Methods for Host-Based Intrusion Detection with Deep Learning

JOHN H. RING IV, University of Vermont, Department of Computer Science, USA and MassMutual, Data Science
COLIN M. VAN OORT, University of Vermont, Department of Computer Science, USA
SAMSON DURST, University of Vermont, Department of Computer Science, USA
VANESSA WHITE, University of Vermont, Department of Computer Science, USA
JOSEPH P. NEAR, University of Vermont, Department of Computer Science, USA
CHRISTIAN SKALKA, University of Vermont, Department of Computer Science, USA

Host-based Intrusion Detection Systems (HIDS) automatically detect events that indicate compromise by adversarial applications. HIDS are generally formulated as analyses of sequences of system events such as bash commands or system calls. *Anomaly-based* approaches to HIDS leverage models of normal (aka baseline) system behavior to detect and report abnormal events, and have the advantage of being able to detect novel attacks. In this paper we develop a new method for anomaly-based HIDS using deep learning predictions of sequence-to-sequence behavior in system calls. Our proposed method, called the *ALAD* algorithm, aggregates predictions at the *application* level to detect anomalies. We investigate the use of several deep learning architectures, including WaveNet and several recurrent networks. We show that *ALAD* empowered with deep learning significantly outperforms previous approaches. We train and evaluate our models using an existing dataset, ADFA-LD, and a new dataset of our own construction, PLAID. As deep learning models are black box in nature we use an alternate approach, allotaxonographs, to characterize and understand differences in baseline vs. attack sequences in HIDS datasets such as PLAID.

CCS Concepts: • **Security and privacy → Intrusion detection systems**; • **Computing methodologies → Neural networks**; Ensemble methods.

Additional Key Words and Phrases: Host-Based Intrusion Detection Systems; Deep Learning; System Calls

1 INTRODUCTION

In this work we improve the state of the art for *Host-based Intrusion Detection Systems* (HIDS) utilizing *anomaly-detection. Intrusion Detection Systems* (IDS) aim to automatically detect events indicating system compromise by malicious adversaries. Due to the growing importance of security threats, this problem has received considerable attention both in academic research [30] and from industry [53, 54, 57]. HIDS are a class of Intrusion Detection Systems (IDS) that monitor a computer system's internals and interfaces to detect intrusions. Systems that utilize anomaly-detection model normal system behavior and report abnormal events. The primary alternative to anomaly-based IDSs is *signature-based*. Signature based approaches operate similarly to a virus scanner: they report events matching the signature of a known attack. For example, the MITRE ATT&CK Framework [55]

Authors' addresses: John H. Ring IV, jhring@uvm.edu, University of Vermont, Department of Computer Science, Burlington, VT, USA, 05401, MassMutual, Data Science; Colin M. Van Oort, cvanoort@uvm.edu, University of Vermont, Department of Computer Science, Burlington, VT, USA, 05401; Samson Durst, Samson.Durst@uvm.edu, University of Vermont, Department of Computer Science, Burlington, VT, USA, 05401; Vanessa White, vwhite1@uvm.edu, University of Vermont, Department of Computer Science, Burlington, VT, USA, 05401; Joseph P. Near, jnear@uvm.edu, University of Vermont, Department of Computer Science, Burlington, VT, USA, 05401; Christian Skalka, ceskalka@uvm.edu, University of Vermont, Department of Computer Science, Burlington, VT, USA, 05401.

2576-5337/2021/1-ART1
https://doi.org/10.1145/3461462

is a set of signatures, expressed as rules for detecting intrusions, that can be used to flag events for further examination. Unlike signature-based approaches, anomaly-based approaches can detect novel attacks, as they are identifying changes in behavior rather than a specific attack.

Network-based Intrusion Detection Systems (NIDS), the primarily alternative to HIDS, examine *network* events (i.e. traffic between hosts), rather than events occurring on a single host, and are thus distinct from HIDS. NIDS have traditionally been simpler to deploy than HIDS, since they do not require modifying individual hosts. However, as important services increasingly migrate to the cloud—where the network is under the control of the cloud provider—deploying a network-based approach for intrusion detection is often not feasible. The relative importance of HIDS research in the intrusion detection space is therefore increasing with the use of cloud computing. We chose to focus on anomaly-based HIDS to create systems compatible with modern cloud deployments that can protect against zero-day attacks.

Automated methods for HIDS are generally formulated as analyses of sequences of system events such as bash commands or system calls [30]. System calls are the interface for userspace programs to request services from the operating system's kernel, such as starting a new process or reading a file. In HIDS research, system call sequences are used as a proxy for understanding the behavior of a running program—we assume that a malicious program will produce a very different pattern of system calls than baseline execution of a benign program. We focus on the use of machine learning to distinguish between malicious and baseline behavior in sequences of system calls.

1.1 Problem Statement & Contributions

Previous work has developed HIDS that operate on individual traces of system call sequences [21, 34] using publicly available datasets [10, 29, 41, 42]. Some of these prior works are also based on anomaly detection [8, 13, 58–61]. All of these works consider system call traces generated by an individual process; however, modern applications often use multiple processes, and modern attacks can impact one or more of these processes. Furthermore, existing system call corpora used to develop these HIDS are limited and outdated. Thus, the problems we address are how to modernize anomaly-based HIDS by incorporating analysis of multi-process applications, how to develop algorithms and evaluation methods more relevant to modern systems and attacks, and overall how to achieve more accurate detection of modern attacks.

We address these problems as follows. *First*, we present a novel approach for building HIDS based on unsupervised deep learning. State-of-the-art in this domain demonstrates that models based on Long Short Term Memory (LSTM) [21], and Gated Recurrent Unit (GRU) [34] architectures outperform prior SVM-based approaches and hence are the most promising technology in this space. The key technical contribution of our approach is an application-level classifier, called *ALAD* (*A*pplication-*L*evel *A*nomaly *D*etection), to distinguish between baseline and malicious behavior. *ALAD* groups system call sequences by *program*—rather than by *process*, as was done in previous work [21, 34]. *ALAD* is simple to implement, and in our experiments produces a statistically significant improvement in classification compared to previous work. We describe the *ALAD* approach in Section 4.5.

Second, we collect and release a new dataset of system call sequences, with modern attacks on multi-process applications, used to support the development of our approach and validate our results. Our new dataset, called PLAID, contains sequences from six modern exploits and penetration techniques as well as a large collection from normal operation. We discuss the creation of PLAID in Section 3.

The *third* main contribution of our paper is the application and evaluation of modern sequence-to-sequence neural network architectures for anomaly detection. In Section 4, we compare a state-of-the-art architecture, WaveNet [43], with replications of the LSTMs and GRUs used in prior work, using both *ALAD* and the trace-level classifiers developed in previous work. We demonstrate our results on PLAID as well as the Australian Defence Force Academy Linux Dataset (ADFA-LD) [10], used by several closely related works [21, 34]. We completed 540

training and evaluation trials over combinations of dataset, model, and replicate. To our knowledge this is the largest comparison of deep learning models used in HIDS to date. We provide open source repositories for all datasets and code[1] to facilitate reproducibility.

In addition, we address a common critique of deep learning, that it is "black-box", in the sense that it structurally obfuscates model details and does not provide practitioners with insights about *why* it works. We show in Section 7, that recent techniques in corpora "divergence" visualization can still provide useful insights into datasets. Specifically, we explore our new dataset along with the popular ADFA-LD to observe differences between normal and malicious sequences. This helps to explain the effectiveness of anomaly detection in this application.

In summary, our primary contributions are as follows:

(1) Application Level Anomaly Detection (ALAD), a new classifier for *groups* of system call sequences.
(2) PLAID, a new dataset of modern system call sequences and attacks.
(3) A comparison of modern sequence-to-sequence neural network architectures for anomaly detection.
(4) The use of rank-turbulence divergence to visualize differences in system-call n-grams.

Note that (3) also subsumes a comparison with historical work, since [21, 34] already demonstrated superiority of deep learning approaches as compared to other historical approaches.

2 BACKGROUND & RELATED WORK

Intrusion detection systems (IDS) aim to automatically detect events indicating system compromise by malicious adversaries and have been studied since at least 1980 [4]. Liu and Lang provide a comprehensive taxonomy of the systems developed since then. IDS are typically classified according to their *sources of data* and *detection methods*.

Network- vs. host-based intrusion detection. There are two major categories of data sources. Network-based intrusion detection systems (NIDS) are deployed at the network level, and detect intrusions by examining network traffic. Host-based intrusion detection systems (HIDS), which are the subject of this work, are deployed on a single host and detect intrusions by examining events on that individual host. NIDS have traditionally received more attention (e.g. [3, 17, 25, 35, 36, 40, 46, 47, 49, 66, 68, 70–73]) because they are easier to deploy, more efficient, and capable of detecting threats across multiple hosts. HIDS have the advantage of being deployable in a cloud setting, in which the cloud provider controls the network infrastructure, and are capable of detecting intrusions that do not produce abnormal network traffic. Our work focuses on HIDS.

Data & datasets. Our work is focused on detecting intrusions using sequences of system calls. System calls are the interface for userspace programs to request services from the operating system's kernel, such as starting a new process or reading a file. Forrest et al. first proposed using these sequences to detect intrusions, by collecting information about "normal" patterns of system calls and detecting system call sequences that deviate from these patterns. Datasets of system call sequences include both *baseline* and *attack* sequences. Baseline sequences are collected from programs running normally; attack sequences are collected from compromised programs behaving abnormally (e.g. while an exploit is being used to attack the program).

Datasets of system call sequences are difficult to construct; as a result, most work in this area is evaluated on just four datasets:

- The DARPA Intrusion Detection Dataset [29] (1998/1999)
- The KDD 99 Dataset [42] (1999)
- The UNM System Call Dataset [41] (1998)
- The ADFA-LD Dataset [10] (2012)

[1] https://gitlab.com/jhring/uvm_ids

Unfortunately, the DARPA, KDD, and UNM datasets are too old to be of practical use as representative of modern host processes and attacks [37]. The ADFA-LD (Australian Defence Force Academy Linux Dataset [10]) dataset was specifically designed to address limitations of previously-collected datasets. In particular, they captured system call traces on a server running a modern operating system (Linux) with realistic workloads (e.g. web browsing and word processing), and attack sequences generated via real vulnerabilities in commonly-used software. For these reasons, the ADFA-LD dataset is often used for HIDS research, and previous work has demonstrated that this realism translates into a much more challenging learning task, suggesting that realistic datasets are vital for designing systems for practical deployment.

Nonetheless, the ADFA-LD dataset has a number of shortcomings. Since its release in 2012, typical workloads on Linux servers have changed, so the dataset is no longer reflective of typical server behavior. The dataset was captured on an i386 host, which though common at the time are rare in modern production environments. This is important because the system calls used by i386 and x86_64 systems differ substantially which makes it difficult to directly compare or integrate ADFA-LD traces with those collected on modern systems. Finally, the normal traces appear to be more reflective of a workstation, rather than server environment and are underspecified. Each attack sequence is labeled with the process which generated it, but the baseline sequences are not similarly labeled—so it is impossible to know what program was used to generate each sequence.

Signature- vs. anomaly-based methods. As mentioned earlier, there are two major methods of detection in HIDS research: *signature-based* methods and *anomaly-based* methods. Signature-based methods are commonly used to detect malware [5, 6, 63]; though they may also be used to detect known patterns of behavior that indicate an intrusion [36, 38]. These methods typically have low false-positive rates and are efficient, but they can only detect known attacks. Anomaly-based methods detect abnormal behavior by comparing against a model of normal behavior; they have higher false positive rates, but are capable of detecting brand-new attacks. Anomaly-based methods have been applied both to sequences of system calls and to other kinds of intrusion detection [8, 13, 58–61]. Our work focuses on anomaly-based intrusion detection.

IDS based on machine learning. A number of machine learning-based intrusion detection systems have been proposed by other authors. Liu and Lang provide a survey of these results. Machine learning approaches based on *supervised learning* (e.g. [2, 36, 38]) correspond to signature-based intrusion detection: they use labeled training data including both baseline behavior and attacks to train classifiers that distinguish between the two. These approaches cannot detect new kinds of attacks. Approaches based on *unsupervised learning* (e.g. [9, 14, 15, 19, 21, 24, 34, 67]) correspond to anomaly-based intrusion detection: they train models of baseline behavior using unlabeled training data containing only baseline behavior. Our work focuses on the use of unsupervised deep learning to perform anomaly-based intrusion detection on system call sequences. Previous work in this area has used both traditional ("shallow") machine learning and deep learning to build models of benign system call sequences. For example, approaches based on Hidden Markov Models [15, 19, 24] and support vector machines (SVM) [14, 67] have both been proposed. These methods worked well on datasets collected in the 1990's but performed poorly on the more recent ADFA-LD [10]. In particular, methods that discard the ordering information in system call sequences, including clustering and "bag of system calls" approaches achieve reasonable accuracy on legacy datasets but fail on ADFA-LDDue to this recent approaches focus on techniques that leverage ordering information, of which deep-learning has been shown to be the most promising. Kim et al. compared a long short-term memory (LSTM) model which k-nearest neighbor and k-means clustering achieving state-of-the-art performance with the LSTM. Chawla et al. use a combined convolutional / recurrent (CNN / RNN) architecture, and obtain similar performance LSTMs with less training time. These deep-learning based approaches represent the state-of-the-art in anomaly-based HIDS, and we use them for comparison in our empirical evaluation.

Visualisation. Various visualization techniques have been used to aid human analysts and users in identifying suspicious activities and emerging threats in the cyber-security realm [11, 62]. Recent work in the field of Complex Systems provides analytical methods and corresponding visualizations for comparing various states of a system [12]. These advances have not previously been applied in the cyber-security domain though the divergent nature of attack vs baseline system call sequences is a natural fit for the application.

3 THE PLAID DATASET

As with all machine learning techniques for IDS, our approach to training and testing models for HIDS relies on corpora of events, in our case system calls. Since we are developing an anomaly detection system, training corpora must contain baseline and attack data as described above in Sections 1 and 2. Given the shortcomings of ADFA-LD discussed in Section 2 we developed a new dataset, named PLAID, with modern system calls, and a richer, more current set of attacks. The **P**LAID **L**ab **A**rtificial **I**ntrusion **D**ataset is an open source dataset intended to support the work described here, and to support research in the broader community. PLAID features modern exploits carried out against a contemporary Linux server deployment, and is publicly available [50].

