F. Araujo, G. Ayoade, K. Al-Naami, Y. Gao, K.W. Hamlen, and L. Khan. Improving Intrusion Detectors by Crook-sourcing. In *Proc. 35th Computer Security Applications Conf. (ACSAC)*. December 2019.

Improving Intrusion Detectors by Crook-sourcing

Frederico Araujo
IBM Research

Gbadebo Ayoade
The University of Texas at Dallas

Khaled Al-Naami
The University of Texas at Dallas

Yang Gao
The University of Texas at Dallas

Kevin W. Hamlen
The University of Texas at Dallas

Latifur Khan
The University of Texas at Dallas

ABSTRACT

Conventional cyber defenses typically respond to detected attacks by rejecting them as quickly and decisively as possible; but aborted attacks are missed learning opportunities for intrusion detection. A method of reimagining cyber attacks as free sources of live training data for machine learning-based intrusion detection systems (IDSes) is proposed and evaluated. Rather than aborting attacks against legitimate services, adversarial interactions are selectively prolonged to maximize the defender's harvest of useful threat intelligence. Enhancing web services with deceptive attack-responses in this way is shown to be a powerful and practical strategy for improved detection, addressing several perennial challenges for machine learning-based IDS in the literature, including scarcity of training data, the high labeling burden for (semi-)supervised learning, encryption opacity, and concept differences between honeypot attacks and those against genuine services. By reconceptualizing software security patches as feature extraction engines, the approach conscripts attackers as free penetration testers, and coordinates multiple levels of the software stack to achieve fast, automatic, and accurate labeling of live web data streams.

Prototype implementations are showcased for two feature set models to extract security-relevant network- and system-level features from servers hosting enterprise-grade web applications. The evaluation demonstrates that the extracted data can be fed back into a network-level IDS for exceptionally accurate, yet lightweight attack detection.

CCS CONCEPTS

• **Security and privacy → Intrusion/anomaly detection and malware mitigation**; *Software security engineering*; *Web application security*; • **Computing methodologies** → *Machine learning*.

KEYWORDS

datasets, neural networks, intrusion detection, honeypots

ACM Reference Format:
Frederico Araujo, Gbadebo Ayoade, Khaled Al-Naami, Yang Gao, Kevin W. Hamlen, and Latifur Khan. 2019. Improving Intrusion Detectors by Crook-sourcing. In *2019 Annual Computer Security Applications Conference (ACSAC '19), December 9–13, 2019, San Juan, PR, USA*. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3359789.3359822

1 INTRODUCTION

Detecting cyber attacks before they reach unpatched, vulnerable web servers (or afterward, for recovery purposes) has become a vital necessity for many organizations. In 2018 alone, the average window of exposure for critical web application vulnerabilities was 69 days, with a new vulnerability found *every hour*—an increase of 13% over the previous year's rate—and over 75% of all legitimate web sites have unpatched vulnerabilities, 20% of which afford attackers full control over victim systems [27, 72]. The cost of data breaches resulting from software exploits is expected to escalate to an unprecedented \$2.5 trillion by 2022 [41].

Intrusion detection [23] is an important means of mitigating such threats. IDSes capitalize on the observation that the most damaging and pernicious attacks discovered in the wild often share similar traits, such as the steps intruders take to open back doors, execute files and commands, alter system configurations, and transmit gathered information from compromised machines [24, 38, 61, 66]. Starting with the initial infection, such malicious activities often leave telltale traces that can be identified even when the underlying exploited vulnerabilities are unknown to defenders. The challenge is therefore to capture and filter these attack trails from network traffic, connected devices, and target applications, and develop defense mechanisms that can effectively leverage such data to disrupt ongoing attacks and prevent future attempted exploits. Specifically, machine learning-based IDSes alert administrators when deviations from a model of *normal* behavior are detected [29, 50, 80].

However, despite its great promise, the advancement of machine learning approaches for web intrusion detection have been hindered by a scarcity of realistic, current, publicly available cyber attack data sets, and by the difficulty of accurately and efficiently labeling such data sets, which are often prohibitively large and complex [75]. This has frustrated comprehensive, timely training of IDSes, and has resulted in an overreliance on unrealistic closed-world assumptions [69], thereby raising IDS false alarm rates and elevating their susceptibility to attacker evasion [11, 17, 31, 63, 69].

This paper proposes and examines a new deception-based approach to enhancing IDS data streams through *crook-sourcing*—the conscription and manipulation of attackers into performing free penetration testing for improved IDS model training and adaptation. Deception has long been recognized as a key ingredient of effective cyber warfare (cf., [81]), but its applications to IDS have heretofore been limited to contexts where the deception is isolated and separate from the data stream in which intrusions must actually be detected. For example, dedicated *honeypots* collect attack-only data streams [76] but have limited IDS training value in that they can mistrain models to recognize only attacks against honeypots, including false positives from scans and accidental connections, or attacks by unsophisticated adversaries unable to identify and avoid honeypots. Attacks with substantial interactivity can be missed, since the honeypot offers no legitimate services, and therefore collects no data characterizing attacks against legitimate services.

Our approach overcomes this limitation by integrating deceptive attack response capabilities directly into live, production server software via *honey-patching* [5]. Honey-patches are software security patches that are modified to avoid alerting adversaries when their exploit attempts fail. Instead of merely blocking the attempted intrusion, the honey-patch transparently redirects the attacker's connection to a carefully isolated decoy environment running an unpatched version of the software. Adversaries attempting to exploit a honey-patched vulnerability observe software responses that resemble unpatched software, even though the vulnerability is actually patched. This allows the system to observe subsequent attack actions until the deception is eventually uncovered. Honey-patches offer equivalent security to conventional patches, but can potentially enhance IDS web data streams with a semantically rich stream of pre-labeled (attack-only) data for training purposes. These crook-sourced data streams thus provide IDSes with concept-relevant, current, feature-filled information with which to detect and prevent sophisticated, targeted attacks.

We demonstrate the potential effectiveness of this new IDS approach through the design, implementation, and analysis of DeepDig (DEcEPtion DIGging), a framework for deception-enhanced web intrusion detection. Evaluation shows that extra information harvested through mini-deceptions (1) improves the precision of anomaly-based IDSes by feeding back attack traces into the classifier, (2) provides feature-rich, multi-dimensional attack data for classification, and (3) can detect exploit variants previously unseen by defenders. Our goal in this work is to assess whether successful deceptions are helpful for intrusion detection, and to what degree. Given the present scarcity of good, current intrusion data sets and the costs of conducting large-scale empirical data collection, we believe that the approach's facility for generating richer, automatically-labeled, web attack data streams offers exceptional promise for future IDS research and deployments.

Our contributions can be summarized as follows:

- We propose a software patching methodology that facilitates semi-supervised learning for intrusion detection, in which deceptive security patches naturally modulate and automate the attack labeling and feature extraction process.
- We present a feature-rich attack classification approach that more accurately characterizes malicious web activities.
- To harness training and test data, we present the design of a framework for the replay and generation of real web traffic, which statistically mutates and injects scripted attacks into the generated output streams.[1]
- We evaluate our approach on large-scale network and system events gathered through simulation and red team evaluations over a test bed built atop production web software, including the Apache web server, OpenSSL, and PHP.

The rest of the paper is organized as follows. Section 2 outlines our approach and presents an overview of the system, followed by a more detailed architecture description in Section 3. Section 4 shows how our approach can support accurate characterization of attacks through decoy data. Implementation is summarized in Section 5, followed by evaluation methodology and results in Section 6. Finally, discussion and related work are presented in Sections 7 and 8 (respectively), and Section 9 concludes with a summary of outcomes and future directions.

2 APPROACH OVERVIEW

We first outline practical limitations of traditional machine learning techniques for intrusion detection, motivating our research. We then overview our approach for automatic attack labeling and feature extraction via honey-patching.

2.1 Intrusion Detection Challenges

Despite the increasing popularity of machine learning in intrusion detection applications, its success in operational environments has been hampered by specific challenges that arise in the cyber security domain. Fundamentally, machine learning algorithms perform better at identifying similarities than at discovering previously unseen outliers. Since normal, non-attack data is usually far more plentiful than realistic, current attack data, many classifiers must be trained almost solely from the former, necessitating an almost perfect model of normality for any reliable classification [69].

Feature extraction [12] is also unusually difficult in intrusion detection contexts because security-relevant features are often not known by defenders in advance. The task of selecting appropriate features to detect an intrusion (e.g., features that generate the most distinguishing intrusion patterns) often creates a bottleneck in building effective models, since it demands empirical evaluation. Identification of attack traces among collected workload traces for constructing realistic, unbiased training sets is particularly challenging. Current approaches usually require manual analysis aided by expert knowledge [11, 17], which severely reduces model evolution and update capabilities to cope with attacker evasion strategies.

A third obstacle is analysis of encrypted streams, which are ubiquitously employed to prevent unauthorized users from accessing sensitive web data transmitted through network links or stored in file systems. Since network-level detectors typically discard cyphered data, their efficacy is greatly reduced by the widespread use of encryption [31]. In particular, attackers benefit from encrypting their malicious payloads, making it harder for standard classification strategies to distinguish attacks from normal activity.

High false positive rates are another practical challenge for adoption of machine learning approaches [63]. Raising too many alarms renders IDSes meaningless in most cases, since actual attacks are lost among the many alarms. Studies have shown that effective intrusion detection therefore demands very low false alarm rates [8].

