Detecting Cobalt Strike beacons in NetFlow data

Vincent van der Eijk, **Coen Schuijt**
University of Amsterdam
{vincent.vandereijk, coen.schuijt}@os3.nl

Supervisor: dr. ing. Ralph Koning
r.koning@uva.nl

***Abstract*—In the current era of cyber security, realistic threat simulation is performed in order to bring the resilience of organizations to real attacks, to a higher level. The goal of a Red Team is to simulate attacks in a realistic manner, whereas the Blue Team tries to keep out adversaries. When analysing threat actors and their tool set, Cobalt Strike is prominent and used in the wild for good and bad. Even Advanced Persistent Threats (APTs) make use of this software. Within this research we provide insights in the approach for detecting beaconing traffic that is generated by Cobalt Strike as part of its attack infrastructure. We propose a detection algorithm based on four identifying network related features, which prove to be able to identify Cobalt Strike TCP beacons with an accuracy of 99.996%.**

***Index Terms*—Cobalt Strike, beacon traffic, Command and Control, botnet analysis, network security, NetFlow.**

I. Introduction

Cybercrime is an emerging business model in nowadays society [1]. Independent of motivation, gaining unauthorized access to computer systems and networks is happening on a daily basis. There exists a variety of malicious software (malware), which can be easily obtained via sources on the internet [2]. One prominent category of malware consists of those which serve the purpose to remotely controlling a system – often referred to as a Remote Access Toolkit (RAT). A combination of compromised systems (bots) can create a network, sometimes referred to as botnet (bot network).

Botnet software can be obtained online relatively easily and used by anyone, even without thorough understanding of IT [2]. Cobalt Strike is a commercial software package that provides the functionality to operate such a botnet. This software is known to be used by both white- and black hat hackers (good and bad), and the software is highly customizable [3]. It is even known to be used by Advanced Persistent Threats; highly skilled, often nation state backed hacking organizations [4]. Additionally, the software comes with impressive features that allow it to change its indicators to avoid detection. Because of the interesting capabilities provided by Cobalt Strike and its widespread use, we focus our research specifically on the Cobalt Strike software. The aim of this research is to determine if we can distinguish obfuscated Cobalt Strike beacons from genuine network traffic based on identifying features. In this paper we propose a method to detect the presence of a Cobalt Strike botnet host (beacon) in network traffic, based on NetFlow data.

A. Research questions

Our main research question is defined as follows:
How can we distinguish obfuscated Cobalt Strike beacons from genuine network traffic based on identifying features?

To answer this question we will look into the following subquestions:

- Which features can we extract from network traffic generated by Cobalt Strike beacons?
- Can we detect a Cobalt Strike beacon using a malleable profile with one or more of those features?

With regards to the research questions, the following hypotheses are defined:

- `H1`: Identifying features such as timing intervals and message length can be used to identify malicious network traffic.
- `H2`: Protocol flow information (metadata) provides valuable features for measurement.

In order to verify these hypotheses, we will conduct a series of experiments in which we analyse a

collection of datasets; NetFlow data containing Cobalt Strike network traffic that is generated by malleable C2 profiles, NetFlow data of genuine network traffic from external sources, and our own generated data. The experiments consist of testing several features of our proposed detection algorithm on these datasets, which is further explained in Section III.

B. Outline

First, in Section II, we provide additional background information regarding security operations, as well as the techniques that are currently used by botnet operators. Section III describes the methodology to setup our own botnet infrastructure using Cobalt Strike to generate botnet traffic in a controlled environment. Next, Section IV provides our results, including the analysis of the network traffic and the outcome of our experiments. A discussion is given in Section V, where we reflect on the results, as well as any limitations in our research. The conclusion is included in Section VI, and we finish with several suggestions for future work in Section VII.

II. Background

When describing types of security teams within a company, a distinction between offensive and defensive security is often made. This is also referred to as Red Teaming and Blue Teaming. Red Teaming is an advanced form of assessment that is used to test the security of a given IT network [5]. The Blue Team, on the other hand, tries its best to prevent the network from being compromised, by taking defensive measures and responding to detected treats.

In general, the Red Team follows the same approach as an arbitrary treat actor would take to perform its malicious operations. This approach often consists of, but is not limited to, the following phases; 1. Reconnaissance, 2. Weaponization, 3. Delivery, 4. Exploitation, 5. Installation, 6. Command & Control, and 7. Actions on Objective [6]. By following these phases and after successfully compromising the target network, this results in access to one or multiple compromised host machine(s) within the target network (achieved within phases 1-5). These compromised hosts are sometimes referred to as `bots`, which can be seen as a zombie machine that awaits instructions from a botnet server and performs certain automated operations (as executed in phase 6-7). A collection of bots is often called a `botnet`. When analysing the amount of botnets over the years, an increase can be observed [7]. We believe that many of today's Blue teams still need to cover a gap to keep up with the Red Teams (and real threat actors) in regards with botnet traffic detection.

A. State of the art

In today's era of Red Teaming operations and adversary simulations, Cobalt Strike is a tool that is commonly used to setup and maintain connections with a target infrastructure [8]. Cobalt Strike can be used to perform Command and Control (C2) operations (e.g. sending commands and instructions to target machines, or receiving information from those targets), much like bots as part of a botnet would do. Current Red Teaming operations show similarities with botnet traffic in the way that C2 network traffic is handled. Although Cobalt Strike is intended to be used from a Red Teaming perspective, it is also known to be used by threat actors to manage their botnet infrastructure [3].

Cobalt Strike has a feature to obfuscate its network traffic, known as `Malleable C2` [9]. This feature gives the Cobalt Strike operator fine grained control over the obfuscation of network traffic (both request and response) between the Cobalt Strike C2 server and target machines (beacons). The Cobalt Strike C2 traffic is controlled by so-called `Malleable C2 profiles`, which can be used to make Cobalt Strike network traffic look as if it is genuine traffic – for example browsing `amazon.com` [10]. Responses can be altered in such a way that the response seems legitimate, e.g. by including the magic bytes (a hexadecimal signature indicating the extension) of an image in a response [9]. Alternatively, there is an option to use profiles that aim to mimic the behaviour of well known malware. This is particularly interesting to hide ones own indicators and make it look like the real malware was present on the network. For companies this may be interesting to test how well they are capable of defending against such threats.

In a realistic assignment, a Red Team tries to remain undetected and operate in a stealthy manner. In this research we setup a realistic test environment in order to simulate Cobalt Strike network traffic as if it is used by real adversaries. The terminology regarding the tools and techniques used, is listed below:

1) Command and Control (C2): This refers to traffic between the target machine (on which the `beacon` is installed) and the Cobalt Strike C2 server. In a broader perspective, this term can be used to describe any traffic related to sending commands to beacons, as well as receiving data from them.

2) Redirectors: Rather than communicating with target machines directly, redirectors can be used to receive incoming connections and forward (proxy) those connections to the Cobalt Strike C2 server. The use of redirectors hides the actual destination of the traffic, and

may point to a genuine domain. The reason for hiding the Cobalt Strike C2 server is to prevent from being listed on a blacklist. Furthermore, real treat actors will apply similar measures in order to hide the origin and therewith their identity. Additionally, it is easier to tear down only the redirector after an assignment, rather than the need the setup a whole new Cobalt Strike infrastructure.

3) Domain redirection: This technique relies on the use of Content Delivery Networks (CDNs). This provides an extra layer of obfuscation in addition to redirectors in such a way that the origin domain is hidden behind a CDN redirected domain. It works by configuring the `Host` header in the HTTP GET request within the malleable profile, so that points to a malicious CDN domain (in our case CloudFront [11]). The CDN domain name resolves to the redirector IP.

4) Malleable profiles: These relate to the set of rules that define; how HyperText Tranfer Protocol (HTTP) traffic (GET and POST) is handled, the Uniform Resource Identifier (URI) parameters being used, how data is encoded, stored and transmitted, the HTTP headers and cookies being used, and so on [10]. Additionally, the beacon characteristics can be adjusted within the profile, such as the target `User Agent` specification, beacon callback interval, jitter (a percentual deviation from the callback time), and the key stores being used for SSL/TLS certificates.

5) Streams and flows: A flow is defined as the unidirectional network communication from one host to another [12]. A stream is defined as the bidirectional set of flows between two hosts.

