Striking Back At Cobalt: Using Network Traffic Metadata To Detect Cobalt Strike Masquerading Command and Control Channels

Clément Parssegny[1,2][0009−0004−2166−0881], Johan Mazel[1][0009−0002−0222−6794], Olivier Levillain[2][0000−0002−0558−5015], and Pierre Chifflier[1]

[1] ANSSI, France
[2] SAMOVAR, Télécom SudParis, Institut Polytechnique de Paris, 91120 Palaiseau, France

Abstract. Off-the-shelf software for *Command and Control* is often used by attackers and legitimate pentesters looking for discretion. Among other functionalities, these tools facilitate the customization of their network traffic so it can mimic popular websites, thereby increasing their secrecy. Cobalt Strike is one of the most famous solutions in this category, used by known advanced attacker groups such as "Mustang Panda" or "Nobelium".
In response to these threats, Security Operation Centers and other defense actors struggle to detect *Command and Control* traffic, which often use encryption protocols such as TLS. Network traffic metadata-based machine learning approaches have been proposed to detect encrypted malware communications or fingerprint websites over Tor network.
This paper presents a machine learning-based method to detect Cobalt Strike *Command and Control* activity based only on widely used network traffic metadata. The proposed method is, to the best of our knowledge, the first of its kind that is able to adapt the model it uses to the observed traffic to optimize its performance. This specificity permits our method to performs equally or better than the state of the art while using standard features. Our method is thus easier to use in a production environment and more explainable.

Keywords: Cobalt Strike, Command and Control, Detection, Machine Learning, Network Metadata

1 Introduction

Command and Control (C2) is used by attackers to control several compromised hosts, forming networks called *botnets*. A usual architecture for these botnets is the client/server botnet. In order to operate, this model of botnet uses communication channels where the server sends commands to infected hosts that will execute them and then transmit the results back. These channels have evolved over the years. First, they used IRC as their transport protocol. Then, HTTP, HTTPS and DNS became commonly employed. Recently, off-the-shelf commercial [48] or open source [25–27] software that were originally designed for Red team audits, have been used by malicious actors [17, 28, 63, 65].

Cobalt Strike [12] is the most popular example of these frameworks [55], used by both legitimate pentesters and real threat actors [16,37]. It can set up a full kill chain targeting Windows systems with several attackers collaborating simultaneously [39]. This situation drives organizations to do everything they can to neutralize identified Cobalt Strike-linked malicious infrastructure [67].

In Cobalt Strike, *Command and Control* communication channels can be customized to mimic benign services, based on different configuration profiles. Moreover, this off-the-shelf software supports TLS to encrypt its communications. Both masquerade and encryption strengthen Cobalt Strike's protection against detection. However, the metadata related to the size, the direction or the timings of the communication can still be leveraged for detection.

Botnet detection has been the goal of previous work through different ways. Many used an approach based on machine learning [4, 18, 24, 32, 36, 49, 52, 69]. However, as these Botnet *Command and Control* channels rely on various protocols such as IRC [23, 36], HTTP [32] or DNS [4, 18], approaches used for the detection vary. While some research monitored a precise network's activity, measuring anomalies as deviations from a computed profile [24, 30, 61], others preferred to inspect packet contents to search for specific characteristics of compromise [49]. As encryption use increased [22], network traffic metadata began to be used to identify

C2 traffic from benign traffic [52]. However, up to our knowledge, malware targeted in existing works did not have a masquerade capability similar to Cobalt Strike. A Cobalt Strike detection method [53] has been proposed. It is however limited regarding Cobalt Strike configurations and the training data is biased. More recently, a deep learning approach has been presented [70]. However, it does not take the masquerading capability of Cobalt Strike into account while using dubious feature construction steps. In this paper, we focus on the capability for Cobalt Strike to obfuscate and hide C2 traffic inside messages imitating benign services. We address the existing work shortcomings and provide a detailed performance evaluation based on configurations used in the wild.

Thus, our goal is to design a method to detect Cobalt Strike C2 traffic despite its masquerade and encryption capabilities. This method works with different configuration profiles and network protocols (namely HTTP, HTTPS and DNS). Since configurations are easily modified and deployed in Cobalt Strike, it is also central for our detection method to be easily extensible to seamlessly include new configurations. Our main contribution is a machine learning-based method to detect Cobalt Strike C2 traffic using only network traffic's metadata. One advantage of this approach is that it is not based on Deep Packet Inspection (DPI) at all but can still produce good performance, even with unsophisticated and standard features. Indeed, we observe mean F_1 scores between 0.78 and 1 with a confidence interval at a 95% threshold, equaling or slightly improving previous approaches. These features also produce good performance on clear and encrypted Cobalt Strike C2 traffic from real-world attacks.

Our paper is structured as follows. Section 2 describes existing work on C2 channel detection. Section 3 portrays the working principle of Cobalt Strike's *Command and Control* feature. Section 4 presents the threat model we consider. Section 5 depicts our method. Section 6 details our results. Finally, Section 7 and Section 8 discuss of our work and of our future work.

2 Related Work

C2 channel detection is inherently linked to botnet detection as it is the communication method for such networks. Decades ago, these botnets were based on IRC channels [1, 29, 30] on which attackers would send commands for victims to execute. Then, HTTP [40] and DNS [41] became widely used protocols for C2.

Several papers searched for botnet by profiling the normal activity of the studied network before measuring the distance of a sample in comparison with the benign one [24, 30, 61].

Another method used is DPI [7,68]. The principle is to inspect payloads to detect botnet messages [29,49] or to identify protocols [30] for correlation detection. However, this method is only possible when the studied traffic is not encrypted [1,23,32,36,49] which is hardly the case now as the use of TLS increases continuously for both benign [22] and C2 traffic. Although methods like entropy computing [18,74] were used to improve detection methods.

To accommodate the increasing use of encryption [22], papers that try to identify malicious traffic within encrypted communications have been published. They focus on machine learning techniques applied to TLS handshake such as `ClientHello`, `ServerHello` or `Certificate` messages [4, 5, 52, 69]. However, TLS 1.3, released in 2018, encrypts most of the handshake messages [19].

An overview of how machine learning has been applied for C2 detection is presented in Table 1. It is noticeable that a majority of the research formalized this problem as a binary classification problem but then differ in the protocols studied. Moreover, while other articles generally study one specific protocol in a known configuration to evaluate and compare different machine learning methods, our work concentrates on a unique method based on traffic metadata applied to a larger panel of configurations and protocols used in the wild. We thus argue that the masquerading capability of Cobalt Strike C2 traffic constitutes a unique challenge and we carefully design our proposed approach to address this challenge. Furthermore, few works detail the number of malicious configurations they are studying, which has a detrimental influence on reproducibility and the generalization capability of their method.

It is also important to note that traffic metadata have also been used in active methods to detect and fingerprint *Command and Control* servers and infected hosts. More specifically, TLS metadata present in the handshake messages were studied by academics [59, 60] and industrial actors whose tools [2, 3, 56] have been integrated to scanning platforms [8] and are routinely used in CTI reports [33]. Finally, industrial research focused on detecting Cobalt Strike and understanding its components to better fight against one of their client's main threat [31, 50, 64].

Table 1: Comparison of machine learning-based methods for C2 channels detection. Clas.: Classification type (B: Binary, M: Multiclass); Mw.: Malware; CS: Cobalt Strike; Feat. type: Feature Type (M: Metadata, D: Deep Packet Inspection); Reprod.: Reproducibility; Feat. imp. analysis: Feature Importance Analysis. Variability is the number of malicious classes used in the datasets e.g. the number of botnets or the number of malleable profiles.

Authors	**Ref.**	**Yr.**	**Clas.**	**Protocols**				**Feat. type**	**Target**		**Reprod.**		**Feat. imp. analysis**
				IRC	**HTTP**	**DNS**	**TLS**		**Type**	**Variability**	**Open code**	**Open data**	
Nivargi et al.	[49]	'06	B	√	-	-	-	M/D	Mw.	?	-	-	-
Livadas et al.	[36]	'06	B/M	√	-	-	-	M	Mw.	1	-	-	-
Kondo et al.	[32]	'07	B	√	√	-	-	M	Mw.	5	-	-	-
Dietrich et al.	[18]	'11	B	-	-	√	-	M	Mw.	11	-	-	-
Warmer	[69]	'11	B	-	-	-	√	M	Mw.	4	-	-	-
Garcia et al.	[24]	'14	B	√	√	-	-	M	Mw.	10	-	√	-
Anderson et al.	[4]	'16	B	-	√	√	√	M/D	Mw.	?	-	-	√
Anderson et al.	[5]	'20	M	-	-	-	√	M	Mw.	?	√	√	√
Pai et al.	[52]	'20	B	-	-	-	√	M	Mw.	?	-	-	-
Van der Eijk et al.	[20]	'20	B	-	√	-	√	M	CS	1	-	-	√
Ramos et al.	[53]	'22	B	-	√	-	√	M	CS	?	-	-	-
Ramos et al.	[54]	'23	B	-	-	-	√	M	CS	29	-	-	-
Yang et al.	[70]	'24	B	-	-	-	√	M/D	CS	?	√	√	-
Our work		**'25**	**B**	**-**	√	√	√	**M**	**CS**	**4**	√	√	√