These significant challenges call for the exploration and development of new, accurate anomaly detection schemes that lift together information from many different layers of the software stack. Toward this end, our work extends machine learning-based intrusion detection with the capability to effectively detect malicious activities bound to the application layer, affording detection approaches an inexpensive tool for automatically and continuously extracting security-relevant features for attack detection.

2.2 Digging Deception-Enhanced Threat Data

DeepDig is a new approach to enhance intrusion detection with threat data sourced from honey-patched [5] applications. Figure 1

[1]The implementation and datasets used in this paper are available in https://github.com/cyberdeception/deepdig.

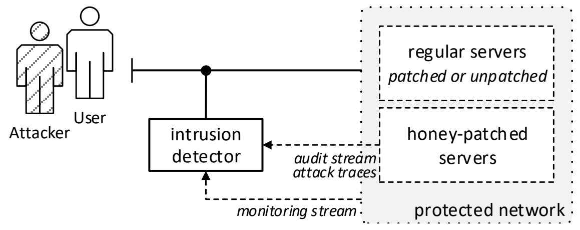

Figure 1: DeepDig approach overview

shows an overview of the approach. Unlike conventional techniques, DeepDig incrementally builds a model of *legitimate* and *malicious* behavior based on audit streams and attack traces collected from honey-patched web servers. This augments the classifier with security-relevant feature extraction capabilities not available to typical network intrusion detectors.

Our central enabling insight is that *software security patches can be repurposed as feature extractors for semi-supervised learning*. The maintenance of the feature extractors is crowd-sourced (by the software development community's ongoing discovery and creation of new security patches), and the data analyzed by the patches is crook-sourced (as attackers contribute their TTP patterns to the data streams processed by the embedded deceptions). Honey-patching transduces these two data sources into a highly accurate, rapidly co-evolving feature extraction module for an IDS. The extractor can effortlessly detect previously unseen payloads that exploit known vulnerabilities at the application layer, which can be prohibitively difficult to detect by a network-level IDS.

These capabilities are transparently built into the framework, requiring no additional developer effort (apart from routine patching) to convert the target application into a potent feature extractor for anomaly detection. Traces extracted from decoys are always contexts of *true* malicious activity, yielding an effortless labeling of the data and higher-accuracy detection models.

By living inside web servers that offer legitimate services, our deception-enhanced IDS can target attackers who use one payload for reconnaissance but reserve another for their final attacks. Deceiving such attackers into divulging the latter is useful for training the IDS to identify the final attack payload, which can reveal attacker strategies and goals not discernible from the reconnaissance payload alone. The defender's ability to thwart these and future attacks therefore derives from a synergy between the application-level feature extractor and the network-level intrusion detector to derive a more complete model of attacker behavior.

2.3 Honey-patching Approach

Prior work has observed that many vendor-released software security patches can be honeyed by replacing their attack-rejection responses with code that instead maintains and forks the attacker's connection to a confined, unpatched decoy [5, 6]. This approach retains the most complex part of the vendor patch (the security check) and replaces the remediation code with some boilerplate forking code [4], making it easy to implement.

Figure 2 demonstrates the approach using pseudo-code for a buffer-overflow vulnerability, a conventional patch, and a honey-patch. The honey-patch retains the logic of the conventional patch's security check, but replaces its remediation with a deceptive fork to a decoy environment. The decoy contains no valuable data; its

```
read $a[i]$
```

```
if ($i \geq length(a)$)
    abort();
read $a[i]$
```

```
if ($i \geq length(a)$)
    fork_to_decoy();
read $a[i]$
```

Figure 2: Pseudo-code for a buffer overflow vulnerability (left), a patch (middle), and a honey-patch (right)

purpose is to monitor attacker actions, such as shellcode or malware introduced by the attacker after abusing the buffer overflow to hijack the software. The infrastructure for redirecting attacker connections to decoys can remain relatively static, so that honey-patching each newly discovered vulnerability only entails replacing the few lines of code in each patch that respond to detected exploits.

This integrated deception offers some important advantages over conventional honeypots. Most significantly, it observes attacks against the defender's genuine assets, not merely those directed at fake assets that offer no legitimate services. It can therefore capture data from sophisticated attackers who monitor network traffic to identify service-providing assets before launching attacks, who customize their attacks to the particular activities of targeted victims (differentiating genuine servers from dedicated honeypots), and who may have already successfully infiltrated the victim's network before their attacks become detected. We next examine how deception-enhanced data harvested in this way can be of particular value to network-level defenses, such as firewalls equipped with machine learning-based intrusion detection.

3 ARCHITECTURE

DeepDig's architecture, depicted in Figure 3, leverages application-level threat data gathered from attacker sessions redirected to decoys to train and adapt a network-level IDS live. Within this framework, honey-patches misdirect attackers to decoys that automatically collect and label monitored attack data. The intrusion detector consists of an *attack modeling* component that incrementally updates the anomaly model data generated by honey-patched servers, and an *attack detection* component that uses this model to flag anomalous activities in the monitored perimeter.

3.1 Monitoring & Threat Data Collection

The decoys into which attacker sessions are forked are managed as a pool of continuously monitored Linux containers. Upon attack detection, the honey-patching mechanism *acquires* the first available container from the pool. The acquired container holds an attacker session until (1) the session is deliberately closed by the attacker, (2) the connection's *keep-alive* timeout expires, (3) the ephemeral container crashes, or (4) a session timeout is reached. The last two conditions are common outcomes of successful exploits. In any of these cases, the container is *released* back to the pool and undergoes a recycling process before becoming available again.

After decoy *release*, the *container monitoring component* extracts the session trace (delimited by acquire and release), labels it, and stores it outside the decoy for subsequent feature extraction. Decoys only host attack sessions, so precisely collecting and labeling their traces (at both the network and OS level) is effortless.

DeepDig distinguishes between three input data streams: (1) the *audit stream*, collected at the target honey-patched server; (2) *attack traces*, collected at decoys; and (3) the *monitoring stream*, the actual test stream collected from regular servers. Each of these streams

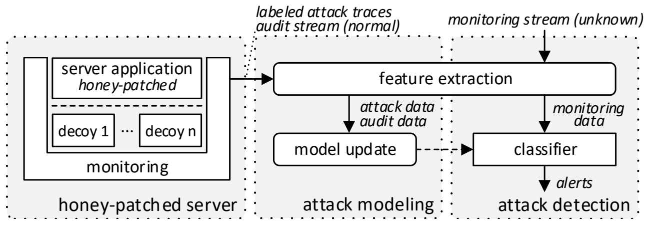

Figure 3: DeepDig system architecture overview

contains network packets and OS events captured at each server environment. To minimize performance impact, we used two powerful and highly efficient software monitors: *sysdig* [73] (to track system calls and modifications made to the file system), and *libpcap* [74] (to monitor ingress and egress of network packets). Specifically, monitored data is stored outside the decoy environments to avoid possible tampering with the collected data.

Deployability. Our monitoring and data collection solution is designed to scale for large, distributed on-premise and cloud deployments. The host-level telemetry leverages a mainstream kernel module that implements non-blocking event collection and memory-mapped event buffer handling for minimal computational overhead. This architecture allows system events to be safely collected (without system call interposition) and compressed by a containerized user space agent that is oblivious to other objects and resources located in the host environment. The event data streams originated from the monitored hosts are conveniently exported to a high-performance, distributed S3-compatible object storage server [59], designed for large-scale data infrastructures.

3.2 Attack Modeling & Detection

Using the continuous audit stream and incoming attack traces as labeled input data, DeepDig incrementally builds a machine learning model that captures legitimate and malicious behavior. The raw training set (*viz.* the audit stream and attack traces) is piped into a feature extraction component that selects relevant, non-redundant features (see §4) and outputs feature vectors—*audit data* and *attack data*—that are grouped and queued for subsequent model update. Since the initial data streams are labeled and have been preprocessed, feature extraction becomes very efficient and can be performed automatically. This process repeats periodically according to an administrator-specified policy. Finally, the *attack detection* module uses the most recently constructed attack model to detect malicious activity in the runtime *monitoring data.*

4 ATTACK DETECTION

To assess our framework's ability to enhance IDS data streams, we have designed and implemented two familiar feature set models: (1) *Bi-Di* detects anomalies in security-relevant network streams, and (2) *N-Gram* finds anomalies in system call traces. Our approach is agnostic to the particular feature set model chosen; we choose these two models for evaluation purposes because they are simple and afford direct comparisons to non-deceptive prior works. The goal of the evaluation is hence to measure the utility of the deception for enhancing data streams for intrusion detection, not to assess the utility of novel feature sets.

Table 1: Packet, uni-burst, and bi-burst features

Category	Features
Packet (Tx/Rx)	Packet length
Uni-Burst (Tx/Rx)	Uni-Burst size
	Uni-Burst time
	Uni-Burst count
Bi-Burst (Tx-Rx/Rx-Tx)	Bi-Burst size
	Bi-Burst time

4.1 Network Packet Analysis

Bi-Di (Bi-Directional) extracts features from sequences of packets and *bursts*—consecutive same-direction packets (*viz.*, uplinks from client *Tx*, or downlinks from server *Rx*) for network behavior analysis. It uses distributions from individual burst sequences (*uni-bursts*) and sequences of two adjacent bursts (*bi-bursts*), constructing histograms using features extracted from packet lengths and directions. To overcome dimensionality issues associated with burst sizes, *bucketization* is applied to group bursts into correlation sets (e.g., based on frequency of occurrence).

