tunneling tools may operate on various operating systems, including Windows, Linux, and Android. Owing to the disparities in traffic collection methods across different operating systems, the traffic input into the DNS tunnel detection model may encounter parsing issues stemming from format variations, thereby impeding the model's detection efficacy.

GraphTunnel considers this and adapts to DNS tunnel traffic collected from different operating systems. Specifically, the DNS tunnel tool traffic used in Table IV comes from different operating system terminals. The Windows terminal collects traffic from the cobaltstrike [52] and tcp-over-dns [34], the Linux terminal collects traffic from the ozymandns [53] and dns2tcp [40], and the Android terminal collects traffic from the

TABLE VI
DETECTION OUTCOMES FOR UNKNOWN TUNNELING TOOLS

Tunnel tools	Accuracy	Precision	Recall	F1 Score
cobaltstrike	1	1	1	1
ozymandns	1	1	1	1
dns2tcp	1	1	1	1
tcp-over-dns	1	1	1	1
AndIodine	1	1	1	1

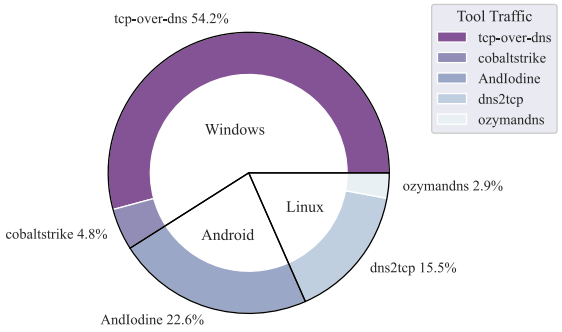

Fig. 4. The proportion of tool traffic on different operating systems.

AndIodine [54]. The distribution of these tools is presented in Figure 4.

To evaluate the detection performance of GraphTunnel against unknown DNS tunneling tools across platforms, we conduct the second experiment using the data from Table IV. Under the same experimental conditions as Q1, we employ the pre-trained model to predict the traffic from these tunneling tools, and the results are presented in Table VI.

The results demonstrate that GraphTunnel maintains a 100% detection accuracy even when facing with unknown tunneling tools. This is attributed to GraphTunnel's consideration of both request and response traffic, matching them one by one and incorporating authoritative domain name servers into the DNS resolution process, effectively constructing the spatial structure of the domain resolution process. Simultaneously, it captures the temporal cost of the DNS resolution process, ingeniously transforming DNS resolution into embeddings of spatiotemporal features.

By intelligently aggregating the features of neighboring nodes through graph neural network algorithms, GraphTunnel efficiently detects and dissects the complex graph structures and patterns within traffic data. This leads to an increasing disparity in the characteristics between benign DNS traffic and DNS tunneling traffic. Therefore, even for unknown DNS tunneling tools, GraphTunnel consistently achieves robust detection performance.

RQ3: Is GraphTunnel superior to baseline methods?

To evaluate the overall performance of GraphTunnel, we conduct experiments using $Dataset_{CIC}$ under the mentioned experimental setup and compare the results with baseline methods. Additionally, we employ GraphTunnel and the CNN model trained on $Dataset_{CIC}$ to predict the C2 and FileTransfer categories of the unknown dataset $Dataset_{korving}$. The detailed comparative outcomes are presented in Table VII.

The tabulated results demonstrate that in the DNS tunnel detection task, all baseline methods achieve F1 Scores exceeding 90%, with Mahdavifar et al. [45] notably reaching an exceptional 99.97%. This outstanding performance can be attributed to the adoption of the random forest algorithm, coupled with bootstrapping and feature random selection, effectively mitigating the risk of overfitting and yielding favorable outcomes on imbalanced datasets. Suman [46] apply various machine learning techniques such as SVM and KNN, achieving an F1 score of 98.9% through meticulous adjustment of hyperparameters and integration of advanced feature selection techniques. However, its robustness is comparatively lower, achieving an F1 score of only 57.82% when confronted with other machine learning algorithms like MLP.

