Lifting the Smokescreen: Detecting Underlying Anomalies During a DDoS Attack

Brian Ricks
Computer Science Department
University of Texas at Dallas
Richardson, USA
Email: bwr031000@utdallas.edu

Bhavani Thuraisingham
Computer Science Department
University of Texas at Dallas
Richardson, USA
Email: bxt043000@utdallas.edu

Patrick Tague
Electrical & Computer Engineering
Carnegie Mellon University
Moffett Field, USA
Email: tague@cmu.edu

***Abstract*—While DDoS attacks have become an ever-growing threat in the last decade, a new variation is taking root in which the DDoS is used as a distraction or smokescreen to hide other malicious activity. This variation, which we call DDoS as a Smokescreen (DaaSS), often results in data theft and financial loss. Furthermore, DaaSS attacks are often only detected because the theft or other malicious activity is discovered independently, long after the DDoS has ceased. In this work, we set out to describe these attacks and introduce a novel approach to detect them using real-world network trace data. We present experimental results showing promise that DaaSS attacks can be detected in a manner conducive to practical deployment.**

***Index Terms*—DDoS, distributed denial-of-service, smokescreen, DaaSS, eMews, anomaly detection, intrusion detection**

I. Introduction

Distributed Denial-of-Service (DDoS) attacks have been a mainstay of the Internet for well over two decades. These disruptive attacks utilize many networked computers, often hijacked to form centrally-controlled botnets, for the purpose of flooding a target computer or network with packet data. Successful attacks may cripple the target from interacting with normal users such as online customers, and cause millions in lost revenue per year.

More recently, DDoS has evolved from a primary mode of attack to something which is *supposed* to look like a primary mode of attack. In this scenario, the goal of the attackers is to utilize the DDoS to provide a smokescreen for the execution of some *other* attack. This new intent of launching a DDoS, which we call *DDoS as a Smokescreen (DaaSS)*, is often employed to steal data, finances, or perform other stealthy activities. The underlying attack which accompanies a DaaSS could, for example, be logins to a user account, withdrawal of funds from an ATM, or an intrusion which exploits a network vulnerability.

One of the earliest documented examples of a DaaSS attack occurred in 2011 on the Sony PlayStation network [1]. The DaaSS attack started with a massive DDoS that lasted several days, followed by an intrusion that resulted in the exfiltration of personal data from 77 million Sony PlayStation customers. The intrusion occurred while Sony was still taking action against the DDoS, and Sony Computer Entertainment America's head at the time stated in a subsequent letter to Congress that the DDoS may have hindered earlier detection of the intrusion.

Since the Sony PlayStation network attack, many other DaaSS attacks have been observed. In 2015, Carphone Warehouse, a UK-based mobile phone retailer, was hit with a large DDoS attack that was later discovered to have served as a smokescreen for the theft of personal and banking details of 2.4 million customers [2].

Traditionally, network intrusion detection systems (IDS) are employed to detect DDoS [3] and perhaps other concurrent attacks. IDS often utilize models trained from data, and the data itself often has a significant class imbalance favoring normal behavior over threats. Misuse-based IDS [4] overcome this issue by using labeled training data when learning a model, enabling detection of previously observed (known) attacks with high accuracy. Unfortunately, they tend to perform poorly when given a previously unobserved (unknown) threat to detect, and thus may not be suitable for DaaSS attacks. Anomaly-based IDS [5]–[10], on the other hand, use *unlabeled* data when learning a model. These approaches typically attempt to generalize from the under-represented classes (threats) to learn a model which can detect both known attacks and potentially unknown threats. Unfortunately, anomaly-based IDS tend to incur a significant false positive rate, which in part hinders them from being deployed outside of academia [11].

In this work, we aim to design an anomaly-based IDS to capture underlying anomalies during an active DDoS which may point to a DaaSS attack, but under the assumptions that our unlabeled training data is balanced and only includes two classes: DDoS behavior, and normal (benign) behavior during an active DDoS. We learn a model which only incorporates representations for these two classes. Underlying anomalies are represented in the model as the entire space outside of the behavior representations we learn from the data, thus giving a well-defined boundary between what we have learned and what we do not know.

To the best of our knowledge, this is the first work that attempts to detect DaaSS type attacks, requiring us to derive novel solutions from data collection to the detection approach.