Table 1 summarizes the features used, including features from prior works [2, 26, 62, 77]. For robustness against encrypted payloads, we here limit feature extraction to packet headers.

Uni-burst features include burst *size* (the sum of the sizes of all packets in the burst), *time* (the duration for the entire burst to be transmitted), and *count* (the number of packets in the burst). Taking direction into consideration, one histogram for each is generated.

Bi-burst features include time and size attributes of *Tx-Rx-bursts* and *Rx-Tx-bursts*. Each is comprised of a consecutive pair of downlink and uplink bursts. The size and time of each are the sum of the sizes and times of the constituent bursts, respectively.

Bi-bursts capture dependencies between consecutive TCP packet flows. Based on connection characteristics, such as network congestion, the TCP protocol applies flow control mechanisms (e.g., window size and scaling, acknowledgement, sequence numbers) to ensure a level of consistency between Tx and Rx. This influences the size and time of transmitted packets in each direction. Each packet flow (uplink and downlink) thereby affects the next flow or burst until communicating parties finalize the connection.

4.2 System Call Analysis

Monitored data also includes system streams comprised of OS events, each containing multiple fields, including event type (e.g., *open*, *read*, *select*), process name, and direction. Our prototype was developed for Linux x86_64 systems, which exhibit about 314 distinct system call events. We build histograms from these using N-Gram, which extracts features from event subsequences. Each feature type consists of between 1 (*uni-events*) and 4 (*quad-events*) consecutive events, with each event classified as an enter or exit.

Bi-Di and N-Gram differ in feature granularity; the former uses coarser-grained bursting while the latter uses individual system call co-occurrences.

4.3 Classification

We evaluate our approach's practicality using two supervised learning models: SVM [22] and deep learning [49]. Our main objective

Algorithm 1: *Ens-SVM*

```
   Data: training data: $TrainX$, testing data: $TestX$
   Result: a predicted label $\mathcal{L}_{\mathcal{I}}$ for each testing instance $\mathcal{I}$
1  begin
2      $\mathbb{B} \leftarrow updateModel(\text{Bi-Di}, TrainX)$;
3      $\mathbb{N} \leftarrow updateModel(\text{N-Gram}, TrainX)$;
4      for each $\mathcal{I} \in TestX$ do
5          $\mathcal{L}_{\mathbb{B}} \leftarrow label(\mathbb{B}, \mathcal{I})$;
6          $\mathcal{L}_{\mathbb{N}} \leftarrow label(\mathbb{N}, \mathcal{I})$;
7          if $\mathcal{L}_{\mathbb{B}} == \mathcal{L}_{\mathbb{N}}$ then
8              $\mathcal{L}_{\mathcal{I}} \leftarrow \mathcal{L}_{\mathbb{B}}$;
9          else
10             $\mathcal{L}_{\mathcal{I}} \leftarrow label\left(\arg\max_{c \in \{\mathbb{B},\mathbb{N}\}} confidence(c, \mathcal{I}),\ \mathcal{I}\right)$;
11         end
12     end
13 end
```

is to show that our deception-enhanced framework facilitates incremental supervised learning for intrusion detection.

Ens-SVM. This method builds SVM models for Bi-Di and N-Gram. Using convex optimization and mapping non-linearly separated data to a higher dimensional linearly separated feature space, SVM separates positive (attack) and negative (benign) training instances by a hyperplane with the maximum gap possible. Prediction labels are assigned based on which side of the hyperplane each monitoring/testing instance resides.

We combine the two classifiers into an *ensemble* that classifies new input data by weighing the classification outcomes of Bi-Di and N-Gram based on their individual accuracy indexes. Ensemble methods tend to exhibit higher accuracy and avoid normalization issues raised by the alternative (brute force) approach of concatenating the dissimilar features into a single feature vector.

Algorithm 1 describes the voting approach for Ens-SVM. For each instance in the monitoring stream, if both Bi-Di and N-Gram agree on the predictive label (line 7), Ens-SVM takes the common classification as output (line 8). Otherwise, if the classifiers disagree, Ens-SVM takes the prediction with the highest SVM confidence (line 10). Confidence is rated using Platt scaling [64], which uses the following sigmoid-like function to estimate confidence:

$$P(y = 1|x) = \frac{1}{1 + \exp\left(Af(x) + B\right)} \tag{1}$$

where y is the label, x is the testing vector, $f(x)$ is the SVM output, and A and B are scalar parameters learned using Maximum Likelihood Estimation (MLE). This yields a probability measure of a classifier's confidence in assigning a label to a testing point.

Metric Learning. To classify instances to classes, we use online adaptive metric learning (OAML) [9, 30]. OAML is better suited to our task than off-line approaches (e.g., k-nearest neighbors), which yield weak predictors when the separation between different class instances is small. Online similarity metric learning (OML) [14, 19, 37, 39, 52] improves instance separation by finding a new latent space to project the original features, learning similarity from a stream of constraints. *Pairwise* and *triplet* constraints are typically employed: a pairwise constraint takes two dissimilar/similar instances, while a triplet constraint (A, B, C) combines similar instances A and B with a dissimilar instance C.

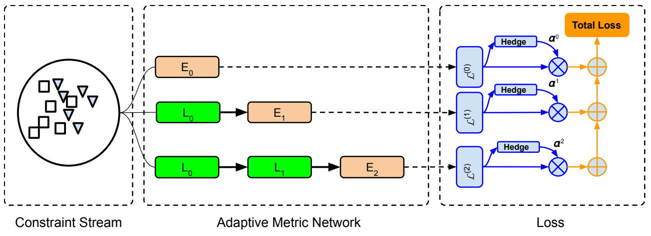

Figure 4: OAML network structure. Each layer L_i is a linear transformation output to a rectified linear unit (ReLU) activation. Embedding layers E_i connect to corresponding input or hidden layers. Linear model E_0 maps the input feature space to the embedding space.

We choose adaptive OML since non-adaptive OML usually learns a pre-selected linear metric (e.g., Mahalanobis distance [79]) that lacks the complexity to learn non-linear semantic similarities among class instances, which are prevalent in intrusion detection scenarios. Moreover, it derives its metric model from well-defined input constraints, leading to bias towards the training data. OAML overcomes these disadvantages by adapting its complexity to accommodate more constraints in the observed data. Its metric function learns a dynamic latent space from the Bi-Di and N-Gram feature spaces, which can include both linear and highly non-linear functions.

OAML leverages artificial neural networks (ANNs) to learn a metric similarity function and can adapt its learning model based on the complexity of its input space. It modifies common deep learning architectures so that the output of every hidden layer flows to an independent metric-embedding layer (MEL). The MELs output an n-dimensional vector in an embedded space where similar instances are clustered and dissimilar instances are separated. Each MEL has an assigned metric weight to determine its importance for the models generated. The output of this embedding is used as input to a k-NN classifier. The approach is detailed below.

Problem Setting. Let $S = \{(\boldsymbol{x_t}, \boldsymbol{x_t^+}, \boldsymbol{x_t^-})\}_{t=1}^{T}$ be a sequence of triplet constraints sampled from the data, where $\{\boldsymbol{x_t}, \boldsymbol{x_t^+}, \boldsymbol{x_t^-}\} \in \mathcal{R}^d$, and $\boldsymbol{x_t}$ (anchor) is similar to $\boldsymbol{x_t^+}$ (positive) but dissimilar to $\boldsymbol{x_t^-}$ (negative). The goal of online adaptive metric learning is to learn a model $\boldsymbol{F} : \mathcal{R}^d \mapsto \mathcal{R}^{d'}$ such that $||\boldsymbol{F}(\boldsymbol{x_t}) - \boldsymbol{F}(\boldsymbol{x_t^+})||_2 \ll ||\boldsymbol{F}(\boldsymbol{x_t}) - \boldsymbol{F}(\boldsymbol{x_t^-})||_2$. Given these parameters, the objective is to learn a metric model with adaptive complexity while satisfying the constraints. The complexity of $\boldsymbol{F}$ must be adaptive so that its hypothesis space is automatically modified.

Overview. Consider a neural network with L hidden layers, where the input layer and the hidden layer are connected to an independent MEL. Each embedding layer learns a latent space where similar instances are clustered and dissimilar instances are separated.

Figure 4 illustrates our ANN. Let $E_\ell \in \{E_0, E_1, E_2, \ldots, E_L\}$ denote the ℓ^{th} metric model in OAML (i.e., the network branch from the input layer to the ℓ^{th} MEL). The simplest OAML model E_0 represents a linear transformation from the input feature space to the metric embedding space. A weight $\alpha^{(\ell)} \in [0, 1]$ is assigned to E_ℓ, measuring its importance in OAML.

For a triplet constraint $(\boldsymbol{x_t}, \boldsymbol{x_t^+}, \boldsymbol{x_t^-})$ that arrives at time t, its metric embedding $f^{(\ell)}(\boldsymbol{x_t^*})$ generated by E_ℓ is

$$f^{(\ell)}(\boldsymbol{x_t^*}) = h^{(\ell)}\Theta^{(\ell)} \tag{2}$$

where $h^{(\ell)} = \sigma(W^{(\ell)}h^{(\ell-1)})$, with $\ell \geq 1$, $\ell \in \mathbb{N}$, and $h^{(0)} = \boldsymbol{x_t^*}$. Here $\boldsymbol{x_t^*}$ denotes any anchor $(\boldsymbol{x_t})$, positive $(\boldsymbol{x_t^+})$, or negative $(\boldsymbol{x_t^-})$ instance, and $h^{(\ell)}$ represents the activation of the ℓ^{th} hidden layer. Learned metric embedding $f^{(\ell)}(\boldsymbol{x_t^*})$ is limited to a unit sphere (i.e., $||f^{(\ell)}(\boldsymbol{x_t^*})||_2 = 1$) to reduce the search space and accelerate training.

