Automating Cyberdeception Evaluation with Deep Learning

Gbadebo Ayoade[†] Frederico Araujo[‡] Khaled Al-Naami[†]
Ahmad M. Mustafa[†] Yang Gao[†] Kevin W. Hamlen[†] Latifur Khan[†]

[†]The University of Texas at Dallas
{*gbadebo.ayoade, khaled.al-naami, ahmad.mustafa, yang.gao, hamlen, lkhan*}*@utdallas.edu*

[‡]IBM Research
frederico.araujo@ibm.com

Abstract

A machine learning-based methodology is proposed and implemented for conducting evaluations of cyberdeceptive defenses with minimal human involvement. This avoids impediments associated with deceptive research on humans, maximizing the efficacy of automated evaluation before human subjects research must be undertaken.

Leveraging recent advances in deep learning, the approach synthesizes realistic, interactive, and adaptive traffic for consumption by target web services. A case study applies the approach to evaluate an intrusion detection system equipped with application-layer embedded deceptive responses to attacks. Results demonstrate that synthesizing adaptive web traffic laced with evasive attacks powered by ensemble learning, online adaptive metric learning, and novel class detection to simulate skillful adversaries constitutes a challenging and aggressive test of cyberdeceptive defenses.

1. Introduction

Cyberdeceptive defenses are increasingly vital for protecting organizational and national critical infrastructures from asymmetric cyber threats. Market forecasts predict an over $2 billion industry for cyberdeceptive products by 2022 [1], including major product releases by Rapid7, TrapX, LogRhythm, Attivo, Illusive Networks, Cymmetria, and many others in recent years [2].

These new defense layers are rising in importance because they enhance conventional defenses by shifting asymmetries that traditionally burden defenders back on attackers. For example, while conventional defenses invite adversaries to find just one critical vulnerability to successfully penetrate the network, deceptive defenses challenge adversaries to discern which vulnerabilities among a sea of apparent vulnerabilities (many of them traps) are real. As attacker-defender asymmetries increase with the increasing complexity of networks and software, deceptive strategies for leveling those asymmetries will become increasingly essential for scalable defense.

Robust evaluation methodologies are a critical step in the development of effective cyberdeceptions; however, cyberdeception evaluation is frequently impeded by the difficulty of conducting experiments with appropriate human subjects. Capturing the diversity, ingenuity, and resourcefulness of real APTs tends to require enormous sample sizes of rare humans having exceptional skills and expertise. Human deception research raises many ethical dilemmas that can lead to long, difficult approval processes [3]. Even when these obstacles are surmounted, such studies are extremely difficult to replicate (and therefore to confirm), and results are often difficult to interpret given the relatively unconstrained, variable environments that are the contexts of real-world attacks.

Progress in cyberdeceptive defense hence demands efficient methods of conducting preliminary yet meaningful evaluations without humans in the loop. Human subject evaluation can then be reserved as a final, high-effort validation of the most promising, mature solutions.

Toward this goal, this paper proposes and critiques a machine learning-based approach for evaluating cyberdeceptive software defenses without human subjects. Although it is extremely difficult to emulate human decision-making automatically for synthesizing attacks, our approach capitalizes on the observation that in practice cyber attackers rely heavily upon mechanized tools for offense. For example, human bot masters rely primarily upon reports delivered by automated bots to assess attack status and reconnoiter targets, and they submit relatively simple commands to the botnet to carry out complex killchains that are largely mechanized as malicious software. In such scenarios, deceiving the mechanized agents goes a long way toward deceiving their human masters. Automating the machine-versus-machine part of the deception evaluation is therefore both feasible and useful.

We therefore propose an evaluation methodology that leverages machine learning to (1) generate realistic streams

URI: https://hdl.handle.net/10125/63975
978-0-9981331-3-3

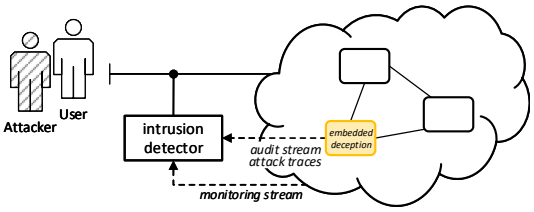

Figure 1: Deceptive IDS training overview

of synthetic traffic comprised of benign interactions and attacks based on real threat data and vulnerabilities, and (2) automatically adapt the synthetic traffic in an effort to evade observed (possibly deceptive) responses to the attacks. The goal is to obtain the maximum evaluative power of adaptive deceptive defenses without explicit human adversarial engagement.[1] As a case study, we apply our technique to evaluate a network-level intrusion detection system (IDS) equipped with embedded honeypots at the application layer. Our contributions include:

- We present the design of a framework for replay and generation of web traffic that statistically mutates and injects scripted attacks into the output streams to more effectively train, test, and evaluate deceptive, concept-learning IDSes.
- We evaluate our approach on large-scale network and system events gathered via simulation over a test bed built atop production web software, including the Apache web server, OpenSSL, and PHP.
- We propose an adaptive deception detector to cope with adaptive defenses to detect outliers in the presence of concept-evolving data streams.

Section 2 first characterizes deceptive defenses that are suitable evaluation subjects of our approach. Section 3 details our technical approach, followed by our evaluation case study and findings in Section 4. Related work is highlighted in Section 5 and Section 6 concludes.

