The Journal of Systems & Software 203 (2023) 111745

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

DongTing: A large-scale dataset for anomaly detection of the Linux kernel[☆,☆☆]

Guoyun Duan [a,b], Yuanzhi Fu [a], Minjie Cai [a], Hao Chen [a], Jianhua Sun [a,*]

[a] CSEE of Hunan University, No.2 Lushan South Road, Changsha 410082, China
[b] Information and Network Center, Hunan University of Science and Engineering, Yongzhou, Hunan 425199, China

ARTICLE INFO

Article history:
Received 9 November 2022
Received in revised form 18 March 2023
Accepted 4 May 2023
Available online 9 May 2023

Keywords:
Anomaly detection
Dataset
Linux kernel
System calls
Kernel BUG
Deep learning

ABSTRACT

Host-based intrusion detection systems (HIDS) can automatically identify adversarial applications by learning models from system events that represent normal system behaviors. The system call is the only way for applications to interact with the operating system (OS). Thus, system call sequences are traditionally used in HIDS to train models to detect novel attacks, and a wide range of datasets has been proposed for this task. However, existing datasets are either built for user-level applications (not for OS kernels), or completely outdated (proposed more than 20 years ago). To address this issue, this paper presents the first large-scale dataset specifically assembled for anomaly detection of the Linux kernel.

The task of creating such a dataset is challenging due to the difficulty both in collecting a diversified set of programs that can trigger bugs in the kernel and in tracing events that may crash the kernel at runtime. In this paper, we describe in detail how to collect the data through an automated and efficient framework. The raw dataset is 85 GB in size, and contains 18,966 system call sequences that are labeled with normal and abnormal attributes. Our dataset covers more than 200 kernel versions (including major/minor releases and revisions) and 3,600 bug-triggering programs in the past five years. In addition, we conduct cross-dataset evaluation to demonstrate that training on our dataset enables superior generalization ability than other related datasets, and provide benchmark results for anomaly detection of Linux kernel on our dataset. Our extensive dataset is both useful for machine learning researchers focusing on algorithmic optimizations and practitioners in kernel development who are interested in deploying deep learning models in OS kernels.

1. Introduction

The Linux kernel is the foundation of many contemporary computer systems, including data-center servers, embedded systems, mobile phones, and network routers, and its reliability and security are critical for these systems. Many efforts such as manual code auditing (Thongtanunam et al., 2014; Oliveira, 2021) and automated analysis (Zhou et al., 2022) are devoted to enhance the security of Linux kernel. However, these endeavors are inadequate to disclose all bugs in the kernel. For example, thousands of kernel vulnerabilities are listed in vulnerability databases like CVE (the common vulnerabilities and exposures), and new bugs are routinely found by kernel fuzzers like syzkaller (Syzbot dashboard, 2022). Therefore, approaches to detecting intrusions against the kernel dynamically are needed. One feasible way is to leverage machine learning to train models of system call sequences to predict possible attacks at runtime, given that the system call is the interface of invoking functionalities provided by the kernel and malicious users often invade the kernel by constructing specific system call patterns. Toward this way, a high-quality dataset containing large amount of system call data is desired.

Currently, publicly available datasets based on system call sequences are built for detecting either network intrusions such as DARPA (Lippmann et al., 2000), KDD99 (McHugh, 2000), CIC-IDS 2018, and IoT-Keeper (Hafeez et al., 2020) or abnormal behaviors of host applications such as UNM (Forrest et al., 1996), FirefoxDS (Murtaza et al., 2013), ADFA-LD (Creech and Hu, 2014), SysCall Dataset (Ezeme et al., 2019), and PLAID (Ring et al., 2021). These datasets have a range of shortcomings including outdated target environments, insufficient comprehensiveness,

[☆] Editor: Burak Turhan.

[☆☆] This research is supported in part by Natural Science Foundation of China under Grant 61972137 and 61772183, in part by the Hunan Provincial Natural Science Foundation of China under Grants 2022JJ20015, and in part by the Research Foundation of Education Department of Hunan Province in China under Grant 20A212.

[*] Corresponding author.
E-mail addresses: dguoyun@hnu.edu.cn (G. Duan), fuyuanzhi@hnu.edu.cn (Y. Fu), caiminjie@hnu.edu.cn (M. Cai), haochen@hnu.edu.cn (H. Chen), jhsun@hnu.edu.cn (J. Sun).

https://doi.org/10.1016/j.jss.2023.111745
0164-1212/© 2023 Elsevier Inc. All rights reserved.

limited types of applications, and long release time. The main shortcoming of existing system call-based datasets is the lack of abnormal data for Linux kernels. For example, UNM contains system calls of only a few Linux command-line tools. FirefoxDS, SysCall Dataset, and ADFA-LD collect system call traces from a small set of applications in a targeted manner. Only PLAID collected about 1100 abnormal, system call sequences regarding the Linux kernel. To the best of our knowledge, no large-scale and comprehensive datasets are currently available for the research of system call-based kernel anomaly detection.

The goal of this work is to develop an automated data collection framework and build a deep learning-based anomaly detection benchmark for Linux kernels. While collecting abnormal system call data for Linux kernel is a demanding task, it is also challenging and time-consuming, considering the diversity of Linux kernels and the complexity of the collection process. With the rapid development of kernel fuzzing techniques in recent years, there is an emerging opportunity to create a large dataset for our purpose. For instance, the popular kernel fuzzing framework `syzkaller` is routinely and widely used to find kernel bugs that are reported to a centralized location, from which we can obtain a large quantity of bug-triggering programs (i.e., POCs) with diversified bug types. With these programs, it is possible to obtain the logs of system call traces that are executed at runtime to trigger specific bugs.

In this work, we propose an automated system call collection framework for anomaly detection of Linux kernels. Using the POCs collected from kernel fuzzing tools (e.g., `syzbot`) as data source, the proposed framework has the ability of efficiently collecting a large scale of abnormal system call data in a distributed architecture. The abnormal data contains a total number of 12,116 system call sequences obtained from running 17,855 bug-triggering POCs. In addition to the abnormal system calls, we also collect normal system calls from four Linux test suites, i.e., Linux Testing Project (LTP) (Linux Testing Project, 2022), Linux Kernel Selftests (Kselftests) (Linux Kernel Selftests, 2022), Open POSIX Tests (Open Posix Test Suite, 2022), and Glibc test suite (Glibc testsuite, 2022). The normal data is derived from 6850 normal programs from the above four test suites. Overall, our dataset, named as DongTing[1] (DT), is composed of 12,116 abnormal system call sequences and 6850 normal system call sequences. It is currently the dataset with the largest amount of kernel attack data, which is 83 times more than ADFA-LD and 10 times more than PLAID. It is further divided into training, validation and testing data subsets, which can be later used to train a deep learning model for automatic anomaly detection of Linux kernels.

To build a benchmark for deep learning-based anomaly detection of Linux kernels, we train representative deep learning models (e.g., CNN, LSTM, Transformer et al.) on the proposed dataset. We also conduct cross-dataset evaluation with existing datasets (ADFA-LD and PLAID) to verify the advantages of collecting a large amount of abnormal data. Experiment results show that training on abnormal data is more effective than on normal data for kernel anomaly detection, while only the proposed dataset support training on abnormal data.

In summary, Our main contributions are as follows:

- **A new large-scale dataset**. The proposed DongTing dataset is the first large-scale system call-based dataset specifically proposed for anomaly detection of Linux kernels. This dataset covers the Linux kernels released in the past five years, including more than 200 Linux kernel versions of major/minor releases and revisions. It involves a total number of 18,966 labeled normal and abnormal system call sequences. The dataset is being actively maintained, guaranteeing correctness, data integrity, and periodical updates.
- **Comprehensive benchmark results.** We build a benchmark for deep learning-based anomaly detection of Linux kernels. We investigate and compare four representative models adapted from recent deep learning research on the proposed dataset. Besides, we also conduct a corss-dataset evaluation to demonstrate the advantage of collecting a large amount of abnormal data.
- **Fully released dataset and source code.** The full dataset with a size of 85 GB is released at Zenodo.[2] The source code for automated system call collection, training deep learning models, and benchmarking performance is available on open source platforms.[3]

2. Background and motivation

The main contribution of this work is a large-scale system call-based dataset for anomaly detection of Linux kernels. In this section, we describe the background of Linux kernel security and system call-based anomaly detection, analyze the limitations in existing system call-based datasets and introduce the motivation behind our proposed dataset.

2.1. Kernel security

The security of the Linux kernel is critical for the security of all applications running on top of the operating system, and is very different from the security of individual applications. Kernel vulnerabilities are more serious than application vulnerabilities (Chen et al., 2011).

Kernel fuzzing. Currently, the main tools for exposing kernel vulnerabilities are genetic algorithm-based American fuzzy lop (AFL) (AFL, 2022; AFL++, 2022), syzkaller (Syzkaller, 2022), and tools based on these platforms, such as kAFL (Schumilo et al., 2017), TeeFuzzer (Duan et al., 2023), and Moonshine (Pailoor et al., 2018), et al. Syzkaller is an open source project initiated by Google in 2016 to discover security vulnerabilities of OS kernels, and it is actively maintained by a community as compared to other kernel fuzzing tools (Schumilo et al., 2017; Kim et al., 2020; Wang et al., 2021). By now, about 4800 Linux kernel bugs have been found, and 3818 vulnerabilities are confirmed. The development of dynamic testing tools has dramatically accelerated the rate of bug discovery in the Linux kernel. For example, Syzkaller has found more kernel vulnerabilities in 5 years using automated methods than in the last 20 years combined. It offers an opportunity for us to collect a dataset for kernel anomaly detection. Syzbot is a system developed by the Syzkaller team for continuous fuzzing of OS kernels, providing services like automated testing services and non-repetitive report generation (Mu et al., 2022). It provides an online dashboard to publish kernel bug reports, which include bug description, patches, and proof of concept (POC) to reproduce bugs.