Our method has common ground with the work of Anderson et al. [4] but we apply it to the Cobalt Strike framework. Van der Eijk et al. [20] designed a threshold-based method to detect Cobalt Strike using network traffic metadata. However, there are limitations to their work. First, they only detect one profile mimicking Amazon. We address this issue by experimenting on four different profiles that mimic four distinct websites. Then, they use empirically and manually chosen values as thresholds for their detection algorithm. Finally, they only use the accuracy as performance metric. As their data contains much more benign instances than malicious ones, their results are biased. We address this potential bias due to the dataset imbalance by using a carefully designed training procedure and appropriate metrics. More recently, Ramos et al. [54] used multi-flow features to detect Cobalt Strike HTTPS sessions. This approach has an objective close to ours, but their method focuses on a different scale of the communication. Before this, Ramos et al. [53] used supervised machine learning to detect Cobalt Strike C2 traffic, but their work suffers from several limitations. First, they do not specify the number nor the details of the publicly available and randomly generated malleable profiles used to train their different models. Hence, there is no assurance that their work is based on a realistic deployment situation. This also restricts the reproducibility of their results. We address this issue by using profiles known to be deployed in the real world for our experiments and by documenting them. Then, their dataset construction process suffers from some limitations. They use both HTTP and HTTPS traffic for their malicious traffic while only HTTPS traffic is used for the benign part. This biases packet size-related network features. We avoid this bias by using both HTTP and HTTPS for benign traffic. We also take the DNS C2 use case into account while they do not. Also, they transform categorical features into integers that encode the rank in terms of decreasing appearance frequency in the dataset. Thus the most frequent value is encoded as "1", the second most frequent value as "2", etc. This brings an order relation between initially unordered values and thus a bias for many methods such as tree-based Random Forest, which is put forward in the paper, or linear models such as Logistic Regression. Thus, we argue that their model is too limited in terms of malleable profiles and biased to be successful in a production environment. Finally, they do not analyze

the feature importance of their model, as shown in Table 1. This prevents from understanding why the model is efficient to detect Cobalt Strike C2 traffic. We address this issue by computing the Mean Decrease in Impurity (MDI) of the different features used in our experiments. More recently, Yang et al. [70] used a deep learning approach to detect HTTPS traffic from Cobalt Strike. Their approach suffers from several issues that we address in this work. First, they are only taking HTTPS traffic into account. By using one protocol, they omit several uses of Cobalt Strike C2 traffic. We address this issue by proposing a method taking all the protocols usable to connect with a C2 server in Cobalt Strike. Then, they do not consider the different malleable profiles they gathered in their datasets. Moreover, by using only the default configuration in their C2 traffic generation lab, they ignore the main difficulty regarding Cobalt Strike C2 traffic detection which is its masquerading capability and the associated configuration diversity [31], as pictured in Subsection 5.2. We address this issue by studying several configurations. Furthermore, their approach based on deep learning may not be suitable in production environment as the detection is harder to explain. Finally, some features, such as tokenized encrypted *Application Data* payload exhibits dubious usefulness. We address this issue by using common metadata features and explainable algorithms.

The detection of Cobalt Strike's C2 traffic can also fall within the more general subject of detecting covert channels. Our use case can be formalized as the detection of a Covert Storage Channel (CSC) with a passive warden as presented in "The prisoner problem" [58]. This problem has been studied in several papers gathered in surveys [38,71]. However, these detection techniques can not apply to covert channels with a high degree of customization and an encryption of the application layer which contains the payload, as in Cobalt Strike. Machine learning approaches have also been explored [21] but mainly for the DNS protocol [6] or timing based covert channels [9]. However, these detection techniques are making the hypothesis that a single method and protocol are used by the covert channel and that this information is known by the defender. As Cobalt Strike is capable of using several protocols and methods to masquerade its malicious traffic, we need a generic method able to adapt to different configurations.

3 Cobalt Strike

Cobalt Strike is an off-the-shelf penetration testing framework written in Java and targeting Windows operating systems. This section details the working principles of *Command and Control* communications in Cobalt Strike. Subsection 3.1 deals with the general architecture. Then, Subsection 3.2 describes the different possibilities brought by malleable profiles and how C2 protocols work.

3.1 Architecture

Command and Control in Cobalt Strike is based on three elements: a server controlled by the attacker, a Beacon on each compromised system and a client used by the attacker to control the server. Figure 1 illustrates how these components interact with each other. The server is at the center of communications between attackers and compromised systems. It forwards commands from the attackers and perform on-demand initial exploitation executable generation. In particular it creates Listeners and Beacons, the two agents of the C2 communication, based on the configured malleable profile. Beacons are executed on a compromised system in order to execute C2 commands selected by the attacker and sent by a Listener that is running on the server side. Finally, the client is a graphical interface used by the attacker to interact with the server.

3.2 Malleable Profiles

Command and Control configuration in Cobalt Strike is based on malleable profiles. These profiles are files that define the behavior used during the communication between the attacker server and the Beacons installed on the target [13]. A malleable profile contains all the parameters structuring the generated traffic, in particular the metadata used to impersonate benign traffic such as the values of the HTTP headers parameters, the encoding method used to obfuscate the payload or the "sleeptime" between two C2 communications. The malleable profile may also define a TLS certificate as Cobalt Strike can use an existing or self-signed certificates to create HTTPS C2 channels. As multiple servers can use the same profile (see Subsection 5.2), detecting a profile may allow one to identify several servers that use similar malleable profiles.

Communication behaviors can be defined for different protocols within a single profile as Cobalt Strike supports HTTP, HTTPS, DNS, SMB and TCP. In the following, we focus only on HTTP(S) and DNS protocols because SMB and TCP are only used in a peer-to-peer connection between two Beacons and not between a Beacon and a server. This Beacon-only communication is designed to limit the number of hosts calling out to the server during lateral movements [15].

The C2 communication can be described in four steps. First, at configurable time intervals (with or without jitter), the Beacon contacts the Listener on the server with identification metadata to check if there are commands to execute. This is called a "check-in". Then, the Listener responds with the attacker's commands or, if there is none, with the default answer defined in the profile.

Once the output of the commands is known, the Beacon sends it to the server with a different request. This request also contains authentication metadata, so the server can sort the requests when several Beacons are used. Finally, the Listener can send a final answer which is optional. For HTTPS, a TLS handshake is executed to fetch the orders and post the outputs. The version of TLS used depends on the OpenJDK version used by Cobalt Strike. The DNS communication is slightly more complex than the HTTP(S) one but follows the same logic. An illustration of these protocols is presented in Figure 2.

4 Threat model

We consider the following threat model. The attacker controls an external C2 server and at least one compromised machine inside the supervised network, running a Cobalt Strike Beacon. Thus, these two hosts establish C2 channels using either HTTP, HTTPS or DNS. We, as defenders, can passively monitor incoming and outgoing network traffic, but we cannot analyze the application layers. To put it in a covert channel detection context, as described by Simmons [58], we are a passive warden trying to detect a Covert Storage Channel between the Cobalt Strike server and the infected host which are the two prisoners.

Our goal is then to distinguish C2 communications from benign traffic.

5 Method

We detail the proposed method in the following parts. Subsection 5.1 outlines its general principle. Subsection 5.2 presents our data generation processes while Subsection 5.3 depicts the feature extraction of the metadata for the datasets' construction. Subsection 5.4 describes our machine learning method.

5.1 Principle

Our objective is to detect Cobalt Strike C2 communications, using only network traffic metadata. This method should work with different malleable profiles and should be extensible to easily take into account new malleable profiles.