We apply the techniques as described above to establish our own Cobalt Strike infrastructure setup, which is described in more detail in Section III.

B. Related work

At the time of writing, there is no scientific research known to us that addresses the detection of Cobalt Strike's malleable profiles specifically. However, there is plenty of research that addresses the detection of malicious network traffic from C2 servers [13, 14, 15, 16, 17].

Kevin P. Dyer et al. developed a programmable network traffic obfuscation system [18]. The research also provides insights in common types of obfuscation techniques. Furthermore, L. Dixon et al. provides an extensive overview of network traffic obfuscation, such as encryption, randomization, mimicry and tunneling [19]. Malleable profiles relate to this research in the way it mimics other network traffic. Mimicry attempts to evade detection by masquerading as whitelisted traffic.

C.J. Dietrich et al. developed an approach to detect botnet C2 channels based on distinct features such as carrier protocols, message length sequences and encoding differences [13]. Similarly, Wang et al. researched several methods for detecting obfuscated network traffic, including a machine learning based approach [14]. Furthermore, P. Prasse et al. provide an approach for detecting malware by analysing HyperText Tranfer Protocol Secure (HTTPS) traffic [20].

Existing efforts for fingerprinting TLS traffic have been made, one notable example being JA3(S) [21]. This approach analyses the cryptographic parameters of the TLS handshake and creates cryptographic hashes; a JA3 hash for the client and a JA3S hash for the server. These hashes are used to uniquely identify the client and server. However, these fingerprints are not completely unique, and making use of a redirector would prevent the JA3S hash to correctly identify the Cobalt Strike C2 server. Additionally, it is possible to modify the cryptographic parameters in order to avoid fingerprinting.

From an offensive point of view, M. Rigaki et al. used a Generative Adversarial Network (GAN) to mimic the behaviour of Facebook Chat in order to let the malicious traffic look realistic [22]. Our research shows similarities to this research in the way that malleable profiles work.

A common Indicator of Compromise (IoC) that is used by researchers is the interval between messages sent and received by the C2 server, where the assumption is made that a beacon connects back to its C2 server periodically in order to obtain commands. This is shown in research by L. van Duin in [23]. Furthermore, B. Jerzman researched Cobalt Strike beaconing behaviour, and also uses beaconing intervals as a means of detection [24]. Our research relies heavily on the interval between messages to the C2 server, which is also used to obtain additional features.

J. Dreijer analysed Cobalt Strike's use of the HTTPS beacon, by using payload length and TTL as indicators [25]. Moreover, this research confirms our findings regarding the RST flags in Cobalt Strike beacon traffic.

C. Scope

The main goal of this research is to be able to detect the TCP variant of the default *Amazon profile*, without making any alterations to it. We can expand on this by making alterations to the profile and test our algorithm on other profiles in order to see how it performs. We only focus on traffic generated by Cobalt Strike's malleable profiles; analysis of other C2 architectures, other payloads or other botnet traffic is out of scope. Additionally, we require that the beacon is not actively used to retrieve data during the capturing of NetFlow data.

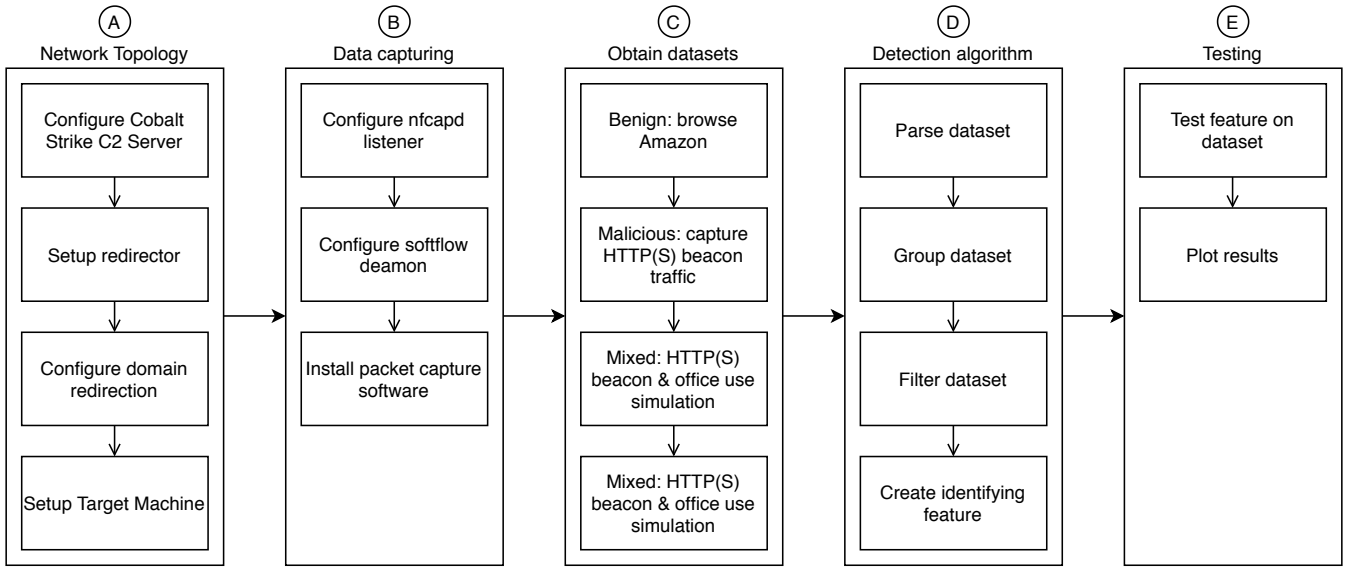

Fig. 1. *The approach of our research illustrated in 5 different steps: A) Creating the Cobalt Strike C2 infrastructure in our controlled environment. B) Setup NetFlow capturing on monitoring interface. C) Generate datasets by running the Cobalt Strike beacon inside the controlled environment. D) Develop detection algorithm, determine feature from the datasets and configure feature thresholds based on the training dataset. E) Test the features on the test dataset using the detection algorithm.*

Although we perform an analysis of the HTTP version of this profile, this research mainly focuses on the HTTPS profile. We expect that Red Teams and threat actors will opt for this option due to its use of Transport Layer Security (TLS), making it harder to get detected. Moreover, the detection algorithm is primarily developed to be able to detect the Transport Control Protocol (TCP) network traffic streams. If the detection algorithm sufficiently performs, it can be extended to also support other protocols (see Section VII-B).

III. Methodology

We develop a detection algorithm and test its capability of successfully alerting on the presence of network communications by a Cobalt Strike C2 beacon/server. We do this by executing the project phases as depicted in Figure 1.

First, we will setup a network topology as described in III-A. This process consists of setting up the Cobalt Strike C2 server, configuring domain redirection, setting up a redirector and creating a target infrastructure. Next, we configure a NetFlow collector and exporter on the virtual network of the target machine in order to capture NetFlow data, which we elaborate on in Section III-B. This setup allows us to generate different types of datasets, consisting of purely benign, as well as mixed NetFlow data. The datasets that we obtain for our experiments are described in detail in Section III-C. Then we construct a detection algorithm that is able to distinguish malicious and benign NetFlow data based on the features that we extract from our training datasets, as explained in Section III-D. Finally, the detection algorithm will be verified against the test datasets to verify its applicability to new data, as elaborated on in III-E.

A. Network topology

We configure a hypervisor host with two Virtual Machines (`VMs`). The first VM is installed with Kali Linux 2020.2 as Operating System (OS). The Kali Linux OS is often used for setting up Red Teaming infrastructures. The Cobalt Strike 4.0 application is installed on this VM as well. The second VM is installed with the Windows 10 OS (Build 1909), which serves as the target machine that we will use to run Cobalt Strike beacon on.

We use VMWare Workstation Pro 15.5.6 as a hypervisor on an Ubuntu 18.04 host. VMWare offers a high level of flexibility regarding remote management, and exposes a Virtual Network Address Translation (NAT) interface to the host operating system; that we can leverage to capture NetFlow data on.

The high level network topology of the infrastructure setup for our testing environment is visualized in Figure 2. This testing environment allows us to simulate a real-world environment as used during adversary simulations, which enables us to obtain realistic network traffic. The following paragraphs will clarify each component of the network infrastructure as depicted in Figure 2.