2.2. System call-based anomaly detection

System calls are the interfaces between user space and kernel space, and analysis of system call sequences is an important way

[1] The name of the second-largest fresh lake in China, located in Hunan province.

[2] http://dx.doi.org/10.5281/zenodo.6627050.

[3] https://github.com/HNUSystemsLab/DongTing.

Table 1
Comparison of related datasets. Datasets related to system call sequences released in the last 10 years.

Dataset	Kernel attack	#Abnormal sequences	#Normal sequences	Attack target	Abnormal training data	Platform	Data size	Release time
Firefox-SD	✗	19	700	Applications	✗	x86_64	0.2700 GB	2013
ADFA-LD	✗	746	5205	Applications	✗	i386	0.0164 GB	2013
PLAID	✗	1145	39,672	Applications	✗	x86_64	0.7800 GB	2021
DongTing	✓	12,116	6850	Linux kernel	✓	x86_64	85.000 GB	2022

of detecting abnormal system behavior (Karn et al., 2021; Han et al., 2021; Zhang et al., 2022).

Anomaly detection using system call sequences. The user-space programs can only interact with the kernel through system calls. Thus, from the attacker's perspective, it is necessary to craft a special program fragment (often called POC) in the user space to trigger the bugs in the kernel. Often a POC needs to invoke a series of system calls in a certain order, and such an ordered system call sequence implies the manifestation of bugs. Thus, system call sequences are the ideal object used to detect potential attacks. In NLP, language models (e.g., RNN) are used to model a probability distribution over sequences of words, and have been successfully used in many applications, such as machine translation and question answering. Similarly, system calls and system call sequences can be regarded as words and sentences in natural languages. Therefore, it is possible to use similar language models for anomaly detection based on system call sequences. For instance, it can be leveraged to estimate the likelihood of different words (system calls) and phrases (a sub-sequence of system calls) in different contexts.

Anomaly detection using abnormal sequences. The Linux kernel provides more than 300 system calls for applications to interact with the kernel. Normal activities use system calls conforming to a standard calling routine. For example, accessing files often have a sequence of `open()`, `write()`, and `close()`. But an abnormal activity may involve a sequence like `open()`, `close()`, and `write()` to trigger an untested path in the kernel. Traditional anomaly detection relies on training data of normal activities, aiming to model the patterns of normal system call sequences, and a system call sequence that deviates from normal patterns is identified as abnormal. However, it is challenging to gather training data that is representative for a wide range of applications. In our case of kernel anomaly detection, it is even more difficult to obtain such a large normal dataset, because of the complexity of OS kernels and the huge space of candidate applications that can be used to collect training data. Our consideration is that collecting abnormal data for the kernel is more straightforward to the task itself. If there is a more convenient and fast way to obtain abnormal system call data, the task can be better solved.

2.3. Motivation

Limitations of existing datasets. Deep learning has been adopted in various anomaly detection tasks, such as log-based anomaly detection (Le and Zhang, 2022; Studiawan et al., 2021), and promising results have been achieved. It is well acknowledged that the power of deep learning models heavily relies on high quality training data. We have investigated system call-based datasets for anomaly detection and find several limitations in existing datasets: (1) Few data samples are related to kernel anomalies. System call sequences in these datasets are obtained by analyzing applications and are not directly related to kernel anomalies. (2) The number of abnormal samples is much less than that of normal samples. Training data contains only normal samples, and abnormal samples are only used for evaluation. The comparison between existing datasets and the proposed dataset is shown in Table 1.

Motivation to build a specialized dataset. Among the currently released datasets (Sánchez et al., 2021), there is no one dedicated to anomaly detection of the Linux kernel. Thus, building a dataset for this purpose is one of our motivation. Existing datasets suffer from various drawbacks such as outdated collection environment and long release time. We take the opportunity of the rapid development of fuzzing tools like syzkaller, which offers great sources to develop a dataset suitable for kernel anomaly detection. Our long term goal is to motivate the deployment of machine learning models in OS kernels for tasks such as identifying attacks against the kernel at runtime.

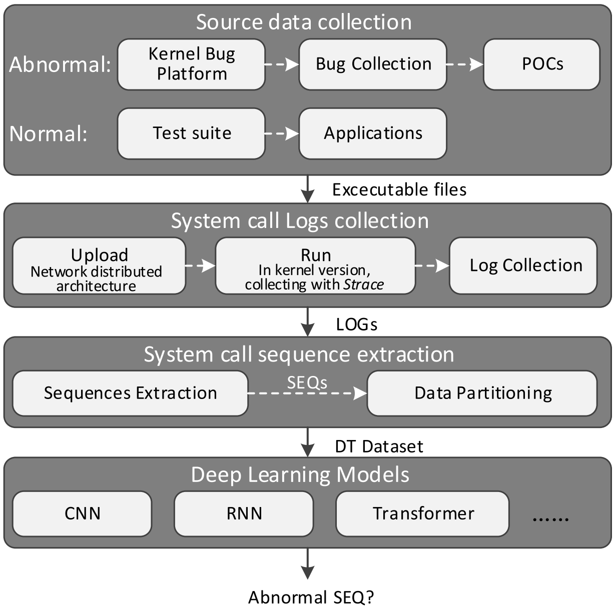


Fig. 1. Data collection framework of the DongTing dataset.

3. The DongTing dataset

In this work, we collect a large system call-based dataset for anomaly detection of Linux kernels. Collecting system call sequences is time-consuming, especially in our context because each POC may need to be traced with a certain version of Linux kernel. We specifically design an automated and distributed framework to accelerate the data collection procedure. In this section, we introduce the details of the data collection framework, and analyze the statistics of the collected dataset.

3.1. Framework overview

The data collection framework for the construction of DongTing dataset is shown in Fig. 1. The framework consists of three stages: (1) source code collection (Section 3.2), (2) system call log collection (Section 3.3), (3) system call sequence extraction (Section 3.4). The constructed dataset after data partitioning can be used to train deep learning models for system call-based anomaly detection. In the following, we would introduce the details of each data collection stage as well as the statistical information of the dataset.

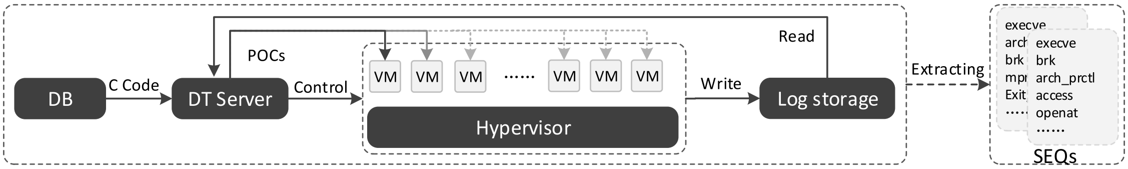


Fig. 2. The distributed architecture with virtual machines for system call log collection. DB is the abbreviation of Database. VM is the abbreviation of Virtual Machine. The database and log storage can be integrated into the DT server if the server has sufficient capacity (as configured in this work).

3.2. Source code collection

To generate real-world system call logs, the source code of relevant programs need to be collected and run in real system environments covering all mainline versions of Linux kernels. The source code data consist of abnormal code and normal code, which are collected in different ways as described next.

Abnormal code is collected from the Syzbot dashboard (2022) platform, which provides a well-organized database of kernel bugs found by the kernel fuzzing tool Syzkaller. As of May 2022, Syzbot has reports for 3656 bugs, providing 17,854 POCs that can be used to reproduce bugs. The POCs span over kernel versions ranging from 4.13 to the latest 5.17. The POCs contain a wide range of attack types, including memory bugs, concurrency bugs, semantics bugs, and error code bugs. For each type, multiple subtypes exist. For example, memory bugs can be classified into the following subtypes, such as user after free, GPF, out of bounds, uninitialized memory, memory leak, NULL-pointer deference, page fault, double free, and user copy error.

Normal code is a collection of normal applications from four regression test suites for Linux, namely LTP (Linux Testing Project) (Linux Testing Project, 2022), Kselftests (Linux Kernel Selftests) (Linux Kernel Selftests, 2022), Open POSIX Tests (Open Posix Test Suite, 2022), and Glibc Test suite (Glibc testsuite, 2022). In total, 6850 executable files are generated and are used to generating traces representing benign behaviors.

3.3. System call log collection

With abnormal and normal source code, system call logs are collected by running the compiled code in Linux systems of different kernel versions. Here, we introduce the log collection procedure and the distributed architecture designed to accelerate the log collection.

3.3.1. Distributed architecture

Each time an executable file is run, a log file is generated which records the system call trace. However, it is time-consuming to run nearly 27,000 executable files and collect system call traces with different kernel versions. According to our preliminary study, it is estimated that the log collection using a single server (16-core CPU, 16 GB memory, 800 GB storage) would cost more than 700 h (nearly 30 days). In this paper, we propose an efficient log collection method to automatically collect all the system call logs with a distributed architecture.

The distributed architecture is shown in Fig. 2. The DT server in the figure is responsible for controlling the data collection procedure, such as uploading and running source code in the target virtual machine, reading logs from the storage. It is also used to control virtual machine restart, shutdown, and other states that cannot be implemented by virtual machines themselves after crashing due to kernel bugs. The *VM server* is used to create virtual machines. In each virtual machine, three Linux kernels are installed. The *log storage* is used to store log data, and the *DB* mainly stores the source code of POCs.