2. Background

2.1. Deception-enhanced Intrusion Detection

Our evaluation approach targets IDS defenses enhanced with deceptive attack-responses (e.g., [4, 5, 6]). Figure 1 depicts the general approach. Unlike conventional intrusion detection, deception-enhanced IDSes incrementally build a model of *legitimate* and *malicious* behavior based on audit streams and attack traces collected from successful deceptions. The deceptions leverage user interactions at the network, endpoint, or application layers to solicit extra communication with adversaries and waste their resources, misdirect them, or gather intelligence. This augments the classifier with security-relevant feature extraction capabilities not available to typical network intrusion detectors.

For example, *honey-patches* [4, 7, 8] introduce application layer deceptions by selectively replacing software security patches with decoy vulnerabilities. Attempted exploits transparently redirect the attacker's session to a decoy environment where the exploit is allowed to succeed. This allows the system to observe subsequent phases of the attacker's killchain without risk to genuine assets.

2.2. Challenges in IDS Evaluation

One of the major challenges for evaluation of deceptive IDSes is the general inadequacy of static attack datasets, which cannot react to deceptive interactions. Testing deceptive defenses with these datasets renders the deceptions useless, missing their value against reactive threats.

To mitigate this problem, a method of dynamic attack synthesis is required. A suitable solution must learn a model of how adversarial agents are likely to react based on their reactions to similar feedback during real-world interactions mined from real attack data. The accuracy of such predictions depends upon the complexity of deceptive responses and the decision logic of the adversaries. For example, when defensive responses are binary (*viz.* accept or reject) or a finite list of error messages, accurate prediction is more feasible than when the output space is large (e.g., arbitrary textual messages). Likewise, automated agents tend to have high predictability (e.g., learnable by emulating their software logic on desired inputs), whereas human agents are far more difficult to predict.

3. Technical Approach

We aim to quantitatively assess the resiliency of adaptive, deceptive, concept-learning defenses for web services against adaptive adversaries. Our approach therefore differs from works that measure only absolute IDS accuracy. We first present our approach for generating web traffic to replay normal and malicious user behavior, which we harness to automatically generate training and test datasets for attack classification (§3.1). We then discuss the testing harness and analysis used to investigate the effects of different attack classes and varying numbers of attack instances on the predictive power and accuracy of intrusion detection (§3.2).

3.1. Traffic Analysis

Our evaluation methodology seeks to create realistic, end-to-end workloads and attack killchains to functionally test cyberdeceptive defenses embedded in commodity server applications and process decoy telemetry for feature

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

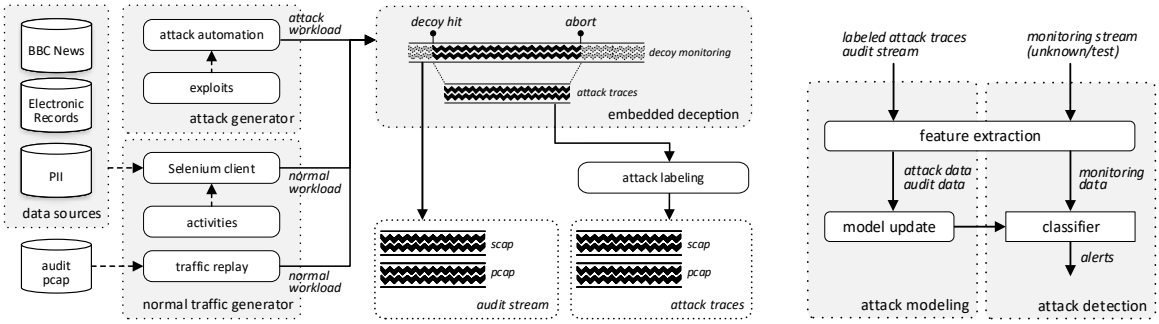

Figure 2: Overview of (a) automated workload generation for cyberdeception evaluation, and (b) deceptive IDS training and testing.

extraction and IDS model evolution. Figure 2a shows an overview of our traffic generation framework. It streams *encrypted* legitimate and malicious workloads onto endpoints enhanced with embedded deceptions, resulting in labeled audit streams and attack traces (collected at decoys) for training set generation.

Workload Generation. Rather than evaluating deception-enhanced IDSes with existing, publicly available intrusion datasets (which are inadequate for the reasons outlined in §2.2), our evaluation interleaves attack and normal traffic following prior work on defense-in-depth [8,9], and injects benign payloads as data into attack packets to mimic evasive attack behavior. The generated traffic contains attack payloads against realistic exploits (e.g., weaponizing recent CVEs for reconnaissance and initial infection), and our framework automatically extracts labeled features from the monitoring network and system traces to (re-)train the classifiers.

Legitimate workload. The framework uses both *real* user sessions and *automated simulation* of various user actions to compose legitimate traffic. Real interactions comprise web traffic that is monitored and recorded as *audit pcap* data in the targeted operational environment (e.g., regular users in a local area network). The recorded sessions are replayed by our framework and streamed as normal workload onto endpoints embedding deceptions.

These regular data streams are enriched with simulated interactions, which are created by automating complex user actions on typical web application, leveraging *Selenium* [10] to automate user interaction with web applications (e.g., clicking buttons, filling out forms, navigating a web page). To create realistic workloads, our framework feeds from online data sources, such as the BBC text corpus [11], online text generators [12] for personally identifiable information (e.g., usernames, passwords), and product names to populate web forms. To ensure diversity, we statistically sample the data sources to obtain user input values and dynamically generate 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.