3.1 Host Configuration

Ubuntu 18.04 LTS [32] was selected as the host Operating System (OS) for PLAID. Ubuntu is a secure modern Linux distribution, and the most popular choice of OS for use on public clouds such as AWS and Microsoft Azure.

Commonly used remote administrative services FTP and SSH [69] were installed through Ubuntu's default package manager and enabled with their default configurations. Redis Version 4.0.14 [52] an open source in-memory data structure store was manually installed on the host and configured to allow connections on the local network. A malicious client side executable [64] was placed on the machine, simulating a successful social engineering attack. Nginx Version 1.14.0 [48] and php-fpm Version 7.1.33 [18] were installed on the host and configured to serve a basic website, a common deployment of the world's most popular web server [33].

This host configuration represents a reasonable approximation of a modern production Linux server offering remote access, high performance data storage, and web hosting.

3.2 Network Setup

Our experiment testbed consists of three Virtual Machines (VMs): our host, an attack machine, and a router. The attack VM is an instance of Kali 2019 [28], a Linux distribution designed for penetration testing. We connected our attack and host VMs on a local network through a bare-bones instance of Ubuntu 18.04 LTS serving as a router. All three VMs were run using VirtualBox Version 6.1 [44] on a single physical machine. Detailed setup instructions of VM network configuration are available on the project GitLab [50].

3.3 Attack Overview

Our host machine was exploited from six different attack vectors.

(1) The **Redis attack** [39] exploits a vulnerability in the "extension" functionality provided in the Redis in-memory database to execute arbitrary code. An exploit for the vulnerability was developed in 2018 and is available in Metasploit.
(2) The **PHP-FPM** attack [26] (CVE-2019-11043) exploits a vulnerability present in the combination of nginx and php-fpm to execute arbitrary code. An exploit for this vulnerability was developed in 2019 and is available on GitHub.
(3) The **privilege escalation** attack [65] (CVE-2016-5195, also called DirtyCow) uses a malicious CSE that exploits a vulnerability in the Linux kernel to obtain a shell with root privileges.

(4) The **brute-force** attacks [27] represent the use of a traditional brute-force password-cracking application (Hydra) to discover users' passwords over SSH and FTP.

3.4 Data Collection

System call traces were generated by starting the target application with `strace`—a userspace utility capable of monitoring interactions between processes and the Linux kernel. Each exploit was run and monitored for ten trials, fully restarting all affected services between each trial. The result of each trial is a series of files containing system calls for example: `execve brk access access openat fstat mmap close. . .` Each individual file corresponds to a single process of the program's execution and is labeled with the process id.

Since the intended use of this dataset is the development of anomaly-based IDSs, we require baseline data approximating normal operation. This baseline dataset was generated by monitoring a wide variety of common operations on the host with no active attacks in progress. Specific items in the set of common operations were chosen for two reasons. The first is to be representative of the wide range of computational tasks performed in modern day enterprise environments. The second is to achieve a high degree of behavioral overlap with the previously described attacks. The chosen baseline operations are:

- Transfer of files to and from the host using FTP
- Host access via SSH and modification of configuration files
- Simulation of web traffic using Apache Bench
- Redis interactions
- Download files from the internet with curl
- Execution of `rustup`, the Rust programming language install script [23]
- PHP and Redis test suites
- Compilation of small and large programs
- Deployment of small programs that involve: reading from disk, non-trivial computation, and standard IO

We encode meta-information in the directory structure in the same manner as the ADFA dataset. The generated data was split into two top level directories- attack and baseline. Inside the attack directory is a subdirectory for each trial labeled with the exploit and trial ID. These subdirectories contain all collected system call trace files from the corresponding exploit trial. Similarly, the baseline directory contains a subdirectory for each baseline operation. These subdirectories contain all collected system call traces associated with the baseline operation.

4 DEEP LEARNING MODELS AND THE ALAD ALGORITHM

In this section we describe the ALAD algorithm, the underlying deep learning models it uses, and our evaluation and experimental methods. We also explicitly state our research hypotheses, as Hypotheses 1 and 2 below. We return to these hypotheses in Section 6 and discuss how our experimental results support or refute them.

4.1 Method Overview and Definitions

Our approach to anomaly-based intrusion detection is a two stage process similar to that of Kim et al. but differs substantially in implementation. We implement a full detection pipeline consisting of two main stages. The first stage models the system call language using deep neural networks trained exclusively on baseline data. The second stage performs anomaly prediction using the model(s) from the first stage as well as an anomaly classifier.

4.1.1 Trace Probability. The first stage in our pipeline is a system call language model, which specifies the probability distribution for the next system call in a sequence given all prior system calls in that sequence. If we have a system call trace $t = x_1, x_2, x_3, \ldots, x_n$, we can calculate the probability of the sequence occurring with

equation 1.

$$p(t) = \prod_{i=1}^{n} p(x_i | x_{1:i-1}) \tag{1}$$

Recall that each event x_i is a system call as described in Section 3. Models trained exclusively with baseline data estimate this probability distribution for a host's normal operation. Thus, we can formally define a model $\mathcal{M}$ as a mapping from traces t to a probability (real number) value. Details of the neural network architectures used, and their training methodologies can be found in Sections Sections 4.2 and 4.4 respectively.

4.1.2 Trace-Level Anomaly Detection (TLAD). The second stage in our pipeline uses the probabilities generated by the first stage to classify a trace as baseline or anomaly. Specifically, a model $\mathcal{M}$ trained on baseline sequences can be used to classify a trace t as anomalistic if it has low probability. Taking the negative log of $\mathcal{M}(t)$ (its *negative log-likelihood*) results in low values if t is not anomalistic, and high values if it is. A standard approach (e.g. [21]) to anomaly detection sets a threshold θ and classifies a trace t as anomalistic if its negative log-likelihood exceeds the threshold. Formally, trace-level anomaly detection ($TLAD$) is defined as follows, given a model $\mathcal{M}$ and threshold θ:

$$TLAD(t) = \begin{cases} 1 \text{ if } -\log(\mathcal{M}(t)) > \theta \\ 0 \text{ otherwise} \end{cases}$$

4.1.3 Application-Level Anomaly Detection (ALAD). A drawback of $TLAD$ is that it considers only a single process at a time, whereas attacks typically target *applications* and can impact multiple processes. We propose an algorithm that aggregates predictions for all processes associated with an application. As discussed above in Section 3, process traces are endowed with application meta-information in corpora, which we can use to group traces into sets A as described below in Section 4.5. Furthermore, there is nothing special about this meta-information, in particular it is easily available to any system in practice. These sets A can be provided as input to our $ALAD$ algorithm to predict whether an application is benign or malicious. Formally:

$$\begin{aligned} ALAD(\mathrm{A}) = \quad & \text{let } \{t_1, \ldots, t_n\} = \mathrm{A} \\ & \text{let } m = \text{median}(-\log \mathcal{M}(t_1), \ldots, -\log \mathcal{M}(t_n)) \\ & 1 \text{ if } m > \theta \text{ otherwise } 0 \end{aligned}$$

Figure 1 illustrates our complete pipeline using $ALAD$.

4.1.4 Research Hypotheses. With the above definitions in place, we can now state our explicit research hypotheses as follows.

Hypothesis 1. *WaveNet will outperform the LSTM and combined CNN/RNN architectures used in prior work [9, 21].*

Hypothesis 2. *ALAD will outperform TLAD as an IDS mechanism.*

We discuss the performance metrics and evaluation methodology for both $TLAD$ and $ALAD$ in Section 4.5. In Section 5 we compare the performance of several models from each architecture (WaveNet, LSTM, CNN/RNN) and show how $ALAD$ yields significant performance improvements compared to $TLAD$.

4.2 Model Architectures

Intrusion detection is a less-explored application for the machine learning community, though many advances in the field are relevant. In particular, if we formulate the anomaly-based IDS as sequence-to-sequence learning problem, then we can leverage cutting-edge techniques from an active area of research in the deep learning community. We investigate and compare several models that are adapted from recent deep learning research.

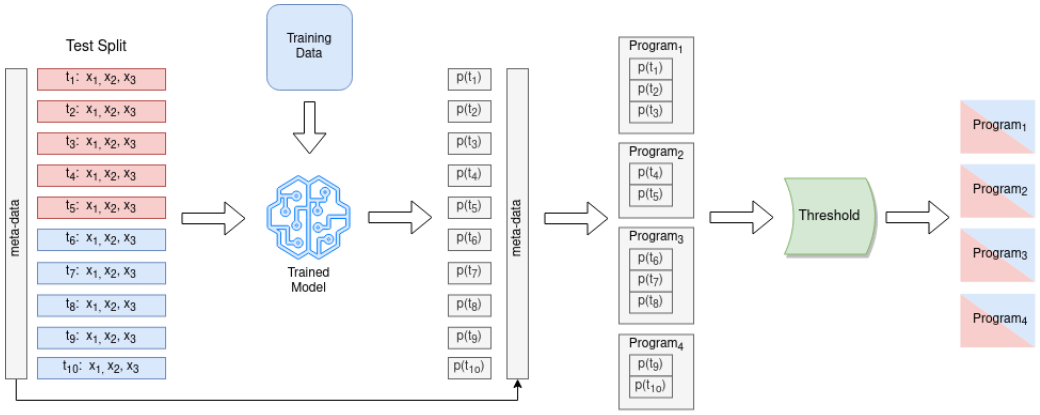


Fig. 1. An illustration of our entire pipeline. Starting on the left is a testing split consisting of attack (red) and baseline (blue) system call traces. These are submitted to a model of normal behavior- the model is a result of training exclusively on baseline traces. The model is first used to obtain the probability of occurrence of each process trace in our test set. Then we use trace metadata to group trace probabilities by application. Finally, we test the aggregation (median) of these grouped probabilities against a threshold θ resulting in a classification for each program.

All models used in this work feature the same high level layout. The integer encoded system calls are fed into a learned embedding layer. The embedding layer is followed by one of the architectures described above which outputs a probability distribution over system calls at each time step.

Our first candidate model is the WaveNet architecture [43], an audio generation model developed by Google DeepMind. WaveNet can serve as a drop-in replacement for LSTM-based architectures, which are commonly used on sequence-to-sequence problems. WaveNet employs discrete convolutions to capture context information and inform predictions, rather than the recurrent connections seen in LSTMs. This allows WaveNet to achieve superior performance with shorter training time as compared to LSTM-based architectures.

Our second and third candidates replicate the architectures from two prior approaches performing anomaly detection on ADFA-LD. They are an LSTM language architecture from Kim et al., and the combined CNN/RNN architecture from Chawla et al.. The LSTM architecture is simply a variable number of LSTM layers followed by dropout leading into a dense layer. The combined CNN/RNN model features multiple one dimension convolutional layers stacked on top of a GRU followed by a dense layer.

We implemented these architectures in Python using TensorFlow version 2.1 [1] and provide our source code on GitLab [50].

4.3 Data

We constructed separate training, testing and validation subsets for both ADFA-LD and PLAID. A separate dense integer encoding was used for each dataset as they were generated on machines using different instruction set architectures. The testing sets feature a 1:1 ratio of attack and normal traces while the training and validation sets contain exclusively normal traces.

4.3.1 ADFA-LD. The ADFA-LD data directory consists of three folders: attack, training and validation. Respectively, these contain 746, 833, and 4,373 system call traces of varying lengths. The 175 unique system calls in

ADFA-LD originally represented by a sparse integer encoding are refactored into a dense encoding for computational efficiency. The training and validation folders contain traces of normal operation while the attack folder features all attack traces.

We use this data to construct our own training, testing, and validation splits as follows. The ADFA training and validation folders are merged, consolidating all normal traces. Our test set was created by combining the attack sequences with 746 randomly selected normal sequences resulting in a 1:1 ratio of attack and normal sequences. The unused normal traces were then randomly split into training and validation sets with an 80:20 ratio resulting in 3,567 sequences selected for training and 892 for validation. Note that the original ADFA data split is not used in this paper and all further references to training, testing and validation refer to our own data splits.

4.3.2 PLAID. Pre-processing of the PLAID dataset was done similarly. PLAID consists of two top-level directories, attack and normal, named for the type of traces they contain. A total of 1,494 traces with a length less than 8 or greater than 4,495 were discarded. The remaining traces consisting of 228 unique system calls were encoded with a dense integer representation. These bounds correspond to the smallest and largest sequences present in ADFA-LD. The test set is constructed by combining all 1,145 remaining attack sequences with an equal number of randomly selected normal traces. The remaining unused normal traces are then randomly split into training and validation sets with an 80:20 ratio resulting in 29,626 sequences selected for training and 7,407 for validation.

4.3.3 Complexity. We note that size of these datasets may seem small for deep learning applications; this observation however fails to consider the size of the overall landscape. There are k^n possible system call traces of length n, where k is the vocabulary size of system calls. The Linux kernel currently features over 300 unique system calls resulting in over 27 million possibilities for traces of length three. With a length of 4,495 the landscape for the largest traces under consideration is much larger than the number of floating-point operations the universe could have performed thus far [31]. Given the complex landscape of system call traces, it is unsurprising that deep learning is required to achieve state-of-the-art performance.

4.4 Model Training & Configuration

For each architecture described in Section 4.2, we build three models with differing hyper-parameters to be used in an ensemble. The models, written M_i for $i \in \{0, 1, 2\}$ and $M \in \{\text{CNN/RNN, LSTM, WaveNet}\}$, are ordered by increasing number of parameters.

Selecting optimal hyper-parameters is a notoriously difficult task due to the large search space and computational cost of exploration. We used a Gaussian process optimizer to inform the search, aiding in the selection of hyper-parameters for our WaveNet models [56]. Ultimately we selected three WaveNet configurations all with 8 WaveNet blocks and no regularization. The models differed only by the number of filters in each convolutional layer which were 128, 256, and 512 respectively.

For the replicated architectures we used the hyper-parameters specified in their respective papers. For the LSTM architecture this was a single LSTM layer with 200 cells, a single LSTM layer with 400 cells, and two LSTM layers with 400 cells. The CNN/RNN models differed in both the number of 1D convolutions 6, 7, 8 and number of GRU units 200, 500, 600 respectively. The number of filters in each convolutional layer was set to mach it's WaveNet counterpart as the value was unspecified in the original work.