978-1-5386-7848-0/18/$31.00

Our main contributions are as follows:

- We provide a formal characterization of the DaaSS threat.
- We propose mechanisms for the capture and feature engineering of datasets incorporating DaaSS behavior on a large-scale network, utilizing a large-scale botnet for the DDoS, and automating client-side user behavior.
- We develop a simple-yet-novel approach for DaaSS detection, using unsupervised methods to learn a model in which no DaaSS-specific training data is required.

The remainder of this paper is structured accordingly. Section II characterizes the DaaSS threat in depth. Section III describes our approach, from dataset collection, feature engineering, model training, and classification. Section IV discusses our experimental methodology and results, followed by limitations and future work in Section V. We finally wrap up our discussion in Section VI.

II. Characterizing the DaaSS Threat

A DaaSS attack incorporates two main components: an *active distributed denial-of-service (DDoS)* and an *intentional underlying anomaly*[1], which could be a single anomaly or a collection of anomalous behaviors. An *anomaly* in this context is defined as an action or set of actions originated by a nefarious agent (an attacker).

DaaSS attacks are similar in nature to the *Coordinated Attack with Role Distribution (CARD)* threat model discussed by Samarji et al. [12], except that we do not assume the underlying anomaly would (or should) be classified as an attack outside of an active DDoS. This key difference compared to traditional simultaneous attack scenarios changes the way we must think about potential solutions. In the following paragraphs, we discuss the two components which comprise a DaaSS attack.

A DDoS can originate from any source, such as a botnet, and may not necessarily be a packet flooding-based attack. For example, TCP SYN flood-based DDoS may consume very little bandwidth [13]. However, an important characteristic a DDoS must incorporate is *easy detectability*, otherwise it will not serve well as a smokescreen.

We define the period of time that a DDoS is considered 'active' as being from the initial start of the DDoS to when any smokescreen effects of the DDoS cease. For example, if the purpose of the DDoS is to distract corporate IT staff, then the active time period extends until IT personnel are no longer diverting resources and attention for mitigation (which often extends past the cessation of the DDoS). If the purpose of the DDoS is, for example, to disable online banking, then the active time period extends until banking services are restored.

The intentional underlying anomaly may resemble behavior that outside the DDoS context could be completely normal. For example, multiple ATM withdrawals from multiple customers over a short time period should not be flagged if no other evidence exists to suggest that these actions are anomalous. Flagging such events outside of an active DDoS could lead to false positives and many angry customers. As another example, a savvy attacker with a working knowledge of traditional anomaly-based IDS may purposely try to engineer an underlying anomaly to resemble normal behavior, providing a possible extra layer of protection in case the DDoS fails as a smokescreen.

In order for an attack to be considered a DaaSS, an intentional underlying anomaly must occur during an active DDoS. Note that the group responsible for the DDoS may not be connected to the group responsible for the underlying anomaly. It is possible that the DDoS is launched independently, and another group takes advantage of the situation to launch an underlying attack, especially given the variety of third-party services that will launch and manage DDoS attacks for a fee [14].

[1]We use the terms 'intentional underlying anomaly' and 'underlying anomaly' interchangeably.

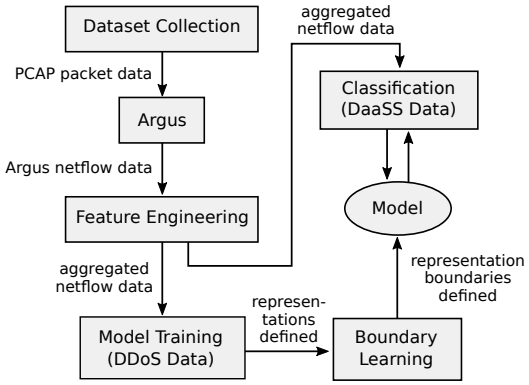

Fig. 1. The pipeline of our approach. We start with dataset collection, then feature engineering, and finally model training with a DDoS training dataset. Model training is a two step process: learning our representations and then their boundaries, which enables DaaSS attack classification.

III. Design of DaaSS Detection Mechanisms

The following sections discuss our approach to DaaSS detection in depth (Figure 1), starting with dataset collection.