Filippo et al. [47] combine unsupervised and supervised methods like SVM for binary classification, achieving a commendable F1 Score of 95.6%. However, this approach slightly lags compared to other methods, possibly due to data truncation during processing, leading to information loss and impacting overall performance. D'Angelo et al. [16] arrange features into two-dimensional images and apply CNN for training, achieving an F1 Score of 99.71% for known DNS tunnels. By automatically learning and extracting relevant features at different abstraction levels, CNN identifies complex patterns in DNS query payloads. However, as indicated by the results in Table VII, CNN struggles to accurately capture the behavioral patterns of entirely unfamiliar DNS tunneling traffic. This limitation leads to a higher rate of false negatives, resulting in a relatively low F1 Score of 57.14% and indicating poor robustness.

In contrast, GraphTunnel captures unique spatiotemporal features by retracing the DNS recursive resolution process. The aggregation of neighbor nodes enhances the spatial structural characteristics of DNS resolution obviously. Regardless of the intensity of the attack, GraphTunnel effectively distinguishes benign traffic from tunnel traffic, resulting in 100% detection accuracy. Even when faced with a completely unknown dataset, GraphTunnel maintains high robustness, achieving at least a 99.37% F1 Score.

RQ4: How effectively does GraphTunnel perform in detecting wildcard DNS resolution scenarios?

To evaluate the robustness of GraphTunnel in the context of wildcard DNS resolution scenarios, we incorporate $Dataset_{wildcard}$ as benign DNS traffic into the original model and conduct a retraining process while keeping other conditions unchanged. In addition, we apply Chowdhary et al. [51] to our dataset and perform the same operations for a more comprehensive comparative analysis with GraphTunnel.

Figure 5 illustrates the detection performance of three ensemble models in Chowdhary et al. [51] and GraphTunnel before incorporating wildcard domain traffic. The first two models display metrics exceeding 90%, whereas the third model exhibits a lower recall of merely 0.8951, albeit its accuracy and precision are the highest. This could be ascribed to the fewer integrated models, accentuating the detection

TABLE VII
PERFORMANCE OF COMPARISON EXPERIMENTS WITH BASELINE METHODS

Model	Dataset	Accuracy	Precision	F1 Score
Samaneh et al. [45]	$Dataset_{CIC}$	0.9997	0.9997	0.9997
Suman et al. [46]	$Dataset_{CIC}$	0.989	0.992	0.989
Filippo et al. [47]	$Dataset_{CIC}$	0.971	0.945	0.956
D'Angelo et al. [16]	$Dataset_{CIC}$	0.9977	0.9995	0.9971
	$Dataset_{korving}$ (C2)	0.3999	1.0	0.5714
	$Dataset_{korving}$ (fileTransfer)	0.4913	1.0	0.6589
GraphTunnel	$Dataset_{CIC}$	1.0	1.0	1.0
	$Dataset_{korving}$ (C2)	0.9876	1.0	0.9937
	$Dataset_{korving}$ (fileTransfer)	0.9994	1.0	0.9997

TABLE VIII
PERFORMANCE COMPARISON OF THE MODEL AFTER WILDCARD DNS RESOLUTION

Model	Algorithm	Accuracy	Precision	Recall	F1 Score
Aastha et al. [51]	DT+KNN+GNB	0.9992 ↑ 0.0153	0.9975 ↑ 0.0271	0.8000 ↓ 0.1261	0.8879 ↓ 0.0598
	RF+DT+SVM	0.9995 ↑ 0.0095	0.9977 ↑ 0.0448	0.8791 ↓ 0.1061	0.9347 ↓ 0.0341
	RF+DT	0.9995 ↑ 0.0001	0.9977 ↓ 0.0023	0.8801 ↓ 0.0150	0.9352 ↓ 0.0095
GraphTunnel	GraphSage	0.9997 ↓ 0.0003	0.9997 ↓ 0.0003	0.9959 ↓ 0.0041	0.9978 ↓ 0.0022

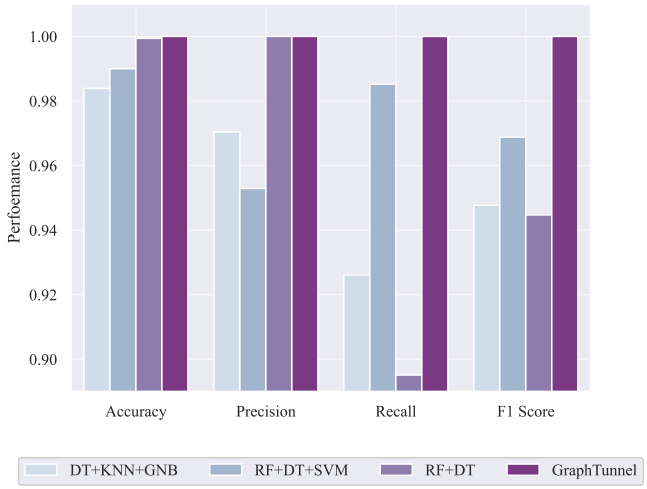

Fig. 5. Performance of comparison experiments with Chowdhary et al. [51].

characteristics of the two algorithms and predisposing them to yield high precision. Conversely, all metrics for GraphTunnel stand at 100%.