We trained all of our models using the Adam optimizer [22] with a learning rate of 0.0001. Gradient clipping with a maximum norm of 5 was applied to ensure training stability [45]. Models were trained for a fixed number of epochs, 300 and 30 for ADFA-LD and PLAID respectively with a batch size of 32. A differing number of training epochs were selected for ADFA-LD and PLAID as the latter contains over eight times for training data. By using both a fixed number of training epochs and batch size for all models we ensured they received the same number of gradient updates allowing for a fair architecture comparison. Sparse categorical cross-entropy was used as the

	Params.	Training Time (h:m:s)	Eval. Time (s)	AUC *TLAD*	FPR *TLAD* (TPR = 1)	AUC *ALAD*	FPR *ALAD* (TPR = 1)
ADFA							
CNN/RNN_0	552096	1:41:55 ± 2:29	29.6 ± 0.9	0.785 ± 0.006	0.843 ± 0.030	$0.981^{\dagger}$ ± 0.003	$0.085^{\dagger}$ ± 0.014
CNN/RNN_1	2528472	2:48:15 ± 2:14	29.5 ± 0.8	0.802 ± 0.005	0.863 ± 0.076	$0.985^{\dagger}$ ± 0.002	$0.112^{\dagger}$ ± 0.037
CNN/RNN_2	7841280	4:53:42 ± 3:25	33.1 ± 4.9	0.800 ± 0.007	0.887 ± 0.082	$0.986^{\dagger}$ ± 0.002	$0.120^{\dagger}$ ± 0.055
$LSTM_0$	391376	1:43:23 ± 2:56	**27.0** ± 0.8	0.726 ± 0.013	0.962 ± 0.068	$0.924^{\dagger}$ ± 0.013	$0.255^{\dagger}$ ± 0.060
$LSTM_1$	1422576	2:50:30 ± 3:08	27.3 ± 0.9	0.759 ± 0.017	0.873 ± 0.070	$0.964^{\dagger}$ ± 0.015	$0.118^{\dagger}$ ± 0.044
$LSTM_2$	2704176	4:36:27 ± 5:05	45.8 ± 0.5	0.793 ± 0.005	**0.795** ± 0.009	$0.983^{\dagger}$ ± 0.002	$0.074^{\dagger}$ ± 0.010
$WaveNet_0$	1111664	**1:19:33** ± 0:48	39.3 ± 3.7	0.815 ± 0.004	0.795 ± 0.050	$0.986^{\dagger}$ ± 0.001	$0.144^{\dagger}$ ± 0.062
$WaveNet_1$	4346736	2:58:54 ± 0:59	38.5 ± 3.3	**0.830** ± 0.007	0.827 ± 0.038	$\mathbf{0.993}^{\dagger}$ ± 0.001	$\mathbf{0.036}^{\dagger}$ ± 0.008
$WaveNet_2$	17206640	8:15:56 ± 3:22	45.9 ± 6.8	0.828 ± 0.017	0.837 ± 0.047	$0.993^{\dagger}$ ± 0.004	$0.048^{\dagger}$ ± 0.065
PLAID							
CNN/RNN_0	569533	1:02:58 ± 1:06	45.7 ± 7.4	0.854 ± 0.024	0.719 ± 0.209	$0.980^{\dagger}$ ± 0.009	$0.220^{\dagger}$ ± 0.189
CNN/RNN_1	2561809	1:41:30 ± 1:36	47.2 ± 3.9	0.844 ± 0.030	0.625 ± 0.147	$0.970^{\dagger}$ ± 0.017	$0.248^{\dagger}$ ± 0.199
CNN/RNN_2	7879917	2:54:41 ± 2:06	48.9 ± 5.4	0.810 ± 0.029	0.683 ± 0.143	$0.945^{\dagger}$ ± 0.039	$0.312^{\dagger}$ ± 0.161
$LSTM_0$	412629	1:01:48 ± 1:34	39.2 ± 4.0	0.886 ± 0.008	0.543 ± 0.096	$\mathbf{0.985}^{\dagger}$ ± 0.004	$\mathbf{0.185}^{\dagger}$ ± 0.056
$LSTM_1$	1465029	1:41:17 ± 2:33	**39.1** ± 6.0	0.883 ± 0.060	0.572 ± 0.136	$0.968^{\dagger}$ ± 0.097	$0.254^{\dagger}$ ± 0.169
$LSTM_2$	2746629	2:42:03 ± 3:28	67.7 ± 4.8	**0.889** ± 0.011	**0.459** ± 0.117	$0.985^{\dagger}$ ± 0.006	$0.198^{\dagger}$ ± 0.135
$WaveNet_0$	1120409	**0:51:48** ± 0:41	68.4 ± 13.8	0.796 ± 0.036	0.661 ± 0.143	$0.936^{\dagger}$ ± 0.046	$0.428^{\dagger}$ ± 0.241
$WaveNet_1$	4362265	1:51:19 ± 0:55	79.4 ± 15.1	0.772 ± 0.024	0.711 ± 0.172	$0.915^{\dagger}$ ± 0.039	0.558 ± 0.202
$WaveNet_2$	17235737	5:01:33 ± 2:35	93.2 ± 20.3	0.798 ± 0.079	0.660 ± 0.142	$0.922^{\dagger}$ ± 0.125	0.523 ± 0.296

Table 1. Accuracy comparison: *ALAD* vs. *TLAD*. We note that our proposed classification methodology (ALAD) results in a significantly higher AUC for all models under consideration. All models were trained and evaluated on a NVIDIA Tesla V100 with 32GB VRAM provided by the Vermont Advanced Computing Core. Training and performance metrics above are reported as the mean of thirty trials ± one standard deviation. In total this table summarizes the results of 540 training and evaluation trials. Total training time for the 540 models, not including hyper-parameter tuning, was over 62 days. We the relative efficiency of WaveNet whose smallest configuration had the fastest training time despite having over twice the parameters of the smallest model. *ALAD* performance metrics marked with † are statistically distinct (two-sided t-test, $p < 0.001$) from their *TLAD* counterpart. Evaluation time is how long it took the model to output the probability distribution for all sequences in the test set. Bolded results are the best in their respective column, and dataset combination.

loss function for all models. The number of parameters, training time, and other summary information for each model is detailed in Table 1.

4.5 ID Classifier Evaluation

We completed 540 evaluation trials over combinations of dataset, model, and replicate. The nine model configurations outlined in Section 4.4 were trained and evaluated for thirty replication trials on both ADFA-LD, and PLAID. Our evaluation compares the *ALAD* and *TLAD* classification algorithms using these underlying models.

Both PLAID and ADFA-LD group traces by attack trial, allowing us to aggregate traces at the application level. The ADFA-LD baseline data does not include program grouping information, so we randomly sampled synthetic programs of equal size from the normal portion of the test set. For sake of consistency, we use the same process on PLAID.

In practice, we bootstrapped the baseline groups with thirty trials for each replicate model. This mitigates statistical errors from the random sampling, such as selection of an unrepresentative grouping. Thus, the single value result is the mean of the bootstrapped operations.

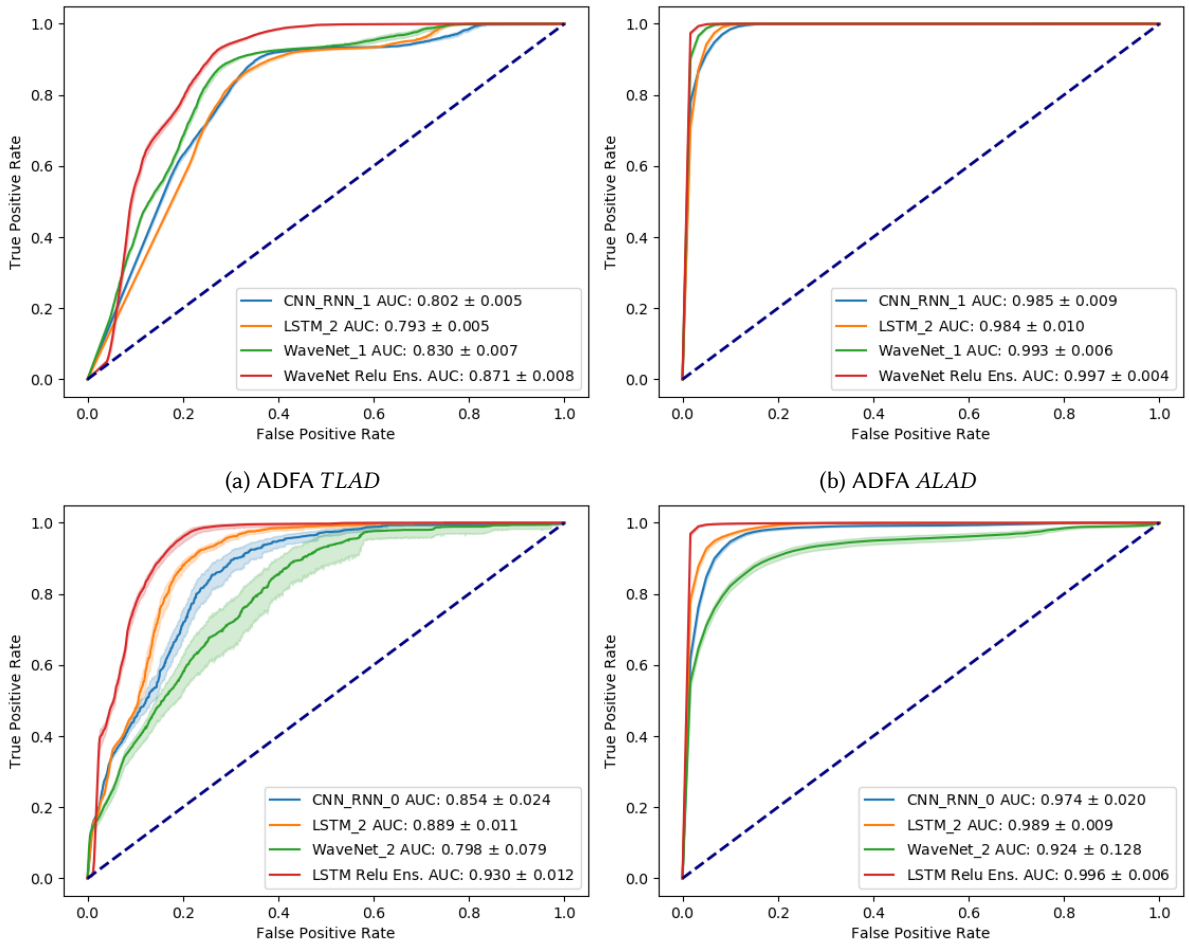
By varying the threshold value θ we obtained Receiver Operating Characteristic (ROC) curve for our classifiers—a common means of evaluating binary classification systems. The x-axis of the curve shows the false positive rate

	AUC *TLAD*	FPR *TLAD* (TPR = 1)	AUC *ALAD*	FPR *ALAD* (TPR = 1)
ADFA				
Avg. CNN/RNN	0.800 ± 0.004	0.842 ± 0.030	0.985† ± 0.002	0.125† ± 0.027
ReLU. CNN/RNN	0.800 ± 0.004	0.847 ± 0.041	0.985† ± 0.002	0.131† ± 0.041
Avg. LSTM	0.765 ± 0.006	0.903 ± 0.079	0.966† ± 0.006	0.228† ± 0.030
ReLU. LSTM	0.766 ± 0.005	0.903 ± 0.079	0.966† ± 0.006	0.231† ± 0.029
Avg. WaveNet	0.870 ± 0.008	0.712 ± 0.071	**0.998**† ± 0.001	**0.026**† ± 0.005
ReLU. WaveNet	**0.871** ± 0.008	0.692 ± 0.079	0.998† ± 0.001	0.027† ± 0.005
$Hybrid_0$	0.800 ± 0.005	0.661 ± 0.023	0.975† ± 0.004	0.153† ± 0.030
ReLU. $Hybrid_0$	0.801 ± 0.005	0.543 ± 0.050	0.976† ± 0.003	0.150† ± 0.030
$Hybrid_1$	0.820 ± 0.009	0.609 ± 0.017	0.981† ± 0.007	0.098† ± 0.039
ReLU. $Hybrid_1$	0.822 ± 0.009	0.504 ± 0.019	0.981† ± 0.007	0.100† ± 0.037
$Hybrid_2$	0.847 ± 0.005	0.547 ± 0.029	0.990† ± 0.002	0.047† ± 0.008
ReLU. $Hybrid_2$	0.848 ± 0.005	**0.485** ± 0.034	0.990† ± 0.002	0.047† ± 0.008
PLAID				
Avg. CNN/RNN	0.919 ± 0.012	0.499 ± 0.058	0.993† ± 0.004	0.119† ± 0.042
ReLU. CNN/RNN	0.919 ± 0.012	0.481 ± 0.050	0.994† ± 0.004	0.127† ± 0.051
Avg. LSTM	0.929 ± 0.020	0.394 ± 0.103	0.994† ± 0.009	0.099† ± 0.141
ReLU. LSTM	**0.930** ± 0.012	**0.380** ± 0.098	0.995† ± 0.006	0.098† ± 0.140
Avg. WaveNet	0.884 ± 0.055	0.559 ± 0.124	0.977† ± 0.055	0.197† ± 0.135
ReLU. WaveNet	0.886 ± 0.047	0.531 ± 0.058	0.978† ± 0.047	0.190† ± 0.098
$Hybrid_0$	0.929 ± 0.003	0.477 ± 0.076	**0.996**† ± 0.001	**0.063**† ± 0.046
ReLU. $Hybrid_0$	0.929 ± 0.003	0.466 ± 0.066	0.996† ± 0.001	0.065† ± 0.046
$Hybrid_1$	0.922 ± 0.037	0.512 ± 0.118	0.989† ± 0.034	0.113† ± 0.165
ReLU. $Hybrid_1$	0.923 ± 0.030	0.485 ± 0.066	0.990† ± 0.026	0.103† ± 0.125
$Hybrid_2$	0.914 ± 0.054	0.479 ± 0.120	0.986† ± 0.050	0.092† ± 0.117
ReLU. $Hybrid_2$	0.915 ± 0.049	0.459 ± 0.067	0.986† ± 0.048	0.089† ± 0.102

Table 2. Performance metrics for all ensembles under consideration. We note that *ALAD* results in a significantly higher AUC for all ensembles under consideration. Homogeneous ensembles, designated by architecture, contain all three model configurations from that architecture. Heterogeneous ensembles, termed hybrid, contain the the model from each architecture at the given configuration level. Performance metrics above are reported as the mean of thirty trials ± one standard deviation. *ALAD* performance metrics marked with † are statistically distinct (two-sided t-test, $p < 0.001$) from their *TLAD* counterpart. Bolded results are the best in their respective column, and dataset combination.

while the y-axis shows the true positive rate. In this case, the curve visualizes the trade-off between detection and false alarm rate. We summarize the performance of a model into a single value using the Area Under Curve (AUC) metric. In addition, we report the False Positive Rate (FPR) where the True Positive Rate (TPR) is one. The reported value for a given metric such as AUC (discussed below in Section 5) is the mean of all 30 replicate trials. For *ALAD* the reported AUC is the mean of 900 operations- thirty replicate trials each with thirty bootstrap groupings.