The use of malleable profiles makes it complex to detect Cobalt Strike C2 channels using patterns in headers or payload. The encryption of the application layers introduced by recent changes in security protocols, such as TLS 1.3, DNS over HTTPS or encrypted `Client Hello` (formerly limited to an encrypted `Server`

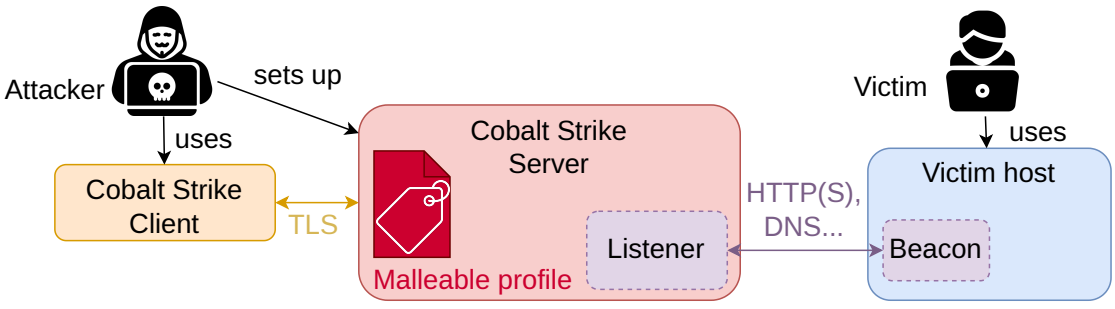
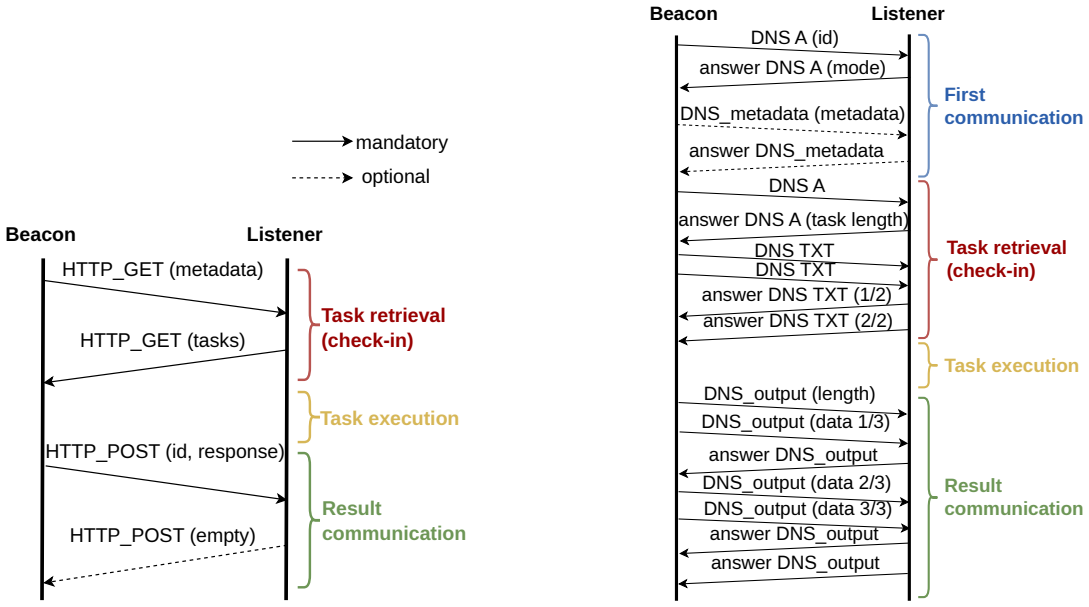


Fig. 1: Cobalt Strike architecture.



(a) HTTP (b) DNS in TXT mode.

Fig. 2: Examples of C2 communications based on HTTP and DNS.

`Name Indication`) also adds to the detection obstacles regarding packet content inspection. Supervised machine learning models based on metadata are thus an answer to masquerade and encryption.

We use a supervised machine learning model for each "group" of malleable profiles we consider, each profile in a "group" mimicking the same benign traffic. This facilitates the extension of the proposed method to new profiles. Our method aims to select the most relevant model regarding the context to obtain the best performance. Thus, with our method, a single model may help detect slightly different malleable profiles, for example derived from the public repository [14].

The proposed method is depicted in Figure 3. First, we make the hypothesis that we know what protocol is used by the traffic to be classified (e.g. based on the UDP/TCP ports). Because the purpose of Cobalt Strike is to be stealthy and overcome firewalls, we suppose attackers use the same ports as benign traffic. Then, we check if a domain name is available in the traffic metadata. This information may be present in different locations such as the "Host" header for HTTP or the *Server Name Indication* for TLS. If we cannot extract a domain name, we check if the remote IP address belongs to a subnet part of a domain present in a known malleable profile (such as `3.0.0.0/15` which belongs to Amazon). If the observed traffic is linked to a known domain, we apply a machine learning model trained with malicious and benign traffic specific to this domain. Otherwise, we use several models, each trained with domain-specific malicious traffic and generic benign traffic.

5.2 Data collection

We set up two platforms to generate network traffic. The first one is based on a virtualized architecture to run *Command and Control* scenarios and capture the malicious traffic generated by the attack. The second one uses an automation framework to mimic user activity and create benign traffic as close as possible to real traffic. We also use external benign traffic datasets to minimize bias. The details of the data used in our experiments are listed in Table 2.

Malicious traffic Because Cobalt Strike is a tool designed for Windows attacks, we use a Windows Server 2022 virtual machine (VM) as the victim. The remaining part of the Vagrant-based architecture is composed of two Debian 11 VMs. One is the Cobalt Strike server, and the other is a `bind9` server used in DNS-based C2 channels experiments.

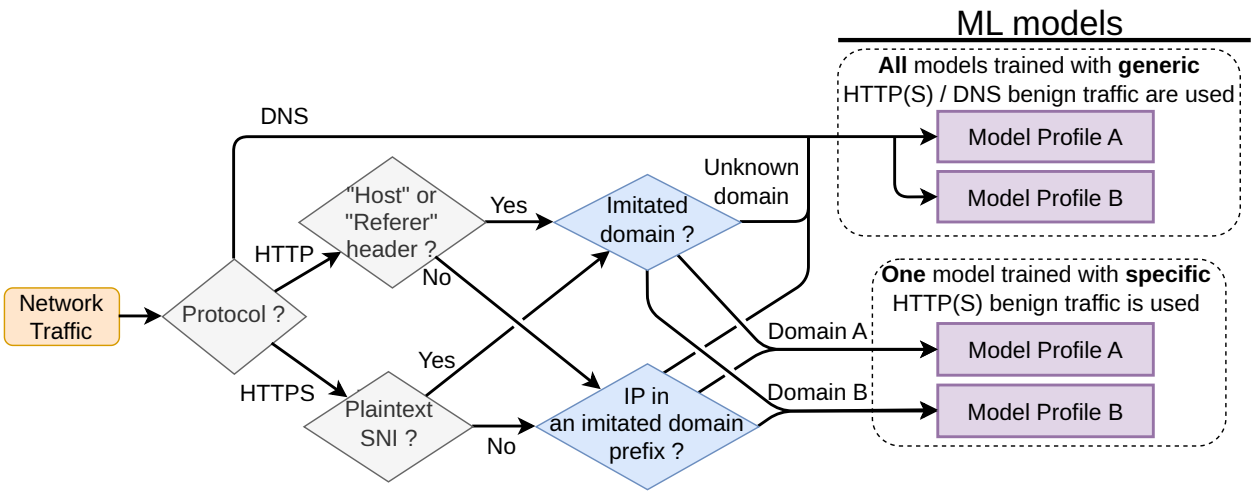


Fig. 3: Proposed method to detect Cobalt Strike C2 traffic. We use machine learning models trained on traffic which uses the same protocol (HTTP, HTTPS or DNS) as the observed traffic. Each model is trained using benign traffic as indicated on the figure and malicious traffic generated using the related profile.

Our detection approach targets network traffic observed between hosts (i.e. not on the host themselves). TCP offloading mechanisms are thus deactivated with the `ethtool` package to respect the default Ethernet *Maximum Transmission Unit* (MTU) value of 1 500 bytes. C2 commands, listed in Table 3, are chosen to be diverse regarding the amount of generated traffic and provided functionality.

We also gather malicious Cobalt Strike C2 traces from Malware Traffic Analysis (MTA) [47] to test our method and features on real-world attacks.

Benign traffic The benign traffic generation process is based on the Python bindings to the Selenium framework. This tool is used to simulate web browsing-related user inputs. Therefore, we write a script adapted to each service spoofed by Cobalt Strike that we study. For example, the script for Amazon search and select items based on predefined search terms as the corresponding studied Cobalt Strike profile uses "item search" like requests. Meanwhile, we capture the generated network traffic. We also use generic traffic from external datasets from the Universitat Politècnica de Catalunya (UPC) [7, 66], the Universidad Pública de Navarra (UPNA) [34, 35] and the Stratosphere IPS project of the Czech Technical University (CTU) [62] to minimize the bias of the benign traffic dataset. To use this traffic with our malicious data and minimize bias again, only flows that respect our condition on MTU are kept as stated in Section 5.2. It is then split by protocol we want to study: HTTP, HTTPS or DNS.

Choice of targeted malleable profiles Cobalt Strike provides its own malleable profile set [14] with an official repository containing 32 malleable profiles based on popular services and known attackers. Because this dataset does not provide any information about its use in attacks, our choice of malleable profiles is based on the Beacon dataset published by NCC [31]. It contains more than 120 000 Cobalt Strike Beacons fetched in the wild between 2018 and 2022.

We extract every profile in the dataset based on the IP/port pair and the month of its collection to avoid duplicates. After determining the domain mimicked for each of them by looking at the headers parameters, we apply a fuzzy hashing algorithm, TLSH, to get a ranking of the most used profiles [51]. We also use DBSCAN to cluster profiles with small differences, even if the mimicked domain value is different. DBSCAN parameters are $\epsilon = 30$ and $minPts = 2$. Here, the ϵ parameter is the maximal distance between two TLSH digests as defined in [51] within a cluster. Based on this information, we select three domains: `amazon.com`, `code.jquery.com` which are the two most mimicked domains and `smashburger.com` which has a significant low number of unique profiles for a great number of instances. The 10 most mimicked domains are listed on Table 4. Finally, for each of these three chosen websites, we select the most common profile that we adapt

Table 2: Number of instances (TCP or UDP flow) used in our experiments.