Table VIII showcases the detection outcomes of the new model on the validation set subsequent to the integration of wildcard domain traffic. By observing the results in the table, we notice an enhancement in precision and accuracy for the three ensemble models following the integration of additional regular wildcard domain traffic data. The reason lies in generating 1-64 character random subdomains according to RFC specifications. When the subdomain length is brief, features such as domain length and information entropy are indistinguishable from benign DNS domains. Moreover, by emulating how real users access domains via browsers, the collected traffic aligns with normal DNS traffic in terms of features like DNS resolution record types and timestamps. Consequently, for models that concentrate on extracting superficial domain or packet features, the majority of positive samples can be accurately classified as the positive class, thereby enhancing precision and accuracy. However, when the subdomain length is excessively elongated, the significance of features such as domain length and information entropy becomes pronounced, predisposing the model to misclassify them as malicious DNS tunneling behavior. This results in some positive samples being erroneously classified as negative, leading to a decline in recall, particularly in the first two models, with recall decreasing by 12.61% and 10.61% respectively.

It is noteworthy that subsequent to the integration of wildcard domain traffic data, the performance metrics of GraphTunnel exhibit a downward trajectory across various aspects. This suggests that wildcard domains exert a substantial impact on the model, which can readily induce bias in the model's detection results and give rise to false positives. However, in contrast to these three integrated models, GraphTunnel takes into account DNS recursive resolution, aggregates domain name node features during the resolution process, and procures unique spatiotemporal structure features. As a result, the influence of features such as subdomain length and information entropy on model detection is relatively minimal, and the recall rate of GraphTunnel has only declined by 0.41%. Furthermore, its F1 Score can still attain 99.78%, thereby demonstrating considerable robustness and reliability.

RQ5: How does GraphTunnel perform in recognizing tunneling tools?

To evaluate the capability of GraphTunnel in identifying tunneling tools, we conduct a multi-classification recognition experiment under the same experimental conditions as previously mentioned, involving ten tunneling tools such as Iodine [39] and dnscat2 [23]. In this experiment, we employ the metric of accuracy to assess the identification performance of GraphTunnel. Figure 6 presents the detailed recognition results for different types of tunneling tool traffic.

The graphical representation reveals the exemplary performance of the model in accurately classifying normal DNS traffic and various tunneling tools. The model demonstrates a near-perfect classification with minimal misclassifications. It exhibits an effective recognition of each category, with

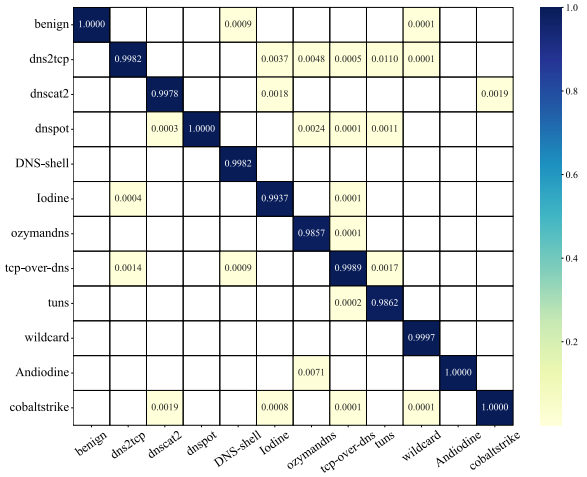

Fig. 6. Identification results for various tunneling tools.

the accuracy rate for each type exceeding 98%. Notably, the traffic generated by the tools dnspot [31], Andiodine [54], and cobaltstrike [52] is identified with an accuracy rate of 100%.

The performance is relatively less impressive for ozymandns [53], where 98.57% of the traffic can be successfully identified. There exists a 0.48% probability of being identified as dns2tcp [40], a 0.24% chance of being identified as dnspot [31], and a 0.71% likelihood of being identified as Andiodine [54]. This suggests that ozymandns is slightly similar to these tools in tunneling data through DNS.