During the training phase, for every arriving triplet $(\boldsymbol{x_t}, \boldsymbol{x_t^+}, \boldsymbol{x_t^-})$, we first retrieve the metric embedding $f^{(\ell)}(\boldsymbol{x_t^*})$ from the ℓ^{th} metric model using Eq. 2. A local loss $\mathcal{L}^{(\ell)}$ for E_ℓ is evaluated by calculating the similarity and dissimilarity errors based on $f^{(\ell)}(\boldsymbol{x_t^*})$. Thus, the overall loss introduced by this triplet is given by

$$\mathcal{L}_{overall}(\boldsymbol{x_t}, \boldsymbol{x_t^+}, \boldsymbol{x_t^-}) = \sum_{\ell=0}^{L} \alpha^{(\ell)} \cdot \mathcal{L}^{(\ell)}(\boldsymbol{x_t}, \boldsymbol{x_t^+}, \boldsymbol{x_t^-}) \tag{3}$$

Parameters $\Theta^{(\ell)}$, $\alpha^{(\ell)}$, and $W^{(\ell)}$ are learned during the online learning phase. The final optimization problem to solve in OAML at time t is therefore:

$$\begin{aligned} \underset{\Theta^{(\ell)}, W^{(\ell)}, \alpha^{(\ell)}}{\text{minimize}} \quad & \mathcal{L}_{overall} \\ \text{subject to} \quad & ||f^{(\ell)}(\boldsymbol{x_t^*})||_2 = 1, \forall \ell = 0, \ldots, L. \end{aligned} \tag{4}$$

We evaluate the similarity and dissimilarity errors using an *adaptive-bound triplet loss* (ABTL) constraint [30] to estimate $\mathcal{L}^{(\ell)}$ and update parameters $\Theta^{(\ell)}$, $W^{(\ell)}$ and $\alpha^{(\ell)}$.

5 IMPLEMENTATION

We developed an implementation of DeepDig for 64-bit Linux (kernel 3.19). It consists of two main components: (1) The monitoring controller performs server monitoring and attack trace extraction from decoys. It consists of about 350 lines of Node.js code, and leverages *tcpdump*, *editcap*, and *sysdig* for network and system call tracing and preprocessing. (2) The attack detection component is implemented as two Python modules: the feature extraction module, comprising about 1200 lines of code and feature generation; and the classifier component, comprising 230 lines of code that references the *Weka* [33] wrapper for LIBSVM [18]. The OAML components comprise about 500 lines of Python code referencing the PyTorch [65] library.

The source-code modifications required to honey-patch vulnerabilities in Apache HTTP, Bash, PHP, and OpenSSL consist of a mere 35 lines of C code added or changed in the original server code, showing that the required deceptive capabilities can be added to production-level web services with very little effort. (The forking framework [4, 5] is fixed, and thus not included in this count.)

6 EVALUATION

A central goal of our research is to quantitatively measure the impact of embedded deception on IDS accuracy. Our evaluation approach therefore differs from works that seek to measure absolute IDS accuracy, or that do not separate the impact of deception from the rest of the detection process. We first present our evaluation framework, which we harness to automatically generate training and test datasets from real web traffic for our experiments. Then we discuss our experimental setup and investigate the effects of different attack classes and varying numbers of attack instances on the predictive power and accuracy of the intrusion detection. Finally, we assess the performance impact of the deception monitoring mechanism that captures network packets and system events.

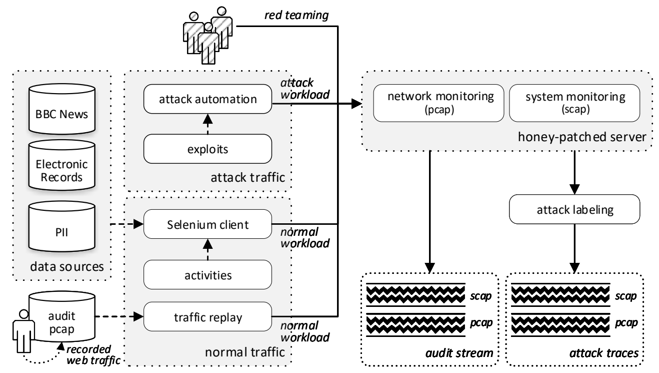

Figure 5: Web traffic generation and testing harness

All experiments were performed on a 16-core host with 24 GB RAM running 64-bit Ubuntu 16.04. Regular and honey-patched servers were deployed as LXC containers [53] running atop the host using the official Ubuntu container image. Red teaming validation was performed on a similar environment deployed on AWS.

6.1 Experimental Framework

Figure 5 shows an overview of our evaluation framework, inspired by related work [9, 13]. It streams *encrypted* legitimate and malicious workloads (both simulated and real) onto a honey-patched web server, resulting in labeled audit streams and attack traces (collected at decoys) for training set generation. This strategy facilitates the reproducibility of our experiments while allowing for the validation of our approach in a realistic setting.

Legitimate workload. In order to collect normal data, we used both real user interactions with a web browser and automated simulation of various user actions on the browser. For the real user interaction, we monitored and recorded web traffic from users in a local area network over a two-day period, resulting in more than 30GB of *audit pcap* data. The recorded sessions are replayed by our framework and include users exhibiting normal browsing activities, such as accessing social media websites, search engines, online shopping websites, web email, video sharing, and news websites.

For the simulated interaction, normal traffic is created by automating complex user actions on a typical web application, leveraging *Selenium* [67] to automate user interaction with a web browser (e.g., clicking buttons, filling out forms, navigating a web page). We generated web traffic for 12 different user activities (each repeated 200 times with varying data feeds), including web page browsing, e-commerce website navigation, blog posting, and interacting with a social media web application. The setup included a CGI web application and a PHP-based Wordpress application hosted on a monitored Apache web server. To enrich the set of user activities, the Wordpress application was extended with *Buddypress* and *Woocommerce* plugins for social media and e-commerce web activities, respectively.

To create realistic interactions with the web applications, our framework feeds from online data sources, such as the BBC text corpus [32], online text generators [60] for personally identifiable information (e.g., usernames, passwords), and product names to populate web forms. To ensure diversity, we statistically sampled the data sources to obtain user input values and dynamically generated web content. For example, blog title and body are statistically sampled from the BBC text corpus, while product names are picked from the product names data source.

Attack workload. Attack traffic is generated based on real-world vulnerabilities. Table 2 lists 22 exploits for nine well-advertised, high-severity vulnerabilities. These include CVE-2014-0160 (Heartbleed), CVE-2014-6271 (Shellshock), CVE-2012-1823 (improper handling of query strings by PHP in CGI mode), CVE-2011-3368 (improper URL validation), CVE-2014-0224 (Change Cipher specification attack), CVE2010-0740 (Malformed TLS record), CVE-2010-1452 (Apache mod_cache vulnerabilty), CVE-2016-7054 (Buffer overflow in openssl with support for ChaCha20-Poly1305 cipher suite), and CVE-2017-5941 (Node.js error handling vulnerability). In addition, nine attack variants exploiting CVE-2014-6271 (Shellshock) were created to carry out different malicious activities (i.e., different attack payloads), such as leaking password files, dropping malware, and invoking bash shells on the remote server. These vulnerabilities are important as attack vectors because they range from sensitive data exfiltration to complete control and remote code execution.

Similarly, we generated attack traffic using CVE-2017-5941 as an additional remote execution attack and executed 6 attack payloads. The post-infection payloads executed tasks such as tool acquisition from remote host, basic environment reconnaissance (e.g., active scanning with Nmap, passive inspection of system logs), remote password file access, root certificate exfiltration, and attempts at gaining access to other machines in the network.

Red teaming. To validate our results, we conducted a small-scale penetration testing experiment using a red team composed of graduate students in cyber security who were tasked with performing reconnaissance and attacking a collection of monitored web server instances deployed on AWS. The deployment comprised unpatched and honey-patched web servers exposing (real or apparent) Shellshock vulnerabilities. Special care was taken not to open unpatched software to the broad Internet. The attack team was composed by 10 students with basic (e.g., script-kidding) to advanced skills (e.g., penetration testing, reverse engineering) in offensive security. After a preliminary leveling and preparation, each student spent an average of 45 minutes completing the exercise on their own time. Students were not permitted to repeat any exercise more than once. Network and system data was collected over a span of 3 days and used as testing and control data.

After successfully completing the initial infection, observed data show that students performed different malicious activities, such as searching for password files, printing user sessions on the system, listing network interfaces, injecting boot loaders, and installing root kits. For example, collected attack samples included modification of `rc.local` to launch a listening process in the background at boot time, network traffic redirection through iptables, file exfiltration over the network, file deletions and access permission modification, and cleanup of event logs and shell history to cover attack paths. No students were able to distinguish honey-patched from unpatched web servers during the experiment.

Table 2: Summary of attack workload

#	Attack Type	Description	Software
1	CVE-2014-0160	Information leak	Openssl
2	CVE-2012-1823	System remote hijack	PHP
3	CVE-2011-3368	Port scanning	Apache
4–10	CVE-2014-6271	System hijack (7 variants)	Bash
11	CVE-2014-6271	Remote Password file read	Bash
12	CVE-2014-6271	Remote root directory read	Bash
13	CVE-2014-0224	Session hijack and information leak	Openssl
14	CVE-2010-0740	DoS via NULL pointer dereference	Openssl
15	CVE-2010-1452	DoS via request that lacks a path	Apache
16	CVE-2016-7054	DoS via heap buffer overflow	Openssl
17–22	CVE-2017-5941*	System hijack (6 variants)	Node.js

*used for testing only, as n-day vulnerability.