Traffic Type / Malleable Profile	**Protocol**	**(B)enign / (M)alicious**	**Instance Count**
Generic traffic (UPC+UPNA+CTU)	HTTP	B	73 873
	HTTPS	B	74 019
	DNS	B	279 455
Default profile	HTTPS	M	247
	DNS	M	7 618
Amazon	HTTP	M	172
	HTTPS	M	160
		B	615
	DNS	M	23 507
Smashburger	HTTPS	M	129
		B	230
jQuery	HTTP	M	2 225
		B	99
MTA Jan. 31st [42]	HTTPS	M	158
MTA May 23th [43]	HTTPS	M	121
MTA Jul. 12th [44]	HTTPS	M	803
MTA Oct. 3rd [45]	HTTPS	M	203
MTA Nov. 6th [46]	HTTP	M	2 208

so the C2 server answers with realistic answers. For example the jQuery profile returns the legitimate code after receiving a command result. The hash of the profiles used are listed in Table 5.

As many web services now enforce the use of HTTPS, the jQuery profile is chosen for the HTTP experiments. Indeed, this service is still accepting HTTP requests while others enforce HTTPS connections. For the HTTPS experiments, we use self-signed certificates generated by Cobalt Strike with mimicked parameters such as the Common Name for more stealthiness. Additionally, we use the default profile for testing HTTPS and DNS detection. This profile is found in several hundreds instances in the NCC dataset but with much variations, making it difficult to establish the correct count.

5.3 Dataset construction

Once the network traffic is captured, we extract the metadata to build the dataset. This is done by using the Zeek analysis framework [73]. Our Zeek scripts fetch the metadata for each flow, i.e. each TCP or UDP connection, from an input network capture. We also make sure to remove flows without any exchanged data, e.g. from a port scanning, as it would bring bias in the learning process.

Netflow is a protocol developed by Cisco to collect information and statistics on an IP network traffic based on *flows*. A *flow* is described using different values that differ between the versions of the protocol. We choose Netflow v5 [10] and Netflow v9 [11] features for our experiments. We limit our work on this metadata as they are standard and easy to collect in a production environment. We also extend these 2 sets with features computed on Netflow information to improve the classification process. Finally, we compute two ratios. One is the ratio of the total size of packets received to the total size of packets sent while the other is the same logic applied to the number of packets. Ramos et al. [53] uses three other features from a standard Zeek log [72]. The first is the ordered history of the TCP flags received in a flow. The second is the transport protocol used (TCP or UDP) while *service* is an inference on the application protocol based on the destination port e.g. 53 for DNS. These feature groups are presented in Table 6.

Table 3: C2 commands used for malicious traffic generation.

Command	Action
`bhashdump`	Dump local passwords hashes
`blogonpassword`	Dump passwords with Mimikatz
`bls`	List current directory
`bmimikatz`	Use a Mimikatz command
`bnet`	Run a network module command
`bportscan`	Scan the network using *Beacon*'s port scanner
`bps`	List processes
`bpwd`	Print current directory path
`brun`	Run a console command
`bpowershell`	Run a console command (brun variant)
`bscreenshot`	Take and send a screenshot

Table 4: Top 10 most common domains mimicked in the Cobalt Strike's Beacons dataset published by NCC group [31]. Each Beacon is counted once per month per (IP, port) couple. Nb: Number.

Spoofed domain	Nb of TLSH-based grouped profiles	Nb of instances	Nb of unique profiles
code.jquery.com	30	4 949	601
www.amazon.com	36	3 662	243
download.windowsupdate.com	42	993	145
locations.smashburger.com	2	827	3
www.google.com	40	722	195
www.bing.com	29	548	92
ocsp.verisign.com	24	520	54
www.microsoft.com	33	360	67
onedrive.live.com	15	316	63
audio-sv5-t1-3.pandora.com	7	281	38

5.4 Machine Learning: pre-processing, algorithm and metrics

After constructing the datasets, we apply supervised machine learning classification techniques. First, we scale the features using standardization. Then, we run a Random Forest algorithm, which is known to give good results and understandable explanation compared to other methods such as deep learning-based ones. We use a stratified k-fold cross-validation with a common value of $k = 10$ for the learning process to limit the bias from the imbalance between the proportion of benign and malicious traffic in the dataset. Finally, we use a grid search for hyperparameter tuning and optimize our performance. This grid search tunes the number of trees (10, 100 or 500), the quality of a split criterion (gini or entropy), the depth achievable by a tree (2, 5, 10, 15 or 20) and the minimum of sample for a split to occur (2, 5, 10 or 50).

Because we are in a security use case where alerts are processed by human beings, it is important to limit the number of false positives. Thus, we choose the F_1 score as a metric. We also use precision and recall

Table 5: Hashes of the different malleable profiles used.

Malleable Profile	SHA-1 hash
Default	`cb8632399e2c07b7b69e2403f3d543ac870176a4`
Amazon	`e3ee5e42845ef28a19fd6dd39b418acab279582d`
jQuery (with realistic answers)	`9b6551fb41c96b47f3e4153a0603458166fe44c6`
Smashburger	`1561b087573c535fc953336b509df88de82b75ba`

Table 6: Used network traffic features. ext.: extended; $\mathcal{B}$: bi-direction (Beacon to Listener and Listener to Beacon jointly) ; $\mathcal{U}$: uni-directional (Beacon to Listener and Listener to Beacon separately) ; $\mathcal{A}$: all directions (bi-direction and two uni-directions) ; R: $\frac{received}{sent}$ ratios ; S: sent packets only (Beacon to Listener); L3/4: Layer 3/4. Byte counts are based on the layer payload.

Feature group name	**Flow information**							
	Nb packets	**Packet size**	**OSI layer used for size**	**Duration**	**TCP flags**	**TCP history**	**Protocol**	**Service**
Netflow v5	$\mathcal{B}$	total ($\mathcal{B}$)	L3	√	√	-	-	-
Netflow v5 ext.	$\mathcal{B}$	total & mean ($\mathcal{B}$)	L3	√	√	-	-	-
Netflow v9	$\mathcal{U}$	total ($\mathcal{U}$); minimum & maximum (S)	L3	√	√	-	-	-
Netflow v9 ext.	$\mathcal{A}$; R	total & mean ($\mathcal{A}$); minimum & maximum (S); R	L3	√	√	-	-	-
Ramos et al. [53]	$\mathcal{B}$	total ($\mathcal{B}$)	L4	-	-	√	√	√

metrics to better analyze the performances of the models. To measure the importance of each feature in the decision of the model, we use the Mean Decrease in Impurity (MDI).

6 Results

The goal of these experiments is to evaluate the performance of the models used in our method which are depicted in purple boxes in Figure 3. We compare them to the Random Forest approach of Ramos et al. [53] which is put forward in their article as it has a good F_1 score and the smallest false positive rate. Yang et al. [70] is not considered as deep learning is not easily explainable.

The different sets of features and malleable profiles used are depicted in Table 6 and Table 7. Netflow uses the layer 3 payload size, designated in the following as "size". To visualize the results, we generate box plots of the F1 scores of the different models in Figure 4. Each box plot is based on the 10 values from the 10 cross-validation folds.

Our preliminary experiments show that the duration feature has a strong impact on the model's performance. This may be explained by the fact that our malicious server and victim are located in a virtualized environment where there is less RTT and jitter whereas the benign traffic is generated using servers in the wild. We also observe a strong impact of CWR and ECE TCP flags, explained by the fact that benign traffic datasets are generated using Linux operating systems which disable these flags by default, whereas the malicious traffic is necessarily produced by Windows hosts where these flags are enabled by default. To limit this bias, we perform experiments without the duration or these two flags.

We present our results in the following parts. First, Subsection 6.1 deals with the case of DNS C2 traffic. Then, Subsection 6.2 compares the performances of the two methods when no known profile can be linked to the observed traffic. Subsection 6.3 analyzes the results when a known domain can be linked to the traffic observed. Finally, in Subsection 6.4, we apply our method and compare the performances of Netflow features to those Ramos et al. [53] propose using publicly shared malicious traces [42–46].

As we observe that the extended feature sets only slightly improve the performance, we focus on the basic sets which are easier to collect in a production environment. Thus we compare three methods: our method with Netflow v5 features, our method with Netflow v9 features (both without the features discarded in the previous paragraph), and Ramos et al. [53] method.

Table 7: Mimicked activities and network traffic used in experiments. The labels of the experiments associated are listed in the last column. JS : JavaScript, Smashb.: Smashburger.