In summary, GraphTunnel maintains high accuracy in identifying traffic generated by various tools. In the face of potential threats, it proves instrumental for analysts to swiftly locate malicious activities based on the identified results and take timely countermeasures.

RQ6: How well does GraphTunnel perform in terms of time and space efficiency?

To comprehensively evaluate the performance of GraphTunnel in real-time monitoring environments, we conduct detailed experiments from multiple dimensions, including time and space consumption. During the model training phase, we utilize the $Dataset_{our}$ to ensure that the model learns sufficient features and information. In the model prediction phase, we employ the $Dataset_{korving}$ to verify the model's generalization ability on unseen data. To better reflect real-world scenarios, we deploy GraphTunnel in a real network environment to monitor the traffic on network cards in real-time. Additionally, we use dnscat2 [23] to construct a real DNS tunnel and launch attacks through this tunnel to evaluate the real-time detection capability of GraphTunnel.

During the experiments, we measure several key performance indicators. We record the Training Time per Epoch (TTE) to evaluate the time consumption of each epoch of the model training process. Memory Usage (TMU) represents the resource requirements of GraphTunnel during training. Prediction Time (PT) is measured to assess the speed of the prediction process, while Prediction Memory Usage (PMU) provides insights into the memory requirements during inference. Attack Detection Time (ADT) measures the time taken from the initiation of an attack to its detection. Each evaluation experiment is repeated 5 times, and the average value is taken. We employ the CNN model [16] to repeat the aforementioned process for comparison. The specific experimental results are presented in the table IX.

TABLE IX
COMPARISON OF TIME AND SPACE CONSUMPTION BETWEEN CNN AND GRAPHTUNNEL

Model	TTE(s)	TMU(MB)	PT(s)	PMU(MB)	ADT(s)
CNN [16]	110.85	1566.94	12.39	1810.9	0.89
GraphTunnel	66.69	1638.42	48.65	126.44	4.46

The experimental findings reveal that GraphTunnel demonstrates higher time efficiency during training compared to the CNN model, with a TTE of 66.69 seconds versus CNN's 110.85 seconds. However, GraphTunnel's TMU of 1638.42 MB compared to CNN's 1566.94 MB is higher. This can be attributed to GraphTunnel's parsing of data into graph structures, which contain richer contextual information than CNN's vector data. This results in higher memory utilization but allows GraphTunnel to cover more data packets, reducing the number of iterations over the graph data in each epoch and thus achieving faster training.

In terms of prediction, GraphTunnel exhibits a longer PT of 48.65 seconds but lower PMU of 126.44 MB, contrasting with CNN's 12.39 seconds of prediction time and 1810.9 MB of memory usage. This aligns with expectations as CNN typically loads and processes the entire dataset at once during prediction, minimizing overhead from memory read/write operations. In contrast, GraphTunnel adopts a data segmentation strategy when handling large-scale datasets, initially dividing data packets into smaller parts before predicting each segment. This approach reduces memory usage but leads to increased processing time.

GraphTunnel requires 4.46 seconds for ADT, exceeding CNN's 0.89 seconds. This difference can be attributed to the CNN model's simpler process of feature extraction from individual data packets, which enables faster real-time detection. However, the CNN model's poor robustness in classifying unknown tunneling traffic poses challenges for accurate traffic classification, potentially increasing the workload for emergency response personnel. In contrast, GraphTunnel's approach of constructing a DNS recursive resolution graph requires more information from data packets for comprehensive traffic analysis and feature extraction. While this leads to slower real-time detection, it offers higher robustness, ensuring accurate detection in complex and dynamic network environments, providing a more reliable basis for emergency response. Therefore, we consider the time acceptable for real-time monitoring.

Indeed, there is potential to optimize our GraphTunnel to shorten traffic detection time and achieve faster response. For offline detection using the $Dataset_{wildcard}$, our GraphTunnel model requires 48 seconds, consuming only 126MB of memory. Consequently, we can feasibly introduce a multithreading mechanism, adopting a "space-for-time" strategy that enables

parallel traffic analysis and integration of processed results into the graph for model detection. For real-time detection, GraphTunnel necessitates gathering sufficient traffic context information to construct a DNS recursive resolution graph, ensuring high robustness, which results in longer processing times. Therefore, reducing the size of the graph is a viable solution, as it implies analyzing a reduced amount of traffic each time. As for determining the optimal graph size to balance robustness and detection efficiency, it needs to be fine-tuned according to the specific circumstances in actual enterprise applications. Additionally, GraphTunnel's performance is influenced by the packet sending frequency of the DNS tunnel tool and the stability of the network environment. We can mitigate the negative impact of these factors on ADT performance by optimizing the network rate processing mechanism, such as adjusting the data flow control strategy or enhancing the packet reception mechanism.