A. Dataset Collection

DDoS datasets are not always readily available to the public. Of those which are, such as CTU-13 [15], the primary focus is on the DDoS itself, and no underlying anomalies which would define a DaaSS attack are present (at least not present on purpose). We are unaware of any datasets which explicitly capture DaaSS attacks, leading us to capture our own.

To this end, we use the CORE network emulator [16] with the EMEWS network automation and management framework [17] to capture the data we require. CORE enables us to build up a large-scale network testbed in our lab, while utilizing the same protocols that are deployed in real-world, physical networks. EMEWS supports client-side user behavior automation and experimental monitoring, allowing us to capture network trace data comprising of hundreds of client-side users without requiring human interaction, while alerting us to issues during a data capture session which may comprise the integrity of the captured data.

We only capture data during an active DDoS, thereby preventing normal behavior before and after the DDoS from being captured. This is partly because, according to the threat characterization in Section II, we only detect underlying anomalies during an active DDoS. Constraining our data in this manner also helps to increase the likelihood of detecting underlying anomalies which may resemble normal behavior outside of an active DDoS, as our model will never learn to classify such behavior as normal.

The resulting data, captured in a controlled environment, is guaranteed to be free from anomalies such as undetected attacks. If such anomalies are accidentally captured in our data, then we risk our model skewing either or both of the normal or DDoS behavior representations during training.

B. Feature Engineering

Our captured data is converted to the *Audit Record Generation and Utilization System (Argus)* network flow (netflow) format [18]. An Argus netflow contains individual traffic flows, in which each flow comprises feature values, including source and destination information, start time, and various aggregated features such as total bytes and packets transmitted. Each flow is broken up into one or more *examples*, with each example from a common flow sharing the same source IP, source port, destination IP, destination port, and protocol. While the netflow data provides a good starting point for training our model as in Section III-C1, additional feature engineering is required to capture specific traffic behavioral properties of interest.

We observe that flooding-based DDoS attacks display much higher throughput compared to benign traffic, but the Argus netflow data tends to be too fine-grained, often splitting up a single flow into many examples of small duration. This leads to complications in differentiating DDoS examples from benign, if not considering example start times.

To prevent unnecessary computations involving example start times and duration during the model training process, we opt instead to aggregate examples which appear to belong to the same flow. For examples $\mathbf{x}_i$ and $\mathbf{x}_j$ with a common source IP, destination IP, destination port, and protocol, we merge the examples if the start times of both fall within a small time interval δ_x. More formally, merge examples $\mathbf{x}_i$ and $\mathbf{x}_j$ if $|t_s(\mathbf{x}_j) - t_s(\mathbf{x}_i)| < \delta_x$. Repeat this process until no more examples can be merged. We let $\mathbf{x} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ represent the final collection of examples after merging is completed.

One issue when aggregating by time in this manner is potential correlations between feature values and the time span in which the network flow data was captured. For example, if DDoS flows are aggregated such that each example comprises all traffic from a specific source IP address (δ_x equal to the entire netflow duration), then feature values such as total packets and bytes will increase proportionally with the capture time. This correlates the netflow data's total duration to specific features values. To prevent this behavior, we set a hard threshold, β_x, for the duration of a single aggregated example. This prevents creation of very long examples such as DDoS flows, enabling feature values independent of network flow capture duration.

Features such as the total number of packets and bytes transmitted during an example need to be aggregated. We accomplish this by using sum-based aggregation, $\sum_i v_{ji}$, where v_{ji} is the value of the j^{th} feature (e.g., total packets or total bytes) for the i^{th} example.

In addition, when examples are merged, the resulting duration is aggregated to include the duration between the original examples, namely as the difference between start times $t_s(\mathbf{x}_j)$ and $t_s(\mathbf{x}_i)$, where $\mathbf{x}_j$ is the last example that we can merge. Note that as we do not have end times for examples, the duration of the final merged example is not recorded.

The features we use in our time-aggregated netflow datasets include example duration (numeric), example protocol (categorical), destination IP (categorical), destination port (categorical), total packets transmitted during an example (numeric), and total bytes transmitted during an example (numeric).

C. Clustering

In keeping with the practicality of our overall approach, we do not assume that our data is labeled, nor do we assume that we have any data on what underlying anomalies look like. Hence, we will use unsupervised methods to cluster the data we do have: benign and DDoS behavioral characteristics. While using unsupervised clustering techniques for anomaly detection is not new [7], [19], we must be mindful of the fact that the model we train will not be aware of what an underlying anomaly looks like, and will not be able to detect them without additional methods.