We use four physical machines with the same hardware for the DT server and the VM server (one for the former and three for the latter). The specification of each physical machine is shown in Table 2. The host operating system of these servers is Ubuntu 21.04 with AppArmor enabled. Servers are connected through Ethernet switches, providing 1Gbps interconnection bandwidth. Nine virtual machines are spawned using VMware ESXi 6.7 on three VM servers. The source code and logs are stored in MongoDB on the DT server.

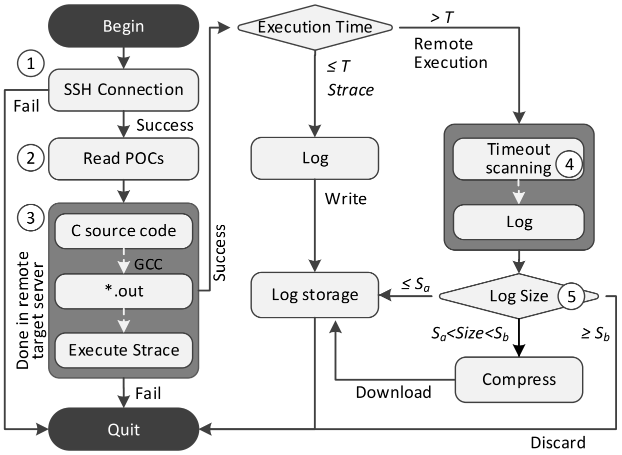


Fig. 3. Details of the work flow for log collection of one POC in the DT server. T is the threshold for execution time of POC and is set to 15 s. S_a and S_b are two thresholds for log size and are set to 10 MB and 30 GB respectively. T, S_a and S_b can be set based on specific requirements during the experiment.

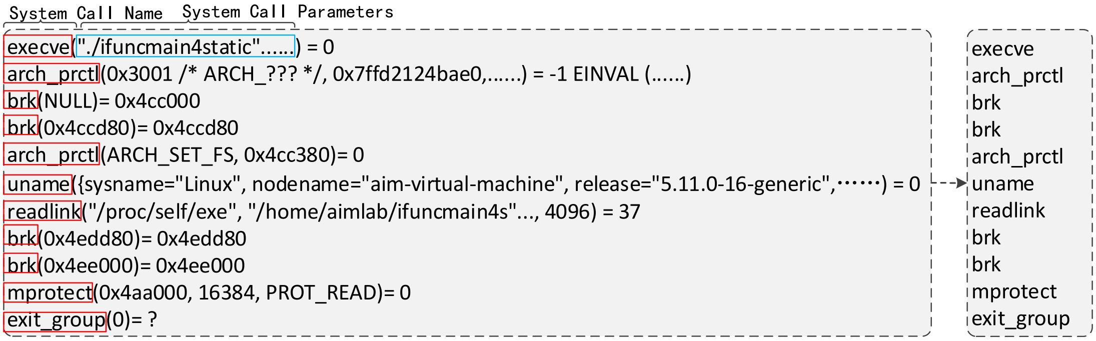
3.3.2. Log collection procedure

The log collection procedure in the DT server is shown in Fig. 3. It has the following main steps: (1) DT server first connects to the virtualization management platform and the target VMs using SSH, and reads the OS kernel version of the target VM. (2) The POC is adapted to the kernel version, and the matched source code is uploaded to the target VM to be compiled. (3) The compiled executable is run, and the system call traces are recorded in log files. (4) The execution time of POCs is checked, and the time-out processes are automatically stopped. (5) Log files are stored in the log storage and exceptions are handled. In particular, the details of system call recording and exception handling are given in the following.

System call recording. Because the POCs (i.e., abnormal source code) span over many different kernel versions (e.g., there are many revisions against a major/minor release of the Linux kernel), we use major or minor releases of the Linux kernel to collect traces instead of managing all revisions required by POCs to accelerate the data collection process. In total, 26 kernel versions are used in our experiments. Normal programs are traced under Ubuntu with the 5.12 kernel. We use `strace` to gather system call traces for both normal programs and POCs. The system call traces of each POC or normal program are recorded to log files individually. We denote a threshold of code execution time as T and two thresholds for the log size as S_a and S_b. When the code execution time exceeds T (15 s), the timeout scanning process

Table 2
Hardware and software configuration.

Classification	Configuration	Parameters
Hardware	CPU	Intel(R) Xeon(R) CPU E5-2683 v4 @ 2.10GHz, 2 Socket
	Memory	ECC DDR4 1600 Hz, 16GB, 4 Pieces
	Hard Drive	SATA 2 TB, 2 Pieces
Software	Operating system	Ubuntu 21.04 LTS
	Virtualization platform	VMware ESXi 6.7, VMware vCenter Server 6.7 U1
	Database	MongoDB 4.4.6
	Development platform	Python 3.9
Network	Ethernet switch	Equipped with 24 1Gbps ports



```
execve("./ifuncmain4static"......) = 0
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffd2124bae0,......) = -1 EINVAL (......)
brk(NULL)= 0x4cc000
brk(0x4ccd80)= 0x4ccd80
arch_prctl(ARCH_SET_FS, 0x4cc380)= 0
uname({sysname="Linux", nodename="aim-virtual-machine", release="5.11.0-16-generic",······) = 0
readlink("/proc/self/exe", "/home/aimlab/ifuncmain4s"..., 4096) = 37
brk(0x4edd80)= 0x4edd80
brk(0x4ee000)= 0x4ee000
mprotect(0x4aa000, 16384, PROT_READ)= 0
exit_group(0)= ?
```

```
execve
arch_prctl
brk
brk
arch_prctl
uname
readlink
brk
brk
mprotect
exit_group
```

Fig. 4. Example of a system call log file and the extracted system call sequence (on the right side). The category of system calls is defined in the Linux 5.17 kernel. The system call category has a total of 362 system call names available for x86_64 systems as used in this work.

is started and run in the background of the remote server. The master process will start the execution of the next POC. Log files whose size exceeds S_b (30 GB) are discarded, considering that infinite loops may result in particularly large logs. Log files smaller than S_a (10 MB) are directly downloaded to the DT server, while logs whose size are between S_a and S_b are first compressed and then downloaded.

Exception handling. Various exceptional events, such as kernel crash, SSH session loss, system not responding, process deadlock, and network disconnection, may occur during the execution of POCs. We use a timeout scanner to monitor the execution of the `strace` process, and forcibly kill the POC process with a timeout (20 min). According to the experimental environment in Table 2, special values such as 20 min and 15 s are defined in the anomaly detection process.

3.4. System call sequence extraction

Sequence extraction. `strace` can be used to obtain information such as system call names, return values, execution time, process IDs, and system calls invoked by sub-processes. Fig. 4 shows a fragment of a typical log produced by `strace`. We focus on the system call names in the logs, and the 362 system calls in the Linux kernel 5.17 (x86) are used as a reference when extracting system call sequences from the logs. Log files are scanned line-by-line and a sequence of system call names is extracted for each log file. There are 1.475 TB abnormal log files and 12.2 GB normal log files, making the sequence extraction quite time-consuming. Using a server with 16 GB memory and 16 cores, we design a parallel extraction strategy to analyze all the log files. After nearly 40 h of sequence extraction, we obtain 18,966 system call sequences with a size of 85.1 GB, which constitute our DongTing dataset.

Data partitioning. To support later usage of DongTing dataset for machine learning-based kernel bug detection, we split the whole dataset into separate subsets for training, validation, and testing. We notice that each bug has multiple different system call sequences (Lin et al., 2022) which are relevant to a certain degree (e.g., a bug may be triggered from different execution paths). To avoid the data correlation between different subsets, we perform data partitioning of the abnormal data based on kernel bugs with a ratio of 80%:10%:10% for the training, validation, and testing set. Specifically, we divide every 10 bugs into a group, from which 2 bugs are randomly selected and marked as validation and testing subsets respectively, the remaining 8 bugs are marked as training subsets. Please notice that the number of system call sequences may not have the same ratio, as the number of system call sequences in different bugs is quite different.

For normal data, since each normal program only generate one system call sequence, we simply split the system call sequences with the same ratio as the abnormal data. The details of data partitioning are summarized in Table 3.

3.5. Statistics of DongTing dataset

Here we present the statistical analysis of the DongTing dataset, including the statistics of different Linux kernel versions, sequence length and system call names. We hope that this analysis may facilitate a better understanding of the dataset and guide further study of system call-based anomaly detection.

Statistics of kernel versions. The abnormal data covers all Linux kernel versions from 4.13 to 5.17, released in the last five years. There are a total of 12,116 system call sequences after running POCs in these kernel versions. The statistics of different Linux kernel versions is shown in Fig. 5. It can be seen that the five mainline versions of 4.13, 4.15, 4.16, 4.19 and 5.3 accounts for nearly 40% of POCs. These kernels are the default kernel versions of mainstream operating systems such as Ubuntu18-20 and SUSE Linux Enterprise Server 15 (Ubuntu, 2022; Suse, 2022), thus more attention need to be paid to these systems. Fig. 5 also describes the average size of system call sequence files in different kernel versions.

Statistics of sequence length. The number of system calls is directly related to the effect of anomaly prediction. POCs containing infinite loops may generate numerous repeated system calls. So, we limit the maximum number of extracted system calls to

Table 3
Data partitioning of DongTing dataset. Bugs stands for the number of Linux kernel bugs, Bug-SEQs stands for the number of system call sequences, Programs stands for the number of complete sets of normal programs, Programs-SEQs stands for the number of normal system calls sequences, and Total-SEQs stands for the total number of system call sequences, DT is the abbreviation of DongTing.