V. Discusion

Although GraphTunnel does not cover all unknown DNS tunnels, it still achieves 100% detection on the unkown samples, which consists of five completely unknown DNS tunneling tools. Moreover, as corroborated by the Q4 experiment, wildcard DNS resolution poses a considerable challenge to accurate detection, and GraphTunnel does not currently achieve 100% accuracy in this scenario. Nevertheless, it maintains an F1 score of 99.78%, surpassing current methods across various metrics. In the realm of application identification research, the challenge of multi-classification looms large. Despite GraphTunnel not achieving 100% identification for every DNS tunneling tool, it currently holds the leading position among existing studies. Our commitment to continuous improvement and optimization in future research endeavors aims to further enhance GraphTunnel's capabilities across all dimensions.

VI. Conclusion

In this study, we introduce GraphTunnel, a robust DNS tunnel detection framework through the DNS recursive resolution graph. This framework is dedicated to the real-time detection of DNS tunnels and has the capability to identify specific tunneling tools from DNS tunnel traffic. Initially, GraphTunnel filters DNS traffic from network traffic and constructs a graph structure representing the DNS recursive resolution process. Subsequently, it utilizes a GNN algorithm to aggregate features of nodes and edges in the graph for effective traffic classification. Moreover, GraphTunnel applies the G2M algorithm for statistical learning of node features in the graph and employs a CNN algorithm to generate intelligent identifiers for DNS tunneling tools. Experimental results demonstrate that GraphTunnel performs admirably in detecting DNS tunnels, with metrics surpassing those of existing baseline methods. GraphTunnel maintains high robustness when facing unknown DNS tunneling tools and generic domain name resolution scenarios. Additionally, it exhibits excellent performance in identifying DNS tunneling tools. In the future, we plan to propose techniques against GraphTunnel to generate adversarial traffic for bypassing, continuously enhancing GraphTunnel's ability to flexibly respond to unknown attack methods.

References

[1] J. Coker. (2021). *72% of Organizations Experienced a DNS Attack in the Past Year*. [Online]. Available: https://www.infosecurity-magazine.com/news/72-orgs-dns-attack-last-year/

[2] EfficientIP. (2022). *IDC 2022 Global DNS Threat Report*. [Online]. Available: https://efficientip.com/resources/idc-dns-threat-report-2022/

[3] Y. Wang, A. Zhou, S. Liao, R. Zheng, R. Hu, and L. Zhang, "A comprehensive survey on DNS tunnel detection," *Comput. Netw.*, vol. 197, Oct. 2021, Art. no. 108322.

[4] S. Sheridan and A. Keane, "Detection of DNS based covert channels," in *Proc. Eur. Conf. Cyber Warfare Secur.*, 2015, p. 267.

[5] S. Adiwal, B. Rajendran, and S. D. Sudarsan, "DNS intrusion detection (DID)—A SNORT-based solution to detect DNS amplification and DNS tunneling attacks," *Franklin Open*, vol. 2, Mar. 2023, Art. no. 100010.

[6] L. Salat, M. Davis, and N. Khan, "DNS tunnelling, exfiltration and detection over cloud environments," *Sensors*, vol. 23, no. 5, p. 2760, Mar. 2023.

[7] T. Ghosh, E. El-Sheikh, and W. Jammal, "A multi-stage detection technique for DNS-tunneled botnets," in *Proc. EPiC Ser. Comput.*, 2019, pp. 137–143.

[8] Y. Ozery, A. Nadler, and A. Shabtai, "Information based heavy hitters for real-time DNS data exfiltration detection," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2024, pp. 1–15.

[9] A. F. Sani and M. A. Setiawan, "DNS tunneling detection using elasticsearch," *IOP Conf. Ser., Mater. Sci. Eng.*, vol. 722, no. 1, 2020, Art. no. 012064.