Our implementation defines different customizable user activities that can be repeated with varying data feeds and scheduled to simulate different workload profiles and temporal patterns. These include web page browsing, e-commerce website navigation, blog posting, and interacting with a social media web application. The setup includes common web software stacks, such as CGI web applications and PHP-based Wordpress applications hosted on a monitored Apache web server.

Attack workload. Attack traffic is generated based on real vulnerabilities. The procedure harnesses a collection of scripted attacks (crafted using Bash, Python, Perl, or Metasploit scripts) to inject malicious client traffic against endpoints in the tested environment. Attacks can be easily extended and tailored to specific test scenarios during evaluation design, without modifications to the framework, which automates and schedules attacks according to parametric statistical models defined by the targeted evaluation (e.g., prior probability of an attack, attack rates, reconnaissance pattern).

In the case study reported in §4, multiple exploits for recent CVEs were scripted to carry out different malicious activities (i.e., different attack payloads), such as leaking password files and invoking shells on the remote web server. These vulnerabilities are important as attack vectors because they range from sensitive data exfiltration to complete control and remote code execution. The post-infection payloads execute tasks such as tool acquisition, basic environment reconnaissance (e.g., active scanning with Nmap, passive inspection of system logs), password file access, root certificate exfiltration, and attempts at gaining access to other machines in the network.

Monitoring & Threat Data Collection. Our framework tracks two lifecycle events associated with monitored decoys: upon a *decoy hit*, the framework records the timestamp that denotes the beginning of an attack session (i.e., when a security condition is met). After the

corresponding *abort* event arrives (i.e., session disconnection), the monitoring component extracts the session trace (delimited by the two events), labels it, and stores the trace outside the decoy for subsequent feature extraction. Since the embedded deceptions should only host attack sessions, precisely collecting and labeling their traces (at both the network and OS level) is effortless using this strategy.

Our approach distinguishes between three separate 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 contains network packets and OS events captured at each server environment. To minimize performance impact, we use two powerful and efficient software monitors: *sysdig* (to track system calls and modifications made to the file system), and *tcpdump* (to monitor ingress and egress of network packets). Specifically, monitored data is stored outside decoy environments to avoid possible tampering with collected data.

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 [13] 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 in the host environment. The event data streams originated from the monitored hosts are exported to a high-performance, distributed S3-compatible object storage server [14], designed for large-scale data infrastructures.

3.2. Data Analysis

Using the continuous audit stream and incoming attack traces as labeled input data, our approach enables concept-learning IDSes to incrementally build supervised models that are able to capture legitimate and malicious behavior. As illustrated in Figure 2b, the raw training set (composed of both audit stream and attack traces) is piped into a feature extraction component that selects relevant, non-redundant features 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.

Network Packet Analysis. Each packet transmitted and received forms the basic unit of information flow for our packet-level analysis. *Bidirectional* (Bi-Di) features are extracted from the patterns observed on this network data. Due to encrypted network traffic opacity, features are extracted from TCP packet headers. Packet data length and transmission time are extracted from network sessions. We extract histograms of packet lengths, time intervals, and directions. To reduce the dimension of the generated features, we apply *bucketization* to group TCP packets into correlation sets based on frequency of occurrence.

Uni-burst features include burst size, time, and count of groups of packets transmitted consecutively in one TCP window. *Bi-burst features* include time and size attributes of consecutive groups of packets transmitted in two consecutive TCP windows.

System Call Analysis. In order to capture events from within the host, we extract features from system-level OS events. Event types include *open*, *read*, *select*, etc., with the corresponding process name. Leveraging N-Gram feature extraction, we build a histogram of the N-Gram occurrences. N-Gram is a contiguous sequences of system call events. We consider four types of such N-Gram: *uni-events*, *bi-events*, *tri-events*, and *quad-events* are sequences of 1–4 consecutive system call events (respectively).

3.3. Classification

Ensemble SVM. After feature extraction, we leverage SVM to classify both Bi-DI and N-Gram features. SVM uses a convex optimization approach by mapping non-linearly separated data to a higher dimensional linearly distinguishing space. With the new linearly separable space, SVM can separate positive (attack) and negative (benign) training instances by a hyperplane with the maximum gap possible. Prediction is assigned based on which side of the hyperplane an instance resides.

The models built from Bi-Di and N-Gram are combined into an ensemble to obtain a better predictive model. Rather than concatenating the features from both Bi-Di and N-Gram, which has the drawback of introducing normalization issues, the ensemble combines multiple classifiers to obtain a better outcome by majority voting. In our case, for each classification output by the classifier models, we obtain the predicted label and the confidence probability of each of the individual classifiers. The outcome of the classifier with the maximum confidence is picked for the predicted instance.

Confidence is rated using Platt scaling [15], which uses the following sigmoid-like function to compute the classification confidence:

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

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 how much a classifier is confident about assigning a label to a testing point.

Online Adaptive Metric Learning. OAML [16] is a recently advanced deep learning approach that improves instance separation by transforming input features to a new latent space. This generates a new latent feature space where similar instances are closer together and dissimilar instances are separated farther. It extends online similarity metric learning (OML) [17,18,19,20,21], which employs *pairwise* and *triplet* constraints: 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.