Noise injection. Rather than testing with existing, publicly available intrusion datasets (which are inappropriate evaluations of DeepDig, since they lack concept-relevance for deception and are generally stripped of raw packet data), our evaluation interleaves attack and normal traffic following prior work on defense-in-depth [13], and injects benign payloads as data into attack packets to mimic evasive attack behavior. The generated traffic contains attack payloads against recent CVEs for which we created and tested realistic exploits, and our framework automatically extracts labeled features from the monitoring network and system traces to (re-)train the classifiers.

Dataset. Web traffic was generated from a separate host to avoid interference with the test bed server. To account for operational and environmental differences, our framework simulated different workload profiles (according to time of day), against various target configurations (including different background processes and server workloads), and network settings, such as TCP congestion controls. In total, we generated 42 GB of (uncompressed) network packets and system events over a period of three weeks. After feature extraction, the training data comprised 1800 normal instances and 1600 attack instances. Monitoring or testing data consisted of 3400 normal and attack instances gathered at unpatched web servers, where the distribution of normal and attack instances varies per experiment.

Detection accuracy. Using this dataset, we trained the classifiers presented in §4 and assessed their individual performance against test streams containing both normal and attack workloads. In the experiments, we measured the true positive rate (*tpr*), where true positive represents the number of actual attack instances that are classified as attacks; false positive rate (*fpr*), where false positive represents the number of actual benign instances classified as attacks; accuracy (*acc*); and F_2 score of the classifier, where the F_2 score is interpreted as the weighted average of the precision and recall, reaching its best value at 1 and worst at 0. We also calculated a base detection rate (*bdr*) to estimate the success of intrusion detection (§6.3). An RBF kernel with $Cost = 1.3 \times 10^5$ and $\gamma = 1.9 \times 10^{-6}$ was used for SVM [62]. OAML employed a ReLU network with n=200, L=1, and k=5 (defined in §4.3).

To evaluate the accuracy of intrusion detection, we verified each classifier after incrementally training it with increasing numbers of attack classes. Each class consists of 100 distinct variants of a single exploit, as described in §6.1, and an n-class model is one trained with up to n attack classes. For example, a 3-class model is

Table 3: Detection rates (%) for scripted attack scenarios ($P_A \approx 1\%$) compared with results from non-deceptive training (parenthesized)

Classifier	*tpr*	*fpr*	*acc*	F_2	*bdr*
Bi-Di OML	91.00 (+13.2)	0.01 (-41.2)	91.14 (+22.2)	90.00 (+30.3)	98.92 (+97.1)
N-Gram OML	65.00 (-19.9)	0.01 (-5.1)	88.58 (+0.0)	80.00 (-8.4)	98.50 (+84.0)
Bi-Di SVM	79.00 (+1.2)	0.78 (-40.5)	89.88 (+20.9)	78.69 (+19.0)	50.57 (+36.1)
N-Gram SVM	92.42 (+7.5)	0.01 (-5.1)	96.89 (+8.3)	93.84 (+5.5)	99.05 (+84.6)
Ens-SVM	**93.63** (+8.8)	**0.01** (-5.1)	**97.00** (+8.4)	**94.89** (+6.5)	**99.06** (+84.6)

Table 4: Detection rates (%) for red team evaluation ($P_A \approx 1\%$) compared with results from non-deceptive training (parenthesized)

Classifier	*tpr*	*fpr*	*acc*	F_2	*bdr*
Bi-Di-OML	94.00 (-4.0)	0.39 (-52.6)	93.10 (+23.1)	94.00 (+7.0)	70.88 (+69.1)
Ngram-OML	99.00 (+1.0)	0.01 (-50.0)	99.90 (+26.9)	94.00 (+5.0)	99.01 (+97.1)
Bi-Di-SVM	99.56 (+1.6)	1.15 (-51.9)	99.19 (+29.2)	99.39 (+12.4)	46.65 (+44.8)
N-Gram-SVM	92.25 (-6.75)	0.01 (-50.0)	96.35 (+23.4)	93.70 (+4.7)	98.94 (+97.0)
Ens-SVM	**99.56** (+0.56)	**0.01** (-50.0)	**99.19** (+26.2)	**99.39** (+10.4)	**99.02** (+97.1)

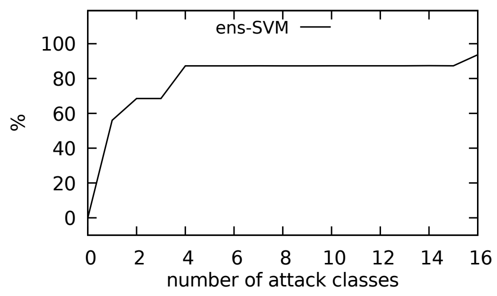

Figure 6: Ens-SVM classification *tpr* for 0–16 attack classes for training on decoy data and testing on unpatched server data.

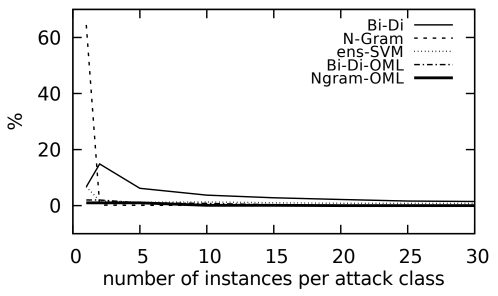

Figure 7: False positive rates for various training set sizes

trained with 300 instances from 3 different attack classes. In each run, the classifier is trained with 1800 normal instances and $100 * n$ attack instances with $n \in [1, 16]$ attack classes sourced from decoys. Each run executes ten experiments where the attacks are shuffled in a cross-validation-like fashion, and the average is reported. This ensures training is not biased toward any specific attacks.

6.2 Experimental Results

Table 3 measures the accuracy of classifiers that were trained using deceptive servers, and then tested on attacks against *unpatched* servers. Attacks are uniformly distributed across all synthetic attack classes and variants described in §6.1. Each result is compared (in parentheses) against the same experiment performed without any deception. The results show that leveraging deception yields an 8–22% increase in classification accuracy, with an 8–20% increase in true positives and a 5–41% reduction in false positives. Env-SVM achieves 97% accuracy with almost no false positives (0.01%).

These significant gains demonstrate that the detection models of each classifier learned from deception-enhanced data generalize beyond data collected in decoys. This showcases the classifier's ability to detect previously unseen attack variants. DeepDig thus enables administrators to add an additional level of protection to their entire network, including hosts that cannot be promptly patched, via the adoption of a honey-patching methodology.

Figure 6 shows that as the number of training attack classes (which are proportional to the number of vulnerabilities honey-patched) increases, a steep improvement in the true positive rate is observed, reaching an average above 93% for Ens-SVM, while average false positive rate in all experiments remains low (< 1%). This demonstrates that deception has a *feature-enhancing* effect—the IDS learns from the prolonged adversarial interactions to detect more attacks.

Testing on an "unknown" vulnerability. We also measured our approach's ability to detect a *previously unseen*, unpatched remote code execution exploit (CVE-2017-5941) carrying attack payloads (classes 17–22) resembling the payloads that have been used to exploit honey-patched vulnerabilities (CVE-2014-6271). In this experiment, CVE-2017-5941 is used as an *n*-day vulnerability for which no patch has been applied. The resulting 98.6–99.8% *tpr* and 0.01–0.67% *fpr* show that crook-sourcing helps the classifier learn attack patterns unavailable at initial deployment, but revealed by deceived adversaries during decoy interactions, to learn exploits for which the classifier was not pre-trained.

Red teaming validation. Table 4 summarizes detection accuracies against the red team. In this experiment, we incrementally trained our previously trained model with new attack instances collected from live decoys, and used it to detect human attacks against unpatched servers. The accuracy rates are much higher against human opponents than against the synthetic attacks, indicating that our synthetic data constitutes a challenging test. This may be in part because replicating the high diversity of the synthetic attacks would require an extremely large-scale human study.

False alarms. Figure 7 plots the false positive rates for classifiers that have undergone 30 incremental training iterations, each with 1–30 normal/attack instances per class. With just a few attack instances (≈ 5 per attack class), the false positive rates drop to almost zero, demonstrating that DeepDig's continuous feeding back of attack samples into classifiers greatly reduces false alarms.

6.3 Base Detection Analysis

In this section we measure the success of DeepDig in detecting intrusions in the realistic scenario where attacks are a small fraction of the interactions. Although risk-level attribution for cyber attacks is difficult to quantify in general, we use the results of a prior study [25] to approximate the probability of attack occurrence for

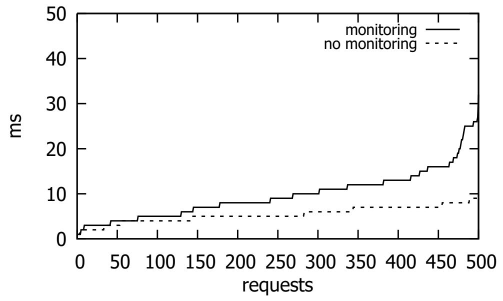

Figure 8: DeepDig performance overhead measured in average round-trip times (workload ≈ 500 req/s)

the specific scenario of *targeted attacks against business and commercial organizations*. The study's model assumes a determined attacker leveraging one or more exploits of known vulnerabilities to penetrate a typical organization's internal network, and approximates the *prior* of a directed attack to $P_A = 1\%$ (based on real-world threat statistics).