[10] W. Ellens, P. Żuraniewski, A. Sperotto, H. Schotanus, M. Mandjes, and E. Meeuwissen, "Flow-based detection of DNS tunnels," in *Proc. 7th IFIP WG 6.6 Int. Conf. Auto. Infrastructure, Manage., Secur.*, Barcelona, Spain. Heidelberg, Germany: Springer, Jun. 2013, pp. 124–135. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-38998-6_16#rightslink

[11] V. Paxson et al., "Practical comprehensive bounds on surreptitious communication over DNS," in *Proc. 22nd USENIX Secur. Symp.*, 2013, pp. 17–32.

[12] N. Ishikura, D. Kondo, V. Vassiliades, I. Iordanov, and H. Tode, "DNS tunneling detection by cache-property-aware features," *IEEE Trans. Netw. Service Manage.*, vol. 18, no. 2, pp. 1203–1217, Jun. 2021.

[13] F. A. Al-Ibraheemi, S. Al-Ibraheemi, and H. Amintoosi, "A hybrid method of genetic algorithm and support vector machine for DNS tunneling detection," *Int. J. Electr. Comput. Eng. (IJECE)*, vol. 11, no. 2, p. 1666, Apr. 2021.

[14] G. Sakarkar, M. K. H. Kolekar, K. P. G. Patil, P. Dutta, R. Chaturvedi, and S. Kumar, "Advance approach for detection of DNS tunneling attack from network packets using deep learning algorithms," *Adv. Distrib. Comput. Artif. Intell. J.*, vol. 10, no. 3, pp. 241–266, 2021. [Online]. Available: https://revistas.usal.es/cinco/index.php/2255-2863/issue/view/1322

[15] A. Lal, A. Prasad, A. Kumar, and S. Kumar, "DNS-tunnet: A hybrid approach for DNS tunneling detection," in *Proc. 4th Int. Conf. Adv. Comput. Technol., Inf. Sci. Commun. (CTISC)*, Apr. 2022, pp. 1–6.

[16] G. D'Angelo, A. Castiglione, and F. Palmieri, "DNS tunnels detection via DNS-images," *Inf. Process. Manage.*, vol. 59, no. 3, May 2022, Art. no. 102930.

[17] H. Bai, W. Liu, G. Liu, Y. Dai, and S. Huang, "Application behavior identification in DNS tunnels based on spatial–temporal information," *IEEE Access*, vol. 9, pp. 80639–80653, 2021.

[18] M. A. Altuncu et al., "Deep learning based DNS tunneling detection and blocking system," *Adv. Electr. Comput. Eng.*, vol. 21, no. 3, pp. 39–48, 2021.

[19] S. Shafieian and M. Zulkernine, "Multi-layer stacking ensemble learners for low footprint network intrusion detection," *Complex Intell. Syst.*, vol. 9, no. 4, pp. 3787–3799, Aug. 2023.

[20] J. Liang, S. Wang, S. Zhao, and S. Chen, "FECC: DNS tunnel detection model based on CNN and clustering," *Comput. Secur.*, vol. 128, May 2023, Art. no. 103132.

[21] S. Wang, L. Sun, S. Qin, W. Li, and W. Liu, "KRTunnel: DNS channel detector for mobile devices," *Comput. Secur.*, vol. 120, Sep. 2022, Art. no. 102818.