We choose OAML since non-adaptive OML usually learns a pre-selected linear metric (e.g., Mahalanobis distance [22]) that lacks the complexity to learn non-linear semantic similarities among class instances, which are prevalent in intrusion detection scenarios. In addition, using a non-adaptive method results in a fixed metric which suffers from bias to a specific dataset. OAML overcomes these disadvantages by adapting its metric learning model 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) which consist of a set of hidden layers where the output is fed as input to an independent metric-embedding layer (MEL). The MELs output an n-dimensional vector in an embedded space that clusters similar instances. The importance of model generated by each MEL layer is determined by a metric weight assigned to each MEL. The output of this embedding is used as input to a k-NN classifier, as 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 OAML 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 3 illustrates our ANN. Let $E_\ell \in \{E_0, \ldots, E_L\}$ denote the ℓ^{th} metric model in OAML (i.e., the network

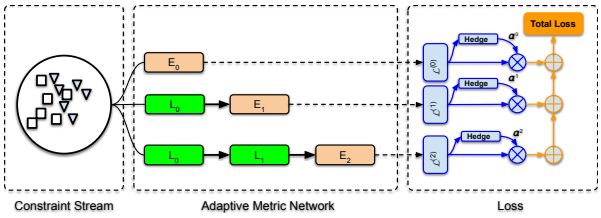

Figure 3: OAML network structure. Each layer L_i is a linear transformation output to a rectified linear unit (ReLU) activation. Embedding layers E_i connect input or hidden layers. Linear model E_0 maps the input feature space to the embedding space.

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 $h^{(\ell)}$ is 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

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

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 [16] to estimate $\mathcal{L}^{(\ell)}$ and update $\Theta^{(\ell)}$, $W^{(\ell)}$ and $\alpha^{(\ell)}$.

Novel Class Detection. Novel classes may appear at any time in real-world monitoring streams (e.g., new attacks and new deceptions). To cope with such *concept-evolving* data streams, we include a deception-enhanced novel class detector that extends traditional classifiers with automatic detection of novel classes before the true labels of the novel class instances arrive.

Data stream classification. Novel class detection observes that data points belonging to a common class are closer to each other (*cohesion*), yet far from data points belonging to other classes (*separation*). Building upon ECSMiner [23, 24], our approach segments data streams into equal, fixed-sized *chunks*, each containing a set of monitoring traces, efficiently buffering chunks for online processing. When a buffer is examined for novel classes, the classification algorithm looks for strong cohesion among outliers in the buffer and large separation between outliers and training data. When strong cohesion and separation are found, the classifier declares a novel class.

Training & model update. A new classifier is trained on each chunk and added to a fixed-sized ensemble of M classifiers, leveraging audit and attack instances (traces). After each iteration, the set of $M+1$ classifiers are ranked based on their prediction accuracies on the latest data chunk, and only the first M classifiers remain in the ensemble. The ensemble is continuously updated following this strategy and thus modulates the most recent concept in the incoming data stream, alleviating adaptability issues associated with concept drift [23]. Unlabeled instances are classified by majority vote of the ensemble's classifiers.

Classification model. Each classifier in the ensemble uses a k-NN classification, deriving its input features from Bi-Di and N-Gram feature set models. Rather than storing all data points of the training chunk in memory, which is prohibitively inefficient, we optimize space utilization and time performance by using a semi-supervised clustering technique based on Expectation Maximization (E-M) [25]. This minimizes both intra-cluster dispersion and cluster impurity, and caches a summary of each cluster (centroid and frequencies of data points belonging to each class), discarding the raw data points.

Feature transformation. To make the learned representations robust to partial corruption of the input patterns and improve classification accuracy, abstract features are generated from the original feature space during training via a *stacked denoising autoencoder* (DAE) [26, 27] using the instances of the first few chunks in the data stream. Stacked DAE builds a deep neural network that aims to capture the statistical dependencies between the inputs by reconstructing a clean input from a corrupted version of it, thus forcing the hidden layers to discover more robust features (yielding better generalization) and prevent the classifier from learning the identity (while preserving the information about the input).

Figure 4 illustrates our approach. The first step creates a corrupted version $\widetilde{x}$ of input $x \in \mathbb{R}^d$ using *additive Gaussian noise* [28]. In other words, a random value v_k is added to each feature in x: $\widetilde{x}_k = x_k + v_k$ where $k = [1 \ldots d]$ and $v_k \sim \mathcal{N}(0, \sigma^2)$ (cf., [29]). The output

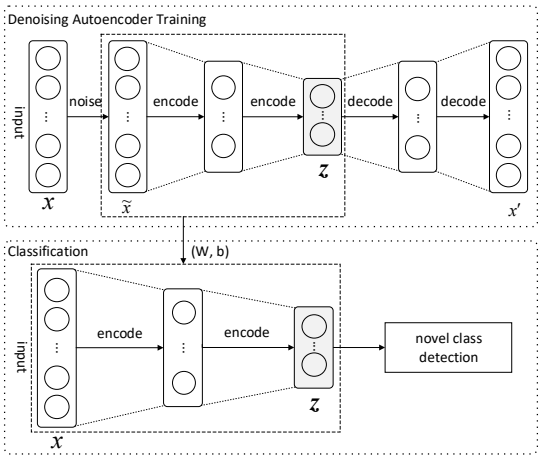

Figure 4: Overview of feature transformation

of the training phase is a set of weights W and bias vectors b. We keep the learned weights and baises to transform the feature values of the subsequent instances of the stream. After transforming the features of stream instances, these are fed back into our novel class detector for training.