To estimate the success of intrusion detection, we use a *base detection rate* (*bdr*) [40], expressed using the Bayes theorem:

$$P(A|D) = \frac{P(A)\ P(D|A)}{P(A)\ P(D|A) + P(\neg A)\ P(D|\neg A)]}, \tag{5}$$

where A and D are random variables denoting the occurrence of a targeted attack and the detection of an attack by the classifier, respectively. We use *tpr* and *fpr* as approximations of $P(D|A)$ and $P(D|\neg A)$, respectively.

The final columns of Tables 3–4 present the *bdr* for each classifier, assuming $P(A) = P_A$. The parenthesized comparisons show how our approach overcomes a significant practical problem in intrusion detection research: Despite exhibiting high accuracy, typical IDSes are rendered ineffective when confronted with their extremely low base detection rates. This is in part due to their inability to eliminate false positives in operational contexts. In contrast, the *fpr*-reducing properties of deception-enhanced defense facilitate much more effective detection of intrusions in realistic settings, with *bdr* increases of up to 97%.

6.4 Monitoring Performance

To assess the performance overhead of DeepDig's monitoring capabilities, we used *ab* (Apache HTTP server benchmarking tool) to create a massive user workload (more than 5,000 requests in 10 threads) against two web server containers, one deployed with network and system call monitoring and another unmonitored.

Figure 8 shows the results, where web server response times are ordered ascendingly. Our measurements show average overheads of 0.2×, 0.4×, and 0.7× for the first 100, 250, and 500 requests, respectively, which is expected given the heavy workload profile imposed on the server. Since server computation accounts for only about 10% of overall web site response delay in practice [70], this corresponds to observable overheads of about 2%, 4%, and 7% (respectively).

While such overhead characterizes feasibility, it is irrelevant to deception because unpatched, patched, and honey-patched servers are all slowed equally by the monitoring activity. The overhead therefore does not reveal which apparent vulnerabilities in a given server instance are genuine patching lapses and which are deceptions, and it does not distinguish honey-patched servers from servers that are slowed by any number of other factors (e.g., fewer computational resources).

7 DISCUSSION

Role of deception. Our approach facilitates supervised learning, whose widespread use in the domain of intrusion detection has been impeded by many challenges involving the manual labeling of attacks and the extraction of security-relevant features [17, 69]. Results demonstrate that even short-term deceptive responses to cyberattacks can significantly ameliorate both of these challenges. Just a few strategically chosen honey-patched vulnerabilities accompanied by an equally small number of honey-patched applications provide a machine learning-based IDS sufficient data to perform substantially more accurate intrusion detection, thereby enhancing the security of the entire network. This suggests that deception can and should play a more significant role in machine learning-based IDS deployments.

Generalization. The results presented in §6 show that our approach substantially improves the accuracy of intrusion detection, reducing false alarms to much more practical levels. This is established experimentally with moderate- to large-scale synthetic attacks and a small-scale red teaming study. Future work should explore larger numbers of attack classes and larger (e.g., industrial scale) datasets to simulate threats to high-profile targets. Due to the high-dimensional nature of the collected data, we chose OAML and SVM in Bi-Di and N-Gram. However, our approach is agnostic to the feature set and classification model; therefore, future work should study the effectiveness of deception for enhancing a variety of learning frameworks.

An avenue of future work is to leverage system call arguments in addition to the features we collected. A common technique is to use pairwise similarity between arguments (as sequences) of different streams [17], and then implement a *k*-NN (*k*-Nearest Neighbors) algorithm with longest common subsequence (LCS) as its distance metric. Generally, packet- and system-level data are very diverse and contain other discriminating features that should be explored.

Online training. The flood of data that is continuously streamed into a typical IDS demands methods that support fast, online classification. Prior approaches update the classification model incrementally using training batches consisting of one or more training instances. However, this strategy necessitates frequently re-training the classifier, and requires a significant number of instances per training. Future research should investigate more appropriate conditions for re-training the model. *Change point detection* (CPD) [34] is one promising approach to determine the optimal re-training predicate, based on a dynamic sliding window that tracks significant changes in the incoming data, and therefore resists concept-drift failures.

Class imbalance. Standard concept-learning IDSes are frequently challenged with imbalanced datasets [35]. Such class imbalance problems arise when benign and attack classes are not equally represented in the training data, since machine learning algorithms tend to misclassify minority classes. To mitigate the effects of class imbalance, sampling techniques have been proposed [20], but they often discard useful data (in the case of under-sampling), or lead to poor generalizations (in the case of oversampling). This scarcity of realistic, balanced datasets has hindered the applicability of machine learning approaches for web intrusion detection. By feeding back labeled attack traces into the classifier, DeepDig alleviates this data drought and enables the generation of adequate, balanced datasets for classification-based intrusion detection.

Intrusion detection datasets. One of the major challenges in evaluating intrusion detection systems is the dearth of publicly available datasets, which is often aggravated by privacy and intellectual property considerations. To mitigate this problem, security researchers often resort to synthetic dataset generation, which affords the opportunity to design test sets that validate a wide range of requirements. Nonetheless, a well-recognized challenge in custom dataset generation is how to capture the multitude of variations and features manifested in real-world scenarios [11]. Our evaluation approach builds on recent breakthroughs in dataset generation for IDS evaluation [13] to create statistically representative workloads that resemble realistic web traffic, thereby affording the ability to perform a meaningful evaluation of IDS frameworks.

Establishing a straight comparison of our results to prior work is frustrated by the fact that the majority of machine learning-based intrusion detection techniques are still tested on extremely old datasets [1, 69], and approaches that account for encrypted traffic are scarce [44]. For instance, recently-proposed SVM-based approaches for network intrusion detection have reported true positive rates in the order of 92% for the DARPA/KDD datasets, with false positive rates averaging 8.2% [54, 82]. Using the model discussed in §6.3, this corresponds to an approximate base detection rate of only 11%, in contrast to 99.06% estimated for our approach. However, the assumptions made by DARPA/KDD do not reflect the contemporary attack protocols and recent vulnerabilities targeted in our evaluation, so this might not be a fair comparison. Future work should consider reevaluating these prior approaches using updated datasets reflective of modern attacks, for reproducible comparisons.

8 RELATED WORK

8.1 ML-based Intrusion Detection

Machine learning-based IDSes (cf., [17, 31, 57, 58, 63]) find patterns that do not conform to expected system behavior, and are typically classified into *host-based* and *network-based* approaches.

Host-based detectors recognize intrusions in the form of anomalous system call trace sequences, in which co-occurrence of events is key to characterizing malicious behavior. For example, malware activity and privilege escalation often manifest specific system call patterns [17]. Seminal work in this area has analogized intrusion detection via statistical profiling of system events to the human immune system [29, 36]. This has been followed by a number of related approaches using histograms to construct profiles of normal behavior [55]. Another frequently-used approach employs a *sliding window* classifier to map sequences of events into individual output values [21, 78], converting sequential learning into a classic machine learning problem. More recently, long call sequences have been studied to detect attacks buried in long execution paths [68].

Network-based approaches detect intrusions using network data. Since such systems are typically deployed at the network perimeter, they are designed to find patterns resulting from attacks launched by external threats, such as attempted disruption or unauthorized access [11]. Network intrusion detection has been extensively studied in the literature (cf., [1, 11]). Major approaches can be grouped into classification-based (e.g., SVM [28], [7], Bayesian network [45]), information-theoretic [51], and statistical [46–48] techniques.

Network-based intrusion detection systems can monitor a large number of hosts at relatively low cost, but they are usually opaque to *local* or *encrypted* attacks. On the other hand, intrusion detection systems operating at the host level have complete visibility of malicious events, despite encrypted network payloads and obfuscation mechanisms [43]. Our approach therefore complements existing techniques and incorporates host- and network-based features to offer protective capabilities that can resist attacker evasion strategies and detect malicious activity bound to different layers of the software stack.

Another related area of research is *web-based* malware detection that identifies drive-by-download attacks using static analysis, dynamic analysis, and machine learning [16, 42]. In addition, other studies focus on *flow-based* malware detection by extracting features from proxy-logs and using machine learning [10].

8.2 Cyber-Deception in Intrusion Detection

Honeypots are information systems resources conceived to attract, detect, and gather attack information [71]. They are designed such that any interaction with a honeypot is likely to be malicious. Shadow honeypots [3] are a hybrid approach in which a front-end anomaly detection system forwards suspicious requests to a back-end instrumented copy of the target application, which validates the anomaly prediction and improves the anomaly detector's heuristics through feedback. Although the target and instrumented programs may share similar states for detection purposes, shadow honeypots make no effort to deceive attackers into thinking the attack was successful—attack detection and the decision of decoying attacker sessions are driven solely by the network anomaly detection component.

8.3 Feature Extraction for Intrusion Detection

A variety of feature extraction and classification techniques have been proposed to perform host- and network-based anomaly detection [56]. Extracting features from encrypted network packets has been intensively studied in the domain of *website fingerprinting*, where attackers attempt to discern which websites are visited by victims. Users typically use anonymous networks, such as Tor, to hide their destination websites [77]. However, attackers can often predict destinations by training classifiers directly on encrypted packets (e.g., packet headers only). Relevant features typically include packet length and direction, summarized as a histogram feature vector. HTML markers, percentage of incoming and outgoing packets, bursts, bandwidth, and website upload time have also been used [26, 62]. Packet-word vector approaches additionally leverage natural language processing and vector space models to convert packets to word features for improved classification [2].