Finally, we also consider the same evaluation strategies for ensembles. We consider two ensemble types: a simple averaging, and the ReLU ensemble method from Kim et al.. An ensemble of each type was constructed for each architecture and configuration level, resulting in 12 total ensembles. All ensembles consist of three models—either the three configurations from a given architecture, or the three different base models with the same configuration index.



(a) ADFA *TLAD*

(b) ADFA *ALAD*

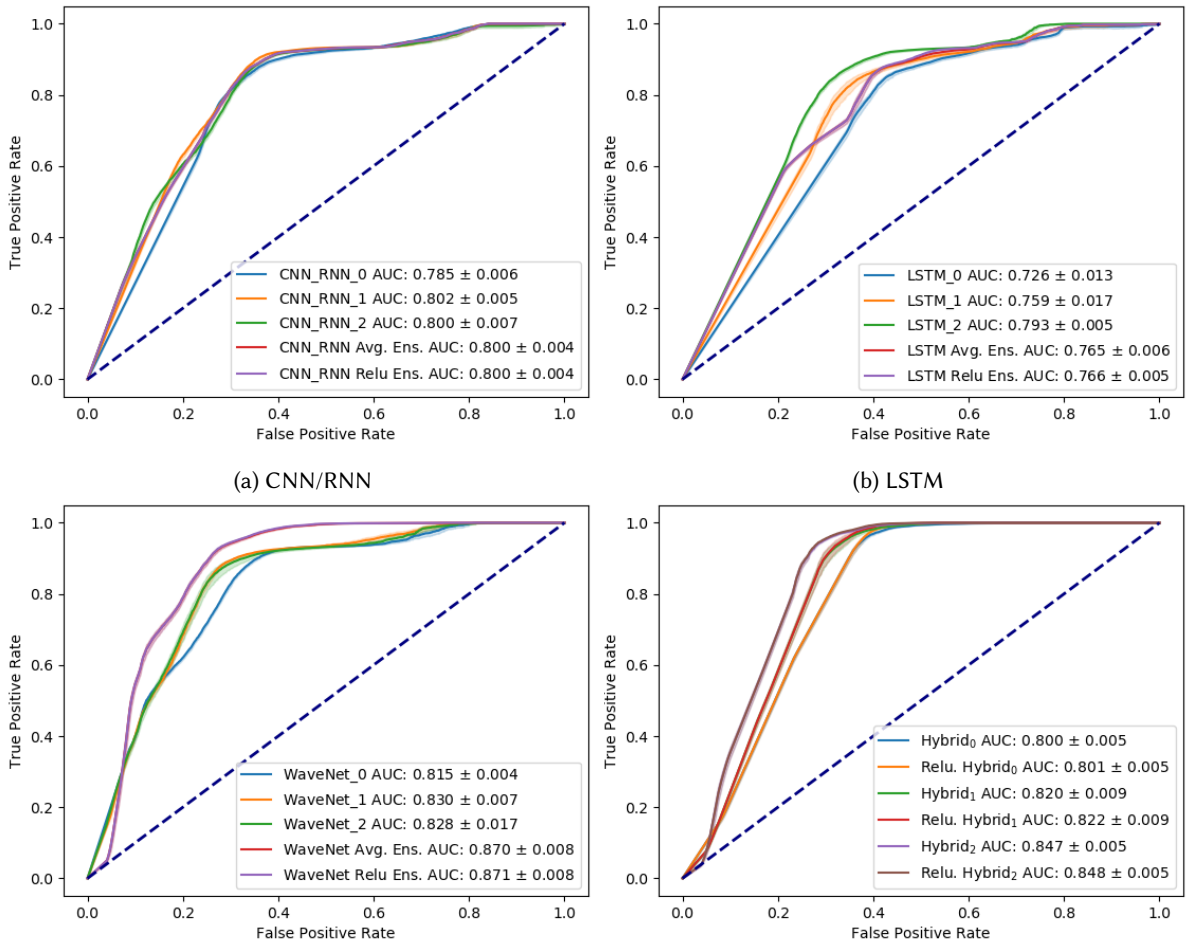
(c) PLAID *TLAD*

(d) PLAID *ALAD*

Fig. 2. ROC curves for the highest performing single model from each architecture along with the highest performing ensemble on ADFA (top) and PLAID (Bottom). Models were evaluated using both the TLAD(left) and *ALAD*(right). ROC curves show the mean and standard deviation for thirty trials. The legend reports the mean AUC and its standard deviation. For all models *ALAD*significantly improved performance.

5 RESULTS

We present performance metrics, namely ROC AUC and FPR at complete detection for all models, in Table 1. Separate columns exist for both metrics over each combination of model, dataset, and classifier method. These metrics are reported as the mean of the thirty replicate trials ± one standard deviation. In all cases *ALAD* significantly increased AUC (two-sided t-test, p-val < 0.001) when compared to TLAD. We also observe a significant reduction in the FPR at complete detection in the vast majority of cases. WaveNet proved to be the strongest performer on ADFA while LSTM models had the strongest performance on PLAID.

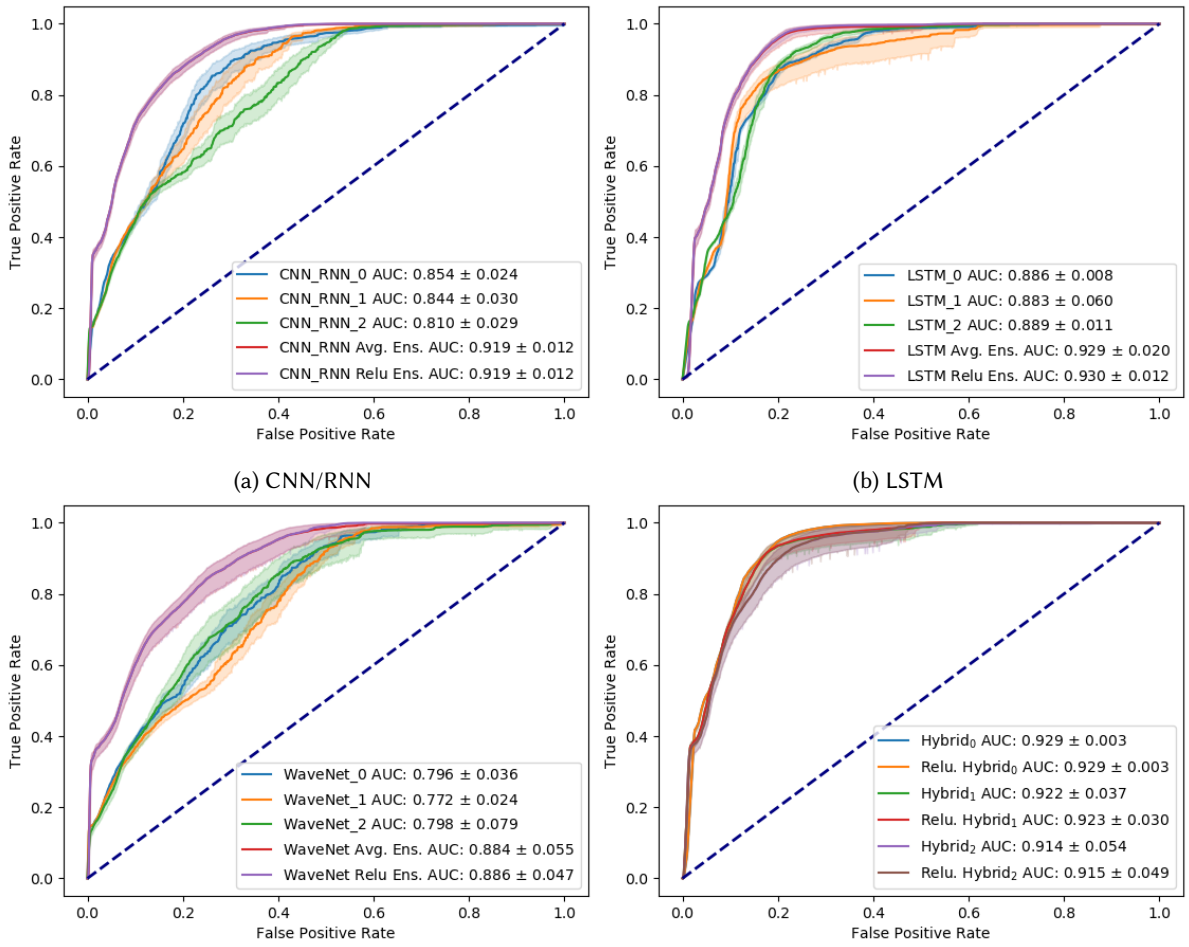


(a) CNN/RNN (b) LSTM

(c) WaveNet (d) Hybrid

Fig. 3. Figures 3a to 3c feature ROC curves for all trained models as well as homogenous ensembles on ADFA. Figure 3d shows the ROC heterogeneous ensembles constructed from model of all three architectures for each hyper-parameter configuration. ROC curves show the mean and standard deviation for thirty trials using *TLAD*. The legend reports the mean AUC and its standard deviation. We note that the LSTM and CNN/RNN ensembles under-performed some of their constituents while the WaveNet ensembles performed better.

In Figure 4.5 we show ROC curves for the highest performing model from each architecture, and the single best ensemble. We present our performance metrics for all 12 ensembles in Table 2. The traditional *TLAD* is shown on the left, and our proposed *ALAD* is on the right. We note the higher ROC curves when using *ALAD* showing the lower false positive rates at all levels of detection. Of additional interest is that there is no clear winner in terms of model architecture or even model size. Models tended to have a higher performance on PLAID compared to ADFA-LD at the trace level, except WaveNet.



(a) CNN/RNN (b) LSTM

(c) WaveNet (d) Hybrid

Fig. 4. Figures 4a to 4c feature ROC curves for all trained models as well as homogenous ensembles on PLAID. Figure 4d shows the ROC heterogeneous ensembles constructed from model of all three architectures for each hyper-parameter configuration. ROC curves show the mean and standard deviation for thirty trials using *TLAD*. The legend reports the mean AUC and its standard deviation.

Figures 3 and 4 show ROC curves for all models and ensembles on ADFA-LD and PLAID respectively using *TLAD*. We use an identical evaluation methodology to Kim et al. and Chawla et al. at the trace level, so we would expect model performance to be similar to the original work despite the differing data splits and training methodology. This was the case for our CNN/RNN models which had AUCs similar to their originally reported values. We failed to replicate the high performance at the trace level of Kim et al., but our smaller LSTM model performed similar to the LSTM model used in Chawla et al.. We did see a performance improvement from the use of ensembles and note that the ReLU ensemble was the top performer for both datasets, beating out the averaging

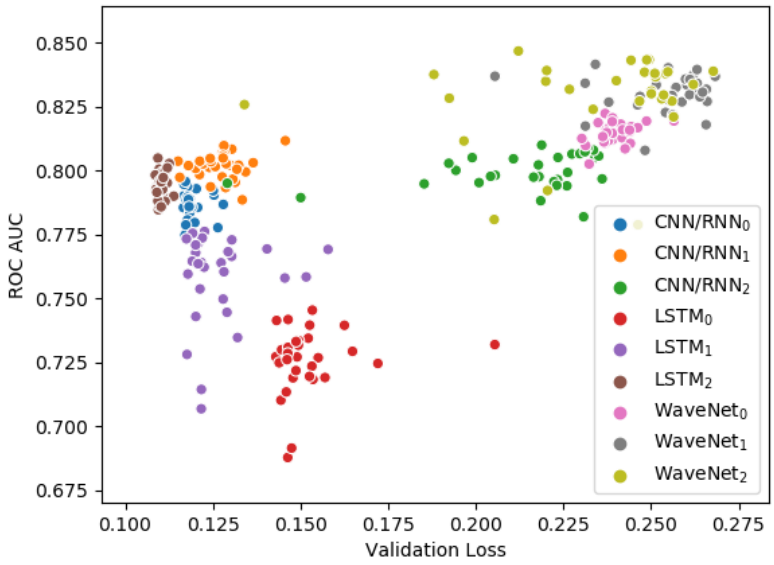


Fig. 5. Validation loss compared to performance for all models on the ADFA dataset. Typically, one expects lower validation loss to correspond with higher performance. Here we see no strong correlation between validation loss and performance. We note that anomaly detection results in a special case as the training task (system call prediction), is not same as the evaluation task (attack classification).

and hybrid ensembles. Despite this we were unable to replicate the strong performance of the ReLU ensemble shown in Kim et al. and note that its performance is virtually indistinguishable from the averaging ensemble.

In Figure 5 we compare validation loss at the final epoch to model performance as measured by the ROC AUC score. One might expect a lower validation loss to correspond with a higher ROC AUC score, however we do not observe this empirically.

In summary, *ALAD* resulted in a significant AUC improvement for all models on all architectures and datasets under consideration. This improvement comes at virtually no additional computational overhead, compared to *TLAD*.

6 DISCUSSION

6.1 Hypotheses

Testing our first of two hypotheses formulated in Section 4.1.4, namely that WaveNet would be the top performing architecture, produced mixed results. On ADFA, the dataset on which all models were tuned, WaveNet was indeed the top performer, supporting our hypothesis. However, WaveNet was the poorest performer on PLAID. There are two plausible explications for this behavior: WaveNet models may have over fit to the training data, or the architecture could be more sensitive to tuning.

Our second hypothesis, namely that *ALAD* would yield superior performance compared to *TLAD*, was fully supported by our analysis. For all models and datasets under consideration there was a statistically significant (two-sided t-test, p-val < 0.001) improvement under *ALAD*. We speculated that this is due to the fact that some attack traces may in fact be benign. This is an unavoidable artifact of the collection methodology. The attack set contains all traces, each representing a distinct process, of a program during a successful attack. The effects of a

modern attack are seen across multiple processes[10]. Precisely identifying the affected processes would require knowing exactly what system calls would have been issued in the absence of an attack.