Data properties	Total	DT-Train				DT-Validation				DT-Test			
		Abnormal data		Normal data		Abnormal data		Normal data		Abnormal data		Normal data	
BUGs	1733	1387	80.035%	–	–	174	10.040%	–	–	172	9.925%	–	–
Bug-SEQs	12,116	9356	77.220%	–	–	1127	9.302%	–	–	1633	13.478%	–	–
Programs	6850	–	–	5487	80.102%	–	–	685	10.000%	–	–	678	9.898%
Programs-SEQs	6850	–	–	5487	80.102%	–	–	685	10.000%	–	–	678	9.898%
Total-SEQs	18,966	9356	49.330%	5487	28.931%	1127	5.942%	685	3.612%	1633	8.610%	678	3.575%

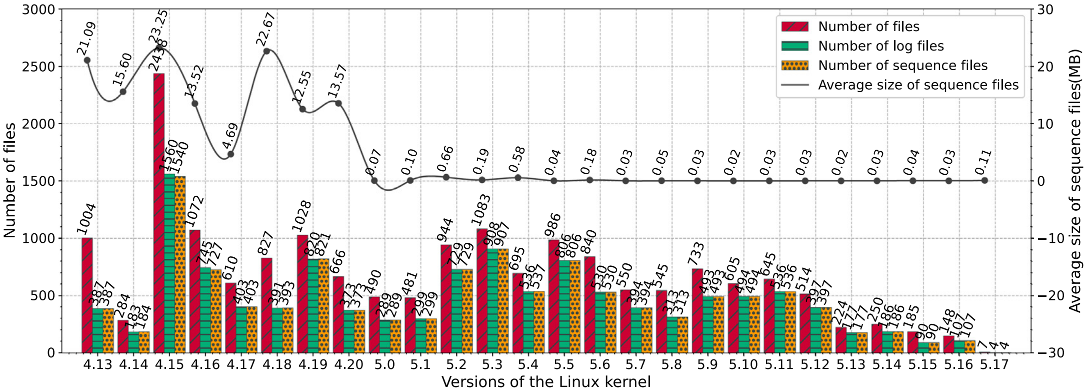


Fig. 5. Statistics of Linux kernel versions for abnormal data.

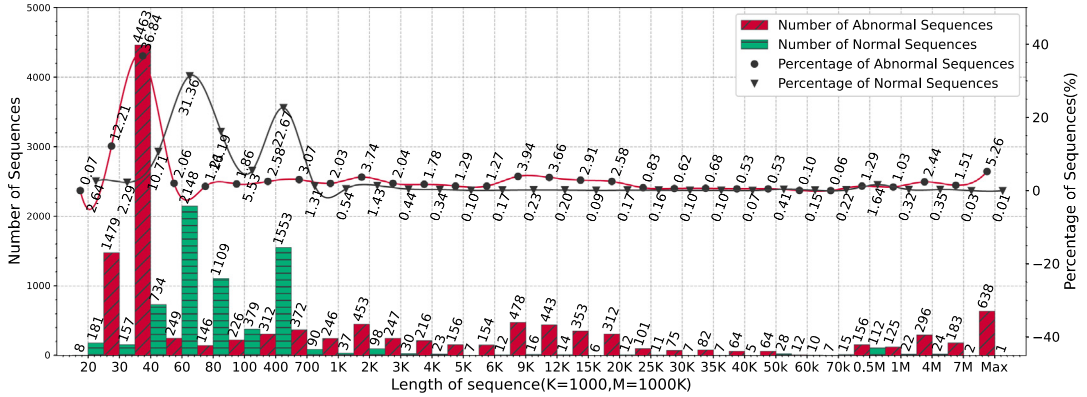


Fig. 6. Statistics of the length of system call sequences.

100 million. According to prior studies (Bridges et al., 2019; Forrest et al., 1996), the length of attack sequences is typically below 100. Thus, we investigate the distribution of sequence length for both attack and normal data in Fig. 6. We can see that abnormal sequences with a length of less than 100 account for more than 54%. For normal sequences, this ratio is about 69%. The length of abnormal sequences is mainly scattered over the interval between 21 and 40. For normal sequences, the length mainly varies from 41 to 60 or from 101 to 400. The length of most normal sequences (96%) is less than 9000. For abnormal sequences, this ratio is 76%. For about 12% of abnormal sequences, their lengths span over the interval from 6000 to 15,000. Sequences whose length exceeds 70,000 are due to loops in POCs.

Statistics of system call names. To compare the difference of normal and abnormal system behaviors from the perspective of system calls, we analyze the statistics of system call names at the sequence-level. First, we calculate the percentage of all system call names occurring in each system call sequence. Then, the average and variance of the percentage of each system call name in all abnormal and normal sequences are calculated separately. The system call names with top 20 highest frequency in the abnormal and normal sequences are shown in Figs. 7 and 8 respectively. One interesting phenomenon is that the system calls *nanosleep*(), *socket*() and *ioctl*() are called much more frequently in abnormal sequences than in normal sequences, which may reflect the outstanding characteristics of conspicuous system behavior. Another interesting phenomenon is that 45.03% of system call names never appear in the data of abnormal sequences, while only 12.98% of that never appear in the normal sequences. Nevertheless, normal and abnormal sequences share most of the frequent system call names in the top 20 most frequent system calls. 15 system call names appear frequently in both normal and abnormal sequences, among which the system call *mmap*() for memory mapping appears the most frequent. This indicates that in addition to the individual system call names, the temporal dependency in the system call sequence also need to be explored.

4. Experiments

To demonstrate the usefulness of the proposed DongTing dataset for kernel bug detection, we first perform cross-dataset evaluation to compare the generalization ability of existing

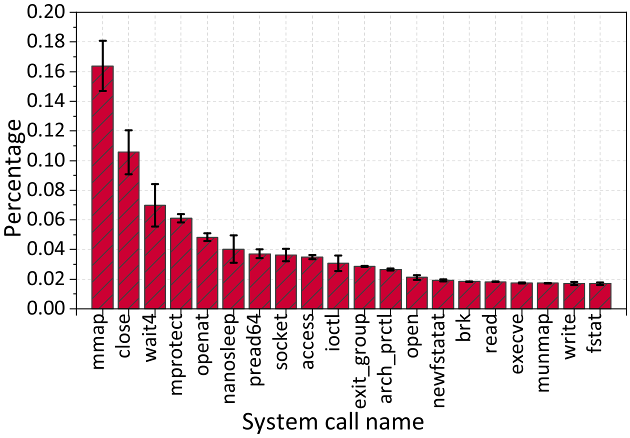


Fig. 7. The top 20 most frequent system calls in the sequence of abnormal system calls. System call *nanosleep* in the figure is an abbreviation of *clock_nanosleep*.

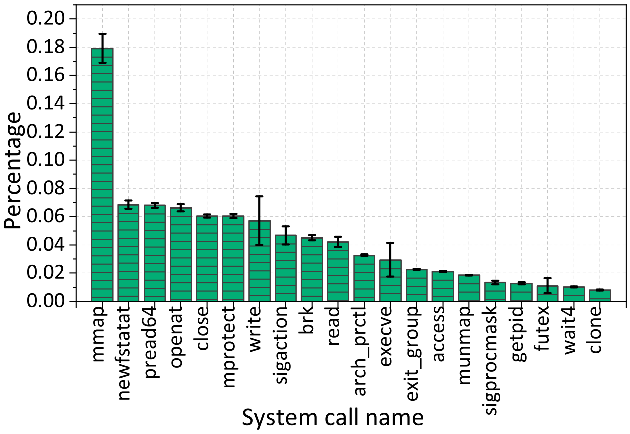


Fig. 8. The top 20 most frequent system calls in the sequence of normal system calls. System call *sigaction* in the figure is an abbreviation for *rt_sigaction* and *sigprocmask* is an abbreviation for *rt_sigprocmask*.

datasets (Section 4.4). We also evaluate existing state-of-the-art models for anomaly detection of system kernels on the test set of DongTing dataset (Section 4.5).

4.1. Task definition and evaluation metric

Task definition. The task is to classify whether a system call sequence is abnormal or not. Formally, given a system all sequence $Q = \{x_i\}_{i=1}^{N}$ of length N where x_i is an index of system call name, the goal is to output a probability $p = f_\theta(Q) \in [0, 1]$, which is close to one when Q is abnormal. We adopt the traditional routine to train a language model based on normal or abnormal system call sequences. The datasets and the baseline models are introduced in Sections 4.2 and 4.3 respectively.

Evaluation metric. We adopt the Area Under ROC Curve (AUC) and F1 score as the evaluation metrics for anomaly detection, with abnormal system call sequences treated as positive. The F1 score is computed using the median anomaly score as the detection threshold.

4.2. Datasets

In the experiment, we select four datasets for cross-dataset evaluation, including ADFA-LD, PLAID, and two variants of DongTing dataset in this work. In the following, we briefly describe how we use them in our experiments.

ADFA-LD. This dataset contains 5951 system call sequences, of which 5205 and 746 are normal sequences and attack sequences, respectively. These sequences cover 175 system calls. We use the dataset splits as in Ring et al. (2021). All 746 attack sequences together with randomly selected 746 normal sequences are used to form the test set. The remaining normal sequences are randomly divided into the training set and validation set with the ratio of 80%:20% (3567 vs. 892).

PLAID. This dataset contains 40,817 system call sequences, of which 39,323 sequences with length between 8 and 4495 are used. The sequences cover 228 system calls. The dataset is divided into three subsets as in Ring et al. (2021). The test set contains all 1145 attack sequences and 1145 randomly chosen normal sequences. The training set and validation set are randomly obtained from the remaining normal sequences with the ratio of 80%:20% (29,626 vs. 7407).