Mimicked activity	**Targeted malleable profile traffic (protocol)**	**Benign traffic (protocol)**	**Experiment label**
Website browsing	Amazon (HTTP)	Amazon (HTTPS) Generic (HTTP)	Az/Az (HTTP) Az/Gen (HTTP)
	Amazon (HTTPS)	Amazon (HTTPS) Generic (HTTPS)	Az/Az (HTTPS) Az/Gen (HTTPS)
	Smashb. (HTTPS)	Smashb. (HTTPS) Generic (HTTPS)	Sb/Sb (HTTPS) Sb/Gen (HTTPS)
	Default (HTTPS)	Generic (HTTPS)	D/Gen (HTTPS)
JS library download	jQuery 3.6 (HTTP)	jQuery (HTTP) Generic (HTTP)	jQ/jQ (HTTP) jQ/Gen (HTTP)
DNS	Default (DNS) Amazon (DNS)	Generic (DNS)	D/Gen (DNS) Az/Gen (DNS)

To visualize the results, we also generate the learning curves with a confidence interval at a 95% threshold, as presented in Figure 5. Then, we group the experiments by protocol used in the C2 traffic. The learning curves are composed of twenty points corresponding to training sizes equally spaced on a logarithmic scale between fifty samples and the size of the overall training dataset. The stratified k-fold cross-validation discussed before guarantees the separation between the training and testing flows for each point of the curve. Since some of the cases studied generate more flows than others, e.g. web browsing vs jQuery downloading, the sample sizes used for the learning curves differ.

6.1 Detection of DNS C2 traffic

We first evaluate the DNS protocol use case. The experiments pictured in Figure 4 (a) and (b), are approaching the maximum F1 score of 1. A straightforward observation is the score improvement when Netflow v9 features are used compared to Netflow v5. This improvement can also be observed in Figure 5 (d) and (g) as the learning curves reach the maximum F_1 score of 1 with ten times less training flows with Netflow v9 features than with Netflow v5 features. The Mean Decrease in Impurity (MDI) used to measure the importance for each feature in Figure 6 (a) shows that the total size of data exchanged is the most important feature whereas the number of packets is not important at all. We manually checked that Cobalt Strike is using standard DNS streams composed mostly of 2 packets which is similar to benign traffic.

Takeaway

To summarize our results on DNS C2, the performance of the three methods are close as they all include the payload size as a feature, which is visibly the biggest difference between malicious and benign DNS traffic. However, our features are easier to collect and more adapted to production environments.

6.2 Detection of HTTP(S) C2 traffic linked to an unknown domain

We then compare both methods when the traffic can not be associated to a known imitated domain. Again, we observe a great improvement in the detection capability of our model using Netflow v9 features compared to Netflow v5. The detection performance is lower for HTTPS Web browsing-masquerading C2 among generic traffic than DNS traffic, as pictured in Figure 4 (e) to (g). This is caused by the diversity of traffic in the generic dataset, having thus more samples similar to the malicious ones. This lowers the recall and thus the F_1 score.

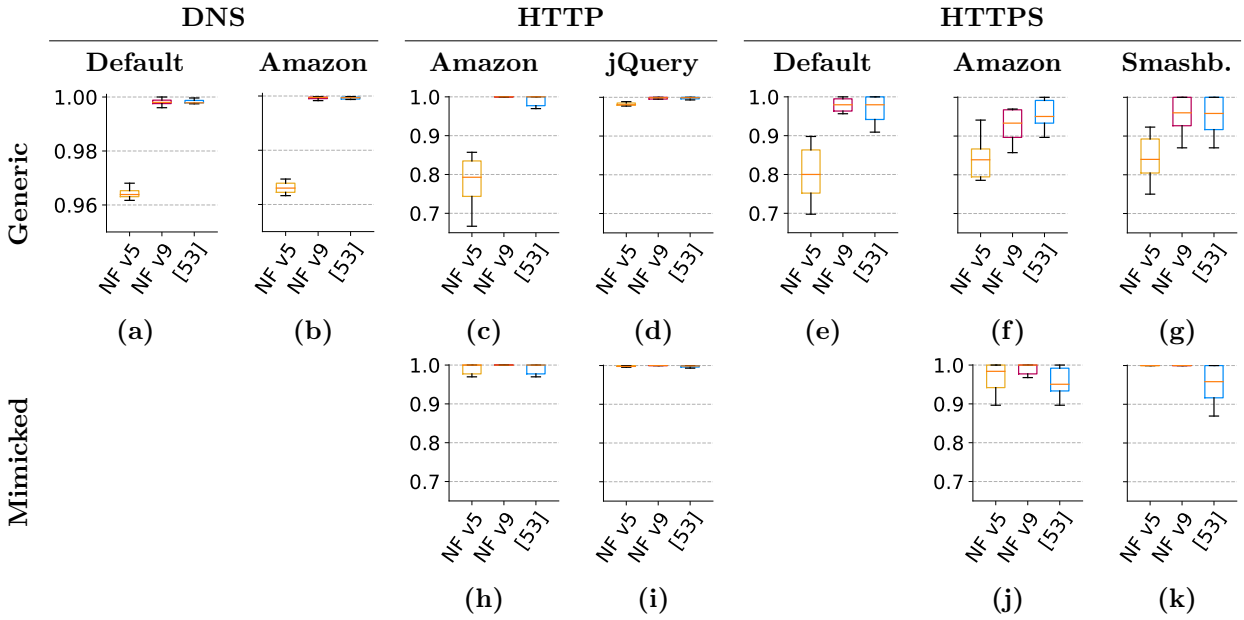


Fig. 4: F_1 score of the different models. Ramos et al. [53] uses generic DNS or HTTP and HTTPS combined benign traffic. Smashb.: Smashburger, NF: NetFlow

The learning curves corresponding to these experiment in Figure 5, using generic benign traffic, thus need significantly more training flows to reach an asymptotic value. It is even hardly reached for HTTPS experiments as shown in Figure 5 (c), (f) and (i).

The MDI for each feature in experiments with Netflow v5 shows that the total size of data exchanged has the most impact. We also observe in Figure 6 (b) and (c) that, in general with Netflow v9 features, the payload size's extremums from the source have a bigger impact on the decision than the number of packets. This is because the size of the payload for a profile has predefined values during check-ins. For instance, Amazon profile based Beacon sends data up to 1480 bytes while the minimum is 20 bytes. Check-ins without commands exhibit this minimum size, and are thus easy to identify. Moreover, the data exfiltration by the Beacon implies that a lot of data is sent by the source, which contrasts with a benign activity where the client receives more data than it sends. Hence, features built on traffic from the source are more crucial for the final decision.

Takeaway

To summarize, the performance of our method with Netflow v9 is similar to the method proposed by Ramos et al. [53]. However, the features we use are easier to collect. Netflow v5 can also be used with lower performance scores.

6.3 Detection of HTTP(S) C2 traffic linked to a mimicked domain

Then we study the case when masquerading traffic can be linked to a known imitated domain. Unlike the approach described in Ramos et al. [53], the method we propose select a specialized model to perform the detection. The improvement of performance by using Netflow v9 compared to Netflow v5 is still observed but, more important, the F1 score is greater than with Ramos et al. [53] in all experiments as the median has an equal or higher value and the box plot is tighter, as seen in Figure 4 (h) to (k).

The corresponding learning curves presented in Figure 5 show easier learning process for the models to reach the same F_1 score than models detecting unknown traffic. They need only several hundreds training flows compared to the several thousands flows used in the previous experiments.

For Netflow v5 features, we note that the feature importance difference between the total size and the number of packets is less significant when the benign traffic is specific to the domain mimicked by Cobalt