Given that our feature space is mixed, we utilize *k-prototypes* [20], [21], an unsupervised hard-clustering learning algorithm which natively allows both categorical and numerical features. k-prototypes is an extension of the categorical k-modes clustering algorithm [21], but comprises two similarity measures, one for categorical features and the other for numerical features. When determining cluster assignment, k-prototypes will use the similarity measure appropriate to the feature for which the measure is being applied. We use Euclidean distance as the similarity measure for numerical features, and feature value matching for categorical features, which utilizes counts of dissimilar feature values.

In our setting, we have two clusters which we know exist: examples belonging to an ongoing DDoS, and benign examples. However, we also have a third, unknown cluster, in which underlying anomalous examples belong. Because we have no training data for this third cluster, we cannot include it during training, but rather treat the space external to all learned clusters as a 'catch-all' to classify examples which do not belong to any cluster.

In the following subsections, we discuss the training and classification processes.

1) Training: Cluster centroid initialization comprises two parts, due to our feature space being mixed. For initialization of our categorical dimensions, we use the method provided by Cao et al. [22], called *density of points (examples)*. This

method calculates the average density $Dens(\mathbf{x}_i)$ of each example $\mathbf{x}_i$, and selects the first cluster centroid $\mathbf{c}_1$ as the example $\mathbf{x}$ with the maximum average density

$$\mathbf{c}_1 = \arg\max_{\mathbf{x}_i \in \mathbf{x}} Dens(\mathbf{x}_i).$$

The second cluster centroid $\mathbf{c}_2$ is selected in a similar fashion, except that the distance from $\mathbf{c}_1$ is also considered, picking an example $\mathbf{x}$ which maximizes the product of its distance to $\mathbf{c}_1$ and its average density

$$\mathbf{c}_2 = \arg\max_{\mathbf{x}_i \in \mathbf{x}} d(\mathbf{x}_i, \mathbf{c}_1) Dens(\mathbf{x}_i).$$

Note that thus far our centroids do not contain any numerical dimensions. Initialization of these dimensions involves sampling from normal distributions whose parameters are derived from our examples. For each numeric feature f_i, we sample two values $f_i(k) \sim N(\overline{x}_i, s_i)$, in which

$$\overline{x}_i = \left(\sum_{j=1}^{|\mathbf{x}_i|} x_{ij} \right) / |\mathbf{x}_i|, \qquad s_i = \sqrt{\left[\sum_{j=1}^{|\mathbf{x}_i|} (x_{ij} - \overline{x}_i)^2 \right] / |\mathbf{x}_i|},$$

and $k = 1, 2$ corresponding to our clusters[2].

Once the clusters centroids $\mathbf{c}_j$ have been initialized, the training phase proceeds through m iterations in which every example is assigned to a cluster, and the cluster centroids updated, respectively. The step of updating the centroids after example assignment can affect which cluster an example is assigned to on the next iteration. The iterations stop upon convergence, defined as when an iteration completes in entirety without any examples changing their cluster assignment. In our experimentation, convergence occurred after $m = 3$ iterations on average.

When the training process is complete, we can identify each cluster in the set $\mathbf{c}$ by its centroid as $\mathbf{c}_j \in \mathbf{c}$ and its associated *radius* r_j [23] that determines its boundary within the space containing the training examples.

Clearly, the performance of the classification step will depend on how this radius is defined. To allow for flexible performance tuning, we define the radius r_j of cluster $\mathbf{c}_j$ as

$$r_j = \alpha_r \max_{\mathbf{x}_i \in \mathbf{c}_j} d(\mathbf{x}_i, \mathbf{c}_j), \tag{1}$$

namely a scaled version of the maximum distance between the cluster centroid and an example assigned to the cluster, where the scaling factor $\alpha_r > 0$ is a tuneable hyperparameter to control model sensitivity. If $\alpha_r = 1.0$, for any cluster $\mathbf{c}_j$, r_j is guaranteed to be the minimum length needed to correctly classify any training example $\mathbf{x}_i$ that was originally assigned to $\mathbf{c}_j$ during training.

Values for α_r in the range $[0, 1.0)$ will result in greater sensitivity for underlying anomaly detection, but with a potentially high false positive rate of underlying anomalies, whereas values of 1.0 or greater limit false positives at the potential expense of missing underlying anomalies.