6.2 Practical Concerns & Use Cases

The information in Tables 1 and 2 allows practitioners considering a deep learning IDS deployment to make informed decisions about the trade-offs between detection, false alarms, and computational cost. These tables show the primary drawback of deep learning powered IDS, long training and non-trivial evaluation times. For real-time detection the time and computational requirements may be too expensive for some applications. However, in addition to real-time detection, IDSs may also be used in a retrospective analysis. In a retrospective analysis IDSs may be used to identify which systems or applications were affected; helping analysts identify the impact of a breach or informing their search.

While PLAID improves upon ADFA-LD there is still a need for more comprehensive datasets. To be effective IDSs must be trained on baseline data reflective of their host. To meet this requirement practitioners must train the systems they wish to deploy on data collected locally. Additionally, the system must be (at least partially) retrained when any significant changes occur, such as the deployment of a new application.

6.3 Implementation Decisions

A deployment of any form of anomaly detection requires practitioners to select a threshold θ. This is an obstacle for practitioners as there is no way to know a priori the estimated probability the model will assign an attack sequence. Fortunately, there are two informed methods through which practitioners may select this value. First, one may use results on an existing corpus such as PLAID or ADFA. Second, one could utilize baseline sequences from their own production system; selecting a threshold that results in FPR they are able to handle. Of course, while neither of these choices guarantee complete detection they provide a means to achieve strong performance with an anomaly-based IDS. There is no wrong choice for a threshold value, only trade-offs between detection and false alarms.

Model selection is yet another obstacle for practitioners deploying a deep learning powered IDS. Typically, in deep learning one performs this task by selecting the model with the lowest validation loss. Unfortunately, we observed no strong correlation between AUC and validation loss. For this reason we recommend practitioners select their models based on their performance on reference datasets such as PLAID and ADFA. Additionally, this result underscores the need for researchers to continue to expand upon existing datasets.

Surprisingly, while we did see improvement from the use of ensemble, the effect was small compared to the performance achieved by the highest performing models. Additionally, while the ReLU ensembles outperformed their average ensemble counterparts, performance gains were marginal. As the creation of an ensemble requires duplicating training and evaluation costs, we believe it to be not worth the effort for this application.

7 VISUALIZING DIFFERENCES BETWEEN BASELINE AND ATTACKS

While deep learning is an effective ML approach in many applications, it suffers from its "black box", uninterpretable nature. Although methods are being developed to interpret deep learning models, they fall short, especially for insights into high-stakes decision making [51]. This is not *necessarily* an argument against the use of deep learning for HIDS since ML models are often just one component of a "observe, orient, decide, act" loop in security operation centers that also incorporate human analysts. However, interpreting data and predictive features in data is often critical for security practitioners. Instead of leveraging ML models for computational insights, we argue that other techniques can be leveraged, orthogonal to model development.

Two recently proposed techniques are "allotaxonometry" and "rank-turbulence divergence" [12]. These highly general methods leverage information-theoretic techniques for visualizing differences in datasets with complex

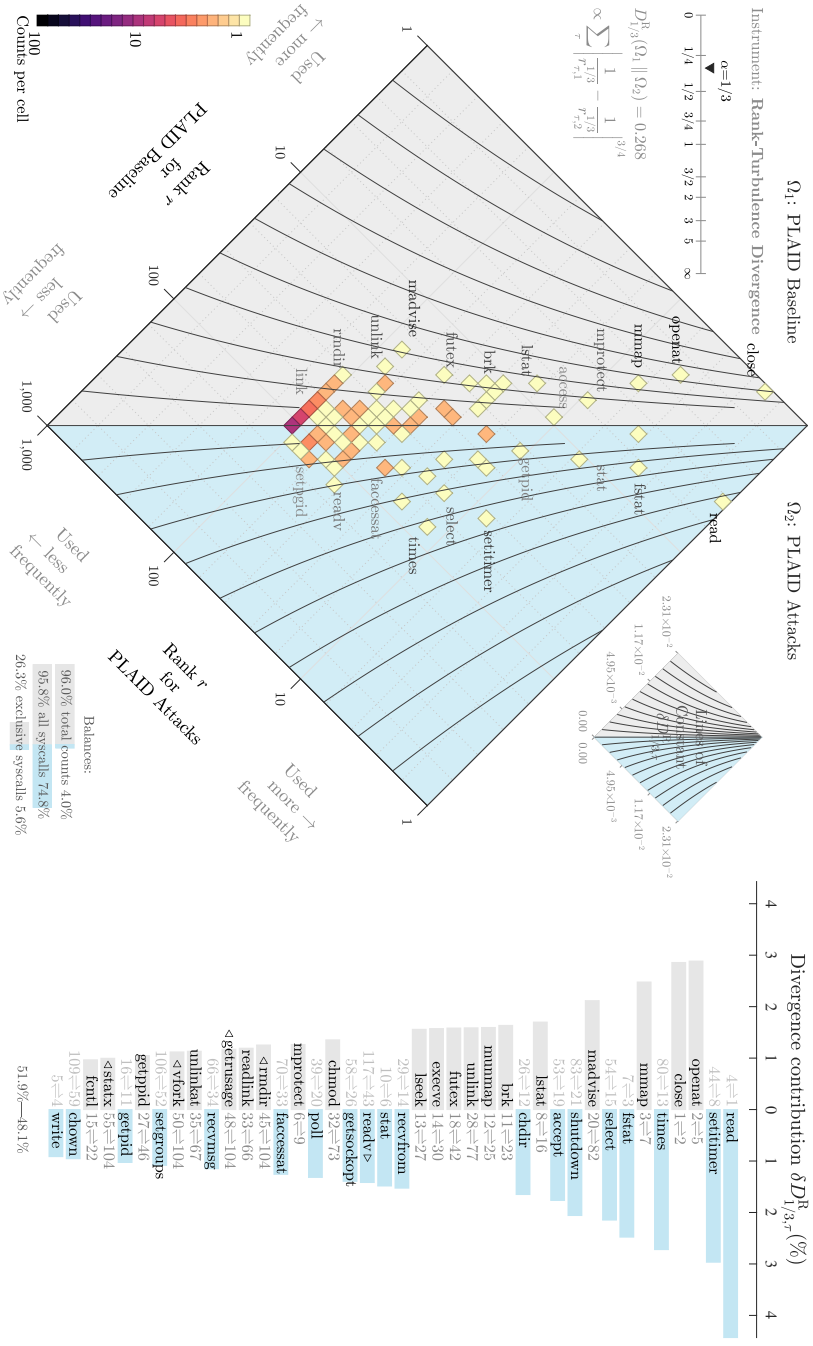


Fig. 6. Comparison of system call rankings between attack and baseline traces in PLAID. Note that some of the most frequently utilized system calls, `read` and `close`, are among the largest contributors to divergence.

structure, such as natural language text, baby names, and mortality cause databases. These techniques are especially relevant in our application space, since anomaly-based HIDS *rely* on the fact that significant differences exist between normal and malicious operations. Quantifying such differences not only sheds light on features potentially exploited by models, but also potentially new types of analysis. In this Section we explore the differences between attack and normal traces for both datasets used in this study, using allotaxonometry and rank-turbulence divergence.

In figure 6 we display the differences between attack and normal uni-grams using an allotaxonograph. This instrument features a rank-turbulence histogram on the left, and a rank-turbulence divergence shift on the right. We compute the relative rate of usage for each uni-gram in the baseline and attack sequences separately, then order system calls using tied-rank. Ranks for system calls that are found in one distribution but not the other are replaced with the maximum rank of the joint distribution. The 2D histogram on the left displays the distribution of uni-grams found in the baseline and attack sequences as well as the overlap between the two distributions. System calls on the left side of the histogram are often used in the baseline sequences, whereas system calls that are highlighted on the right side of the histogram are often used in attack sequences. System calls which are used in both systems equivalently can be seen in the middle.

Of particular interest is that commonly used system calls (e.g., `open`, `close`, and `times`) display relatively high rank-turbulence divergence in both datasets. This is in contrast to natural language where rankings of the most common words tend to be stable across corpora [12]. Additionally, the most dangerous system calls [7] are not top contributors to divergence. This suggests that focusing exclusively on dangerous system calls could result in failures to detect intrusions. Additional allotaxonographs of uni through tri grams of both datasets are in Appendix A. We also contrast the raw frequencies of system calls found in baseline and attack traces for both datasets in Appendix B.

In figure 7 we show that system call usage roughly follows an exponential rank frequency distribution. The rank frequency system call bi and tri-grams appears to approximate a power-law with an exponential cutoff in the tail. Natural language corpora tend to be and stay power-law like for uni- through tri-grams with the tail starting to flatten [20]. Thus, system call corpora become more power-law, while not quite reaching a power-law distribution while natural language corpora continue to follow a power-law distribution. Additional rank frequency plots for bi and trigrams are located in Appendix B. In all of these figures we clearly see substantial differences between attack and normal system call distributions.

8 CONCLUSION

In this paper we developed new methods for host-based intrusion detection systems (HIDS). Our fundamental approach to intrusion detection is to develop models for predicting "normal" aka baseline behavior, and then leveraging those models to detect malicious behavior as anomalistic. This approach has the benefit of being able to detect novel attacks, as well as known ones. We used deep learning models to achieve high levels of prediction performance.

Our work makes four primary contributions in the area of HIDS research. First, we collected and publicly released PLAID, a new system-call dataset for developing and evaluating IDS. Second, we developed *ALAD* (*A*pplication-*L*evel *A*nomaly *D*etection), a new classification method for anomaly-based IDS. Third, we presented the largest comparison to date of deep learning architectures applied to this domain. Fourth, we explored new visualization methods, based on information-theoretic corpus divergence measures, for exploring HIDS datasets.

Evaluating the performance of advanced methods, such as alternative deep learning models, requires comprehensive benchmarking that cannot be accomplished with the use of a single dataset. In our own architecture comparison, the use of either PLAID or ADFA-LD independently might lead to a conclusive answer that is different

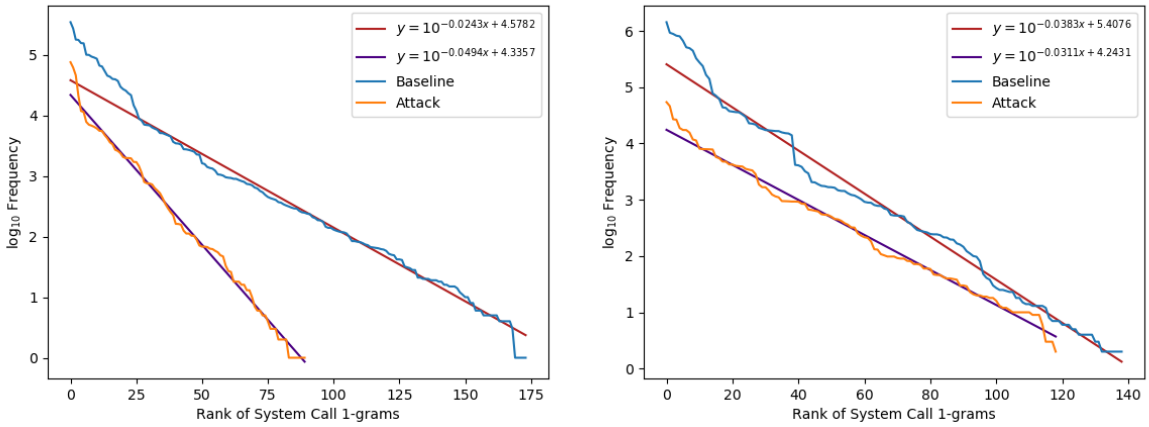


Fig. 7. Rank frequency plots of system calls for attack and baseline traces in ADFA-LD (left) and PLAID (right). Fit lines were obtained using Huber regression. Observe that system call usage roughly follows an exponential rank frequency distribution. This is differs from natural language where word frequencies follow a power law distribution [20].

from the relatively inconclusive results that we observed during a comprehensive evaluation. By introducing PLAID, we hope to empower the community to better evaluate new and existing HIDS models.

ALAD offered significantly better performance than *TLAD* regardless of the selected deep learning architecture or training dataset. This indicates that the inclusion of a relatively minimal piece of meta-data, application-level labels, can greatly impact IDS performance. The consistent benefit of *ALAD* begs the question, what other data or meta-data elements should be considered when constructing HIDS?

The results of our architecture search were fairly inconclusive with respect to classification performance, with WaveNet performing best on ADFA-LD and the LSTM model performing best on PLAID. However, WaveNet required approximately 60% less training time to converge on both ADFA-LD and PLAID when compared with similarly sized LSTM and GRU models. Thus, practitioners looking to train deep learning empowered HIDS quickly or scale up to massive data sets may prefer architectures composed primarily of convolutions over those composed of recurrent layers.

In our application of allotaxonographs to ADFA-LD and PLAID we identified clear differences between system calls created by baseline and malicious behavior. These differences may lead to additional insights into datasets why deep learning models out perform traditional machine learning models for HIDS. Future work should continue to investigate quantitative methods for corpus divergence in order to improve the interpretability of HIDS.

Overall, our results represent a significant improvement in the state-of-the-art in anomaly-based HIDS. We provide a useful new dataset for the broader HIDS research community, and a blueprint for developing deep learning empowered HIDS by presenting clear evaluation methodologies and reproducible results. Finally, we highlight opportunities for adapting these tools to particular domains.

ACKNOWLEDGMENTS

The authors are grateful for the computational facilities provided by the Vermont Advanced Computing Core. We are thankful to Thayer Alshaabi for assistance generating allotaxonographs and for helpful conversations. Finally, we are grateful for feedback provided by David R. Dewhurst.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.

[2] Ehsan Aghaei and Gursel Serpen. 2017. Ensemble classifier for misuse detection using N-gram feature vectors through operating system call traces. *International Journal of Hybrid Intelligent Systems* 14, 3 (2017), 141–154.

[3] Ahmed Ahmim, Leandros Maglaras, Mohamed Amine Ferrag, Makhlouf Derdour, and Helge Janicke. 2019. A novel hierarchical intrusion detection system based on decision tree and rules-based models. In *2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS)*. IEEE, 228–233.