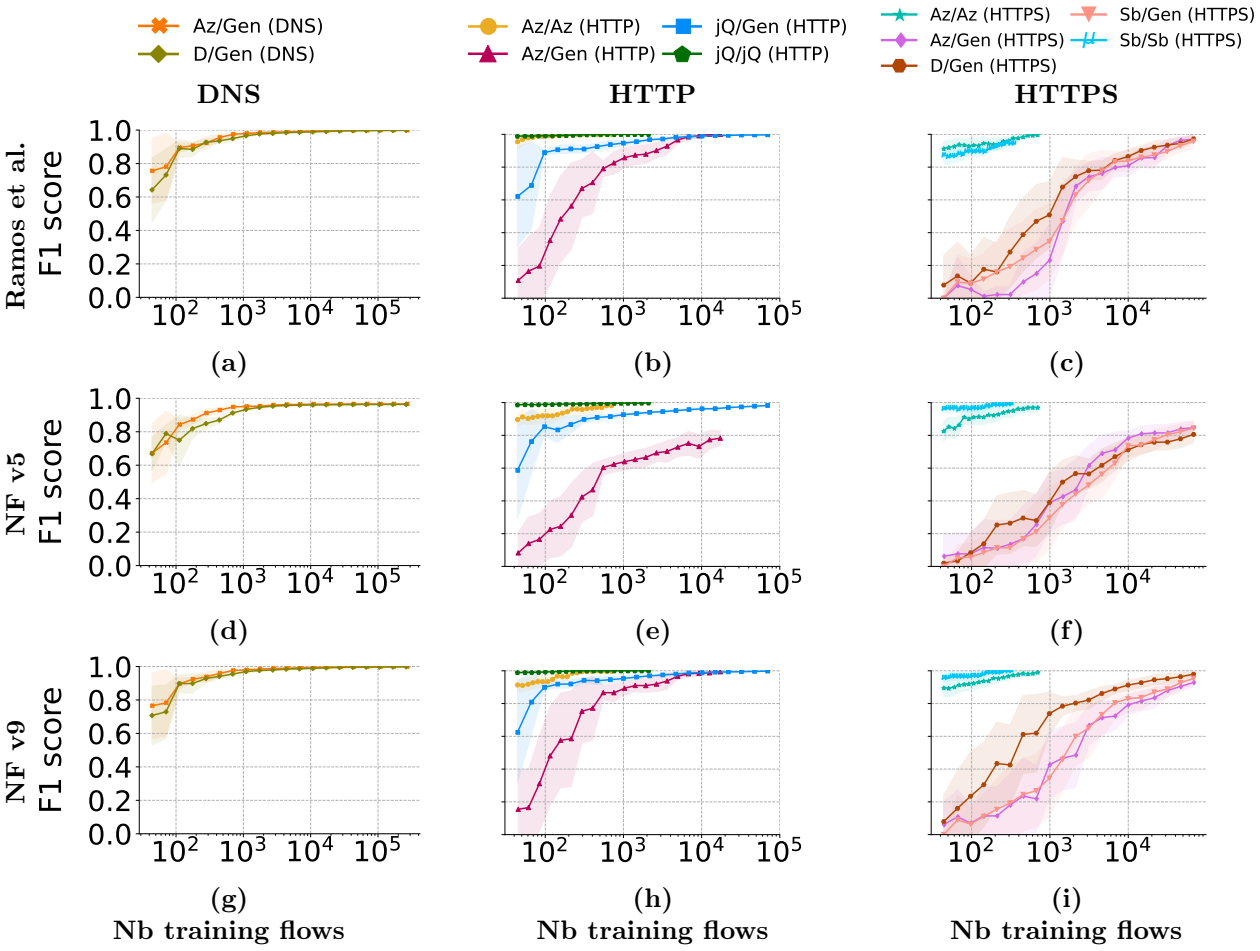


Fig. 5: Learning curves using F_1 score with a confidence interval at a 95% threshold for the different feature groups. NF: NetFlow, Az: Amazon, D: default, Gen: generic, jQ: jQuery, Sb: smashburger.

Strike. We suppose that specific benign traffic has a smaller variation in numbers of packets than generic benign traffic, justifying closer feature importance values. Moreover, Figure 6 (d) shows the impact of the maximum size sent by the client for HTTP experiments. Again, check-ins without commands stand out when compared to specific benign traffic.

Takeaway

To summarize our results on generated C2 traffic, we obtain good performances with the Netflow v9 based feature sets. Furthermore, the use of specific traffic significantly improves the different scores compared to generic traffic. Thus, a method that selects a specific model matching the observed traffic provides the best performances and outperforms the approach proposed by Ramos et al. [53].

6.4 Detection of documented real-world Cobalt Strike traffic

Finally, we apply our method to Cobalt Strike C2 traffic collected in 2023 [42–46]. To minimize bias, we study traces with at least a hundred Cobalt Strike TCP flows. This malicious traffic has been collected during real-world attacks by Malware Traffic Analysis [47], as presented in Section 5.2. The four traces with HTTPS traffic, detailed in Table 2, contain unencrypted SNIs. However, as they do not correspond to any targeted profile, our method uses several models trained with generic HTTPS benign traffic, as depicted on the bottom of Figure 3.

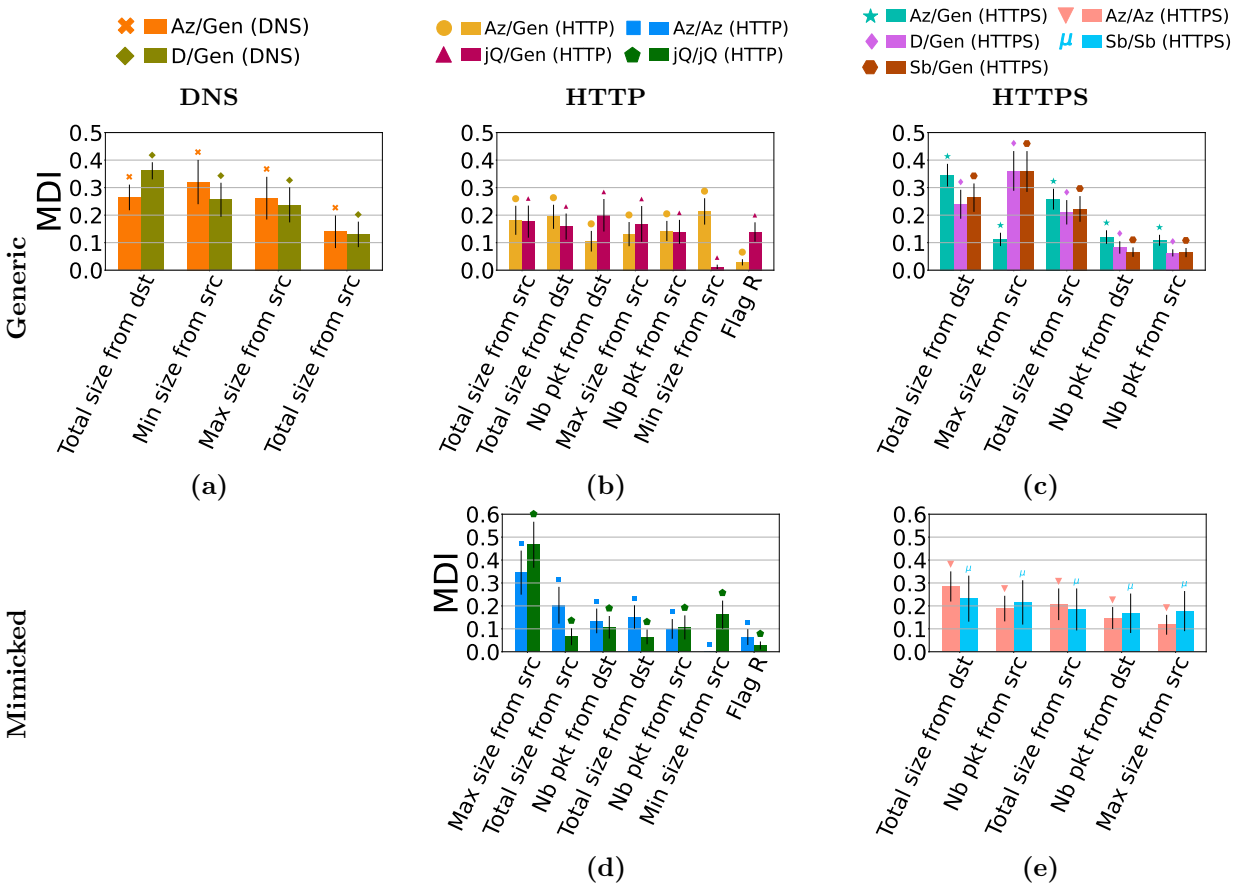


Fig. 6: Feature importance based on MDI with a confidence interval at a 99% threshold for Netflow v9. Only features with a mean value greater than 0.04 are kept. Az: Amazon, D: Default, Gen: Generic, jQ: jQuery, Sb: Smashburger, pkt: packet, src: source (client), dst: destination (server), R: Reset.

We observe that our method obtains downgraded performances. This may be occurring because the correct profile is not present inside training data. On the contrary, for a trace containing HTTP traffic with a known referer header corresponding to jQuery [46], our method uses one model trained with jQuery HTTP benign traffic. We then obtain a F_1 score of 0.99, as pictured in Figure 7 (a). We manually extract and check afterwards that the malleable profile used in this trace is, in fact, a jQuery mimicking profile.

To evaluate the capability of Netflow features-based models to detect real-world HTTPS Cobalt Strike traffic, we train four models on the HTTPS selected traces. We then compare our results to those obtained with the method proposed by Ramos et al. [53], as pictured in Figure 7 (b) to (e). Metrics values are presented in appendix in Table 8, including precision and recall. We observe that in most cases, models based on Netflow v9 features equals or outdo the performances reached by Ramos et al. [53]. Thus, Netflow v9 based models are proficient to detect external HTTPS *Command and Control* but Netflow v5 can also be used to create simpler yet effective models.

Takeaway

To summarize, our method applied to real-world traces performs well when the profile can be identified (see results for the jQuery profile [46]). Then, we show that Netflow features-based models' performance roughly equals Ramos et al. [53] and surpass it in some cases. Thus our method's performance is optimal when diverse profiles are used for training, as this increases the probability to use a specific model matching the traffic studied. These experiments show similar results as those in virtualized environment.