[2]Bessel's correction was not used in our s_i derivation due to it having no effect on our learned cluster centroids after training.

With the addition of cluster radii, we can now formally define the *external space*, denoted as $\mathbf{c}_{\text{ext}}$, as the portion of the space that is not contained in any of the clusters $\mathbf{c}_j \in \mathbf{c}$.

2) Classification: With the addition of the external space $\mathbf{c}_{\text{ext}}$, k-prototypes can assign each test example $\mathbf{x}_i$ according to the cluster radii for $\mathbf{c}_j \in \mathbf{c}$ or to $\mathbf{c}_{\text{ext}}$ if there is no cluster $\mathbf{c}_j$ such that $d(\mathbf{x}_i, \mathbf{c}_j) \leq r_j$, thereby declaring $\mathbf{x}_i$ as an underlying anomaly. More simply stated, an example $\mathbf{x}_i$ is anomalous if $\forall j \; d(\mathbf{x}_i, \mathbf{c}_j) > r_j$. If an example $\mathbf{x}_i$ satisfies $d(\mathbf{x}_i, \mathbf{c}_j) \leq r_j$ for multiple clusters $\mathbf{c}_j$, then we assign it to the cluster $\mathbf{c}_j$ with minimum distance $d(\mathbf{x}_i, \mathbf{c}_j)$ to the centroid.

IV. Experiments

Our experiments are performed with datasets captured using the eMEWS network automation and management framework [17] and underlying CORE network emulator [16] (Section III-A). We discuss our experimental methodology next, followed by discussion of our results.

A. Experimental Methodology

Our experimental network consists of 312 nodes, of which 130 are benign HTTPS clients, 31 are benign SSH clients, and 89 are bots which comprise the botnet for DDoS attacks. The benign nodes randomly connect to one of three HTTPS servers for web crawling, whereas the botnet nodes only connect to a single *target* HTTPS server for the purpose of launching and sustaining a request-based DDoS attack. This target HTTPS server is located on a LAN referred to as the *target network*, which along with the aforementioned HTTPS server, contains an SSH server and workstation nodes for hypothetical employees (or insiders). Our data is captured from the target network's primary router.

We partition our experimental scenarios into two categories: (1) DDoS attacks with no underlying anomalies and (2) DaaSS scenarios which incorporate a DDoS with underlying anomalies. For our experiments, we consider two underlying anomalies. The first incorporates SSH logins and data exfiltration originating from two attackers within the target network, emulating an insider-type of DaaSS attack. The second attack originates external from the target network, with multiple attackers performing large-volume data exfiltration from the same server as which the active DDoS is targeting, and with traffic patterns that resemble the DDoS itself.

A training set comprising two hours of captured network traces is used to learn our clusters, and nine additional datasets each comprising 30 minutes (except for scenario *DDoS-5* which is two hours) of captured network traces make up our test data. Our test sets comprise five DDoS scenarios without any underlying anomalies and four DaaSS scenarios. Each dataset collected is processed using the methods discussed in Section III-B, with $\delta_x = 30$ seconds and $\beta_x = 180$ seconds.

The metric we aim to minimize is the number of false positives in terms of underlying anomalies. Because the number of examples corresponding to these anomalies is largely imbalanced compared to benign and DDoS examples, we use precision and recall to display our results for the DaaSS

Scn	Exmple Ct	$\alpha_r = 1.0$		$\alpha_r = 1.25$		$\alpha_r = 1.5$	
		#FP	Rate	#FP	Rate	#FP	Rate
DDoS-1	1499	1	0.07%	0	0%	0	0%
DDoS-2	1538	2	0.13%	1	0.07%	0	0%
DDoS-3	1646	12	0.73%	4	0.24%	2	0.12%
DDoS-4	1725	0	0%	0	0%	0	0%
DDoS-5	5490	19	0.35%	8	0.16%	4	0.07%

TABLE I

RESULTS OF THE DDOS SCENARIOS WITH NO INTENTIONAL UNDERLYING ANOMALIES. 'SCN' IS THE EXPERIMENTAL SCENARIO WHICH WE ARE EVALUATING, 'EXMPLE CT' IS THE NUMBER OF EXAMPLES THE SCENARIO COMPRISES, '#FP' IS THE NUMBER OF FALSE POSITIVES PRODUCED BY A SCENARIO, AND 'RATE' IS THE FALSE POSITIVE RATE.

scenarios. The DDoS scenarios naturally cannot use these metrics due to comprising only benign and DDoS examples, so we use false positive rate to display their results.