One-class SVM Ensemble. Our approach builds an ensemble of one-class SVM classifiers. One-class SVM is an unsupervised learning method that learns the decision boundary of training instances and predicts whether an instance is inside it. We train one classifier for each class. For instance, if our training data consists of instances of k classes, our ensemble must contain k one-class SVM classifiers, each trained with one of the k class's instances.

During classification, once a new unlabeled instance x emerges, we classify it using all the one-class SVM classifiers in the ensemble.

We build our ensemble using the first few chunks of instances. During the classification of the stream, once novel class's instances emerge, we train a new one-class SVM classifier with the new novel class instances. Then we add the new classifier to the ensemble.

Attacker Evasion. To properly challenge deceptive defenses, it is essential to simulate adversaries who adapt and obfuscate their behaviors in response to observed responses to their attacks. Attackers employ various evasion techniques to bypass protections, including packet size padding, packet timing sequence morphing, and modifying data distributions to resemble legitimate traffic.

In our study, we considered three encrypted traffic evasion techniques published in the literature: *Pad-to-MTU* [30], *Direct Target Sampling* [31], and *Traffic Morphing* [31]. Pad-to-MTU (pMTU) adds extra bytes to each packet length until it reaches the Maximum Transmission Unit (1500 bytes in the TCP protocol). Direct Target Sampling (DTS) is a distribution-based technique that uses statistical random sampling from benign traffic fol-

lowed by attack packet length padding. Traffic Morphing (TM) is similar to DTS but it uses a convex optimization methodology to minimize the overhead of padding. Each of these are represented using the traffic modeling approach detailed in §3.1 and analyzed using the machine learning approaches detailed above.

4. Case Study

As a case study of our evaluation approach, we applied it to test DEEPDIG [8], an IDS platform protecting deceptively honey-patched [4] web servers. DEEPDIG is an anomaly-based IDS that improves its detection model over time by feeding attack traces that trigger honey-patch traps back into a classifier. This core feature makes it an advanced, intelligent defense that cannot be properly evaluated using static datasets.

4.1. Implementation

We implemented our evaluation framework atop 64-bit Linux. The data generation component is implemented using Python and Selenium [10]. The monitoring controller is 350 lines of node.js code, and leverages *tcpdump* [32], *editcap* [33], and *sysdig* [13] for network and system call tracing and preprocessing. The machine learning modules are implemented in Python using 1200 lines of scikit-learn [34] code for data preprocessing and feature generation. The novel class detection component comprises of about 250 lines of code to reference the Theano deep learning library [35] and ECSMiner [23]. Finally, the OAML module was implemented with 500 lines of PyTorch [36] deep learning development framework code.

4.2. Experimental Setup

The traffic generator was deployed on 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.

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 (§4.3).

Table 1: Base detection rate percentages for an approximate targeted attack scenario ($P_A \approx 1\%$) [37]

Classifier	*tpr*	*fpr*	*acc*	F_2	*bdr*
1SVM Bi-Di	77.78	41.23	68.96	59.69	1.87
1SVM N-Gram	84.88	5.11	88.57	88.38	14.47
VNG++	46.81	0.83	69.25	52.31	36.29
Panchenko	47.69	0.17	70.04	53.24	73.92
Bi-Di OML	91.00	0.01	91.14	90.00	98.92
N-Gram OML	65.00	0.01	88.58	80.00	98.50
Bi-Di SVM	79.00	0.78	89.88	78.69	50.57
N-Gram SVM	92.42	0.01	96.89	93.84	99.05
Ens-SVM	**93.63**	**0.01**	**97.00**	**94.89**	**99.06**

Model Parameters. In our experiments, SVM uses RBF kernel with Cost 1.3×10^5, and gamma is 1.9×10^{-6}. OAML employs a ReLU network with $n = 200$, $L = 1$, $k = 5$, learning rate of 0.3, lr decay of 1×10^{-4}, and ADAM optimizer. One-class SVM uses RBF kernel and Nu = 0.5. Novel class detection uses the DAE denoising autoencoder with $L = 2$, input feature size = 6000, first layer = $\frac{2}{3}$ of input size, second layer = $\frac{1}{3}$ of input size, and additive Gaussian noise where $\sigma = 1.1$.

4.3. IDS Evaluation

Using the dataset shown in Table 2, we trained and assessed the individual performances of the classifiers presented in §3.3 and two other state-of-the-art supervised approaches, VNG++ [30] and Panchenko (P) [38], which are widely used in the literature on encrypted traffic analysis [39]. To obtain different baselines, 1SVM, VNG++, and Panchenko were trained non-deceptively (i.e., trained exclusively on normal data, as outlier detectors), while the OML and SVM classifiers were trained atop DEEPDIG.

Table 1 summarizes our results, which confirm our intuition that deceptively-trained IDSes are able to curtail false positives and achieve better detection rates than non-deceptive outlier detectors by 25.1–97.2%. Figure 5 also illustrates the performances of the different IDS approaches when trained incrementally with the first 1–16 attack classes. Specifically, the results shown in Fig. 5(a)–(d) underscore perennial challenges encountered in conventional anomaly-based intrusion detection: reduced detection accuracy and high incidence of false alarms. Conversely, Ens-SVM is able to achieve high accuracy after being trained with just a few attack classes (Fig. 5(e)–(f)).