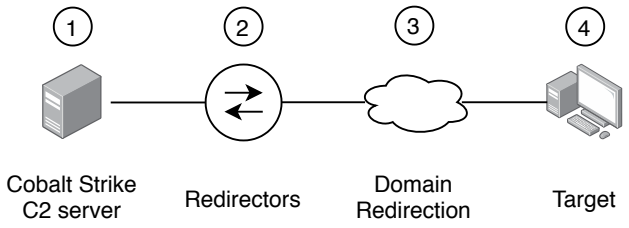

Fig. 2. *The connection process from beacon to server (simplified): 1) The Cobalt Strike C2 server. 2) Local redirectors to relay network traffic to the Cobalt Strike C2 server. 3) The CDN used for domain redirection. 4) The target workstation that is connected with a Virtual Network Address Translation (NAT) network where we attach the Cobalt Strike beacon.*

1) Cobalt Strike C2 server: The Cobalt Strike C2 server (depicted `1` in Figure 2) is the main part of the infrastructure setup, which is used to generate the payloads and to setup the listeners that the beacons on the Target will call back to. The VM running the Cobalt Strike C2 server is configured with its own public IP address. On the Cobalt Strike C2 server, we issued certificates with the `Let's Encrypt certbot`. The use of these certificates avoids HTTPS certificate warnings when one would resolve the redirector IP address that the beacon connects back to. We configure two listeners on the Cobalt Strike C2 server. A listener actively "listens" to incoming requests from beacons and stages (provides) the payload. The first listener is an HTTP listener, and the second one an HTTPS listener. The parameter options for both listeners are included in Table I.

TABLE I
CONFIGURATION PARAMETERS AS BEING USED FOR HTTP AND HTTPS LISTENERS (HTTPS PARAMETERS THAT DIFFER FROM THEIR HTTP EQUIVALENTS, ARE DENOTED BETWEEN PARENTHESES)

Parameter	**HTTP (HTTPS)**
Payload	Beacon HTTP (Beacon HTTPS)
HTTP(S) Host	Redirector URL
HTTP(S) Host (Stager)	Cloudfront domain name
Profile	Default
HTTP(S) Port (C2)	80 (443)
HTTP(S) Port (Bind)	<empty>
HTTP(S) Host Header	<empty>
HTTP(S) Proxy	<empty>

2) Redirectors: In order to prevent the public IP address of our Cobalt Strike C2 server to be disclosed, we configure one or multiple redirector servers (depicted as `2` in Figure 2) that will act as a proxy between the Cobalt Strike C2 server and our target. This adds another layer of complexity to our infrastructure. The utility `socat` allows us to establish a bidirectional byte stream between the redirector(s) and our Cobalt Strike C2 server [26]. By proxying the traffic, we can disclose multiple public IP addresses (one for each redirector), that all point to the same Cobalt Strike C2 server. This way, we can configure our beacon payload in such a way that it calls back to several redirectors in a round-robin fashion. Each of the redirectors are configured with two `socat` instances for both port 443 (HTTPS) and port 80 (HTTP). The redirectors will forward the traffic to the Cobalt Strike C2 server on these respective ports.

3) Domain redirection: We apply a technique called domain redirection (shown as `3` in Figure 2) in order to allow our target machines to connect to trusted external domains of Amazon CloudFront [11]. Although Amazon CloudFront is intended to be used as a CDN, we can leverage it to perform domain redirection. The reason we choose Amazon CloudFront is the fact that it provides fine grained control, and there is no need to change DNS records. This technique introduces four advantages for our Cobalt Strike C2 infrastructure; 1) introducing another layer of complexity that hides the original source IP address, 2) the network traffic blends in with other CDN traffic, 3) CloudFront is often whitelisted by firewalls, 4) we can easily deploy new domains if others are compromised and placed on a blacklist.

4) Target: The target environment (depicted as `4` in Figure 2) is simulated with a virtual Network Address Translation (NAT) network [27]. This type of network is commonly used for both home and corporate networks. The reason for choosing NAT relies on the fact that the target machine maintains connected to the internet, and additionally, a correlation between internal host IP and external IP addresses can be made. Inside the target environment we have a single workstation that is running Windows 10 version 1909.

On the Cobalt Strike C2 server we create two payloads, one for each listener. We opted for the `PowerShell Command` payload type, because it is easy to deploy on the target. We checked (set) the `x64` option, but this is not mandatory for the payload to work. We deliver the payloads by hosting them

as files via integrated Cobalt Strike web server, and downloading them within the target machine. In order to serve both payloads, we used the following Cobalt Strike settings; the `File` points to the locally generated payload, for each payload we specify an URL path at the `Local URI` parameter. Furthermore, we configure the CloudFront domain name as the `Local Host` and the `Local Port` is set to "443" (also when serving the HTTP payload). The `Mime Type` parameter is set to "Automatic" and the `SSL` setting is checked (set).

B. Data capturing

With the systems and configurations in place as described in Section III-A, we provide all the required functionality to operate our Cobalt Strike C2 infrastructure. In order to get insights in the network traffic between the Cobalt Strike beacon and the Cobalt Strike C2 server, we setup several monitoring tools. First, we will configure the `nfdump 1.6.16` utility together with `softflowd 0.9.9` to capture and export NetFlow data on the Virtual NAT interface. We use a sampling rate of 1, because we do not want the flows to be aggregated. We specifically capture network traffic on the Virtual NAT interface, as this interface simulates the edge router of the network. This would also be the location where an administrator of the network captures NetFlow data, as it contains information/statistics about ingress and egress connections.

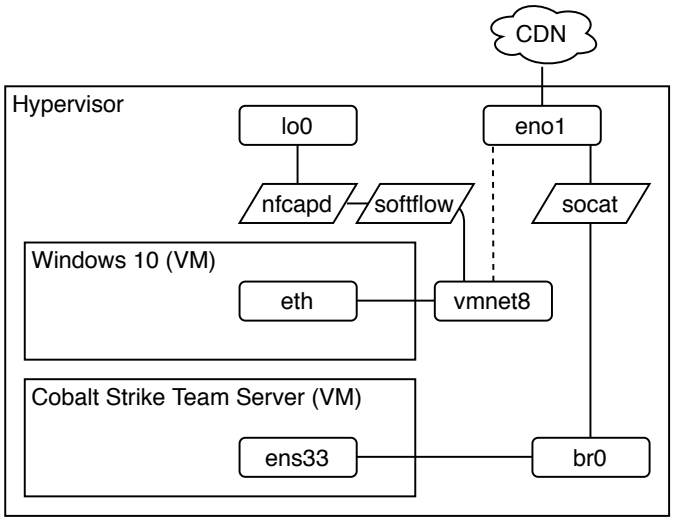

Fig. 3. *Detailed illustration of the hypervisor components and processes. The figure shows the hypervisor with its primary physical interface eno1. Interfaces vmnet8 and br0 are virtual interfaces to connect the guest operating systems to a NAT network and a bridged network respectively. Nfcapd and Softflowd processes run on the hypervisor to directly export NetFlow data from the interface vmnet8. The socat process proxies incoming HTTP(S) traffic to the Cobalt Strike VM.*

The reason for capturing NetFlow data relies on the fact that this provides us with metadata about the network traffic, rather than the payload data. This is especially useful when analysing encrypted traffic such as HTTPS. Additionally, we run `Wireshark 3.2.4` on the target machine to obtain packet captures of the raw network traffic. The internal hypervisor configuration is depicted in Figure 3, and provides a detailed overview of the internals of the hypervisor setup. The hypervisor has one primary interface called `eno1`, which is the interface that is exposed to the internet. The *socat* processes listen on incoming requests on port 80 and 443, and proxies them via `br0` to the IP address of the `ens33` interface of the Cobalt Strike Team Server VM. The Windows 10 VM has an interface, depicted as `eth`. Furthermore, there is a Virtual NAT interface, depicted as `vmnet8`. This interface simulates the NAT router as being used in an office environment, and uses `eno1` to communicate with the internet. Finally, there is a loopback interface called `lo0`. The *nfcapd* daemon – which is part of the nfdump package – is used as a NetFlow collector, and listens on `lo0` for incoming flow data. The *softflow* daemon is configured to tap the `vmnet8` interface and sends the generated flow data to the `lo0` interface over UDP port 2055.