B. Experimental Results

Table I gives the results for our DDoS scenarios. The goal of these experiments is to observe the false positive count of our approach, in terms of underlying anomalies, during DDoS attacks *without* any underlying anomalies. For our system to be considered for practical deployment, a very low number of false positives is a baseline requirement.

Scenarios *DDoS-1* and *DDoS-2* comprised benign user behavior resembling that captured in our training set. All false positives were eliminated with $\alpha_r = 1.5$. The results for $\alpha_r = 1.0$ may seem acceptable, as two false positives for scenario *DDoS-2* is rather low, but considering that our examples are aggregated and may comprise a large length of time, even one false positive may be significant enough to trigger a false alarm, depending on how (or if) outliers are filtered. Thus, we consider any number of false positives to be significant.

Scenarios *DDoS-3*, *DDoS-4*, and *DDoS-5* were captured with slightly different benign behavior, to emulate drift in user behavior over time from when the model was originally trained. A particularly resilient model would not have to be retrained as often. As expected, these scenarios produced a larger number of false positives, with the exception of scenario *DDoS-4*, which produced zero false positives even with an α_r value of 1.0, surpassing that of even the non-drift scenarios. Scenario *DDoS-5* was captured for two hours, as we were curious if the length of an active DDoS affected our false positive rate. In that regard, this scenario resulted in a lower false positive rate than scenario *DDoS-3*, which is promising even though we still consider four false positives with $\alpha_r = 1.5$ to be significant.

How well does our approach actually detect DaaSS attacks? Our results, given in Table II, show promise that our original goal of reliable DaaSS detection while minimizing false positives can be realized.

Scenarios *DaaSS-1* and *DaaSS-2* comprised benign user behavior resembling that captured in our training set, and used the insider threat discussed in Section IV-A. An α_r value of 1.25 eliminated all false positives, which is promising given that an α_r value of 1.5 was needed to completely eliminate false positives in DDoS scenarios *DDoS-1* and *DDoS-2*. While our recall for scenario *DaaSS-2* is a bit low, detecting two-thirds of the underlying anomalies is arguably enough to reliably detect the DaaSS attack.

Scenarios *DaaSS-3* and *DaaSS-4* incorporate drift in user behavior, with scenario *DaaSS-3* using the insider threat, and scenario *DaaSS-4* using the external threat discussed in Section IV-A. Expectedly, these scenarios had higher false positives, but using an α_r value of 1.5 eliminated all of them. Surprisingly, we were able to achieve perfect classification on scenario *DaaSS-4*, in spite of the underlying anomaly resembling the active DDoS.

How high of an α_r value can be set before we lose recall? For scenario *DaaSS-3*, we were able to set $\alpha_r = 4.0$ before our recall started decreasing, and for scenario *DaaSS-4*, we could go all the way up to $\alpha_r = 26.0$. This may imply that as benign behavior changes over time, simply setting a higher α_r value may at least buy some time from having to retrain the model. It may also imply that our underlying anomalies are distant enough from our clusters to be reliably detected given that the external space shrinks as α_r increases.

V. LIMITATIONS AND FUTURE WORK

A natural limitation when using a hard-clustering approach with a small number of clusters involves separation of distinct behavioral patterns. Using only one cluster to represent all benign behavioral patterns naturally results in a cluster with a large radius, and can potentially lower recall for DaaSS attacks. As our DDoS cluster only represents one type of DDoS, our approach was able to correctly classify all underlying anomaly examples for scenario *DaaSS-5*, even though the underlying anomaly itself resembled the DDoS. By subdividing the benign (normal) cluster by specific patterns of benign behavior, we should end up with clusters of smaller radius as compared to our single benign cluster. This should enable our approach to detect a wider range of underlying anomalies, and provide better recall for scenarios such as *DaaSS-2*.