Base Detection Analysis. We measure the success of detecting intrusions assuming a realistic scenario in which attacks are only a small fraction of the interactions. Al-

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 info leak	OpenSSL
14	CVE-2010-0740	DoS via NULL pointer deref	OpenSSL
15	CVE-2010-1452	DoS via request lacking path	Apache
16	CVE-2016-7054	DoS via heap buffer overflow	OpenSSL
17–22	CVE-2017-5941	System hijack (6 variants)	Node.js

though risk-level attribution for cyber attacks is difficult to quantify in general, we use the results of a recent study [37] to approximate the probability of attack occurrence for 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 as $P_A = 1\%$ based on real-world threat statistics.

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

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

where A and D are random variables denoting targeted attacks and their detection by the classifier, respectively.

Table 1 presents the accuracy values and *bdr* for each classifier, assuming $P(A) = P_A$. The numbers expose a practical problem with the defense that is typical in intrusion detection research: Despite having high accuracy values, the IDS is ineffective when confronted with extremely low base detection rates. This is in part due to its inability to eliminate false positives in operational contexts where the attacks are such a tiny fraction of the total traffic available for learning.

4.4. Resistance to Attack Evasion Techniques

Table 3 shows the results of the deceptive defense against our evasive attack techniques compared with results when no evasion is attempted. In each experiment, the classifier is trained and tested with 1800 normal instances and 1600 *morphed* attack instances.

Our evaluation shows that the tpr drops slightly and the fpr increases with the introduction of attacker evasion techniques. This shows that the system could resist some of the evasions but not all. However, we can conclude that an increase in the frequency of classifier retraining may be needed to accommodate the drop in performance. This may be a challenge as shorter time interval results in fewer data points to retrain the classifier to maintain their detection performance.

Table 3: Detection performance in adversarial settings

Evasion technique	*tpr*	*fpr*	*acc*	F_2
No evasion	**93.63**	**0.01**	**97.00**	**99.06**
pMTU	75.84	0.96	85.78	79.57
DTS	82.78	6.02	87.58	84.91
TM	79.29	6.17	85.52	81.91

Table 4: Novel attack class detection performance

Features	Classifier	*tpr*	*fpr*
Bi-Di	OneSVM	44.06	31.88
	DAE & OneSVM	76.54	85.61
	ECSMiner	74.91	26.66
	DAE & ECSMiner	**84.73**	**0.01**
N-Gram	OneSVM	54.25	45.13
	DAE & OneSVM	80.09	71.49
	ECSMiner	76.36	34.89
	DAE & ECSMiner	**89.67**	**2.95**

4.5. Novel Class Detection Accuracy

To test the ability of our novel class classifier to detect novel classes emerging in the monitoring stream, we split the input stream into equal-sized chunks. A chunk of 100 instances is classified at a time where one or more novel classes may appear along with existing classes. We measured the *tpr* (total incremental number of actual novel class instances classified as novel classes) and the *fpr* (total number of existing class instances misclassified as belonging to a novel class).

Table 4 shows the results for OneSVM and ECSMiner. Here ECSMiner outperforms OneSVM in all measures. For example, for Bi-Di features, ECSMiner observes an *fpr* of 26.66% while OneSVM reports an *fpr* of 31.88%, showing that the binary-class nature of ECSMiner is capable of modeling the decision boundary better than OneSVM. To achieve better accuracy, we augmented ECSMiner with extracted deep abstract features using our stacked denoising autoencoder approach (DAE & ECSMiner). For DAE, we used two hidden layers (where the number of units in the first hidden layer is $2/3$ of the original features, and the number of units in the second hidden layer is $1/3$ of the first hidden layer units). For the additive Gaussian noise, which is used for data corruption, we assigned $\sigma = 1.1$. As a result, *fpr* reduced to a minimum (0.01%), showing a substantial improvement over ECSMiner. Notice that using the abstract features with OneSVM does not help as shown in the table.

While effective in detecting concept drifts, our novel class detection technique requires a (semi-)manual labeling of novel class instances. In our future work, we plan to investigate how to automatically assign labels (e.g., deceptive vs. non-deceptive defense response) to previously unseen classes.

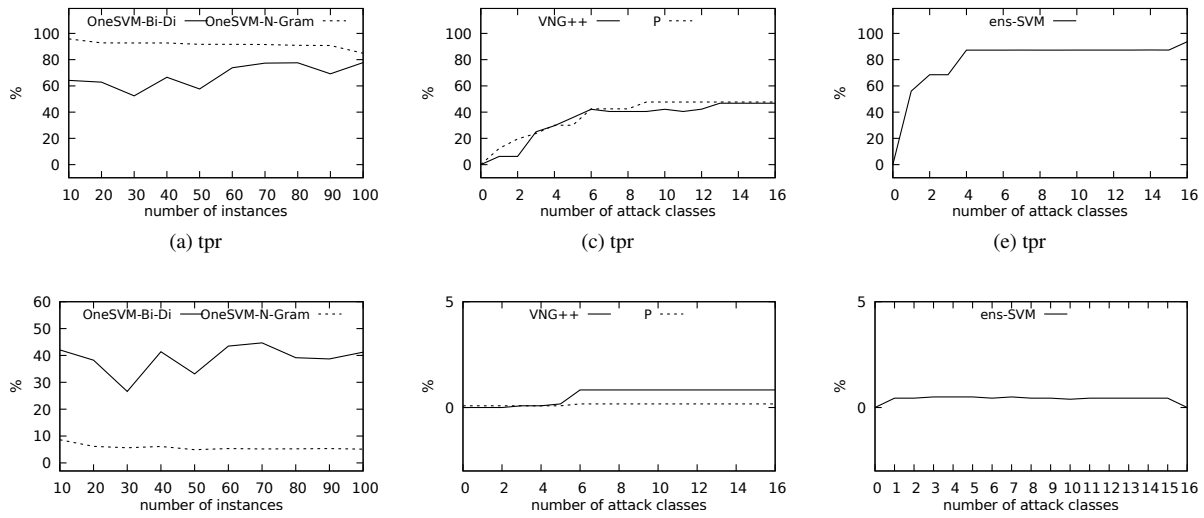

Figure 5: Baseline evaluations: (a)–(b) OneSVM Bi-Di and OneSVM N-Gram, and (c)–(d) VNG++ and Panchenko. Classification: (e)–(f) Ens-SVM classification tpr for 0–16 attack classes.