Bi-Di leverages packet and uni-burst data and introduces bi-directional bursting features for better classification of network streams. On unencrypted data, host-based systems have additionally extracted features from co-occurrences and sequences of system events, such as system calls [15, 55]. DeepDig uses a hybrid scheme that combines both host- and network-based approaches via a modified ensemble technique.

9 CONCLUSION

This paper introduced, implemented, and evaluated a new approach for enhancing web intrusion detection systems with threat data sourced from deceptive, application-layer, software traps. Unlike conventional machine learning-based detection approaches, DeepDig incrementally builds models of legitimate and malicious behavior based on audit streams and traces collected from these traps. This augments the IDS with inexpensive and automatic security-relevant feature extraction capabilities. These capabilities require no additional developer effort apart from routine patching activities. This results in an effortless labeling of the data and supports a new generation of higher-accuracy detection models.

ACKNOWLEDGMENTS

The research reported herein was supported in part by ONR award N00014-17-1-2995; NSA award H98230-15-1-0271; AFOSR award FA9550-14-1-0173; an endowment from the Eugene McDermott family; NSF FAIN awards DGE-1931800, OAC-1828467, and DGE-1723602; NSF awards DMS-1737978 and MRI-1828467; an IBM faculty award (Research); and an HP grant. Any opinions, recommendations, or conclusions expressed are those of the authors and not necessarily of the aforementioned supporters.

REFERENCES

[1] Mohiuddin Ahmed, Abdun Naser Mahmood, and Jiankun Hu. 2016. A Survey of Network Anomaly Detection Techniques. *Journal of Network and Computer Applications* 60 (2016), 19–31.
[2] Khaled Alnaami, Gbadebo Ayoade, Asim Siddiqui, Nicholas Ruozzi, Latifur Khan, and Bhavani Thuraisingham. 2015. P2V: Effective Website Fingerprinting Using Vector Space Representations. In *Proceedings of the IEEE Symposium on Computational Intelligence*. 59–66.
[3] Kostas G. Anagnostakis, Stelios Sidiroglou, Periklis Akritidis, Michalis Polychronakis, Angelos D. Keromytis, and Evangelos P. Markatos. 2010. Shadow Honeypots. *International Journal of Computer and Network Security (IJCNS)* 2, 9 (2010), 1–15.
[4] Frederico Araujo and Kevin W. Hamlen. 2015. Compiler-instrumented, Dynamic Secret-Redaction of Legacy Processes for Attacker Deception. In *Proceedings of the 24th USENIX Security Symposium*.
[5] Frederico Araujo, Kevin W. Hamlen, Sebastian Biedermann, and Stefan Katzenbeisser. 2014. From Patches to Honey-Patches: Lightweight Attacker Misdirection, Deception, and Disinformation. In *Proceedings of the 21st ACM Conference on Computer and Communications Security (CCS)*. 942–953.
[6] Frederico Araujo, Mohammad Shapouri, Sonakshi Pandey, and Kevin Hamlen. 2015. Experiences with Honey-patching in Active Cyber Security Education. In *Proceedings of the 8th USENIX Conference on Cyber Security Experimentation and Test (CSET)*.
[7] Mamoun Awad, Latifur Khan, Farokh Bastani, and I-Ling Yen. 2004. An Effective Support Vector Machines (SVMs) Performance Using Hierarchical Clustering. In *Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*. 663–667.
[8] Stefan Axelsson. 1999. The Base-rate Fallacy and its Implications for the Difficulty of Intrusion Detection. In *Proceedings of the 6th ACM Conference on Computer and Communications Security (CCS)*. 1–7.
[9] Gbadebo Ayoade, Frederico Araujo, Khaled Al-Naami, Ahmad M. Mustafa, Yang Gao, Kevin W. Hamlen, and Latifur Khan. 2020. Automating Cyberdeception Evaluation with Deep Learning. In *Proceedings of the 53rd Hawaii International Conference on System Sciences (HICSS)*.
[10] Karel Bartos, Michal Sofka, and Vojtech Franc. 2016. Optimized Invariant Representation of Network Traffic for Detecting Unseen Malware Variants. In *Proceedings of the 25th USENIX Security Symposium*. 807–822.
[11] Monowar H. Bhuyan, Dhruba Kumar Bhattacharyya, and Jugal Kumar Kalita. 2014. Network Anomaly Detection: Methods, Systems and Tools. *IEEE Communications Surveys & Tutorials* 16, 1 (2014), 303–336.
[12] Avrim L. Blum and Pat Langley. 1997. Selection of Relevant Features and Examples in Machine Learning. *Artificial Intelligence* 97, 1 (1997), 245–271.
[13] Nathaniel Boggs, Hang Zhao, Senyao Du, and Salvatore J. Stolfo. 2014. Synthetic Data Generation and Defense in Depth Measurement of Web Applications. In *Proceedings of the 17th International Symposium on Recent Advances in Intrusion Detection (RAID)*. 234–254.
[14] Casey Breen, Latifur Khan, and Arunkumar Ponnusamy. 2002. Image Classification Using Neural Networks and Ontologies. In *Proceedings of the 13th International Workshop on Database and Expert Systems Applications*. 98–102.
[15] João BD Cabrera, Lundy Lewis, and Raman K Mehra. 2001. Detection and Classification of Intrusions and Faults Using Sequences of System Calls. *ACM SIGMOD Record* 30, 4 (2001), 25–34.
[16] Davide Canali, Marco Cova, Giovanni Vigna, and Christopher Kruegel. 2011. Prophiler: A Fast Filter for the Large-scale Detection of Malicious Web Pages. In *Proceedings of the 20th International World Wide Web Conference (WWW)*. 197–206.
[17] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly Detection: A Survey. *ACM Computing Surveys (CSUR)* 41, 3 (2009), 15.
[18] Chih-Chung Chang and Chih-Jen Lin. 2011. LIBSVM: A Library for Support Vector Machines. *ACM Transactions on Intelligent Systems and Technology (TIST)* 2, 3 (2011).
[19] Gal Chechik, Varun Sharma, Uri Shalit, and Samy Bengio. 2010. Large Scale Online Learning of Image Similarity Through Ranking. *Journal of Machine Learning Research (JMLR)* 11 (2010), 1109–1135.
[20] David A. Cieslak, Nitesh V. Chawla, and Aaron Striegel. 2006. Combating Imbalance in Network Intrusion Datasets. In *Proceedings of the IEEE International Conference on Granular Computing (GrC)*. 732–737.
[21] William W. Cohen. 1995. Fast Effective Rule Induction. In *Proceedings of the 12th International Conference on Machine Learning*. 115–123.
[22] Corinna Cortes and Vladimir Vapnik. 1995. Support-vector Networks. *Machine Learning* 20, 3 (1995), 273–297.
[23] Dorothy E. Denning. 1987. An Intrusion-detection Model. *IEEE Transactions on Software Engineering (TSE)* 13, 2 (1987), 222–232.
[24] Jon DiMaggio. 2015. The Black Vine Cyberespionage Group. Symantec Security Response.
[25] Dmitry Dudorov, David Stupples, and Martin Newby. 2013. Probability Analysis of Cyber Attack Paths Against Business and Commercial Enterprise Systems. In *Proceedings of the IEEE European Intelligence and Security Informatics Conference (EISIC)*. 38–44.
[26] Kevin P. Dyer, Scott E. Coull, Thomas Ristenpart, and Thomas Shrimpton. 2012. Peek-a-boo, I Still See You: Why Efficient Traffic Analysis Countermeasures Fail. In *Proceedings of the 33rd IEEE Symposium on Security & Privacy (S&P)*. 332–346.
[27] Edgescan. 2019. Vulnerability Statistics Report.
[28] Eleazar Eskin, Andrew Arnold, Michael Prerau, Leonid Portnoy, and Salvatore Stolfo. 2002. A Geometric Framework for Unsupervised Anomaly Detection. In *Applications of Data Mining in Computer Security*. Springer, 77–101.
[29] Stephanie Forrest, Steven A. Hofmeyr, Aniln Somayaji, and Thomas A. Longstaff. 1996. A Sense of Self for Unix Processes. In *Proceedings of the 17th IEEE Symposium on Security & Privacy (S&P)*. 120–128.
[30] Yang Gao, Yi-Fan Li, Swarup Chandra, Latifur Khan, and Bhavani Thuraisingham. 2019. Towards Self-adaptive Metric Learning on the Fly. In *Proceedings of the 28th International World Wide Web Conference (WWW)*. 503–513.
[31] Pedro Garcia-Teodoro, J Diaz-Verdejo, Gabriel Maciá-Fernández, and Enrique Vázquez. 2009. Anomaly-based Network Intrusion Detection: Techniques, Systems and Challenges. *Computers & Security* 28, 1 (2009), 18–28.
[32] Derek Greene and Pádraig Cunningham. 2006. Practical Solutions to the Problem of Diagonal Dominance in Kernel Document Clustering. In *Proceedings of the 23rd International Conference on Machine learning (ICML)*. 377–384.
[33] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. 2009. The WEKA Data Mining Software: An Update. *ACM SIGKDD Explorations Newsletter* 11, 1 (2009), 10–18.
[34] Ahsanul Haque, Latifur Khan, and Michael Baron. 2016. SAND: Semi-supervised Adaptive Novel Class Detection and Classification Over Data Stream. In *Proceedings of the 30th Conference on Artificial Intelligence (AAAI)*. 1652–1658.
[35] Haibo He and Edwardo A. Garcia. 2009. Learning From Imbalanced Data. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 21, 9 (2009), 1263–1284.
[36] Steven A. Hofmeyr, Stephanie Forrest, and Anil Somayaji. 1998. Intrusion Detection Using Sequences of System Calls. *Journal of Computer Security* 6, 3 (1998), 151–180.
[37] Prateek Jain, Brian Kulis, Inderjit S. Dhillon, and Kristen Grauman. 2008. Online Metric Learning and Fast Similarity Search. In *Proceedings of the 21st International Conference on Neural Information Processing Systems (NIPS)*. 761–768.
[38] Allen Jeng. 2015. Minimizing Damage From J.P. Morgan's Data Breach. *InfoSec Reading Room* (2015).
[39] Rong Jin, Shijun Wang, and Yang Zhou. 2009. Regularized Distance Metric Learning: Theory and Algorithm. In *Proceedings of the 22nd International Conference on Neural Information Processing Systems (NIPS)*. 862–870.
[40] Marc Juarez, Sadia Afroz, Gunes Acar, Claudia Diaz, and Rachel Greenstadt. 2014. A Critical Evaluation of Website Fingerprinting Attacks. In *Proceedings of the 21st ACM Conference on Computer and Communications Security (CCS)*. 263–274.