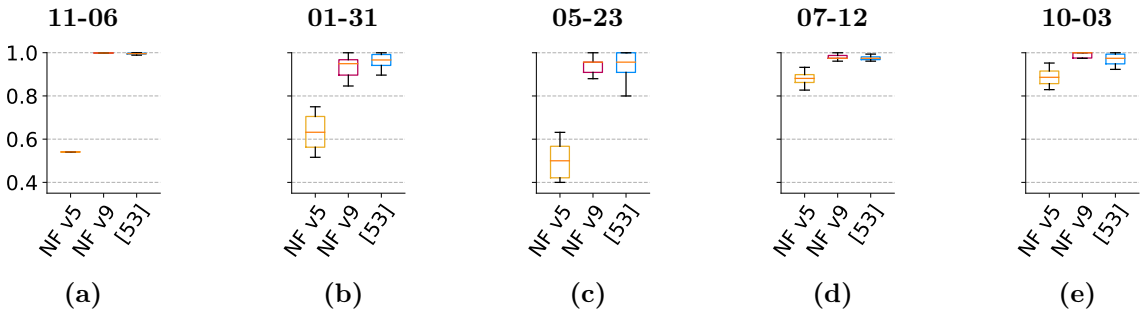


Fig. 7: F_1 score with a confidence interval at a 95% threshold for detecting masqueraded Cobalt Strike C2 observed in the wild. Ramos et al. [53] uses generic HTTP and HTTPS combined benign traffic. NF: NetFlow.

7 Ethics, discussion and future work

We acknowledge that inspecting packets has an ethical impact on privacy. However, we argue that we minimize this impact by searching only for the domain name and that our method still provides good results when no domain is linked to the observed traffic. Moreover, the traffic generated in our work is not captured from real users' activity. We identify three main limitations to our work. First, the number of malicious samples in our experiments is low as we have a limited number of attack scenario to run on our platform. This prevents us from obtaining optimal results but collecting more information about real use of Cobalt Strike can help to improve our datasets. Second, the number of malleable profiles studied is still limited while being higher than in most related works. This is because we limit our experiments to three popular profiles detected in the wild. This may explain the downgraded performances observed during the evaluation of our method on external HTTPS traces. Finally, the generic traffic datasets used are a few years old. More recent datasets would result to models closer to the current state of network traffic. These limitations will be addressed in future work.

We plan to extend this work along three axes. The first axis is to continue our experiments on other kinds of profiles. In a *Command and Control* communication, the victim sends more data than it receives. However, the tested profiles mimic activities where the victim receives more data than they send, making the detection easier. By studying the capability to detect data-sending oriented profiles such as cloud storage uploads or videoconferencing, we suppose it would help to better estimate the efficiency of other models and improve the coverage offered by our method. The second axis is the evaluation of a general performance of our method. We are also interested in the domain generalization of our method inside malleable profile groups (see Section 5.2), i.e. how well a model trained on data from a single profile performs on samples from close-related malleable profiles. This work will be based on the TLSH-based groups presented in Table 4. Finally, the third axis is to extend this work towards other frameworks [27] and Cobalt Strike extensions that are used in the wild [57]. We hypothesize that the performance may be preserved as the proposed method is not based on Cobalt Strike's specific parameters.

8 Conclusion

In this paper, we propose a new network traffic metadata-based machine learning method to detect Cobalt Strike masquerading C2 traffic. This passive detection method, in opposition with previous work, is based on unsophisticated but widely used features so it can be easily deployed in production environments. It can also adapt the model used to the observed traffic to optimize its performance. We evaluate the performance of this method on data generated using several widely used Cobalt Strike configurations deployed in a realistic virtualized architecture. We are the first paper providing such detailed, documented and reproductible evaluation. We show that a Random Forest model can detect Cobalt Strike masquerading *Command and Control* with or without encryption. We also show that this method performs better than the state of art when mimicked websites or services can be identified, and provides similar results to previously proposed approaches when it is not the case. Finally, we show that our method performance is similar to a previously

proposed method on real-world attacks while using more standard features. The artifacts to reproduce the results of this paper are released at `https://github.com/cp-tsp/ares2025-artifacts`.

Appendix

Table 8: Metrics computed and rounded down to one hundredth with a confidence interval at a 95% threshold of the different experiments in comparison with Ramos et al. [53]. Exp: Experiment, P: precision, R: recall, H: HTTP, HS: HTTPS, D: DNS.

Exp.	**Netflow v5**			**Netflow v9**			**Ramos et al. [53]**		
	F_1	P	R	F_1	P	R	F_1	P	R
Az/Az (HTTP)	.98±.02	1±0	.96±.04	**.99±0**	1±0	.99±.01	.98±.01	.99±.01	.98±.02
Az/Gen (HTTP)	.78±.04	.85±.05	.72±.06	**.99±.01**	.99±.01	.99±.01			
jQ/jQ (HTTP)	.99±0	.99±0	.99±0	**1±0**	1±0	1±0	.99±0	.99±0	.99±0
jQ/Gen (HTTP)	.98±0	.98±0	.97±0	.99±0	1±0	.99±0			
Az/Az (HTTPS)	.96±.02	1±0	.94±.04	**.99±.01**	.99±.01	.98±.01	.95±.02	1±0	.91±.04
Az/Gen (HTTPS)	.84±.05	.88±.05	.81±.06	.92±.02	.99±.01	.87±.05			
Sb/Sb (HTTPS)	.99±.01	.98±.02	1±0	**1±0**	1±0	1±0	.95±.03	1±0	.91±.06
Sb/Gen (HTTPS)	.84±.04	.88±.05	.81±.07	.95±.03	1±0	.91±.06			
D/Gen (HTTPS)	.80±.05	.87±.05	.75±.07	**.97±.01**	1±0	.95±.02	.96±.02	1±0	.94±.04
D/Gen (D)	.96±0	.95±0	.97±0	**.99±0**	.99±0	.99±0	**.99±0**	.99±0	.99±0
Az/Gen (D)	.96±0	.96±0	.97±0	**.99±0**	.99±0	.99±0	**.99±0**	.99±0	.99±0
01-31 (HTTPS)	.60±.10	.68±.11	.63±.09	.92±.05	1±0	.88±.06	**.94±.05**	1±0	.89±.05
05-23 (HTTPS)	.49±.05	.54±.08	.48±.09	.93±.04	.99±.01	.87±.07	**.94±.04**	1±0	.89±.05
07-12 (HTTPS)	.88±.02	.87±.03	.88±.02	**.98±0**	1±0	.96±.01	.97±0	1±0	.95±.02
10-03 (HTTPS)	.88±.02	.91±.03	.87±.04	**.98±0**	1±0	.97±.02	.96±.02	1±0	.93±.04
11-06 (HTTP)	.56±.04	.99±0	.40±.06	**.99±0**	1±0	.99±0	**.99±0**	1±0	.99±0
Exp.	**Netflow v5 extended**			**Netflow v9 extended**			**Ramos et al. [53]**		
	F_1	P	R	F_1	P	R	F_1	P	R
Az/Az (HTTP)	.98±.01	1±0	.97±.03	**1±0**	1±0	1±0	.98±.01	.99±.01	.98±.02
Az/Gen (HTTP)	.82±.03	.88±.06	.78±.05	**.98±.01**	.98±.01	.97±.02			
jQ/jQ (HTTP)	**.99±0**	.99±0	.99±0	**.99±0**	1±0	.99±0	**.99±0**	.99±0	.99±0
jQ/Gen (HTTP)	.98±0	.98±0	.97±0	**.99±0**	.99±0	.99±0			
Az/Az (HTTPS)	.97±.01	.97±.03	.97±.02	**.99±0**	.99±.01	1±0	.95±.02	1±0	.91±.04
Az/Gen (HTTPS)	.86±.03	.91±.02	.81±.07	.91±.03	.98±.01	.85±.06			
Sb/Sb (HTTPS)	.99±.01	.98±.02	1±0	**1±0**	1±0	1±0	.95±.03	1±0	.91±.06
Sb/Gen (HTTPS)	.86±.05	.92±.05	.82±.08	.94±.02	1±0	.89±.05			
D/Gen (HTTPS)	.80±.05	.88±.05	.74±.07	**.96±.01**	1±0	.93±.02	.96±.02	1±0	.94±.04
D/Gen (D)	.96±0	.95±0	.97±0	**.99±0**	.99±0	.99±0	**.99±0**	.99±0	.99±0
Az/Gen (D)	.96±0	.96±0	.97±0	**.99±0**	.99±0	.99±0	**.99±0**	.99±0	.99±0