5. Related Work

Deception-enhanced IDS. Our evaluation methodology is designed to assess adaptive, deception-enhanced IDS systems protecting web services. Examples from the literature include shadow honeypots [41, 42], Argos [43], Honeycomb [44], and DAW [45].

Synthetic Attack Generation. Our approach was inspired by WindTunnel [9], which is a synthetic data generation framework for evaluating (non-deceptive) security controls. WindTunnel acquires data from network, system call, file access, and database queries and evaluates which of the data sources provides better signal for detection remote attacks. The DETER [46] testbed provides a framework for designing repeatable experiments for evaluating security of computer systems.

6. Conclusion

Effective evaluation of cyberdeceptive defenses is notoriously challenging. Our attempts to conduct such an evaluation without resorting to human subjects experimentation indicates that dynamic, synthetic attack generation powered by deep learning is a promising approach. In particular, a combination of ensemble learning leveraging multiple classifier models, online adaptive metric learning, and novel class detection suffices to model aggressively adaptive adversaries who respond to deceptions in a variety of ways. Our case study evaluating an advanced, deceptive IDS shows that the resulting synthetic attacks can expose both strengths and weaknesses in modern embedded-deception defenses.

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] Mordor Intelligence, "Global cyber deception market," tech. rep., Mordor Intelligence, 2018.

[2] G. Sadowski and R. Kau, "Improve your threat detection function with deception technologies," Tech. Rep. G00382578, Gartner, March 2019.

[3] D. Baumrind, "IRBs and social science research: The costs of deception," *IRB: Ethics & Human Research*, vol. 1, no. 6, pp. 1–4, 1979.

[4] F. Araujo, K. W. Hamlen, S. Biedermann, and S. Katzenbeisser, "From patches to honey-patches: Lightweight attacker misdirection, deception, and disinformation," in *Proc. ACM Conf. Computer and Communications Security*, pp. 942–953, 2014.

[5] J. Avery and E. H. Spafford, "Ghost patches: Fake patches for fake vulnerabilities," in *Proc. IFIP Int. Conf. ICT Systems Security and Privacy Protection*, pp. 399–412, 2017.

[6] S. Crane, P. Larsen, S. Brunthaler, and M. Franz, "Booby trapping software," in *Proc. New Security Paradigms Work.*, pp. 95–106, 2013.

[7] F. Araujo, M. Shapouri, S. Pandey, and K. Hamlen, "Experiences with honey-patching in active cyber security

education," in *Proc. Work. Cyber Security Experimentation and Test*, 2015.

[8] F. Araujo, G. Ayoade, K. Al-Naami, Y. Gao, K. W. Hamlen, and L. Khan, "Improving intrusion detectors by crook-sourcing," in *Proc. Annual Computer Security Applications Conf.*, December 2019.

[9] N. Boggs, H. Zhao, S. Du, and S. J. Stolfo, "Synthetic data generation and defense in depth measurement of web applications," in *Proc. Int. Sym. Recent Advances in Intrusion Detection*, pp. 234–254, 2014.

[10] Selenium, "Selenium browser automation." http://www.seleniumhq.org, 2016.

[11] D. Greene and P. Cunningham, "Practical solutions to the problem of diagonal dominance in kernel document clustering," in *Proc. Int. Conf. Machine learning*, pp. 377–384, 2006.

[12] Mockaroo, "Product data set." www.mockaroo.com, 2018.

[13] Sysdig, "Linux system exploration and troubleshooting tool." https://github.com/draios/sysdig, 2019.

[14] MinIO, "MinIO object storage." https://min.io, 2019.

[15] J. C. Platt, *Probabilities for SV Machines*, ch. 5, pp. 61–74. Neural Information Processing, MIT Press, 2000.

[16] Y. Gao, Y.-F. Li, S. Chandra, L. Khan, and B. Thuraisingham, "Towards self-adaptive metric learning on the fly," in *Proc. Int. World Wide Web Conf.*, pp. 503–513, 2019.

[17] W. Li, Y. Gao, L. Wang, L. Zhou, J. Huo, and Y. Shi, "OPML: A one-pass closed-form solution for online metric learning," *Pattern Recognition*, vol. 75, pp. 302–314, 2018.

[18] G. Chechik, V. Sharma, U. Shalit, and S. Bengio, "Large scale online learning of image similarity through ranking," *J. Machine Learning Research*, vol. 11, pp. 1109–1135, 2010.

[19] P. Jain, B. Kulis, I. S. Dhillon, and K. Grauman, "Online metric learning and fast similarity search," in *Proc. Annual Conf. Neural Information Processing Systems*, pp. 761–768, 2008.