[41] Juniper Research. 2017. The Future of Cybercrime and Security: Key Takeaways and Juniper Leaderboard.
[42] Alexandros Kapravelos, Yan Shoshitaishvili, Marco Cova, Christopher Kruegel, and Giovanni Vigna. 2013. Revolver: An Automated Approach to the Detection of Evasive Web-based Malware. In *Proceedings of the 22nd USENIX Security Symposium*. 637–652.
[43] Jungwon Kim, Peter J. Bentley, Uwe Aickelin, Julie Greensmith, Gianni Tedesco, and Jamie Twycross. 2007. Immune System Approaches to Intrusion detection—A Review. *Natural Computing* 6, 4 (2007), 413–466.
[44] Tiina Kovanen, Gil David, and Timo Hämäläinen. 2016. Survey: Intrusion Detection Systems in Encrypted Traffic. In *Proceedings of the 16th International Conference on Next Generation Wired/Wireless Networking (NEW^{2}AN)*. 281–293.
[45] Christopher Kruegel, Darren Mutz, William Robertson, and Fredrik Valeur. 2003. Bayesian Event Classification for Intrusion Detection. In *Proceedings of the 19th Annual Computer Security Applications Conference (ACSAC)*. 14–23.
[46] Christopher Kruegel and Giovanni Vigna. 2003. Anomaly Detection of Web-based Attacks. In *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS)*. 251–261.
[47] Christopher Kruegel, Giovanni Vigna, and William Robertson. 2005. A Multi-model Approach to the Detection of Web-based Attacks. *Computer Networks* 48, 5 (2005), 717–738.
[48] Christopher Krügel, Thomas Toth, and Engin Kirda. 2002. Service Specific Anomaly Detection for Network Intrusion Detection. In *Proceedings of the 17th ACM Symposium on Applied Computing (SAC)*. 201–208.
[49] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep Learning. *Nature* 521 (2015), 436–444.
[50] Wenke Lee and Salvatore J. Stolfo. 1998. Data Mining Approaches for Intrusion Detection. In *Proceedings of the 7th USENIX Security Symposium*. 79–93.
[51] Wenke Lee and Dong Xiang. 2001. Information-theoretic Measures for Anomaly Detection. In *Proceedings of the 22nd IEEE Symposium on Security & Privacy (S&P)*. 130–143.
[52] Wenbin Li, Yang Gao, Lei Wang, Luping Zhou, Jing Huo, and Yinghuan Shi. 2018. OPML: A One-pass Closed-form Solution for Online Metric Learning. *Pattern Recognition* 75 (2018), 302–314.
[53] LXC. 2019. Linux Containers. http://linuxcontainers.org.
[54] Prajowal Manandhar and Zeyar Aung. 2014. Towards Practical Anomaly-based Intrusion Detection by Outlier Mining on TCP Packets. In *Proceedings of the 25th International Conference on Database and Expert Systems Applications (DEXA)*. 164–173.
[55] Carla Marceau. 2001. Characterizing the Behavior of a Program Using Multiple-length N-grams. In *Proceedings of the New Security Paradigms Workshop (NSPW)*. 101–110.
[56] Mehedy Masud, Latifur Khan, and Bhavani Thuraisingham. 2011. *Data Mining Tools for Malware Detection*. CRC Press.
[57] Mohammad M. Masud, Tahseen M. Al-Khateeb, Kevin W. Hamlen, Jing Gao, Latifur Khan, Jiawei Han, and Bhavani Thuraisingham. 2008. Cloud-based Malware Detection for Evolving Data Streams. *ACM Transactions on Management Information Systems (TMIS)* 2, 3 (2008).
[58] Mohammad M. Masud, Jing Gao, Latifur Khan, Jiawei Han, and Bhavani Thuraisingham. 2010. Classification and Novel Class Detection in Data Streams with Active Mining. In *Proceedings of the 14th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*. 311–324.
[59] MinIO. 2019. MinIO Object Storage. https://min.io/.
[60] Mockaroo. 2018. Product data set. https://www.mockaroo.com.
[61] Novetta Threat Research Group. 2016. Operation Blockbuster: Unraveling the Long Thread of the Sony Attack.
[62] Andriy Panchenko, Lukas Niessen, Andreas Zinnen, and Thomas Engel. 2011. Website Fingerprinting in Onion Routing Based Anonymization Networks. In *Proceedings of the 10th Annual ACM Workshop on Privacy in the Electronic Society (WPES)*. 103–114.
[63] Animesh Patcha and Jung-Min Park. 2007. An Overview of Anomaly Detection Techniques: Existing Solutions and Latest Technological Trends. *Computer Networks* 51, 12 (2007), 3448–3470.
[64] John C. Platt. 1999. Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods. In *Advances in Large Margin Classifiers*. MIT Press, 61–74.
[65] PyTorch. 2019. Open Source Deep Learning Platform. https://pytorch.org.
[66] Tony Sager. 2014. Killing Advanced Threats in Their Tracks: An Intelligent Approach to Attack Prevention. *InfoSec Reading Room* (2014).
[67] Selenium. 2019. Selenium Browser Automation. http://www.seleniumhq.org.
[68] Xiaokui Shu, Danfeng Yao, and Naren Ramakrishnan. 2015. Unearthing Stealthy Program Attacks Buried in Extremely Long Execution Paths. In *Proceedings of the 22nd ACM Conference on Computer and Communications Security (CCS)*. 401–413.
[69] Robin Sommer and Vern Paxson. 2010. Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. In *Proceedings of the 31st IEEE Symposium on Security & Privacy (S&P)*. 305–316.
[70] Steve Souders. 2007. *High Performance Web Sites: Essential Knowledge for Front-End Engineers*. O'Reilly.
[71] Lance Spitzner. 2002. *Honeypots: Tracking Hackers*. Addison-Wesley.
[72] Symantec. 2018. Internet Security Threat Report, Vol. 23.
[73] Sysdig. 2019. Universal System Visibility Tool. https://github.com/draios/sysdig.
[74] tcpdump. 2019. Tcpdump and Libpcap. https://www.tcpdump.org/.
[75] Chih-Fong Tsai, Yu-Feng Hsu, Chia-Ying Lin, and Wei-Yang Lin. 2009. Intrusion Detection By Machine Learning: A Review. *Expert Systems with Applications* 36, 10 (2009), 11994–12000.
[76] Emmanouil Vasilomanolakis, Shankar Karuppayah, Max Mühlhäuser, and Mathias Fischer. 2015. Taxonomy and Survey of Collaborative Intrusion Detection. *Comput. Surveys* 47, 4 (2015).
[77] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. 2014. Effective Attacks and Provable Defenses for Website Fingerprinting. In *Proceedings of the 23rd USENIX Security Symposium*.
[78] Christina Warrender, Stephanie Forrest, and Barak Pearlmutter. 1999. Detecting Intrusions Using System Calls: Alternative Data Models. In *Proceedings of the 20th IEEE Symposium on Security & Privacy (S&P)*. 133–145.
[79] Shiming Xiang, Feiping Nie, and Changshui Zhang. 2008. Learning a Mahalanobis Distance Metric for Data Clustering and Classification. *Pattern Recognition* 41, 12 (2008), 3600–3612.
[80] Danfeng Yao, Xiaokui Shu, Long Cheng, Salvatore J. Stolfo, Elisa Bertino, and Ravi Sandhu. 2017. *Anomaly Detection as a Service: Challenges, Advances, and Opportunities*. Morgan & Claypool Publishers.
[81] Jim Yuill, Dorothy Denning, and Fred Feer. 2006. Using Deception to Hide Things From Hackers: Processes, Principles, and Techniques. *Journal of Information Warfare* 5, 3 (2006), 26–40.
[82] Ming Zhang, Boyi Xu, and Dongxia Wang. 2015. An Anomaly Detection Model for Network Intrusions Using One-class SVM and Scaling Strategy. In *Proceedings of the 11th International Conference on Collaborative Computing: Networking, Applications, and Worksharing (CollaborateCom)*. 267–278.