C. Obtain datasets

The approach can be categorized in the following sub-categories; packet based analysis and flow based analysis. As mentioned in Section II-C, we first analyse the Cobalt Strike HTTP beacon in order to get an understanding of the generic behaviour of malleable profiles used by Cobalt Strike. This analysis is performed on the packet captures (PCAPS) of the Cobalt Strike HTTP beacon. As the HTTP protocol doesn't use encryption, everything is sent in plain text. This makes analysis via PCAPS a feasible approach for this protocol.

However, analysis of the Cobalt Strike HTTPS beacon with PCAPS is less feasible due to the encryption being used within HTTPS connections. As such, we analyze the NetFlow data when looking at HTTPS beacon network traffic. The NetFlow data provides us with metadata rather than information about the payload (the actual data) being sent. An example of a NetFlow entry is given in Table II.

We created several datasets in order to develop and test our algorithm. The packet captures are generated on a clean Target machine (Windows 10 Developer Edition, obtained via Microsoft [29]). The NetFlow data is generated on the Virtual NAT interface (`vmnet8`), and

TABLE II
EXAMPLE OF THE HEADER AND ONE ENTRY IN A NETFLOW LOG FILE

Date first seen	Duration	Proto	Src IP Addr:Port	Dst IP Addr:Port	Flags	Tos	Packets	Bytes	pps	bps	Bpp	Flows
2020-07-02 20:39:39.472	0.064	TCP	172.16.22.129:50223	145.100.104.47:443	.APRSF	0	13	1297	203	162125	99	1

collected on the loopback address (`lo0`) on the hypervisor. Before the start of each data capture we revert the Target machine back to its initial snapshot, resulting in a consistent clean state. We divided the datasets into the following categories; Benign and Mixed. For all datasets that we recorded ourselves, we defined the fact that the beacon is active throughout the entire recording as a prerequisite. Another prerequisite is the fact that the beacon is not actively used, e.g. it only calls back in regular time intervals.

1) Benign: The datasets with ID (`3`) and (`4`) are originally a single dataset that we split into both a training and a testing dataset [28]. We use the first 2 million flows in the dataset to train the algorithm, which we label as dataset (`3`). This is shown in Table III. The remainder of the dataset is used for testing, which we label (`4`). Although this dataset is originally a mixed dataset it does not contain a Cobalt Strike beacon, which is the reason that we mark it as benign. This dataset is the only source of NetFlow data that does not contain a Cobalt Strike beacon at all. Our algorithm should therefore not produce any alerts when using this dataset.

2) Mixed: We created several mixed datasets. First of all, we created two training data sets with ID (`1`) and (`2`). These datasets contain active Cobalt Strike beacons that periodically call back to the Cobalt Strike C2 server; (`1`) over HTTPS and (`2`) over both HTTP and HTTPS. Although these datasets mainly contain malicious data, the fact that background traffic was included could not be avoided. After the initial executable (stager) is executed on the target machine, the target machine connects back to the C2 Server to download and execute the beacon payload, also known as `the stage`. During the initial callback, the connection to the Cobalt Strike C2 Server is established. When the beacon becomes active, the infected target machine calls back to the Cobalt Strike C2 Server periodically, based on the interval that is defined in the malleable profile. It is not mandatory to capture the initial callback; the (periodic) callback data is the data we are interested in for this research. The reason for creating this malicious dataset is mainly to define and train our algorithm. The final detection algorithm is tested based on other mixed datasets, which is further explained in the next paragraph.

Secondly, we created a mixed dataset that contains realistic office use. This dataset is labeled with ID (`5`), and is used for testing the detection algorithm. This dataset is generated by capturing network traffic at the moment the HTTPS beacon is active. During this time, normal workstation activities (web browsing, watching videos, e-mail, fetching and installing updates, etc.) will be simulated as well. This results in the beacon traffic being blended in with the benign traffic. A complete overview of applications that are used to generate benign network traffic is provided in Appendix A.

Furthermore, the remainder of the datasets all contain mixed network traffic, with at least one Cobalt Strike beacon present in the dataset. The Cobalt Strike beacons have a destination IP address of

TABLE III
DESCRIPTION OF DATASETS AND NETFLOW INFORMATION USED FOR TRAINING AND TESTING THE DETECTION ALGORITHM

ID	Function	Content	Description	Flows	Unique	Beacons
1	Train	Mixed	Amazon profile, HTTPS, 5 minute interval, 60% jitter	1047	481	4
2	Train	Mixed	Amazon profile, HTTP and HTTPS, 1 minute interval, 99% jitter	1652	784	4
3	Train	Benign	CTU-Malware-Capture-Botnet-43 [28]	2000000	55101	0
4	Test	Benign	CTU-Malware-Capture-Botnet-43 [28]	4351187	120778	0
5	Test	Mixed	Default Amazon profile, HTTPS beacon	338	176	1
6	Test	Mixed	Amazon profile, HTTPS beacon, 99% jitter	476	242	1
7	Test	Mixed	Amazon profile, HTTP and HTTPS beacons, 2 redirectors	862	448	4
8	Test	Mixed	Amazon profile, HTTP and HTTPS beacons, 2 redirectors, 5 minute interval	2122	941	4
9	Test	Mixed	Default Amazon profile, HTTPS, mixed with self-generated benign network traffic.	8869	4508	1
10	Test	Mixed	Default Gmail profile, HTTP and HTTPS [10]	1952	996	2
11	Test	Mixed	Default Pitty Tiger profile, HTTP and HTTPS [10]	1249	645	2
12	Test	Mixed	Default Cobalt Strike, HTTP and HTTPS, configuration without malleable profile	393	199	2

either `145.100.104.174`, `145.100.104.47`, or `136.144.149.0`. The last octet of the last IP address is removed in order to anonymize the original IP address. Table III provides an overview of all the datasets that we used for our experiments, along with a description of the contents of the datasets. The description gives an indication of the malleable profiles that are present in each dataset, along with profile specific configurations of the beacon. The column **Flows** contains the total amount of NetFlow records in the dataset. The column **Unique** contains the total amount of flows from a given source IP address to a destination socket, creating a 4-tuple (source IP, destination IP, destination port, protocol). We will refer to this 4-tuple as a *flow collection*. These flow collections are later used by the detection algorithm in order to identify a set of flows that are related to each other as either malicious or benign. The column **Beacons** shows the amount of Cobalt Strike beacons that can be observed in the dataset.

D. Detection algorithm

The algorithm that we develop in order to detect network traffic between the Cobalt Strike beacon and the Cobalt Strike C2 server is depicted in Figure 4. The reason for choosing for a static algorithm rather than using a machine learning approach, relies on the fact that we would not be able to address problems to the algorithm in case it would not perform as desired. With this approach we have a greater control over the specific parameters for the detection algorithm [30].

Before processing the data, we export the raw NetFlow data to ASCII files with the `nfcapd` tool. These ASCII files contain the following flow data: date first seen, duration, protocol, source IP, source port, destination IP, destination port, flow size in bytes, bits per packet, and all TCP flags, amongst others. The algorithm first parses the ASCII files (**1** in Figure 4).

The dataset is grouped in a `host table`, using the source IP address as the key element. The source IP address of a flow is added to the host table if it does not exist yet (**2** in Figure 4), and we attach the flow information to the host object (**3** in Figure 4). If the source IP already exists in the host table, we only append the flow to the host object (**3** in Figure 4). This data structure allows us to programmatically search, filter, and apply statistical operations on a large set of hosts that have one or more flows originating from the IP address of the host object.

The current features implemented in the model allow to iterate through all known hosts or to only filter on hosts with a private IP address. It is also possible to select just a single host in order to run the detection algorithm against a specific target that requires further analysis.