References

1. Abu Rajab, M., Zarfoss, J., Monrose, F., Terzis, A.: A multifaceted approach to understanding the botnet phenomenon. In: ACM SIGCOMM IMC (2006)
2. Althouse, J.: Tls fingerprinting with ja3 and ja3s. https://engineering.salesforce.com/tls-fingerprinting-with-ja3-and-ja3s/ (2019)
3. Althouse, J.: Ja4+ network fingerprinting. https://blog.foxio.io/ja4%2B-network-fingerprinting (2023)
4. Anderson, B., McGrew, D.: Identifying encrypted malware traffic with contextual flow data. In: AISec (2016)
5. Anderson, B., McGrew, D.: Accurate tls fingerprinting using destination context and knowledge bases (2020)
6. Buczak, A.L., Hanke, P.A., Cancro, G.J., Toma, M.K., Watkins, L.A., Chavis, J.S.: Detection of tunnels in pcap data by random forests. CISRC '16 (2016)
7. Bujlow, T., Carela-Español, V., Barlet-Ros, P.: Independent comparison of popular dpi tools for traffic classification. Comput. Nets. (2015)
8. Censys: Jarm in censys search. https://docs.censys.com/docs/ls-jarm
9. Chen, S., Lang, B., Liu, H., Li, D., Gao, C.: Dns covert channel detection method using the lstm model (2021)
10. Cisco: NetFlow v1, v5, v7 and v8. https://www.cisco.com/c/en/us/td/docs/net_mgmt/netflow_collection_engine/3-6/user/guide/format.html (2007)
11. Cisco: NetFlow v9. https://www.cisco.com/en/US/technologies/tk648/tk362/technologies_white_paper09186a00800a3db9.html (2011)
12. CobaltStrike: https://www.cobaltstrike.com
13. CobaltStrike: https://www.cobaltstrike.com/help-malleable-c2
14. CobaltStrike: Official malleable profiles repository. https://github.com/Cobalt-Strike/Malleable-C2-Profiles (2014)
15. CobaltStrike: https://hstechdocs.helpsystems.com/manuals/cobaltstrike/current/userguide/content/topics/listener-infrastructure_peer-2-peer.htm (2022)
16. CrowdStrike: https://www.crowdstrike.com/blog/meet-crowdstrikes-adversary-of-the-month-for-june-mustang-panda/ (2018)
17. Cybereason: https://www.cybereason.com/blog/sliver-c2-leveraged-by-many-threat-actors
18. Dietrich, C., Rossow, C., Freiling, F., Bos, H., van Steen, M., Pohlmann, N.: On botnets that use dns for command and control. In: EC2ND (2011)
19. E. Rescorla, Mozilla: RFC on TLS 1.3. https://www.rfc-editor.org/rfc/rfc8446
20. Vincent van der Eijk, C.S.: Detecting Cobalt Strike beacons in NetFlow data (2020)
21. Elsadig, M.A., Gafar, A.: Covert channel detection: Machine learning approaches. IEEE Access pp. 38391–38405 (2022)
22. Felt, A.P., Barnes, R., King, A., Palmer, C., Bentzel, C., Tabriz, P.: Measuring https adoption on the web. In: USENIX Security (2017)
23. Freiling, F., Holz, T., Wicherski, G.: Botnet tracking: Exploring a root-cause methodology to prevent distributed denial-of-service attacks. In: ESORICS (2005)
24. García, S., Grill, M., Stiborek, J., Zunino, A.: An empirical comparison of botnet detection methods. Elsevier Advanced Technology Publications (2014)
25. Github: https://github.com/looCiprian/GC2-sheet
26. Github: https://github.com/YDHCUI/manjusaka
27. Github: https://github.com/BishopFox/sliver
28. Google Cybersecurity Action Team: https://services.google.com/fh/files/blogs/gcat_threathorizons_full_apr2023.pdf (2023)
29. Gu, G., Porras, P., Yegneswaran, V., Fong, M.: Bothunter: Detecting malware infection through ids-driven dialog correlation. In: USENIX Security (2007)
30. Gu, G., Zhang, J., Lee, W.: Botsniffer: Detecting botnet command and control channels in network traffic. In: NDSS Symposium (2008)
31. Hu, Y.Z.: Mining data from cobalt strike beacons. https://www.nccgroup.com/us/research-blog/mining-data-from-cobalt-strike-beacons (2022)
32. Kondo, S., Sato, N.: Botnet traffic detection techniques by c&c session classification using svm. In: Advances in Information and Comput. Secur. (2007)
33. Kravensecurity: C2 hunting: How to find c2 servers with shodan. https://kravensecurity.com/c2-hunting-using-shodan/ (2024)
34. Labayen, V., Magaña, E., Morató, D., Izal, M.: Network traffic and code for machine learning classification. In: Machine Learning for Networking (2020)
35. Labayen, V., Magaña, E., Morató, D., Izal, M.: Online classification of user activities using machine learning on network traffic. Comput. Nets. (2020)

36. Livadas, C., Walsh, R., Lapsley, D., Strayer, W.T.: Using machine learning techniques to identify botnet traffic. In: LCN (2006)
37. Microsoft: https://www.microsoft.com/en-us/security/blog/2021/05/27/new-sophisticated-email-based-attack-from-nobelium/ (2021)
38. Mileva, A., Panajotov, B.: Covert channels in tcp/ip protocol stack - extended version-. Open Computer Science (2014)
39. MitreAtt&ck: https://attack.mitre.org/software/S0154/
40. MitreAtt&ck: https://attack.mitre.org/techniques/T1071/001/
41. MitreAtt&ck: https://attack.mitre.org/techniques/T1071/004/
42. MTA: https://www.malware-traffic-analysis.net/2023/01/31/
43. MTA: https://www.malware-traffic-analysis.net/2023/05/23/
44. MTA: https://www.malware-traffic-analysis.net/2023/07/12/
45. MTA: https://www.malware-traffic-analysis.net/2023/10/03/
46. MTA: https://www.malware-traffic-analysis.net/2023/11/06/
47. MTA: https://www.malware-traffic-analysis.net/2023/
48. Nayak, C.: https://bruteratel.com/
49. Nivargi, V., Bhaowa, M., Lee, T.: Machine Learning Based Botnet Detection (2006)
50. N.Mavis: The art and science of detecting Cobalt Strike (2020)
51. Oliver, J., Hagen, J.: Designing the elements of a fuzzy hashing scheme. In: EUC 2021
52. Pai, K., Shubhodeep, M., Madhusoodhana, S.: Novel tls signature extraction for malware detection (2020)
53. Ramos, F.M., Wang, X.: A machine learning based approach to detect stealthy cobalt strike c&c activities from encrypted network traffic. In: Machine Learning for Networking (2022)
54. Ramos, F.M., Wang, X.: Detecting stealthy cobalt strike c&c activities via multi-flow based machine learning. In: ICMLA (2023)
55. Red Canary: Threat detection report. https://resource.redcanary.com/rs/003-YRU-314/images/2022_ThreatDetectionReport_RedCanary.pdf (2022)
56. Salesforce: Easily identify malicious servers on the internet with jarm. https://engineering.salesforce.com/easily-identify-malicious-servers-on-the-internet-with-jarm/ (2020)
57. SentinelOne: https://www.sentinelone.com/blog/geacon-brings-cobalt-strike-capabilities-to-macos-threat-actors/ (2023)
58. Simmons, G.J.: The prisoners' problem and the subliminal channel. In: Advances in Cryptology. pp. 51–67 (1984)
59. Sosnowski, M., Zirngibl, J., Sattler, P., Carle, G.: DissecTLS: A Scalable Active Scanner for TLS Server Configurations, Capabilities, and TLS Fingerprinting. In: PAM 2023 (2023)
60. Sosnowski, M., Zirngibl, J., Sattler, P., Carle, G., Grohnfeldt, C., Russo, M., Sgandurra, D.: Active TLS Stack Fingerprinting: Characterizing TLS Server Deployments at Scale. In: TMA 2022 (2022)
61. Staniford-chen, S., Cheung, S., Crawford, R., Dilger, M., Frank, J., Hoagl, J., Levitt, K., Wee, C., Yip, R., Zerkle, D.: Grids : A graph based intrusion detection system for large networks (1998)
62. Stratosphere: Stratosphere laboratory datasets. https://www.stratosphereips.org/datasets-overview (2015)
63. Talos: https://blog.talosintelligence.com/manjusaka-offensive-framework
64. T.J.O'Leary, T.Bonner, M.Janus, D.Given, E.Wickens, J.Simpson: Finding Beacons In The Dark (2021)
65. Unit42: https://unit42.paloaltonetworks.com/brute-ratel-c4-tool/ (2022)
66. UPC: https://historic.cba.upc.edu/monitoring/traffic-classification.html (2015)
67. US district court for the eastern district of New York: https://noticeofpleadings.com/crackedcobaltstrike/ (2023)
68. Wang, K., Stolfo, S.: Anomalous payload-based network intrusion detection. In: RAID (2004)
69. Warmer, M.: Detection of web based command & control channels. Ph.D. thesis (2011)
70. Yang, X., Ruan, S., Yue, Y., Sun, B.: Petnet: Plaintext-aware encrypted traffic detection network for identifying cobalt strike https traffics. Comput. Nets. (2024)
71. Zander, S., Armitage, G., Branch, P.: A survey of covert channels and countermeasures in computer network protocols (2007)
72. Zeek: conn.log. https://docs.zeek.org/en/master/logs/conn.html
73. Zeek: Zeek framework official website. https://zeek.org/
74. Zhang, H., Papadopoulos, C., Massey, D.: Detecting encrypted botnet traffic (2013)