Outlier detection for our setting in terms of β_x may not be practical, as a single outlier could in reality encompass an entire underlying anomaly, due to 180 seconds potentially being adequate enough to finish the attack. Further experimentation with lower β_x values (and potentially lower δ_x values) is needed to determine how this would affect DDoS detection and false positives related to underlying anomalies. From an issue of trust, we feel it is better for the system to occasionally miss a DaaSS attack, than to falsely alert the presence of one. On one hand, we do not want IT personnel diverted to handling a potential underlying attack which did not actually occur. On the other hand, we do not want the system to be so cautious that it rarely detects underlying anomalies.

Setting α_r may require some knowledge or intuition of what kinds of anomalies are possible or expected during an active DDoS. Fortunately, as our experiments show (Section IV-B), there seems to be a wide range of values that α_r can take

Scn	Exmple Ct	Exmple DaaSS	$\alpha_r = 1.0$			$\alpha_r = 1.25$			$\alpha_r = 1.5$		
			#FP	Precision	Recall	#FP	Precision	Recall	#FP	Precision	Recall
DaaSS-1	1406	3	1	75%	100%	0	100%	100%	0	100%	100%
DaaSS-2	1497	9	1	85.71%	66.67%	0	100%	66.67%	0	100%	66.67%
DaaSS-3	1405	14	6	68.42%	92.86%	1	92.86%	92.86%	0	100%	92.86%
DaaSS-4	1687	14	1	93.33%	100%	1	93.33%	100%	0	100%	100%

TABLE II

RESULTS OF THE DAASS SCENARIOS. AS IN TABLE I, 'SCN' IS THE EXPERIMENTAL SCENARIO WHICH WE ARE EVALUATING, 'EXMPLE CT' IS THE NUMBER OF EXAMPLES THE SCENARIO COMPRISES, 'EXMPLE DAASS' GIVES THE NUMBER OF EXAMPLES CORRESPONDING TO AN INTENTIONAL UNDERLYING ANOMALY, AND '#FP' IS THE NUMBER OF FALSE POSITIVES PRODUCED BY A SCENARIO. PRECISION IS DEFINED AS $TP/(TP + FP)$ AND RECALL IS DEFINED AS $TP/(TP + FN)$. TP IS THE NUMBER OF EXAMPLES CORRECTLY CLASSIFIED AS AN INTENTIONAL UNDERLYING ANOMALY, FP IS THE NUMBER OF EXAMPLES INCORRECTLY CLASSIFIED AS AN INTENTIONAL UNDERLYING ANOMALY, AND FN IS THE NUMBER OF EXAMPLES THAT SHOULD HAVE BEEN CLASSIFIED AS AN INTENTIONAL UNDERLYING ANOMALY.

to lower the number of false positives while maintaining an acceptable recall, though this is dependent on how close the underlying anomalies represent other types of behavior.

Much of our future work is centered around learning more expressive models that can capture a wide variety of benign and DDoS behaviors while maintaining a very low false positive rate in terms of DaaSS attacks. Ideally we would like to accomplish this while maintaining the mostly high recall numbers shown in Table II. We also plan to expand our network testbed to include more diverse network traffic sources and larger botnets, enabling us to capture a wider range of benign, DDoS, and DaaSS behavior.

VI. CONCLUSION

In this work we discussed a recent variation of a DDoS attack, named DaaSS, and presented a novel approach on detecting these attacks, from dataset collection all the way up to a model suitable for detection. Our experimental results show promise that 'lifting the smokescreen' of a DDoS is indeed possible, and we hope that this work becomes one of many to solve practical DaaSS problems.

VII. ACKNOWLEDGEMENTS

This work was supported in part by a grant from the National Science Foundation (#ACI-1443019).

REFERENCES

[1] P. Wagenseil, "Sony Blames Anonymous for PlayStation Network Attack," http://www.nbcnews.com/id/42909386/ns/technology_and_science-security/t/sony-blames-anonymous-playstation-network-attack/, 2011.

[2] C. Williams, "Carphone Warehouse hackers 'used traffic bombardment smokescreen'," https://www.telegraph.co.uk/finance/newsbysector/epic/cpw/11794521/Carphone-Warehouse-hackers-used-traffic-bombardment-smokescreen.html, 2011.

[3] G. Carl, G. Kesidis, R. R. Brooks, and S. Rai, "Denial-of-Service Attack-Detection Techniques," *IEEE Internet Computing*, vol. 10, no. 1, pp. 82–89, Jan 2006.