[4] James P Anderson. 1980. Computer security threat monitoring and surveillance. *Technical Report, James P. Anderson Company* (1980).

[5] Ahmad Azab, Mamoun Alazab, and Mahdi Aiash. 2016. Machine learning based botnet identification traffic. In *2016 IEEE Trustcom/BigDataSE/ISPA*. IEEE, 1788–1794.

[6] Ahmad Azab, Robert Layton, Mamoun Alazab, and Jonathan Oliver. 2014. Mining malware to detect variants. In *2014 fifth cybercrime and trustworthy computing conference*. IEEE, 44–53.

[7] Massimo Bernaschi, Emanuele Gabrielli, and Luigi V Mancini. 2000. Operating system enhancements to prevent the misuse of system calls. In *Proceedings of the 7th ACM conference on Computer and communications security*. 174–183.

[8] Atul Bohara, Uttam Thakore, and William H Sanders. 2016. Intrusion detection in enterprise systems by combining and clustering diverse monitor data. In *Proceedings of the Symposium and Bootcamp on the Science of Security*. 7–16.

[9] Ashima Chawla, Brian Lee, Sheila Fallon, and Paul Jacob. 2018. Host based intrusion detection system with combined CNN/RNN model. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 149–158.

[10] Gideon Creech and Jiankun Hu. 2013. Generation of a new IDS test dataset: Time to retire the KDD collection. In *2013 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 4487–4492.

[11] Tran Khanh Dang and Tran Tri Dang. 2013. A survey on security visualization techniques for web information systems. *International Journal of Web Information Systems* (2013).

[12] Peter Sheridan Dodds, Joshua R Minot, Michael V Arnold, Thayer Alshaabi, Jane Lydia Adams, David Rushing Dewhurst, Tyler J Gray, Morgan R Frank, Andrew J Reagan, and Christopher M Danforth. 2020. Allotaxonometry and rank-turbulence divergence: A universal instrument for comparing complex systems. *arXiv preprint arXiv:2002.09770* (2020).

[13] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 1285–1298.

[14] Eleazar Eskin, Andrew Arnold, Michael Prerau, Leonid Portnoy, and Sal Stolfo. 2002. A geometric framework for unsupervised anomaly detection. In *Applications of data mining in computer security*. Springer, 77–101.

[15] Eleazar Eskin, Wenke Lee, and Salvatore J Stolfo. 2001. Modeling system calls for intrusion detection with dynamic window sizes. In *Proceedings DARPA Information Survivability Conference and Exposition II. DISCEX'01*, Vol. 1. IEEE, 165–175.

[16] Stephanie Forrest, Steven A Hofmeyr, Anil Somayaji, and Thomas A Longstaff. 1996. A sense of self for unix processes. In *Proceedings 1996 IEEE Symposium on Security and Privacy*. IEEE, 120–128.

[17] Kathleen Goeschel. 2016. Reducing false positives in intrusion detection systems using data-mining techniques utilizing support vector machines, decision trees, and naive Bayes for off-line analysis. In *SoutheastCon 2016*. IEEE, 1–6.

[18] The PHP Group. 2016. PHP Hypertext Processor. https://www.php.net/releases/7_1_0.php Accessed: 2020-05-08.

[19] XA Hoang and Jiankun Hu. 2004. An efficient hidden Markov model training scheme for anomaly intrusion detection of server applications based on system calls. In *Proceedings. 2004 12th IEEE International Conference on Networks (ICON 2004)(IEEE Cat. No. 04EX955)*, Vol. 2. IEEE, 470–474.

[20] Martin Joos. 1936. The Psycho-Biology of Language.

[21] Gyuwan Kim, Hayoon Yi, Jangho Lee, Yunheung Paek, and Sungroh Yoon. 2016. LSTM-based system-call language modeling and robust ensemble method for designing host-based intrusion detection systems. *arXiv preprint arXiv:1611.01726* (2016).

[22] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[23] Steve Klabnik and Carol Nichols. 2019. *The Rust Programming Language (Covers Rust 2018)*. No Starch Press.
[24] Andrew P Kosoresow and SA Hofmeyer. 1997. Intrusion detection via system call traces. *IEEE software* 14, 5 (1997), 35–42.
[25] Phuangpaka Kuttranont, Kobkun Boonprakob, Comdet Phaudphut, Songyut Permpol, Phet Aimtongkhamand, Urachart KoKaew, Boonsup Waikham, and Chakchai So-In. 2017. Parallel KNN and neighborhood classification implementations on GPU for network intrusion detection. *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)* 9, 2-2 (2017), 29–33.
[26] Emil Lerner. 2019. PHP-FPM Attack. https://github.com/neex/phuip-fpizdam Accessed: 2020-06-02.
[27] OffSec Services Limited. 2013. Brute-Force Password Attack. https://tools.kali.org/password-attacks/hydra Accessed: 2020-06-02.
[28] OffSec Services Limited. 2019. Kali Linux. https://www.kali.org/ Accessed: 2020-05-08.
[29] Massachusetts Institute of Technology Lincoln Laboratory. 1998/1999. DARPA Intrusion Detection Evaluation Dataset. https://www.ll.mit.edu/r-d/datasets/1999-darpa-intrusion-detection-evaluation-dataset, https://www.ll.mit.edu/r-d/datasets/1998-darpa-intrusion-detection-evaluation-dataset, Accessed: 2020-05-12.
[30] Hongyu Liu and Bo Lang. 2019. Machine learning and deep learning methods for intrusion detection systems: A survey. *Applied Sciences* 9, 20 (2019), 4396.
[31] Seth Lloyd. 2002. Computational capacity of the universe. *Physical Review Letters* 88, 23 (2002), 237901.
[32] Canonical Ltd. 2018. Ubuntu Linux. https://releases.ubuntu.com/18.04.4/ Accessed: 2020-05-08.
[33] Netcraft Ltd. 2020. April 2020 Web Server Survey. https://news.netcraft.com/archives/2020/04/08/april-2020-web-server-survey.html Accessed: 2020-05-08.
[34] ShaoHua Lv, Jian Wang, YinQi Yang, and Jiqiang Liu. 2018. Intrusion prediction with system-call sequence-to-sequence model. *IEEE Access* 6 (2018), 71413–71421.
[35] Tao Ma, Fen Wang, Jianjun Cheng, Yang Yu, and Xiaoyun Chen. 2016. A hybrid spectral clustering and deep neural network ensemble algorithm for intrusion detection in sensor networks. *Sensors* 16, 10 (2016), 1701.
[36] Steven McElwee, Jeffrey Heaton, James Fraley, and James Cannady. 2017. Deep learning for prioritizing and responding to intrusion detection alerts. In *MILCOM 2017-2017 IEEE Military Communications Conference (MILCOM)*. IEEE, 1–5.
[37] John McHugh. 2000. Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory. *ACM Transactions on Information and System Security (TISSEC)* 3, 4 (2000), 262–294.
[38] Weizhi Meng, Wenjuan Li, and Lam-For Kwok. 2015. Design of intelligent KNN-based alarm filter using knowledge-based alert verification in intrusion detection. *Security and Communication Networks* 8, 18 (2015), 3883–3895.
[39] Metasploit. 2019. Redis Attack. https://www.exploit-db.com/exploits/47195 Accessed: 2020-06-02.
[40] Erxue Min, Jun Long, Qiang Liu, Jianjing Cui, and Wei Chen. 2018. TR-IDS: Anomaly-based intrusion detection through text-convolutional neural network and random forest. *Security and Communication Networks* 2018 (2018).
[41] University of New Mexico Computer Science Department. 1998. UNM System Call Dataset. https://www.cs.unm.edu/~immsec/systemcalls.htm Accessed: 2020-05-12.
[42] ACM Special Interest Group on Knowledge Discovery and Data Mining. 1999. KDD Cup 1999: Computer Network Intrusion Detection. https://www.kdd.org/kdd-cup/view/kdd-cup-1999/Data, http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html, Accessed 2020/07/10.
[43] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. 2016. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499* (2016).
[44] Oracle. 2019. Virtual Box. https://www.virtualbox.org/ Accessed: 2020-05-08.
[45] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *International conference on machine learning*. 1310–1318.
[46] Sasanka Potluri, Shamim Ahmed, and Christian Diedrich. 2018. Convolutional neural networks for multi-class intrusion detection system. In *International Conference on Mining Intelligence and Knowledge Exploration*. Springer, 225–238.
[47] Benjamin J Radford, Leonardo M Apolonio, Antonio J Trias, and Jim A Simpson. 2018. Network traffic anomaly detection using recurrent neural networks. *arXiv preprint arXiv:1803.10769* (2018).
[48] Will Reese. 2008. Nginx: the high-performance web server and reverse proxy. *Linux Journal* 2008, 173 (2008), 2.
[49] Maria Rigaki and Sebastian Garcia. 2018. Bringing a gan to a knife-fight: Adapting malware communication to avoid detection. In *2018 IEEE Security and Privacy Workshops (SPW)*. IEEE, 70–75.
[50] John H. Ring IV. 2020. UVM IDS GitLab Repository. https://gitlab.com/jhring/uvm_ids.
[51] Cynthia Rudin. 2019. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence* 1 (05 2019), 206–215. https://doi.org/10.1038/s42256-019-0048-x
[52] Salvatore Sanfilippo. 2009. Redis. https://redis.io/ Accessed: 2020-05-08.
[53] LLC. SolarWinds Worldwide. 2020. Solarwinds Security Event Manager. https://www.solarwinds.com/security-event-manager Accessed: 2020-06-16.
[54] Splunk. 2020. Splunk Intrusion Detection System. https://www.splunk.com/ Accessed: 2020-06-16.
[55] Blake E Strom, Andy Applebaum, Doug P Miller, Kathryn C Nickels, Adam G Pennington, and Cody B Thomas. 2018. Mitre att&ck: Design and philosophy. *Technical report* (2018).

[56] Keras Team. 2020. Keras Tuner. https://keras-team.github.io/keras-tuner/ Accessed: 2020-05-10.
[57] OSSEC Project Team. 2020. OSSEC: Host Intrusion Detection for Everyone. https://www.ossec.net/ Accessed: 2020-06-16.
[58] Nam Nhat Tran, Ruhul Sarker, and Jiankun Hu. 2017. An approach for host-based intrusion detection system design using convolutional neural network. In *International Conference on Mobile Networks and Management*. Springer, 116–126.
[59] Aaron Tuor, Samuel Kaplan, Brian Hutchinson, Nicole Nichols, and Sean Robinson. 2017. Deep learning for unsupervised insider threat detection in structured cybersecurity data streams. In *Workshops at the Thirty-First AAAI Conference on Artificial Intelligence*.
[60] Solomon Ogbomon Uwagbole, William J Buchanan, and Lu Fan. 2017. Applied machine learning predictive analytics to SQL injection attack detection and prevention. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE, 1087–1090.
[61] Ali Moradi Vartouni, Saeed Sedighian Kashi, and Mohammad Teshnehlab. 2018. An anomaly detection method to detect web attacks using stacked auto-encoder. In *2018 6th Iranian Joint Congress on Fuzzy and Intelligent Systems (CFIS)*. IEEE, 131–134.
[62] Sitalakshmi Venkatraman and Mamoun Alazab. 2018. Use of data visualisation for zero-day malware detection. *Security and Communication Networks* 2018 (2018).
[63] Sitalakshmi Venkatraman, Mamoun Alazab, and R Vinayakumar. 2019. A hybrid deep learning image-based analysis for effective malware detection. *Journal of Information Security and Applications* 47 (2019), 377–389.
[64] Robin Verton. 2016. cowroot.c. https://gist.github.com/rverton/e9d4ff65d703a9084e85fa9df083c679 Accessed: 2020-05-08.
[65] Robin Verton. 2019. Privilege Escalation Attack. https://gist.github.com/rverton/e9d4ff65d703a9084e85fa9df083c679 Accessed: 2020-06-02.
[66] Wei Wang, Yiqiang Sheng, Jinlin Wang, Xuewen Zeng, Xiaozhou Ye, Yongzhong Huang, and Ming Zhu. 2017. HAST-IDS: Learning hierarchical spatial-temporal features using deep neural networks to improve intrusion detection. *IEEE Access* 6 (2017), 1792–1806.
[67] Yanxin Wang, Johnny Wong, and Andrew Miner. 2004. Anomaly intrusion detection using one class SVM. In *Proceedings from the Fifth Annual IEEE SMC Information Assurance Workshop, 2004.* IEEE, 358–364.
[68] Kehe Wu, Zuge Chen, and Wei Li. 2018. A novel intrusion detection model for a massive network using convolutional neural networks. *IEEE Access* 6 (2018), 50850–50859.
[69] Tatu Ylonen. 1996. SSH–secure login connections over the Internet. In *Proceedings of the 6th USENIX Security Symposium*, Vol. 37.
[70] Xiaoyong Yuan, Chuanhuang Li, and Xiaolin Li. 2017. DeepDefense: identifying DDoS attack via deep learning. In *2017 IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE, 1–8.
[71] Yi Zeng, Huaxi Gu, Wenting Wei, and Yantao Guo. 2019. $Deep - Full - Range$: A deep learning based network encrypted traffic classification and intrusion detection framework. *IEEE Access* 7 (2019), 45182–45190.
[72] Baoan Zhang, Yanhua Yu, and Jie Li. 2018. Network intrusion detection based on stacked sparse autoencoder and binary tree ensemble method. In *2018 IEEE International Conference on Communications Workshops (ICC Workshops)*. IEEE, 1–6.
[73] He Zhang, Xingrui Yu, Peng Ren, Chunbo Luo, and Geyong Min. 2019. Deep adversarial learning in intrusion detection: A data augmentation enhanced framework. *arXiv preprint arXiv:1901.07949* (2019).