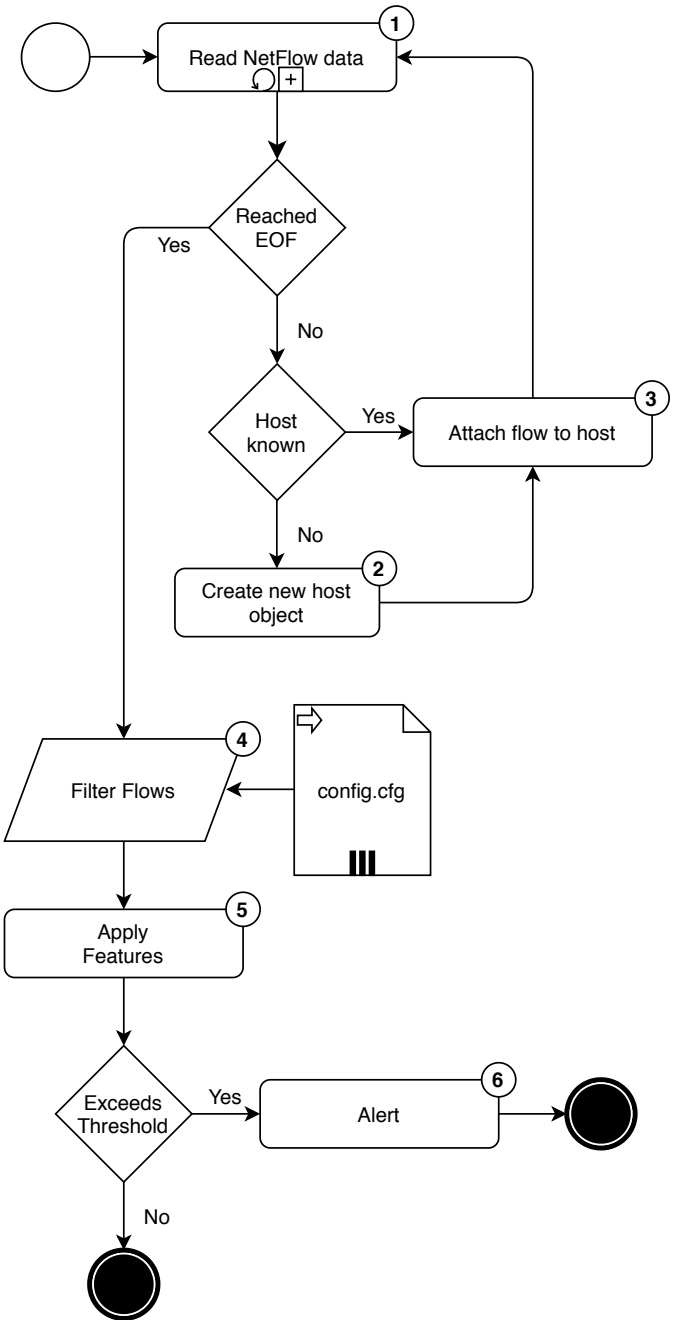

Fig. 4. *A flow chart illustrating the main steps of the detection algorithm. 1) The NetFlow data is parsed. 2) A host database is maintained. 3) Flows are appended to host objects. 4) Flows are filtered. 5) Features are applied. 6) Alerts are given.*

If no options are specified, the detection algorithm iterates over all known host objects and groups unique flows based on the 4-tuple: `(source IP, destination IP, destination port, protocol)`, which we will call a *flow collection*. This

means that such a flow collection will contain one or more flows originating from a host to a remote socket. We then apply a basic filter where all flow collections are discarded if `n < 10`, where `n` represents the number of flows for a specific host (**4** in Figure 4). Secondly we apply a filter that discards the flow collection if the absolute time Δ`t < 30`, where `t` represents the total duration of all flows for that host (**4** in Figure 4). This ensures we have sufficient data points to work with. Besides, we assume that a Cobalt Strike beacon is unlikely to have a lifespan of less than 30 seconds, or less than 10 callbacks to the Cobalt Strike C2 server.

After the preliminary filtering, the detection algorithm iterates over the remaining flow collections, and determines the standard deviation of the byte size of the flow collection. Additionally, for each host object, we count the amount of flows with the `ACK`, `PSH`, `SYN` and `FIN` (abbreviated `APSF`) TCP flags set. Then the agorithm checks if the thresholds are reached for the standard deviation of the byte size ($>$ 100 bytes) and the percentage of flows with the `APSF` flags set (`0.95`). If this is the case, the algorithm calculates the correlation between the absolute time *t* and the mean of the start and end time for each of the flows. This correlation indicates if there is a consistent time interval between each separate flow in the collection, and takes the duration of flows into account. The correlation is determined using the `scipy.stats.linregress` package (**5** in Figure 4), and provides us the `r-value`, which indicates the correlation of the data (with 0 meaning no correlation and 1 meaning totally correlated). If the `r-value` drops below a the defined threshold of `0.98`, this means that the NetFlow data has a weak correlation and is unlikely to contain beacon traffic. No alert is generated and the algorithm continues with the next host object. On the other hand, if the `r-value` exceeds the threshold, the detection algorithm generates an alert (**6** in Figure 4). This alert gives an indication that the specific host where the flow originated from potentially contains a Cobalt Strike beacon. The specific threshold values for the standard deviation of the byte size, amount of flows with `APSF` flags, and the `r-value` are configured based on running the algorithm against the test dataset where the optimal results are achieved. The source code of the detection algorithm is available on the internal GitLab server of OS3[1].

E. Testing

The first step in testing the capabilities of our algorithm consists of analysing the predictions on the training datasets with IDs (`1-3`) from Table III. We configure the thresholds for each of the parameters based on the results we gather from the training dataset.

Then, we test our algorithm on completely unknown datasets, indicated with the IDs `(4-12)` in Table III. Our goal is to test (verify) the algorithm on a dataset of approximately 4.3M flows consisting of benign traffic with ID `(4)`, five variations of the *Amazon profile* with IDs `(5-9)`, and two other profiles such as the *Gmail profile* with ID `(10)` and the *default Cobalt Strike profile* with ID `(12)`. The *default Cobalt Strike profile* is the standard Cobalt Strike profile that does not use a malleable profile. Additionally, we test a different kind of profile, mimicking a realistic threat with the *Pitty Tiger* profile with ID `(11)`.

The algorithm plots the correlation for hosts that have a set of flows that show a high correlation between absolute time and mean of start end end time, expressed as the `r-value`). Example figures are included in the results of the algorithm, which are included in Section IV.

IV. Results

For this research we analysed the main characteristics of Cobalt Strike's beacon traffic for both regular HTTP and HTTPS network connections made by the Cobalt Strike beacon. In Section IV-A we present the features that identify a Cobalt Strike beacon, together with the threshold values that we specified to enable to detection

[1]https://gitlab.os3.nl/veijk/nfanalyzer

TABLE IV
Overview of features including description and threshold values

Feature	**Description**	**Threshold**
Time interval	Time between two consecutive flows	0.98
Flow duration	Mean of start and end time of a flow	0.98
Byte size of flow	Total bytes sent within a flow	stddev $<$100 bytes
Amount of APSF flows	Amount of flows with APSF flags set	0.95
Amount of RST flows	Amount of flows with RST flags set	N/A
Amount of DNS Requests	Ratio of DNS requests as part of total flows	N/A

algorithm to filter on these features. Next, we illustrate the results of the detection algorithm based on figures that are generated by the detection algorithm itself in Section IV-B. These figures provide us with additional understanding of the patterns and behaviour of the NetFlow data that we analyse. Finally, we present the results of the detection algorithm against our test dataset in a confusion matrix in Section IV-C.

A. Features

After analysing the packets and flows we were able to identify six features that show characteristics specific to a Cobalt Strike beacon. Table IV provides an overview of these six identified features, along with the threshold values that are used by the detection algorithm, which we will elaborate on in this section.

First, we noticed a recurring and consistent time interval between each callback of the beacon to the Cobalt Strike C2 server. Second, we observed that the flow duration of a beacon callback is very short; in the order of tenths of seconds. These two observations were combined in order to measure the correlation between the flow occurrence and the absolute time. For the correlation coefficient (`r-value`) between absolute time t and the occurrence of flows we defined the optimal threshold, to be `0.98`. This value allows us to discard false positive results that show a weak linear correlation, while selecting the flows that have a high linear correlation – a characteristic typical for beaconing behaviour.

Moreover, we observed that the byte sizes of flows originating from the Cobalt Strike beacon show similarities for each flow. Because of the similar pattern for each beacon callback, we observe that the byte sizes for the flows show a narrow distribution. We are able to identify beaconing behaviour by calculating the standard deviation of the flows in a flow collection for a specific host. The standard deviation indicates the spread of the byte sizes of the flows. During calibration of the detection algorithm (based on our training dataset), we empirically discovered that the byte sizes of Cobalt Strike beaconing flows always have a standard deviation of less than 100 bytes. Therefore we use this value as the threshold (upper bound).