[4] J. Cannady, "Artificial Neural Networks for Misuse Detection," in *National Information Systems Security Conference*, 1998, pp. 443–456.

[5] D. E. Denning, "An Intrusion-Detection Model," *IEEE Transactions on Software Engineering*, vol. SE-13, no. 2, pp. 222–232, Feb 1987.

[6] N. Ye, "A Markov Chain Model of Temporal Behavior for Anomaly Detection," in *In Proceedings of the 2000 IEEE Workshop on Information Assurance and Security*, 2000, pp. 171–174.

[7] L. Portnoy, E. Eskin, and S. Stolfo, "Intrusion Detection with Unlabeled Data Using Clustering," in *In Proceedings of ACM CSS Workshop on Data Mining Applied to Security (DMSA-2001*, 2001, pp. 5–8.

[8] A. Patcha and J. M. Park, "An Overview of Anomaly Detection Techniques: Existing Solutions and Latest Technological Trends," *Computer Networks*, vol. 51, no. 12, pp. 3448 – 3470, 2007.

[9] X. Xu, Y. Sun, and Z. Huang, "Defending DDoS Attacks Using Hidden Markov Models and Cooperative Reinforcement Learning"," in *Intelligence and Security Informatics*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 196–207.

[10] C. J. Dietrich, C. Rossow, and N. Pohlmann, "CoCoSpot: Clustering and Recognizing Botnet Command and Control Channels Using Traffic Analysis," *Comput. Netw.*, vol. 57, no. 2, pp. 475–486, Feb. 2013.

[11] R. Sommer and V. Paxson, "Outside the Closed World: On Using Machine Learning for Network Intrusion Detection," in *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, ser. SP '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 305–316.

[12] L. Samarji, F. Cuppens, N. Cuppens-Boulahia, W. Kanoun, and S. Dubus, "Situation Calculus and Graph Based Defensive Modeling of Simultaneous Attacks," in *Cyberspace Safety and Security*. Springer International Publishing, 2013, pp. 132–150.

[13] B. Ricks and P. Tague, "Isolation of Multiple Anonymous Attackers in Mobile Networks," in *Network and System Security*. Springer International Publishing, 2015, pp. 32–45.

[14] M. Karami and D. McCoy, "Understanding the Emerging Threat of DDoS-as-a-Service," in *Presented as part of the 6th USENIX Workshop on Large-Scale Exploits and Emergent Threats*. Washington, D.C.: USENIX, 2013.

[15] S. Garcia, M. Grill, J. Stiborek, and A. Zunino, "An Empirical Comparison of Botnet Detection Methods," *Computers & Security*, vol. 45, pp. 100–123, 2014.

[16] J. Ahrenholz, "Comparison of CORE Network Emulation Platforms," in *MILCOM 2010 Military Communications Conference*, Oct 2010, pp. 166–171.

[17] B. Ricks, P. Tague, and B. Thuraisingham, "Large-Scale Realistic Network Data Generation on a Budget," in *19th International Conference on Information Reuse and Integration (IRI)*. IEEE, 2018.

[18] Qosient, "Audit Record Generation and Utilization System (Argus)," https://qosient.com/, 2018.

[19] K. Leung and C. Leckie, "Unsupervised Anomaly Detection in Network Intrusion Detection Using Clusters," in *Proceedings of the Twenty-eighth Australasian Conference on Computer Science*, ser. ACSC '05, Darlinghurst, Australia, 2005, pp. 333–342.

[20] Z. Huang, "Clustering Large Data Sets with Mixed Numeric and Categorical Values," in *In The First Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 1997, pp. 21–34.

[21] ——, "Extensions to the k-Means Algorithm for Clustering Large Data Sets with Categorical Values," *Data Mining and Knowledge Discovery*, vol. 2, no. 3, pp. 283–304, Sep 1998.

[22] F. Cao, J. Liang, and L. Bai, "A New Initialization Method for Categorical Data Clustering," *Expert Syst. Appl.*, vol. 36, no. 7, pp. 10 223–10 228, Sep. 2009.

[23] M. Masud, J. Gao, L. Khan, J. Han, and B. M. Thuraisingham, "Classification and Novel Class Detection in Concept-Drifting Data Streams Under Time Constraints," *IEEE Trans. on Knowl. and Data Eng.*, vol. 23, no. 6, pp. 859–874, Jun. 2011.