A ALLOTAXONOGRAPHS

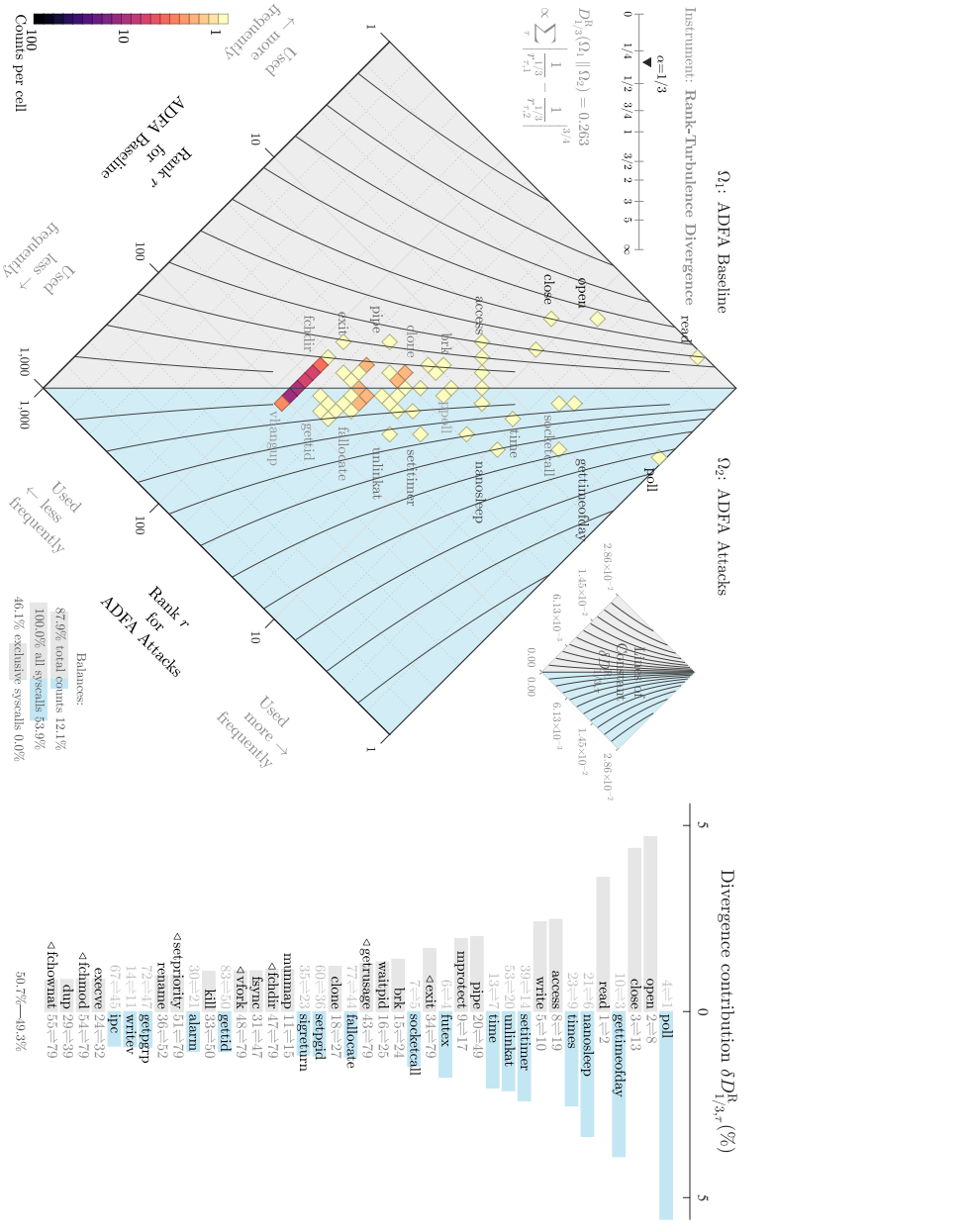


Fig. 8. Comparison of system call rankings between attack and baseline traces in ADFA-LD. Note that some of the most frequently utilized system calls, `poll` and `read`, are among the largest contributors to divergence. Of additional interest is that the most dangerous system calls are not top contributors to divergence.

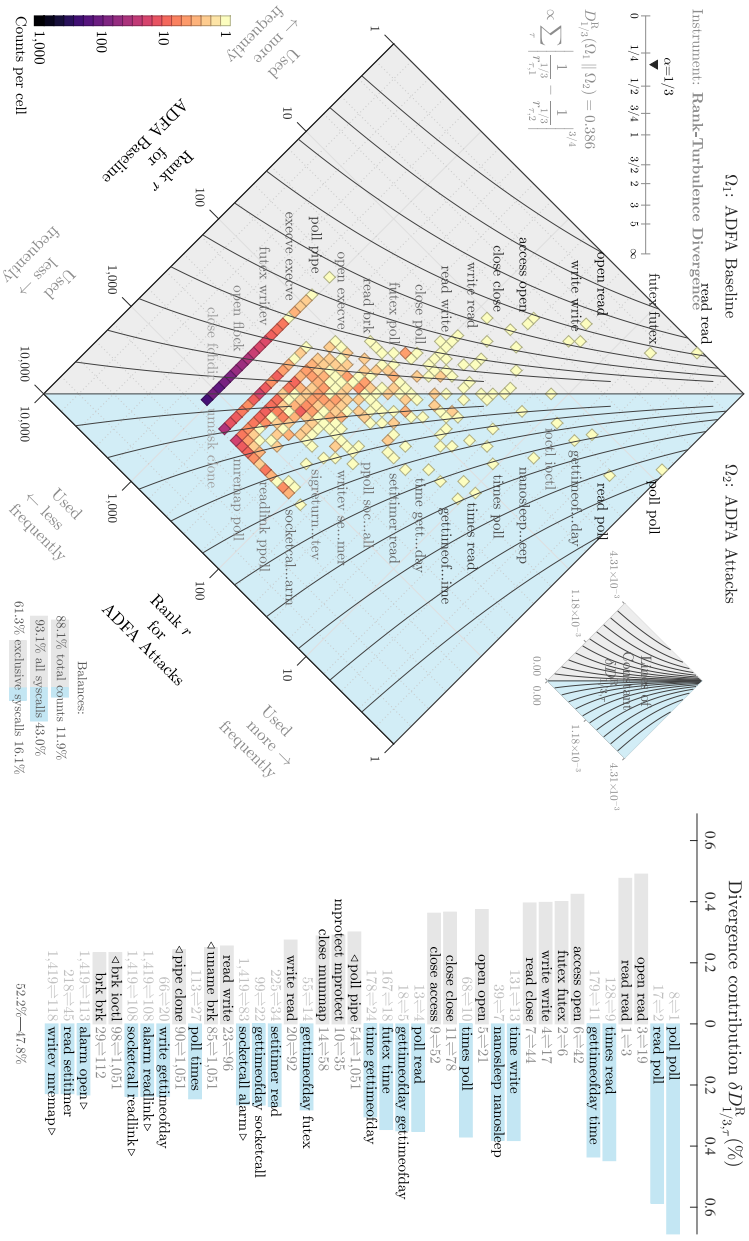


Fig. 9. Comparison of system call bi-gram rankings between attack and baseline traces in ADFA-LD. Similar to uni-grams frequent bi-grams remain top contributors to divergence. We see a larger portion of bi-grams appearing only in one split compared to uni-grams.

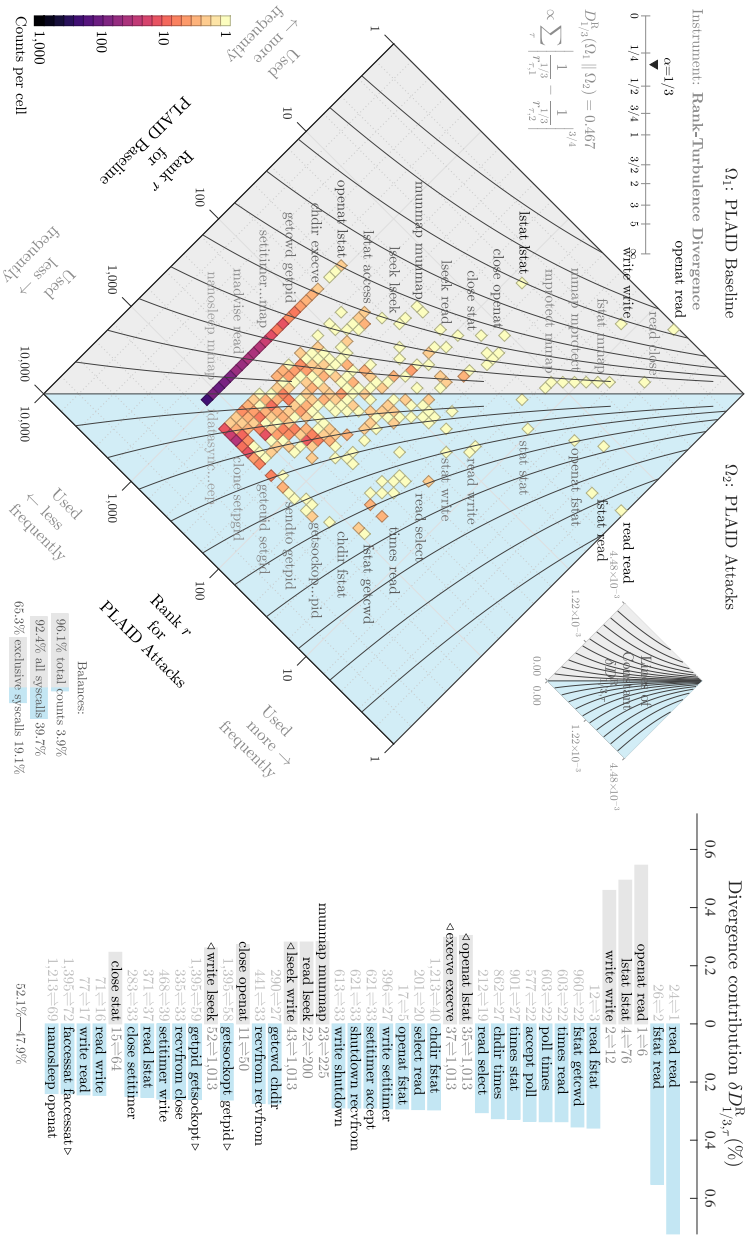


Fig. 10. Comparison of system call bi-gram rankings between attack and baseline traces in PLAID. Similar to uni-grams frequent bi-grams remain top contributors to divergence. We see a larger portion of bi-grams appearing only in one split compared to uni-grams.

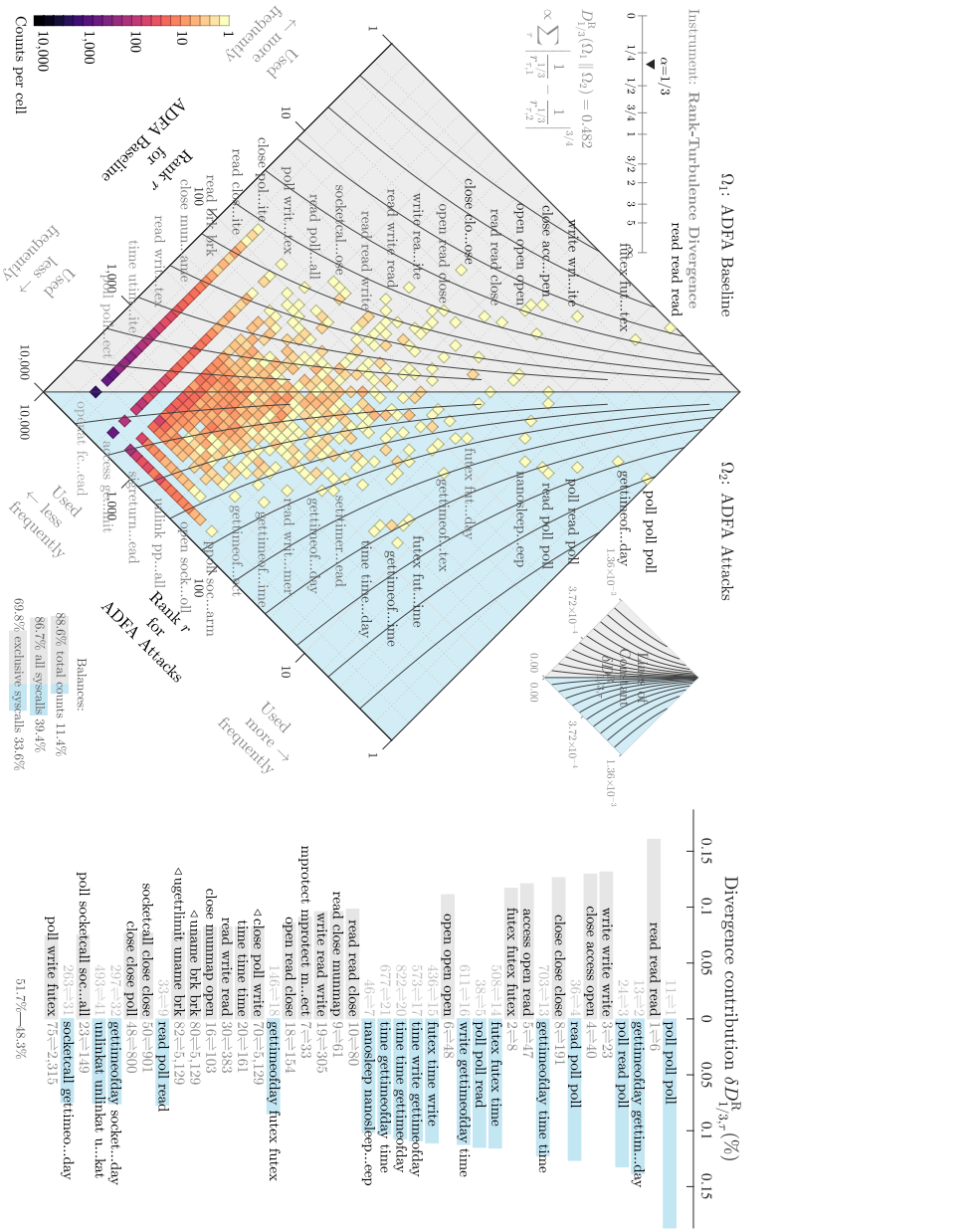


Fig. 11. Comparison of system call tri-gram rankings between attack and baseline traces in ADFA-LD. A slightly larger portion of tri-grams are present only in one set compared to bi-grams. This suggests that longer n-grams help to differentiate between sets.