DT-Normal and DT-Abnormal. It is important to notice that both ADFA-LD and PLAID have no attack sequences in their training set, therefore only normal model can be trained with ADFA-LD and PLAID. Our DongTing dataset have both attack and normal sequences in three subsets. This enables us to train both normal and abnormal models with two variants of DongTing, named as DT-Normal and DT-Abnormal respectively. For training the normal model, we use all normal sequences in the training set of DongTing. Similarly, we use all abnormal sequences for training the abnormal model. For evaluation, the same test set of DongTing is used. To make fair comparison, we only use sequences with length between 8 and 4495. In addition, we sample 678 abnormal sequences from the test set to make 50%:50% ratio of normal and abnormal sequences.

4.3. Models and implementation details

Baseline models. The use of deep learning models for anomaly detection of kernels is a less explored topic. We select and compare four model architectures of CNN/RNN (Chawla et al., 2019), LSTM (Kim et al., 2016), WaveNet (Oord et al., 2016) and ECOD (Li et al., 2022) that are recently investigated for host-based intrusion detection. In particular, CNN/RNN, LSTM and WaveNet are constructed with increasing parameters following Ring et al. (2021), resulting in three different models for each architecture (e.g., LSTM0, LSTM1, LSTM2). The three CNN/RNN models have (6, 7, 8) 1D convolutional layers and (200, 500, 600) GRU units respectively. The three LSTM models are one with a single LSTM layer of 200 cells, one with a single LSTM layer of 400 cells, and one with two LSTM layers of 400 cells. All three WaveNet models are configured with eight WaveNet blocks (Oord et al., 2016). They are different in the number of filters in each convolutional layer, which are 128, 256 and 512, respectively. In addition, we also adopt ECOD (Li et al., 2022) for evaluation, which is a parameter-free outlier detection method using an empirical cumulative distribution function to measure the outliers of data points.

The models are not only used to evaluate cross-dataset generalization performance of existing datasets, but also compared to establish a benchmark for anomaly detection of system kernel on DongTing.

Implementation details. The system call sequences in all datasets are encoded according to the system call table in Linux kernel 5.17 (Linux Kernel, 2022) (system call index value plus 1), which is essential for cross-dataset evaluation. We train the models as a sequence-to-sequence prediction task in a self-supervised manner with only normal sequences (or abnormal sequences). The trained models are expected to learn normal (or abnormal) system behaviors from corresponding system call sequences, and can be used to predict the probability of belonging to the learned

Table 4
Cross-dataset evaluation. Each row and column represent the training and testing set respectively. The top three scores of mean Area under curve (AUC) on each dataset are marked as **first**, **second** and **third**.

DataSet		Test Data			
		ADFA-LD	PLAID	DongTing-Normal	DongTing-Abnormal
Training data	ADFA-LD	(**0.810**, 0.0011)	(**0.447**, 0.0038)	(0.293, 0.0006)	(0.293, 0.0006)
	PLAID	(**0.498**, 0.0006)	(**0.865**, 0.0022)	(**0.296**, 0.0005)	(**0.296**, 0.0005)
	DT-Normal	(0.468, 0.0008)	(0.345, 0.0037)	(**0.916**, 0.0032)	(**0.916**, 0.0032)
	DT-Abnormal	(**0.539**, 0.0027)	(**0.658**, 0.0042)	(**0.879**, 0.0012)	(**0.879**, 0.0012)

behavior given a test sequence. We set the hyperparameters of the models following Ring et al. (2021). The training epoch is set to 30 for PLAID, and 300 for ADFA-LD, DT-Normal and DT-Abnormal. The batch size is set to 16 for training $wavenet_2$ and 32 for the rest of models. We conduct our experiments on 2 physical servers installed with Ubuntu 21.04. The hardware configuration of each server is Intel Xeon E5-2640 V4, DDR4 ECC 2400 MHz with 128G memory, and 4 GPU cards (NVIDIA Tesla V100 16G).

4.4. Cross-dataset evaluation

To demonstrate that the model trained using our DongTing dataset has better generalization ability to unseen system behaviors, we perform cross-dataset evaluation on existing datasets as described in Section 4.2. The models trained on four training sets separately are evaluated on the test sets respectively. For consistency of comparison, we adopt the nine models as used in previous datasets. The average AUC computed from nine models is reported in Table 4. Since DT-Normal and DT-Abnormal share the same testing set, the evaluation results on them are the same.

It is generally expected that a model could achieve the best score on the dataset it is trained on, thus we focus on the generalization scores obtained by evaluating on other datasets. Table 4 shows that the model trained on DT-Abnormal achieves the best generalization scores on ADFA-LD and PLAID (0.539 and 0.658), demonstrating strong generalization ability of our dataset. In contrast, the generalization scores of the models trained on ADFA-LD and PLAID are quite low (0.293 and 0.296) when evaluated on DongTing dataset. One interesting and important observation is that the model trained on DT-Abnormal has obviously superior generalization ability than the model trained on DT-Normal, as evaluated on ADFA-LD and PLAID. This indicates that training on abnormal data is more effective than on normal data for anomaly detection. Meanwhile, only the proposed DongTing dataset has sufficient data for training on abnormal data.

4.5. Benchmark results

To build a benchmark for anomaly detection of OS kernel, we evaluate 10 models (see Section 4.3) trained on DT-normal and DT-abnormal and compare the anomaly detection performance as well as the model size and inference speed. Table 5 provides quantitative results for the selected models. Comparing models trained on DT-Normal with that trained on DT-Abnormal, while individual models have different performance, their overall performance is close. It also can be seen that F1 scores are relatively low compared with AUC scores. F1 score reflects the detection accuracy when certain classification threshold is selected in practice. For models trained on DT-Abnormal, the highest F1 score is 0.844, indicating that there is still much space for improvement in the problem of the dataset.

In order to study the uncertainty of the reported performance, error bars of 10 models trained on normal and abnormal sequences in DongTing dataset are also drawn in Fig. 9. Each model is trained 10 times with different random seeds using DT-normal and DT-abnormal respectively, and then the mean and standard deviation of AUC scores are calculated. It can be seen that the biggest standard deviation is within 0.05. Among all models, CNN/RNN1 achieves the most balanced performance in accuracy and stability.

In addition to AUC which evaluates the overall performance of a model, ROC curve is also an important metric which plots the false positive rates at different detection levels (true positive rates). The ROC curves of four models (one highest performing model from each model architecture) trained on DT-Normal and DT-Abnormal are drawn in Figs. 10 and 11. It can be seen from Fig. 10 that CNN/RNN clearly outperforms other architectures when trained on DT-Normal. It has lower false positive rates at nearly all true positive rates. However, there is no clear winner of model architecture when trained on DT-Abnormal. The ROC curves also show that all models tend to have increasing false detection when the true positive rate is over 0.75.

4.6. Evaluation with long sequence

In the above cross-dataset evaluation Section 4.4, we selected sequences of length 8–4495 from DongTing dataset, for consistent sequence length comparison with ADFA-LD and PLAID. However, the sequence range was much wider in the DongTing dataset. To study the effect of long sequence data on model training, we further select sequences of length 3–9000 in the DongTing dataset to train the models respectively and compare the results with those of length 8–4495. The results are shown in Table 6. It can be seen that models trained with sequences of length 8–4495 have slightly higher AUC values than models trained with sequences of length 3–9000. We conclude that longer sequence may add difficulty in training models, although with a limited impact.

5. Discussion

In this section, we discuss the possible application scenarios in which our dataset can be used, as well as the limitations.

Usage Scenarios. Machine learning models trained from our dataset can be used in two ways. First, we can deploy the models in user space for detecting OS kernel intrusions by monitoring the system call invocations of the whole system or individual process. This can be realized by developing a monitor leveraging kernel features like eBPF that is lightweight and flexible. eBPF has been widely used in many tasks such as kernel-level monitoring or enhancing kernel functionalities. Second, the models trained with our dataset can be deployed in the OS kernel for anomaly detection with the support of techniques like model compression (Akgun et al., 2020; Zhang and Huang, 2019). There are a few recent studies that exploit AI to enhance the kernel subsystems like storage and scheduler (Umit Akgun et al., 2021; Qiu et al., 2021). Some even developed specific frameworks for using AI models in the kernel. Therefore, it is a promising direction to promoting the deployment of deep learning based anomaly detection models in OS kernels to protect the OS from kernel attacks at runtime.

Performance Impact. Because we collect system call traces in a non-production environment, the impact of tracing on performance is not the main concern. However, if we deploy the

Table 5
Performance evaluation of 10 models trained on DongTing-Normal and DongTing-Abnormal. The top three AUC and F1 scores in each column are marked as **first**, **second** and **third**.

Model	#Params	Inference time (s)	AUC		F1	
			DongTing-Normal	DongTing-Abnormal	DongTing-Normal	DongTing-Abnormal
CNN/RNN0	0.7M	44.0	0.882	0.889	0.786	0.778
CNN/RNN1	2.7M	66.6	**0.959**	0.867	**0.795**	0.776
CNN/RNN2	8.1M	60.2	**0.975**	**0.908**	0.788	**0.814**
LSTM0	0.5M	16.6	0.804	0.800	0.590	0.734
LSTM1	1.7M	20.8	0.863	0.888	0.642	0.784
LSTM2	3.0M	33.3	0.934	**0.903**	0.781	0.804
WaveNet0	1.2M	219.9	0.956	0.875	**0.801**	**0.806**
WaveNet1	4.4M	243.7	**0.958**	**0.919**	0.784	**0.844**
WaveNet2	17.4M	364.5	0.912	0.864	**0.812**	0.706
Ecod	–	1.4	0.637	0.700	0.727	0.721

Table 6
The performance of 10 models trained with abnormal sequences with length range of 8–4495 and 3–9000 respectively.