Another feature that we observed for the Cobalt Strike beacon is the high amount of `RST` flags being set for the HTTPS beacon, and – in general – the combination of `APSF` consistently occurring in every NetFlow record for the Cobalt Strike beacon communication for both the HTTP and HTTPS beacons. After analysing the results for our algorithm based on the training dataset, we found that flow collections originating from a Cobalt Strike beacon show at least 95% of the flows have the `APS` and `Fin` flag set. We used the threshold of `0.95` to distinguish a set of flows containing beacon traffic from a set of flows that does not contain this behaviour.

Lastly, we observed a difference in the amount of DNS queries being made. Benign traffic to the Amazon domain shows DNS requests in regular intervals; mostly in order to resolve third party website components. On the contrary, there are no DNS requests made by the Cobalt Strike beacon that mimics HTTP or HTTPS network traffic using the *Amazon profile*, other than the initial request to resolve the beacon URI.

B. Detection results

The detection algorithm described in Section III-D allows us to calculate the probability of a network stream being Cobalt Strike beacon traffic.

Analysis of the Cobalt Strike beacon traffic for both HTTP and HTTPS shows us a linear pattern when plotting the mean of start and end time for each flow in a series of NetFlow data flows. Figure 5 shows the mean of start and end time for each flow, sent from the Target machine (containing a Cobalt Strike beacon) to the Cobalt Strike C2 server. This figure clearly illustrates a strong correlation between the absolute time and the occurrence of the flows that we can observe in the beaconing behaviour of network traffic. The `r-value` in this case is `0.999`.

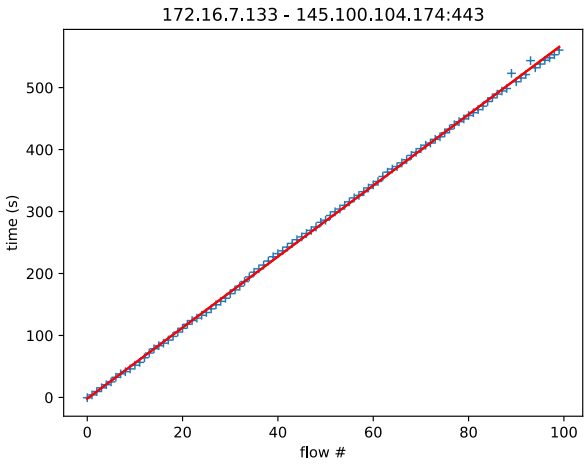

Fig. 5. *The mean of start and end time for each flow between the target (172.16.7.133) and the C2 server (145.100.104.174). The detection algorithm will alert on this flow as it has a low deviation from the linear regression (r=0.999, n=100).*

On the other hand, when our detection algorithm is used against the sets of flows of regular HTTPS

connections with the same analysis method, no alert is given. When we plot the flows of a regular HTTPS connection, we can observe that there is a weak correlation between the absolute time and the occurrence of the flows for the network connections to a specific host, as shown in Figure 6. The `r-value` that was measured for this connection is `0.854`, and is lower than the defined threshold of `0.98`; so no alert will be given for this connection. Based on this principle the detection algorithm will iterate over each flow collection to label it as either benign or malicious and send an alert accordingly.

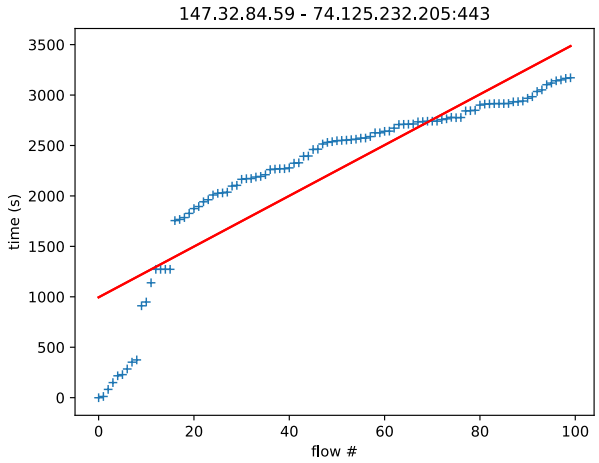

Fig. 6. *The mean of start and end time for each flow of a regular HTTPS connection between the target (147.32.84.59) and a remote server (74.125.232.205). The detection algorithm will not alert on this flow as it deviates too much from the linear regression. (r=0.891, n=100)*

Figure 7 shows the results for the *Amazon profile* with an introduced jitter of 99%, which is the maximum amount. Even with the highest amount of jitter introduced to the Cobalt Strike beacon, we are still able to observe a strong linear correlation.

Figure 8 shows the relationship between the amount of jitter we introduce using the malleable profile and the linear correlation of the NetFlow data. We observe a minimal decrease in the linear correlation of the data, even if we increase the amount of jitter to 99%, which is the maximum value allowed by Cobalt Strike. The time interval of 5 seconds shown in Figure 8 is the default value for the *Amazon profile*, which we sampled for for a duration of 10 minutes. Note that the Y-axis of the figure starts at 98%, which is also the threshold that we defined for the linear correlation as described in Section IV-A.

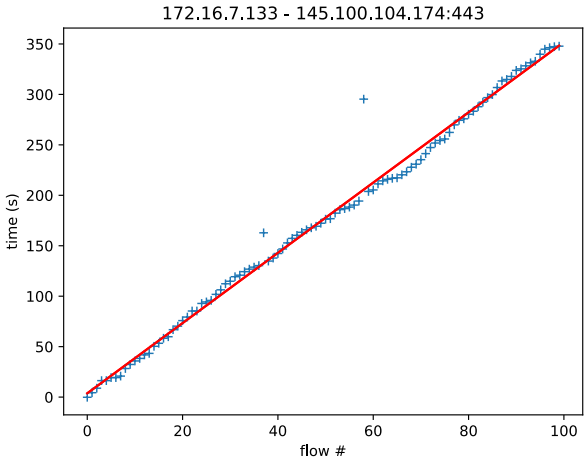

Fig. 7. *Mean of start and end time of each flow for flows between the target (172.16.7.133) and the C2 server (145.100.104.147) with an introduction of 99% jitter to the callback interval (the maximum amount). The detection algorithm will alert on this flow as it has a low deviation from the linear regression (r=0.994, n=100).*

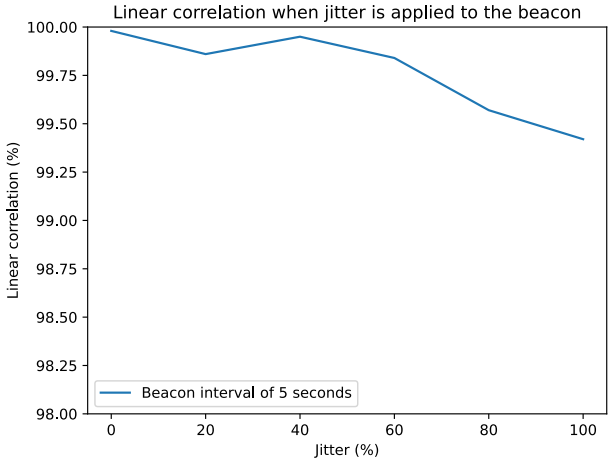

Fig. 8. *The percentage of linear correlation for different percentages of jitter applied to the Cobalt Strike beacon using the Amazon profile (n=100). The threshold for the detection algorithm is set at 98%. Despite introducing high amounts of jitter, the correlation remains strong.*

C. Confusion matrix

The detection algorithm is tested against the datasets as listed in Table III, which combines 9 different datasets that are used for testing the detection algorithm. The datasets are recorded in different conditions and contain different malleable profile to test the detection algorithm against. The results of the detection algorithm after it was tested against the different datasets is provided in Table V. These results show how the algorithm is able

to detect 15 out of 17 Cobalt Strike beacons that are included in our test datasets.