[20] R. Jin, S. Wang, and Y. Zhou, "Regularized distance metric learning: Theory and algorithm," in *Proc. Annual Conf. Neural Information Processing Systems*, pp. 862–870, 2009.

[21] C. Breen, L. Khan, and A. Ponnusamy, "Image classification using neural networks and ontologies," in *Proc. Int. Work. Database and Expert Systems Applications*, pp. 98–102, 2002.

[22] S. Xiang, F. Nie, and C. Zhang, "Learning a mahalanobis distance metric for data clustering and classification," *Pattern Recognition*, vol. 41, no. 12, pp. 3600–3612, 2008.

[23] M. M. Masud, T. M. Al-Khateeb, L. Khan, C. Aggarwal, J. Gao, J. Han, and B. Thuraisingham, "Detecting recurring and novel classes in concept-drifting data streams," in *Proc. Int. IEEE Conf. Data Mining*, pp. 1176–1181, 2011.

[24] T. Al-Khateeb, M. M. Masud, K. M. Al-Naami, S. E. Seker, A. M. Mustafa, L. Khan, Z. Trabelsi, C. Aggarwal, and J. Han, "Recurring and novel class detection using class-based ensemble for evolving data stream," *IEEE Trans. Knowledge and Data Engineering*, vol. 28, no. 10, pp. 2752–2764, 2016.

[25] M. M. Masud, J. Gao, L. Khan, J. Han, and B. Thuraisingham, "A practical approach to classify evolving data streams: Training with limited amount of labeled data," in *Proc. Int. Conf. Data Mining*, pp. 929–934, 2008.

[26] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proc. Int. Conf. Machine Learning*, pp. 1096–1103, 2008.

[27] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *J. Machine Learning Research*, vol. 11, pp. 3371–3408, 2010.

[28] M. Chen, K. Q. Weinberger, F. Sha, and Y. Bengio, "Marginalized denoising auto-encoders for nonlinear representations," in *Proc. Int. Conf. Machine Learning*, pp. 1476–1484, 2014.

[29] Y. Bengio, *Learning Deep Architectures for AI*. Now Foundations and Trends, 2009.

[30] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton, "Peek-a-boo, I still see you: Why efficient traffic analysis countermeasures fail," in *Proc. IEEE Sym. Security & Privacy*, pp. 332–346, 2012.

[31] C. V. Wright, S. E. Coull, and F. Monrose, "Traffic morphing: An efficient defense against statistical traffic analysis," in *Proc. IEEE Network and Distributed Security Sym.*, pp. 237–250, 2009.

[32] TcpDump, "Network packet capture and analyzer." www.tcpdump.org, 2019.

[33] Linux Manual, *editcap: Edit and/or Translate the Format of Capture Files*, 2019. https://linux.die.net/man/1/editcap.

[34] Scikit-learn, "Scikit-learn: Machine learning in Python." https://scikit-learn.org, 2011.

[35] Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions," *arXiv*, vol. abs/1605.02688, 2016.

[36] PyTorch, "An open source deep learning framework." https://pytorch.org, 2019.

[37] D. Dudorov, D. Stupples, and M. Newby, "Probability analysis of cyber attack paths against business and commercial enterprise systems," in *Proc. IEEE European Intelligence and Security Informatics Conf.*, pp. 38–44, 2013.

[38] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel, "Website fingerprinting in onion routing based anonymization networks," in *Proc. Annual ACM Work. Privacy in the Electronic Society*, pp. 103–114, 2011.

[39] T. Kovanen, G. David, and T. Hämäläinen, "Survey: Intrusion detection systems in encrypted traffic," in *Int. Conf. Internet of Things, Smart Spaces, and Next Generation Networks and Systems*, pp. 281–293, 2016.

[40] M. Juarez, S. Afroz, G. Acar, C. Diaz, and R. Greenstadt, "A critical evaluation of website fingerprinting attacks," in *Proc. ACM Conf. Computer and Communications Security*, pp. 263–274, 2014.

[41] K. G. Anagnostakis, S. Sidiroglou, P. Akritidis, K. Xinidis, E. Markatos, and A. D. Keromytis, "Detecting targeted attacks using shadow honeypots," in *Proc. USENIX Security Sym.*, 2005.

[42] K. G. Anagnostakis, S. Sidiroglou, P. Akritidis, M. Polychronakis, A. D. Keromytis, and E. P. Markatos, "Shadow honeypots," *Int. J. Computer and Network Security*, vol. 2, no. 9, pp. 1–15, 2010.

[43] G. Portokalidis, A. Slowinska, and H. Bos, "Argos: An emulator for fingerprinting zero-day attacks for advertised honeypots with automatic signature generation," *ACM SIGOPS Operating Systems Review*, vol. 40, no. 4, pp. 15–27, 2006.

[44] C. Kreibichi and J. Crowcroft, "Honeycomb – creating intrusion detection signatures using honeypots," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 1, pp. 51–56, 2004.

[45] Y. Tang and S. Chen, "Defending against internet worms: A signature-based approach," in *Proc. Annual Joint Conf. IEEE Computer and Communications Societies*, pp. 1384–1394, 2005.

[46] T. Benzel, R. Braden, D. Kim, C. Neuman, A. Joseph, K. Sklower, R. Ostrenga, and S. Schwab, "Experience with DETER: A testbed for security research," in *Proc. Int. Conf. Testbeds and Research Infrastructures for the Development of Networks and Communities*, 2006.