Model	Inference time (s)		AUC	
	Abnormal(8–4495)	Abnormal(3–9000)	Abnormal(8–4495)	Abnormal(3–9000)
CNN/RNN0	44.0	88.4	0.889	0.914
CNN/RNN1	66.6	160.4	0.867	0.876
CNN/RNN2	60.2	132.5	0.908	0.937
LSTM0	16.6	51.3	0.800	0.797
LSTM1	20.8	39.5	0.888	0.849
LSTM2	33.3	86.9	0.903	0.854
WaveNet0	219.9	443.5	0.875	0.884
WaveNet1	243.7	460.4	0.919	0.913
WaveNet2	364.5	681.3	0.864	0.791
Ecod	0.7	0.8	0.721	0.720
Average value	107.03	214.5	0.863	0.854

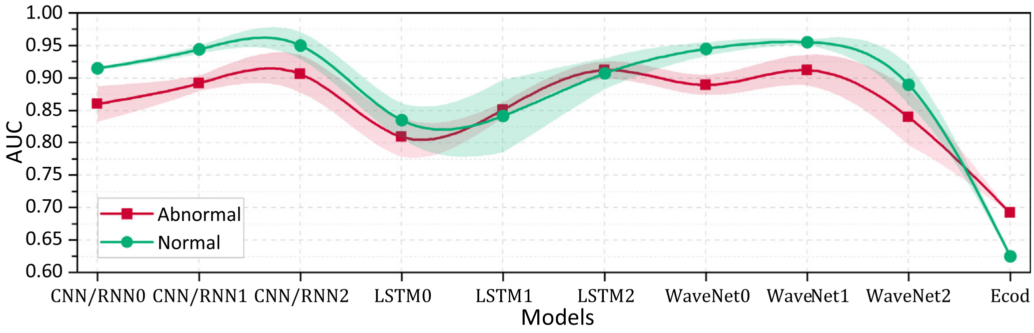


Fig. 9. Error bars of 10 models trained on normal and abnormal sequences in DongTing dataset.

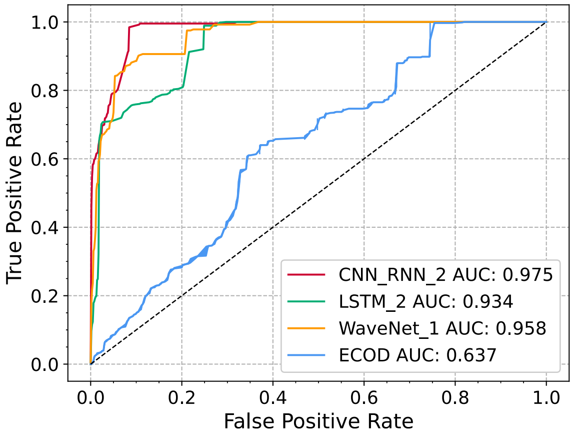


Fig. 10. ROC curves of three models trained on DongTing-Normal.

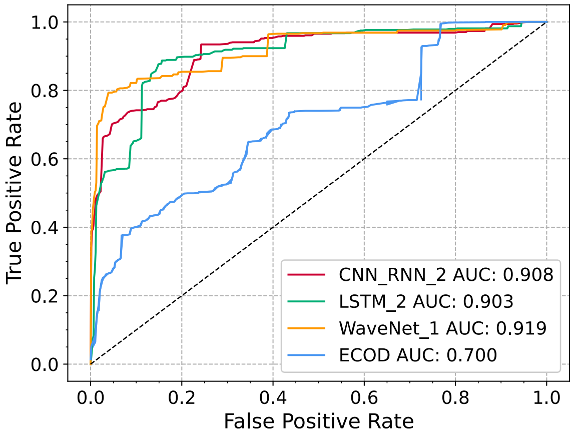


Fig. 11. ROC curves of three models trained on DongTing-Abnormal.

trained models to perform real-time anomaly detection (tracing system calls from applications) in a production environment, and the overhead of tracing need to be considered. Previous work like QTRACE (Cucinotta et al., 2010) have shown that the overhead of tracing can be reduce significantly by batching data transfer from the kernel to user space. We plan to use such techniques in our future work.

Training data selection. In the cross-dataset evaluation, we train models with either normal data (DT-Normal) or abnormal data (DT-Abnormal) of the training set or DongTing dataset. This is for a fair comparison, since other related datasets only support training models with normal data. As show in Table 4, the models trained with DT-Abnormal has significantly better generalization ability than the models trained with DT-Normal or other datasets. Since our DongTing dataset involves both normal and abnormal data, it also enables training a model with the mixture of normal and abnormal data (i.e., the whole training set of DongTing dataset). With additional experiments, we have evaluated the performance of the model trained with both normal and abnormal data (denoted as "DongTing-Train"). The model trained on DongTing-Train obtains AUC score of 0.919 on the testing data of DongTing, which is the best among all training data selection choices. This is reasonable since more information is used for training. With respect to the generalization ability, the model trained on DongTing-Train obtains the second-best AUC scores of 0.498 and 0.499 on the testing data of ADFA-LD and PLAID respectively.

Limitations. One limitation is that the number of abnormal sequences is constrained by the number of reported kernel bugs on syzbot. We can overcome this limitation by constantly updating our dataset with our automated data collection framework. Another limitation is that we do not label abnormal sequences with fine-grained subtypes of bugs. For example, memory bugs have multiple different subtypes. Manual classification for all POCs is labor-intensive. We are currently developing automated tools for this task. One possible technical bias is that some kernel bugs can be triggered more easily than others, so more abnormal sequences are collected for these bugs. This may result in prediction biases for certain bugs with the trained models.

6. Related work

In this section, we discuss related work in two aspects. The first is the brief introduction of the development of datasets built from system call traces. The second is about the recent advances in employing machine learning for anomaly detection.

Datasets based on system calls. By investigating the datasets released in the past 25 years (Hafeez et al., 2020; Liu et al., 2018), we can find that these datasets are mainly used for anomaly detection of network services and user-level host applications. The representative ones are UNM, Firefox-SD, SysCall Dataset, ADFA-LD, and PLAID. Although these datasets have shortcomings, they are still widely used by researchers because high quality datasets in specific domains are often not available.

UNM (Forrest et al., 1996) is a dataset released 26 years ago, and consists of 6 subsets. It contains normal and abnormal sequences of only four command-line Linux programs. This dataset far lags behind the development of OS kernels, and may even have errors (Yang et al., 2017).

Firefox-SD (Murtaza et al., 2013) and ADFA-LD (Creech and Hu, 2014) were released in 2013. Firefox-SD is a collection of system call sequences obtained by tracing the execution of Firefox with normal and attack payloads. The types of attacks include memory corruption, integer overflow, null pointer reference, etc. ADFA-LD is built based on a 32-bit Linux platform. The normal data is collected from common network server applications such as Apache, PHP, and FTP. The abnormal data is obtained from running approximate 60 attack payloads such as the initiation of Web exploits, social engineering, remote trigger vulnerabilities, remote password brute force cracking, etc. Both datasets are created for user-level applications and not suitable for intrusion detection of OS kernels.

SysCall (Ezeme et al., 2019) and PLAID (Ring et al., 2021) are two latest datasets. SysCall extracts not only sequences, but also the type and timestamp of system calls. It is proposed for modern cyber threat analysis, but is not publicly available. As for intrusion detection of OS kernels, the latest PLAID dataset was released in 2021 and only contains about 1100 system call sequences of attack data. Thus, its attack data is only used for testing the deep learning models trained from normal data.

The comparison of our dataset with the related datasets is summarized in Table 1. We mainly compare datasets released in the last 10 years which used system call sequences for anomaly detection.

Machine learning-based anomaly detection. Anomaly detection techniques are a well studied research domain and has various applications in malware detection (Forrest et al., 1996), network intrusion detection (Ring et al., 2021; Yehezkel et al., 2021), concurrency errors (Bridges et al., 2019; Zaman et al., 2019; Jewell and Beaver, 2011), Internet of Things (IoT) network (Huertas Celdrán et al., 2022), intelligent driving (Di Biase et al., 2021; Singh et al., 2021), and high-dimensional time series data (Deng and Hooi, 2021). Recently, machine learning has been widely used to identify the presence of malware (Wang et al., 2017) and malicious network traffic (Carter et al., 2022; Karn et al., 2021; Ahmed et al., 2016), and is also a common approach for log analysis and vulnerability classification (Du et al., 2017; Suciu et al., 2022; Wen et al., 2019; Fournier et al., 2021).

Zaman et al. developed a tool called SCMiner (Zaman et al., 2019) for detecting concurrency errors, which uses an unsupervised learning approach to identifying abnormal system call sequences from logs. Ring et al. (2021) built a dataset based on user-level programs, and three deep learning models, CNN/RNN (Chawla et al., 2019), LSTM (Kim et al., 2016), WaveNet (Oord et al., 2016) and ECOD (Li et al., 2022), are selected to train and predict anomalies. Le and Zhang (2022) provided an in-depth analysis of five state-of-the-art deep learning-based models, such as CNN and LSTM, for detecting system anomalies on four public log datasets. Remaining issues to be addressed are also summarized.

7. Conclusion

In this paper, we present the first large-scale dataset for anomaly detection of the Linux kernel. Our dataset covers a wide range of versions of the Linux kernel, and contains 18,966 well-labeled normal and attack sequences. In addition, we introduce in detail how this large-scale dataset is build through a specifically designed automated framework, which can significantly reduce the time consumption for producing our dataset. As for performance evaluation, we investigate and compare four deep learning models, i.e., CNN/RNN, LSTM, WaveNet, and ECOD. Through cross-dataset evaluation, two important observations are made. The first is that the model trained on the our dataset achieves the best generalization scores. The second is that the model trained with the abnormal data shows superior generalization ability than the model trained with normal data. This has important implications for future development of anomaly based intrusion detection. In the future, we plan to collect more abnormal system call sequences and evaluate them in more large-scale models.