[22] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–11.
[23] *Dnscat2*. Accessed: Sep. 23, 2023. [Online]. Available: https://github.com/iagox86/dnscat2
[24] J. Ahmed, H. H. Gharakheili, Q. Raza, C. Russell, and V. Sivaraman, "Real-time detection of DNS exfiltration and tunneling from enterprise networks," in *Proc. IFIP/IEEE Symp. Integr. Netw. Service Manage. (IM)*, Apr. 2019, pp. 649–653.
[25] *Scapy*. Accessed: Sep. 27, 2023. [Online]. Available: https://github.com/secdev/scapy
[26] D. Herrmann, C. Banse, and H. Federrath, "Behavior-based tracking: Exploiting characteristic patterns in DNS traffic," *Comput. Secur.*, vol. 39, pp. 17–33, Nov. 2013.
[27] P. Mockapetris, *Domain Names—Implementation and Specification*, document RFC 1035, 1987. [Online]. Available: https://www.zytrax.com/books/dns/apd/rfc1035.txt
[28] S. Uddin and H. Lu, "Dataset meta-level and statistical features affect machine learning performance," *Sci. Rep.*, vol. 14, no. 1, p. 1670, Jan. 2024.
[29] C. Guo, M. Lu, and J. Chen, "An evaluation of time series summary statistics as features for clinical prediction tasks," *BMC Med. Informat. Decis. Making*, vol. 20, no. 1, pp. 1–20, Dec. 2020.
[30] M. Altaf, T. Akram, M. A. Khan, M. Iqbal, M. M. I. Ch, and C.-H. Hsu, "A new statistical features based approach for bearing fault diagnosis using vibration signals," *Sensors*, vol. 22, no. 5, p. 2012, Mar. 2022.
[31] *Dnspot*. Accessed: Sep. 23, 2023. [Online]. Available: https://github.com/mosajjal/dnspot
[32] *DNS-Shell*. Accessed: Sep. 23, 2023. [Online]. Available: https://github.com/sensepost/DNS-Shell
[33] *Tuns*. Accessed: Sep. 23, 2023. [Online]. Available: https://members.loria.fr/LNussbaum/tuns.html
[34] *TCP-Over-DNS*. Accessed: Sep. 23, 2023. [Online]. Available: https://analogbit.com/software/tcp-over-dns/
[35] K. Žiža, P. Tadić, and P. Vuletić, "DNS exfiltration detection in the presence of adversarial attacks and modified exfiltrator behaviour," *Int. J. Inf. Secur.*, vol. 22, no. 6, pp. 1865–1880, Dec. 2023.
[36] (2021). *CIC-Bell-DNS-EXF-2021 Dataset*. Accessed: Oct. 20, 2023. [Online]. Available: https://www.unb.ca/cic/datasets/dns-exf-2021.html
[37] F. Korving and R. Vaarandi, "Daca: Automated attack scenarios and dataset generation," in *Proc. Int. Conf. Cyber Warfare Secur.*, 2023, vol. 18, no. 1, pp. 550–559.
[38] (2023). *Cloudflare*. Accessed: October 6, 2023. [Online]. Available: https://radar.cloudflare.com/domains
[39] *Iodine*. Accessed: Sep. 23, 2023. [Online]. Available: https://code.kryo.se/iodine/
[40] *DNS2TCP*. Accessed: Sep. 23, 2023. [Online]. Available: https://github.com/alex-sector/dns2tcp
[41] *Ladon*. Accessed: Oct. 6, 2023. [Online]. Available: https://github.com/k8gege/Ladon
[42] *LinEnum*. Accessed: Oct. 6, 2023. [Online]. Available: https://github.com/rebootuser/LinEnum
[43] (2023). *Gather*. Accessed: Oct. 6, 2023. [Online]. Available: https://github.com/wwl012345/gather
[44] *OneForAll*. Accessed: Oct. 13, 2023. [Online]. Available: https://github.com/shmilylty/OneForAll
[45] S. Mahdavifar et al., "Lightweight hybrid detection of data exfiltration using DNS based on machine learning," in *Proc. 11th Int. Conf. Commun. Netw. Secur.*, Dec. 2021, pp. 80–86.
[46] O. P. Suman, "A novel approach for malicious domain classification based on DNS traffic analysis and machine learning," Oct. 2023.
[47] F. Sobrero, B. Clavarezza, D. Ucci, and F. Bisio, "Towards a near-real-time protocol tunneling detector based on machine learning techniques," *J. Cybersecurity Privacy*, vol. 3, no. 4, pp. 794–807, Nov. 2023.
[48] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2016, *arXiv:1609.02907*.
[49] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," 2017, *arXiv:1710.10903*.
[50] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" 2018, *arXiv:1810.00826*.
[51] A. Chowdhary, M. Bhowmik, and B. Rudra, "DNS tunneling detection using machine learning and cache miss properties," in *Proc. 5th Int. Conf. Intell. Comput. Control Syst. (ICICCS)*, May 2021, pp. 1225–1229.
[52] *Cobalt Strike*. Accessed: Sep. 23, 2023. [Online]. Available: https://www.cobaltstrike.com/
[53] *OzymanDNS*. Accessed: Sep. 23, 2023. [Online]. Available: https://github.com/splitbrain/dnstunnel
[54] *AndIodine*. Accessed: Sep. 23, 2023. [Online]. Available: https://github.com/yvesf/andiodine