TABLE V
OVERVIEW OF AGGREGATED NETFLOW DATA THAT THE DETECTION ALGORITHM WAS ABLE TO IDENTIFY CORRECTLY AS EITHER MALICIOUS OR BENIGN

		Actual	
		Good	Bad
Predicted	Good	128910	2
	Bad	5	15

Table V displays the predicted values of the detection algorithm against the actual values of the accumulated NetFlow data as it is present in the datasets. If the actual NetFlow data is benign (good) and the predicted value by the algorithm is also benign, the result is marked as a True Negative (TN). This indicates we are able to correctly identify non-malicious traffic as such. When the detection algorithm wrongly identifies actual benign data this results in a False Positive (FP). Table V shows that the detection algorithm incorrectly identifies 5 different flow collections, resulting in 5 False Positives. If the actual NetFlow data originates from a Cobalt Strike beacon, this is labeled as malicious (bad). When the detection algorithm identifies the malicious NetFlow data correctly, this results in a True Positive (TP). If the actual malicious data is incorrectly identified as benign, this results in a False Negative (FN).

To calculate the performance of our algorithm we aggregated the results for all test datasets with ID (`4-12`). Table VI provides an overview of the results for each separate dataset used for testing the algorithm. We then calculate the accuracy *ACC* of our detection method using the following formula:

$$ACC = \frac{TP+TN}{TP+TN+FP+FN} = \frac{15+128911}{15+128911+5+2} = 99,995\%$$

TABLE VI
RESULTS OF THE DETECTION ALGORITHM FOR EACH DATASET

ID	Unique	Beacons	FN	FP	TP	TN
4	120778	0	0	3	0	120775
5	176	1	0	0	1	175
6	242	1	0	0	1	241
7	448	4	0	0	4	444
8	941	4	2	2	2	935
9	4508	1	0	0	1	4507
10	996	2	0	0	2	994
11	645	2	0	0	2	643
12	199	2	0	0	2	197

V. DISCUSSION

As can be seen in the matrix as provided in Section IV-C, the performance of our algorithm is very high with an achieved accuracy of 99,995%. However, we have to make some remarks regarding these results as the accuracy might provide a distorted view due to the large unevenness in the distribution among actual benign and malicious samples in our testing dataset. The accuracy of 99,995% is primarily caused by the high amount of True Negatives that we are able to identify correctly. The 2 False Negatives can be clarified in the following manner; the redirector disconnected during the traffic capture, causing the behaviour of the beacon to change. We observed the beacon only sending `SYN` flags from that moment onwards. It is harder to clarify the 5 False Negatives; these streams show a similar behaviour to the beacon traffic. One of those 5 False Negatives shows such a high level of similarity with recorded beacon traffic, that we investigated the destination IP further. Unfortunately we can not determine with certainty whether this was an actual False Negative; this may be due to a mislabeled benign flow in our dataset.

It should be noted that the two beacons that were not detected partially contained invalid data because the redirector failed during the capture of NetFlow data for the dataset with ID (`8`). Although this dataset is partially invalid as it does not meet the preconditions as specified in Section III-C we still believe the results are valuable to our research as it indicates certain conditions in which the detection algorithm is unable to identify a Cobalt Strike beacon.

We encountered difficulties while obtaining and generating datasets during this research. First of all, there are no datasets widely available to our knowledge that contain truly benign and/or malicious data which is accurately described and labeled. The datasets with ID (`3`) and (`4`) that we used in our experiments are labeled, but only on a per-host basis and did not contain any Cobalt Strike beacons. Secondly, regarding the external datasets; we cannot be completely sure that flows labeled as benign are actually benign as the creator of the dataset could be unaware of any malware infections in the network. Lastly, to our knowledge there is no dataset available that contains labeled NetFlow data of a Cobalt Strike beacon. As such, we are not sure whether the malicious Cobalt Strike datasets that we created are representative for all Cobalt Strike beacon traffic. However, we followed the Cobalt Strike documentation and its best practices while setting up the infrastructure, which is why we make the assumption that our infrastructure and Cobalt Strike beacons are representative for this research.

The biases we encountered and their mitigation are included in the list below:

- `Selection bias`: we tried to avoid selecting a dataset that is suited towards our algorithm by means of including external datasets, representing a realistic environment. However, we are aware of the fact that we tested our detection algorithm only on Cobalt Strike beacon samples that were generated within our own infrastructure.
- `Confirmation bias`: We tend to look for information that confirms our perceptions. When encountering TCP RST flags for HTTPS beacons, we researched whether this was unique to our environment. However, the behaviour could be explained by an expert in the field and therewith confirmed our observations.
- `Clustering illusion`: We tend to see patterns in random data. We observed a seemingly linear correlation between the absolute time and the mean of start and end time of flows. After further analysis of the correlation - in our case linear regression - we found out that this correlation is descriptive/unique enough for the behaviour we want to analyse.
- `Choice supportive bias`: Choosing for a certain solution makes you feel confident in that specific solution even if it has flaws. We encountered this while testing our detection algorithm when using the timing interval as the only feature. After testing the algorithm on another dataset we found our solution was not that promising as it seemed. However, combining all different features that we identified solved this problem.

In regard to the parameters we used in the preliminary filtering, this heavily depends on the provided dataset. We found out that a minimum of 10 flows (similar to [24]) and a minimum total flow duration of 300 seconds were necessary to provide enough sample points to measure correlation.

Furthermore, a correlation coefficient of `0.98` or higher was used to detect Cobalt Strike beacons. This parameter seems to be uniform Cobalt Strike beacon traffic; due to testing the detection algorithm on external datasets, other profiles and different jitter values. Even when introducing high amounts of jitter with a beacon interval of 5 seconds, we observed a strong correlation. We expect that these results are the same for beacons with a longer interval, as long as sufficient data points are available (e.g. the duration of the capture needs to be longer).

We can reason about the RST behaviour of the HTTPS beacon as well. When we dived deeper in the packet captures, we found out that the sequence numbers for the stream only changed once (after the initial callback). These repetitive sequence numbers for consecutive flows could give an indication of the fact that the network socket on the server side is reused. If the beacon uses a new handle for each callback on the same socket, this causes the server to drop the previous handle and reset the connection.

VI. Conclusion

Based on the results provided in Section IV we are able to answer our research question. However, we would like to address our secondary research questions first.

Which features can we extract from network traffic generated by Cobalt Strike beacons? We discovered six features of Cobalt Strike's C2 network traffic, including; time intervals between C2 server communication, the specific TCP flags `Ack`, `Psh`, `Syn`, and `Fin` that are sent by the client, the consistent byte sizes of flows, flows with a short duration in the order of tenths of seconds, the lack of DNS requests for HTTP(S) beacon network traffic, and the TCP RST flags sent by the beacon.

Can we detect a Cobalt Strike beacon using a malleable profile with one or more of those features?

When combining the four features: timing intervals of Cobalt Strike C2 network traffic, flow duration, the specific TCP flags sent by the client, and the consistent byte size of flows we are able to detect Cobalt Strike beacons with a high accuracy.

Based on the results provided in Section IV we are able to answer our main research question: How can we distinguish obfuscated Cobalt Strike payloads from genuine network traffic based on identifying features?

The features that we identified in the network communication of a Cobalt Strike C2 server allow us to extract and identify potentially malicious network communications. We are able to achieve this due to the high amount of linearity that is present in the behaviour of an infected machine, even if techniques are applied to reduce the linear behaviour by introducing jitter in the callback period or by load-balancing the network traffic to the Cobalt Strike C2 server over multiple hosts. Because this linear behaviour is unusual for regular network traffic we are able to identify malicious C2 network traffic with a high accuracy.

Herewith we are also able to prove our hypotheses; `H1` Identifying features such as timing intervals and message length can be used to identify malicious network traffic, and `H2`: protocol flow information (metadata) provides valuable features for measurement. Both timing intervals and duration (`H1`), as well as protocol flow information (byte sizes, flags set, byte length of flow).

VII. Future Work

Within this section we provide some insights in future works. In Section VII-A we give advise regarding testing other profiles. In Section VII-B we elaborate on the other protocols that could be researched. Section VII-C proposes an upgrade to the current detection algorithm regarding programmability, and Section VII-D shows possible optimizations for expanding the algorithm to perform real-time alerting.

A. Profile expansion

At the moment we constructed a detection algorithm that is able to detect beacon traffic, as generated with the *Amazon profile*. Future research could focus on expanding the detection algorithm for use with other profiles; both the normal profiles (mimicking legitimate internet services) as well as threat profiles (mimicking threats being used in the wild).