CRediT authorship contribution statement

Guoyun Duan: Conceptualization, Methodology, Software, Validation, Writing – original draft. **Yuanzhi Fu:** Validation, Investigation, Formal analysis, writing – original draft. **Minjie Cai:** Funding acquisition, Methodology, Writing – review & editing. **Hao Chen:** Conceptualization, Investigation, Writing – review & editing. **Jianhua Sun:** Supervision, Funding acquisition, project administration.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgment

We would like to thank the anonymous reviewers for their valuable suggestions.

References

2022. AFL. Website. https://github.com/google/AFL (June 2022).

2022. AFL++. Website. https://aflplus.plus/ (June 2022).

Ahmed, M., Naser Mahmood, A., Hu, J., 2016. A survey of network anomaly detection techniques. J. Netw. Comput. Appl. 60, 19–31. http://dx.doi.org/10.1016/j.jnca.2015.11.016.

Akgun, I.U., Aydin, A.S., Zadok, E., 2020. KMLIB: Towards machine learning for operating systems. In: Proceedings of the on-Device Intelligence Workshop, Co-Located with the MLSys Conference. pp. 1–6.

Bridges, R.A., Glass-Vanderlan, T.R., Iannacone, M.D., Vincent, M.S., Chen, Q.G., 2019. A survey of intrusion detection systems leveraging host data. ACM Comput. Surv. 52 (6), http://dx.doi.org/10.1145/3344382.

Carter, J., Mancoridis, S., Galinkin, E., 2022. Fast, lightweight IoT anomaly detection using feature pruning and PCA. In: Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing. SAC '22, Association for Computing Machinery, New York, NY, USA, pp. 133–138. http://dx.doi.org/10.1145/3477314.3508377.

Chawla, A., Lee, B., Fallon, S., Jacob, P., 2019. Host based intrusion detection system with combined CNN/RNN model. In: Alzate, C., Monreale, A., Assem, H., Bifet, A., Buda, T.S., Caglayan, B., Drury, B., García-Martín, E., Gavaldà, R., Koprinska, I., Kramer, S., Lavesson, N., Madden, M., Molloy, I., Nicolae, M.-I., Sinn, M. (Eds.), ECML PKDD 2018 Workshops. Springer International Publishing, Cham, pp. 149–158. http://dx.doi.org/10.1007/978-3-030-13453-2_12.

Chen, H., Mao, Y., Wang, X., Zhou, D., Zeldovich, N., Kaashoek, M.F., 2011. Linux kernel vulnerabilities: State-of-the-art defenses and open problems. In: Proceedings of the Second Asia-Pacific Workshop on Systems. APSys '11, Association for Computing Machinery, New York, NY, USA, http://dx.doi.org/10.1145/2103799.2103805.

Creech, G., Hu, J., 2014. A semantic approach to host-based intrusion detection systems using contiguousand discontiguous system call patterns. IEEE Trans. Comput. 63 (4), 807–819. http://dx.doi.org/10.1109/TC.2013.13.

Cucinotta, T., Checconi, F., Abeni, L., Palopoli, L., 2010. Self-tuning schedulers for legacy real-time applications. In: Proceedings of the 5th European Conference on Computer Systems. EuroSys '10, Association for Computing Machinery, New York, NY, USA, pp. 55–68. http://dx.doi.org/10.1145/1755913.1755921.

Deng, A., Hooi, B., 2021. Graph neural network-based anomaly detection in multivariate time series. In: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 35. pp. 4027–4035.

Di Biase, G., Blum, H., Siegwart, R., Cadena, C., 2021. Pixel-wise anomaly detection in complex driving scenes. In: 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition. CVPR, pp. 16913–16922. http://dx.doi.org/10.1109/CVPR46437.2021.01664.

Du, M., Li, F., Zheng, G., Srikumar, V., 2017. DeepLog: Anomaly detection and diagnosis from system logs through deep learning. CCS '17, Association for Computing Machinery, New York, NY, USA, pp. 1285–1298. http://dx.doi.org/10.1145/3133956.3134015.

Duan, G., Fu, Y., Zhang, B., Deng, P., Sun, J., Chen, H., Chen, Z., 2023. TEEFuzzer: A fuzzing framework for trusted execution environments with heuristic seed mutation. Future Gener. Comput. Syst. 144, 192–204. http://dx.doi.org/10.1016/j.future.2023.03.008.

Ezeme, O.M., Mahmoud, Q.H., Azim, A., Michael, L., 2019. SysCall dataset: A dataset for context modeling and anomaly detection using system calls. http://dx.doi.org/10.17632/vfvw7g8s8h.2.

Forrest, S., Hofmeyr, S., Somayaji, A., Longstaff, T., 1996. A sense of self for unix processes. In: Proceedings 1996 IEEE Symposium on Security and Privacy. pp. 120–128. http://dx.doi.org/10.1109/SECPRI.1996.502675.

Fournier, Q., Aloise, D., Azhari, S.V., Tetreault, F., 2021. On improving deep learning trace analysis with system call arguments. In: 2021 IEEE/ACM 18th International Conference on Mining Software Repositories. MSR, pp. 120–130. http://dx.doi.org/10.1109/MSR52588.2021.00025.

2022. Glibc testsuite. Website. https://sourceware.org/glibc/wiki/Testing/Testsuite/ (May 2022).

Hafeez, I., Antikainen, M., Ding, A.Y., Tarkoma, S., 2020. IoT-KEEPER: Detecting malicious IoT network activity using online traffic analysis at the edge. IEEE Trans. Netw. Serv. Manag. 17 (1), 45–59. http://dx.doi.org/10.1109/TNSM.2020.2966951.

Han, W., Xue, J., Wang, Y., Zhang, F., Gao, X., 2021. APTMalInsight: Identify and cognize APT malware based on system call information and ontology knowledge framework. Inform. Sci. 546, 633–664. http://dx.doi.org/10.1016/j.ins.2020.08.095.

Huertas Celdrán, A., Sánchez Sánchez, P.M., Sisi, F., Bovet, G., Martínez Pérez, G., Stiller, B., 2022. Creation of a dataset modeling the behavior of malware affecting the confidentiality of data managed by IoT devices. In: Nedjah, N., Abd El-Latif, A.A., Gupta, B.B., Mourelle, L.M. (Eds.), Robotics and AI for Cybersecurity and Critical Infrastructure in Smart Cities. Springer International Publishing, Cham, pp. 193–225. http://dx.doi.org/10.1007/978-3-030-96737-6_11.

Jewell, B., Beaver, J., 2011. Host-based data exfiltration detection via system call sequences. In: ICIW2011-Proceedings of the 6th International Conference on Information Warfare and Secuirty: ICIW. Academic Conferences Limited, p. 134.

Karn, R.R., Kudva, P., Huang, H., Suneja, S., Elfadel, I.M., 2021. Cryptomining detection in container clouds using system calls and explainable machine learning. IEEE Trans. Parallel Distrib. Syst. 32 (3), 674–691. http://dx.doi.org/10.1109/TPDS.2020.3029088.

Kim, K., Jeong, D.R., Kim, C.H., Jang, Y., Shin, I., Lee, B., 2020. HFL: Hybrid fuzzing on the linux kernel. In: NDSS. http://dx.doi.org/10.14722/ndss.2020.24018.

Kim, G., Yi, H., Lee, J., Paek, Y., Yoon, S., 2016. LSTM-based system-call language modeling and robust ensemble method for designing host-based intrusion detection systems. arXiv preprint arXiv:1611.01726.

Le, V.-H., Zhang, H., 2022. Log-based anomaly detection with deep learning: How far are we? In: Proceedings of the 44th International Conference on Software Engineering. ICSE '22, Association for Computing Machinery, New York, NY, USA, pp. 1356–1367. http://dx.doi.org/10.1145/3510003.3510155.

Li, Z., Zhao, Y., Hu, X., Botta, N., Ionescu, C., Chen, G., 2022. ECOD: Unsupervised outlier detection using empirical cumulative distribution functions. IEEE Trans. Knowl. Data Eng. 1. http://dx.doi.org/10.1109/TKDE.2022.3159580.

Lin, Z., Chen, Y., Wu, Y., Yu, C., Mu, D., Xing, X., Li, K., 2022. GREBE: Unveiling exploitation potential for linux kernel bugs. In: 2022 2022 IEEE Symposium on Security and Privacy (SP). SP, IEEE Computer Society, Los Alamitos, CA, USA, pp. 1217–1234. http://dx.doi.org/10.1109/SP46214.2022.00071.

2022. Linux kernel. Website. https://elixir.bootlin.com/linux/v5.17/source/arch/x86/entry/syscalls/syscall_64.tbl (May 2022).

2022. Linux kernel selftests. Website. https://www.kernel.org/doc/html/v4.16/dev-tools/kselftest.html (May 2022).

2022. Linux testing project. Website. https://linux-test-project.github.io/ (May 2022).

Lippmann, R., Fried, D., Graf, I., Haines, J., Kendall, K., McClung, D., Weber, D., Webster, S., Wyschogrod, D., Cunningham, R., Zissman, M., 2000. Evaluating intrusion detection systems: the 1998 DARPA off-line intrusion detection evaluation. In: Proceedings DARPA Information Survivability Conference and Exposition. DISCEX'00, Vol. 2. pp. 12–26. http://dx.doi.org/10.1109/DISCEX.2000.821506, vol.2.

Liu, M., Xue, Z., Xu, X., Zhong, C., Chen, J., 2018. Host-based intrusion detection system with system calls: Review and future trends. ACM Comput. Surv. 51 (5), http://dx.doi.org/10.1145/3214304.