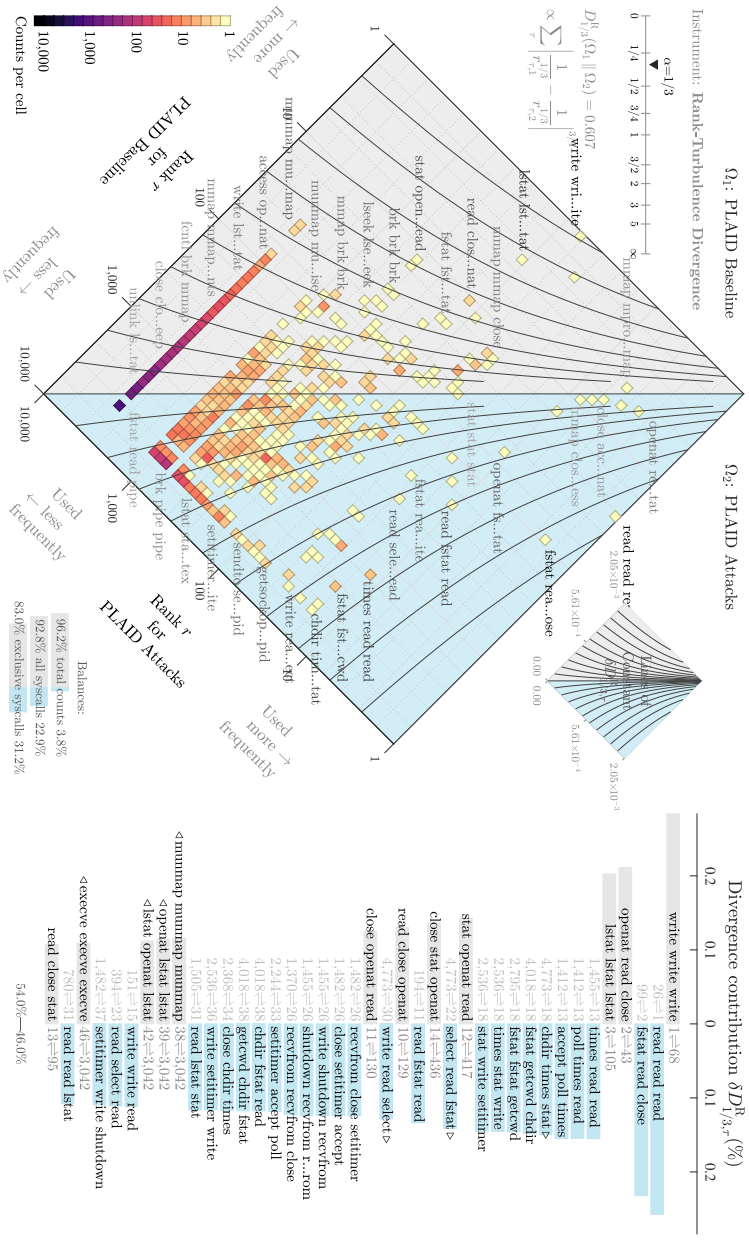


Fig. 12. Comparison of system call tri-gram rankings between attack and baseline traces in PLAID. A slightly larger portion of tri-grams are present only in one set compared to bi-grams. This suggests that longer n-grams help to differentiate between sets.

B SYSTEM CALL FREQUENCIES

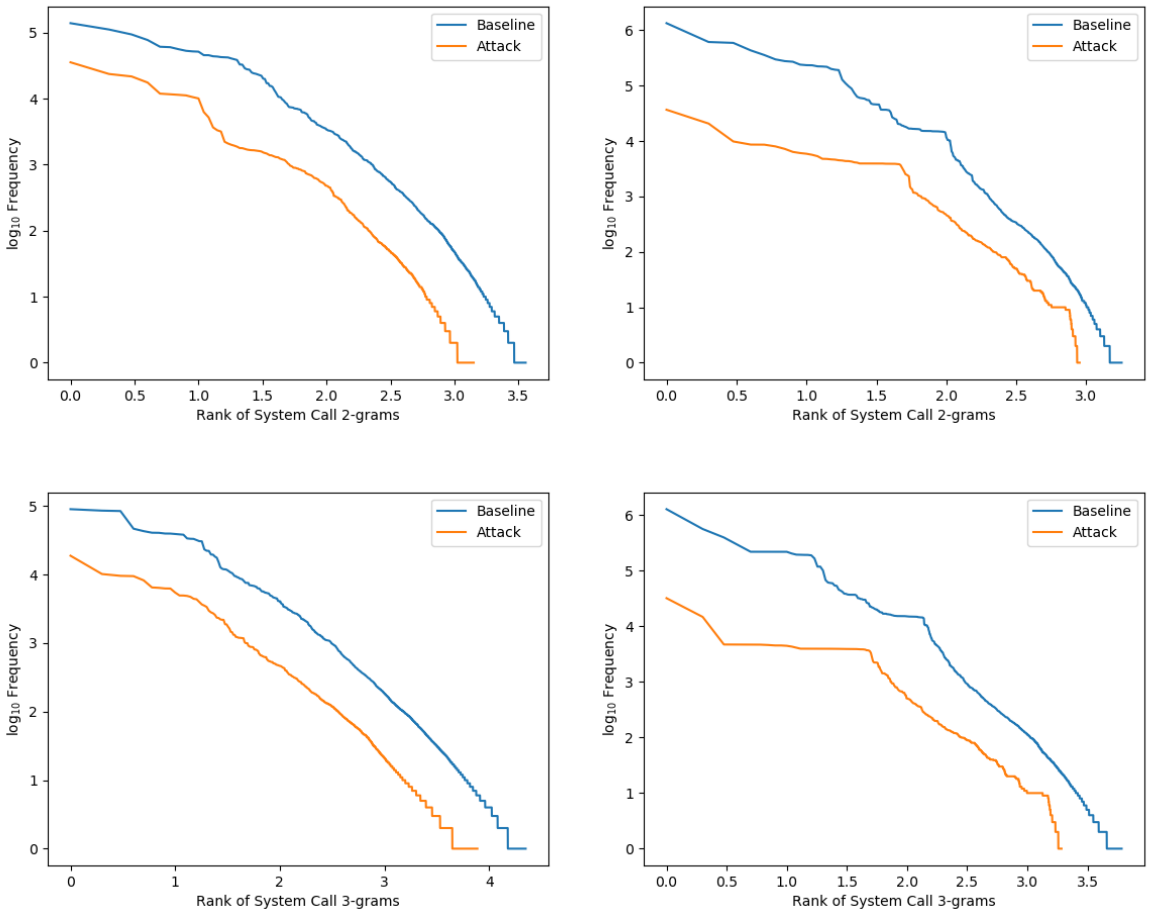


Fig. 13. Rank frequency plots of system call bi (top) abd tri (bottom) grams for attack and baseline traces in ADFA-LD (left) and PLAID (right). The rank frequency appears to approximate a power-law with an exponential cutoff in the tail. Natural language corpora tend to be and stay power-law like for uni through tri-grams with the tail starting to flatten. In contrast to system call corpora which become more power law like.

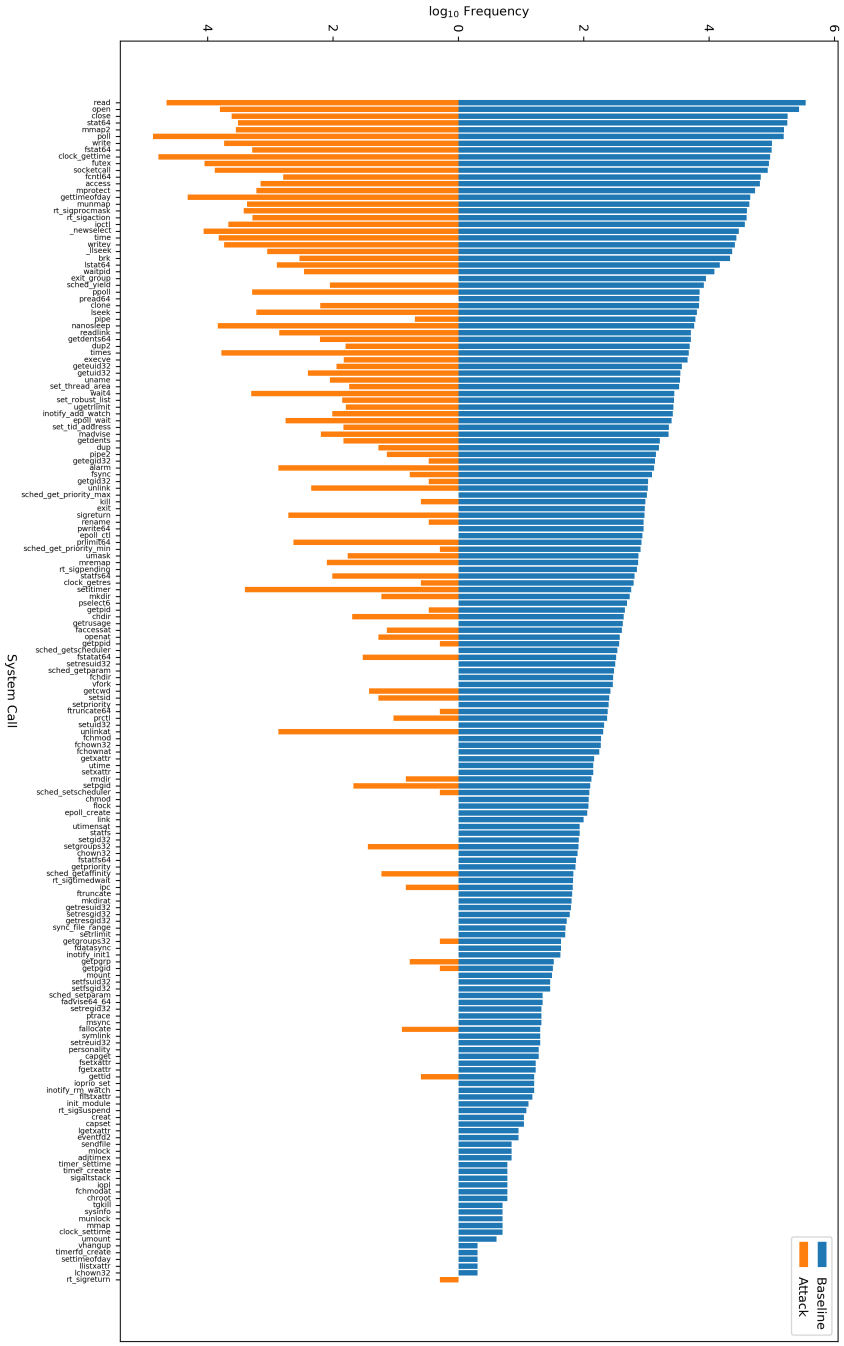


Fig. 14. Comparison of system call usage between baseline and attack traces in ADFA-LD. System calls are in monotonically non-increasing order base on their frequency in baseline traces. Notice that usages of individual system calls differ significantly between sets.

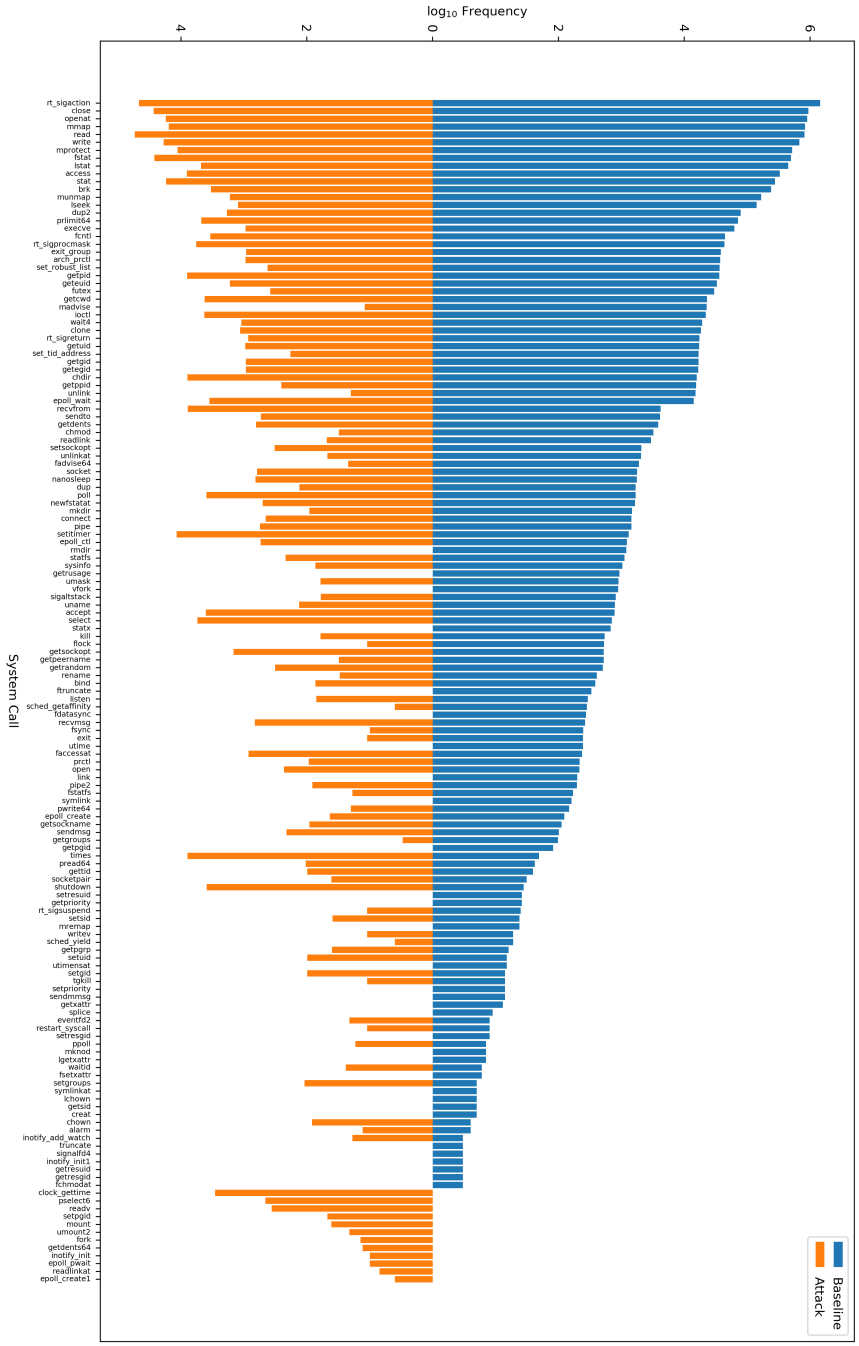


Fig. 15. Comparison of system call usage between baseline and attack traces in PLAID. System calls are in monotonically non-increasing order base on their frequency in baseline traces. Notice that usages of individual system calls differ significantly between sets. Of additional interest is the amount of `clock_gettime` calls in the attack split.