B. Protocol expansion

Similar to the expansion of the profiles the detection algorithm is able to detect, future work could focus on being able to detect other protocols being used for the Cobalt Strike C2 channel. The current algorithm is able to detect TCP beacons, however, since Cobalt Strike also supports other listener protocols (e.g. SMB, UDP) it is interesting to see if those protocols generate similar network traffic, and whether and how the detection algorithm would be able to detect these kind of protocols.

C. Modularity

As an extension to the points mentioned in Section VII-A and Section VII-B, it is feasible to introduce additional modularity and programmability to the detection algorithm. For example, to give the option to provide a configuration file with certain parameters set per profile and protocol to detect, as these may vary per protocol. These parameters could include the r-value, p-value and range for the standard error (min-max). Testing ideal parameters could be a research on its own.

D. (Near) real-time detection

The current detection algorithm allows us to alert on potential Cobalt Strike beacon traffic based on historical NetFlow data. This means the algorithm is limited to detect the presence of a Cobalt Strike beacon server in a reactive manner, without the possibility of real-time detection.

E. Researching other use-cases

It would be interesting to further research different beacon behaviour. For example, our detection algorithm could be tested against a Cobalt Strike beacon that is actively used. Additionally, it would be interesting to see how the algorithm performs when a Cobalt Strike operators selects a callback time in the order of hours for a beacon that otherwise calls back every couple of seconds. Moreover, it would be interesting to know if the algorithm performs well if a Cobalt Strike operator changes the sleep times often; introducing fluctuations in callback – independent from the jitter parameter.

F. Researching Machine learning approaches

In this research we show that our algorithm is able to achieve a high accuracy of 99.998% in regards to the datasets being used. It would be interesting to see how the results would differ if a machine learning approach is taken. Another promising approach for detecting anomalies in time series is by using a tool called `stumpy`; which works by creating matrices of time stamped data and calculated the Euclidean distance over sub-sequences with the data using a sliding window [31].

References

[1] Keman Huang, Michael Siegel and Stuart Madnick. *Cybercrime-as-a-service: identifying control points to disrupt*. 2017.

[2] Wentao Chang et al. "Characterizing botnets-as-a-service". In: *Proceedings of the 2014 ACM conference on SIGCOMM*. 2014, pp. 585–586.

[3] Josh Abraham. *Cobalt Strike, Software S0154*. 6 June 2019. URL: https://attack.mitre.org/software/S0154/ (visited on 05/06/2020).

[4] Adel Alshamrani et al. "A survey on advanced persistent threats: Techniques, solutions, challenges, and research opportunities". In: *IEEE Communications Surveys & Tutorials* 21.2 (2019), pp. 1851–1877.

[5] Bradley J Wood and Ruth A Duggan. "Red teaming of advanced information assurance concepts". In: *Proceedings DARPA Information Survivability Conference and Exposition. DISCEX'00*. Vol. 2. IEEE. 2000, pp. 112–118.

[6] Eric M Hutchins, Michael J Cloppert, Rohan M Amin et al. "Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains". In: *Leading Issues in Information Warfare & Security Research* 1.1 (2011), p. 80.

[7] Spamhaus. *Botnet Threat Report 2019*. 1 Jan. 2020. URL: https://www.spamhaus.org/news/images/full-2019/spamhaus-botnet-threat-report-2019.pdf (visited on 02/05/2020).

[8] Raphael Mudge. *Adversary Simulation and Red Team Operations Software*. 2 May 2020. URL: https://www.cobaltstrike.com (visited on 02/05/2020).

[9] Raphael Mudge. *Malleable Command and Control Language*. URL: https://www.cobaltstrike.com/help-malleable-c2 (visited on 02/05/2020).

[10] Raphael Mudge. *rsmudge/Malleable-C2-Profiles*. 7 Apr. 2018. URL: https://github.com/rsmudge/Malleable-C2-Profiles/tree/master/ (visited on 02/05/2020).

[11] amazon.com. *Amazon Cloudfront*. URL: https://aws.amazon.com/cloudfront/.

[12] Ed. B. Claise. *Cisco Systems NetFlow Services Export Version 9*. 2004.

[13] Christian J Dietrich, Christian Rossow and Norbert Pohlmann. "CoCoSpot: Clustering and recognizing botnet command and control channels using traffic analysis". In: *Computer Networks* 57.2 (2013), pp. 475–486.

[14] Liang Wang et al. "Seeing through network-protocol obfuscation". In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 2015, pp. 57–69.

[15] Satoshi Kondo and Naoshi Sato. "Botnet traffic detection techniques by C&C session classification using SVM". In: *International Workshop on Security*. Springer. 2007, pp. 91–104.

[16] Chen Lu and Richard Brooks. "Botnet traffic detection using hidden markov models". In: *Proceedings of the Seventh Annual Workshop on Cyber Security and Information Intelligence Research*. 2011, pp. 1–1.

[17] Sajad Homayoun et al. "BoTShark: A deep learning approach for botnet traffic detection". In: *Cyber Threat Intelligence*. Springer, 2018, pp. 137–153.

[18] Kevin P. Dyer, Scott E. Coull and Thomas Shrimpton. "Marionette: A Programmable Network Traffic Obfuscation System". In: *24th USENIX Security Symposium (USENIX Security 15)*. Washington, D.C.: USENIX Association, Aug. 2015, pp. 367–382. ISBN: 978-1-939133-11-3. URL: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/dyer.

[19] Lucas Dixon, Thomas Ristenpart and Thomas Shrimpton. "Network traffic obfuscation and automated internet censorship". In: *IEEE Security & Privacy* 14.6 (2016), pp. 43–53.

[20] Paul Prasse et al. "Malware detection by HTTPS traffic analysis". In: (2017).

[21] Lee Brotherston. *TLS fingerprinting*. 25 Sept. 2015. URL: https://blog.squarelemon.com/tls-fingerprinting/ (visited on 25/06/2020).

[22] Maria Rigaki and Sebastian Garcia. "Bringing a gan to a knife-fight: Adapting malware communication to avoid detection". In: *2018 IEEE Security and Privacy Workshops (SPW)*. IEEE. 2018, pp. 70–75.

[23] Leendert van Duijn. *Beacon detection in PCAP files*. 2014.

[24] Bartosz Jerzman. *Hunting beacons*. URL: https://www.x33fcon.com/archive/2019/slides/x33fcon19_Hunting_Beacons_Bartek.pdf (visited on 04/07/2020).

[25] Joey Dreijer. *StealthWare - Social Engineering Malware*. 2015.

[26] Raphael Mudge. *Cloud-based Redirectors for Distributed Hacking*. 14 Jan. 2014. URL: https://blog.cobaltstrike.com/2014/01/14/cloud-based-redirectors-for-distributed-hacking/ (visited on 27/06/2020).

[27] Pyda Srisuresh and Matt Holdrege. *IP network address translator (NAT) terminology and considerations*. 1999.

[28] Stratosphere Lab. *The CTU-13 Dataset. A Labeled Dataset with Botnet, Normal and Background traffic.* URL: https://www.stratosphereips.org/datasets-ctu13 (visited on 04/07/2020).

[29] Microsoft. *Een virtuele machine voor Windows verkrijgen*. URL: https://developer.microsoft.com/nl-nl/windows/downloads/virtual-machines/ (visited on 05/06/2020).

[30] Jasper Snoek, Hugo Larochelle and Ryan P Adams. "Practical Bayesian Optimization of Machine Learning Algorithms". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 2951–2959. URL: http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms.pdf.

[31] TD Ameritrade IP Company, Inc. *The Matrix Profile*. URL: https://stumpy.readthedocs.io/en/latest/Tutorial_The_Matrix_Profile.html (visited on 05/07/2020).

APPENDIX A
BENIGN DATASET GENERATION

The following list shows the programs executed in order to create the mixed dataset (in chronological order).

```
0. (Beacon active)
1. Start Firefox
2. Play Youtube video
3. Browse the web
  3.1 https://www.tweakers.net
  3.2 https://www.reddit.com
  3.3 https://www.github.com
4. Download Chrome
  4.1 Install Chrome
5. Update Windows
  5.1 Feature Update version 2004
6. Open outlook
  6.1 Register email
  6.2 Send email
  6.3 Receive email
```