McHugh, J., 2000. Testing intrusion detection systems: A critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by lincoln laboratory. ACM Trans. Inf. Syst. Secur. 3 (4), 262–294. http://dx.doi.org/10.1145/382912.382923.

Mu, D., Wu, Y., Chen, Y., Lin, Z., Yu, C., Xing, X., Wang, G., 2022. An in-depth analysis of duplicated linux kernel bug reports. In: NDSS. http://dx.doi.org/10.14722/ndss.2022.24159.

Murtaza, S.S., Khreich, W., Hamou-Lhadj, A., Couture, M., 2013. A host-based anomaly detection approach by representing system calls as states of kernel modules. In: 2013 IEEE 24th International Symposium on Software Reliability Engineering. ISSRE, pp. 431–440. http://dx.doi.org/10.1109/ISSRE.2013.6698896.

Oliveira, D., 2021. Recommending code understandability improvements based on code reviews. In: 2021 36th IEEE/ACM International Conference on Automated Software Engineering Workshops. ASEW, pp. 131–132. http://dx.doi.org/10.1109/ASEW52652.2021.00035.

Oord, A.v.d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., Kavukcuoglu, K., 2016. Wavenet: A generative model for raw audio. arXiv preprint arXiv:1609.03499.

2022. Open posix test suite. Website. http://posixtest.sourceforge.net/ (May 2022).

Pailoor, S., Aday, A., Jana, S., 2018. Moonshine: Optimizing OS fuzzer seed selection with trace distillation. In: Proceedings of the 27th USENIX Conference on Security Symposium. SEC '18, USENIX Association, USA, pp. 729–743.

Qiu, Y., Liu, H., Anderson, T., Lin, Y., Chen, A., 2021. Toward reconfigurable kernel datapaths with learned optimizations. In: Proceedings of the Workshop on Hot Topics in Operating Systems. pp. 175–182.

Ring, J.H., Van Oort, C.M., Durst, S., White, V., Near, J.P., Skalka, C., 2021. Methods for host-based intrusion detection with deep learning. Digit. Threats 2 (4), http://dx.doi.org/10.1145/3461462.

Sánchez, P.M.S., Valero, J.M.J., Celdrán, A.H., Bovet, G., Pérez, M.G., Pérez, G.M., 2021. A survey on device behavior fingerprinting: Data sources, techniques, application scenarios, and datasets. IEEE Commun. Surv. Tutor. 23 (2), 1048–1077. http://dx.doi.org/10.1109/COMST.2021.3064259.

Schumilo, S., Aschermann, C., Gawlik, R., Schinzel, S., Holz, T., 2017. kAFL: Hardware-assisted feedback fuzzing for OS kernels. In: 26th USENIX Security Symposium (USENIX Security 17). USENIX Association, Vancouver, BC, pp. 167–182.

Singh, Y., Kuruvila, A.P., Basu, K., 2021. Hardware-assisted detection of malware in automotive-based systems. In: 2021 Design, Automation & Test in Europe Conference & Exhibition. DATE, pp. 1763–1768. http://dx.doi.org/10.23919/DATE51398.2021.9474053.

Studiawan, H., Sohel, F., Payne, C., 2021. Anomaly detection in operating system logs with deep learning-based sentiment analysis. IEEE Trans. Dependable Secure Comput. 18 (5), 2136–2148. http://dx.doi.org/10.1109/TDSC.2020.3037903.

Suciu, O., Nelson, C., Lyu, Z., Bao, T., Dumitras, T., 2022. Expected exploitability: Predicting the development of functional vulnerability exploits. In: 31st USENIX Security Symposium (USENIX Security 22). USENIX Association, Boston, MA.

2022. Suse. Website. https://www.suse.com/zh-cn/support/kb/doc/?id=000019587 (May 2022).

2022. Syzbot dashboard. Website. https://syzkaller.appspot.com/upstream/fixed (Access May 2022).

2022. Syzkaller. Website. https://github.com/google/syzkaller (June 2022).

Thongtanunam, P., Kula, R.G., Cruz, A.E.C., Yoshida, N., Iida, H., 2014. Improving code review effectiveness through reviewer recommendations. In: Proceedings of the 7th International Workshop on Cooperative and Human Aspects of Software Engineering. In: CHASE 2014, Association for Computing Machinery, New York, NY, USA, pp. 119–122. http://dx.doi.org/10.1145/2593702.2593705.

2022. Ubuntu. Website. https://ubuntu.com/about/release-cycle#ubuntu-kernel-release-cycle (May 2022).

Umit Akgun, I., Selman Aydin, A., Shaikh, A., Velikov, L., Burford, A., McNeill, M., Arkhangelskiy, M., Zadok, E., 2021. KML: Using machine learning to improve storage systems. arXiv e-prints arXiv–2111.

Wang, Q., Guo, W., Zhang, K., Ororbia, A.G., Xing, X., Liu, X., Giles, C.L., 2017. Adversary resistant deep neural networks with an application to malware detection. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD '17, Association for Computing Machinery, New York, NY, USA, pp. 1145–1153. http://dx.doi.org/10.1145/3097983.3098158.

Wang, D., Zhang, Z., Zhang, H., Qian, Z., Krishnamurthy, S.V., Abu-Ghazaleh, N., 2021. {SyzVegas}: Beating kernel fuzzing odds with reinforcement learning. In: 30th USENIX Security Symposium (USENIX Security 21). pp. 2741–2758.

Wen, Y., Cao, J., Cheng, S., 2019. PTracer: A linux kernel patch trace bot. In: 2019 34th IEEE/ACM International Conference on Automated Software Engineering. ASE, pp. 1210–1211. http://dx.doi.org/10.1109/ASE.2019.00140.

Yang, C., Puthal, D., Mohanty, S.P., Kougianos, E., 2017. Big-sensing-data curation for the cloud is coming: A promise of scalable cloud-data-center mitigation for next-generation IoT and wireless sensor networks. IEEE Consum. Electron. Mag. 6 (4), 48–56. http://dx.doi.org/10.1109/MCE.2017.2714695.

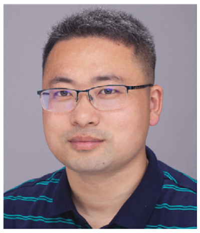
Yehezkel, A., Elyashiv, E., Soffer, O., 2021. Network anomaly detection using transfer learning based on auto-encoders loss normalization. In: Proceedings of the 14th ACM Workshop on Artificial Intelligence and Security. AISec '21, Association for Computing Machinery, New York, NY, USA, pp. 61–71. http://dx.doi.org/10.1145/3474369.3486869.

Zaman, T.S., Han, X., Yu, T., 2019. SCMiner: Localizing system-level concurrency faults from large system call traces. In: 2019 34th IEEE/ACM International Conference on Automated Software Engineering. ASE, pp. 515–526. http://dx.doi.org/10.1109/ASE.2019.00055.

Zhang, Y., Huang, Y., 2019. "Learned": Operating systems. SIGOPS Oper. Syst. Rev. 53 (1), 40–45. http://dx.doi.org/10.1145/3352020.3352027.

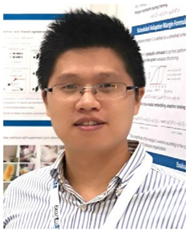
Zhang, L., Morin, B., Baudry, B., Monperrus, M., 2022. Maximizing error injection realism for chaos engineering with system calls. IEEE Trans. Dependable Secure Comput. 19 (4), 2695–2708. http://dx.doi.org/10.1109/TDSC.2021.3069715.

Zhou, J., Zhang, T., Shen, W., Lee, D., Jung, C., Azab, A., Wang, R., Ning, P., Ren, K., 2022. Automatic permission check analysis for linux kernel. IEEE Trans. Dependable Secure Comput. 1. http://dx.doi.org/10.1109/TDSC.2022.3165368.



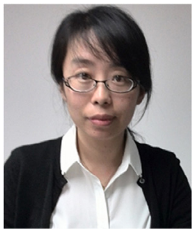
Guoyun Duan (Member, IEEE) is a Ph.D. student at the College of Computer Science and Electronic Engineering, Hunan University, China. His research interests are systems security, heterogeneous systems and privacy security. He is a member of the IEEE and CCF.

Yuanzhi Fu is a Ph.D. student at the College of Computer Science and Electronic Engineering, Hunan University, China. His research interests are in systems security.



Minjie Cai received the B.S. and M.S. degrees in electronics and information engineering from Northwestern Polytechnical University, China, in 2008 and 2011, respectively, and the Ph.D. degree in information science and technology from The University of Tokyo, in 2016. He is currently an Associate Professor with the College of Computer Science and Electronic Engineering, Hunan University, China. He also serves as a cooperative research fellow at The University of Tokyo. His research interests include computer vision, multimedia analysis, and human–computer interaction.

Hao Chen (Member, IEEE, and ACM) received the B.S. degree in chemical engineering from Sichuan University, China, in 1998, and the Ph.D. degree in computer science from Huazhong University of Science and Technology, China in 2005. He is now a professor at the College of Computer Science and Electronic Engineering, Hunan University, China. His current research interests include parallel and distributed computing, operating systems, cloud computing and systems security. He has publication in prestigious international journals and conferences, such as IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Computers, Future Generation Computer Systems, ASPLOS, SIGMOD, VLDB, USENIX ATC, PACT, ICS, IPDPS and ICPP. He is a member of the IEEE and the ACM.



Jianhua Sun is a professor at the College of Computer Science and Electronic Engineering, Hunan University, China. She received the Ph.D. degree in Computer Science from Huazhong University of Science and Technology, China in 2005. Her research interests are in systems security, parallel and distributed computing, and operating systems. She has published more than 70 papers in prestigious journals and conferences, such as IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Computers, FGCS, VLDB, PACT, ICPP, and IPDPS.