

Automated Linux Hardening Through Spatiotemporal Kernel Telemetry and Combinatorial Policy Synthesis

Yarlagadda Bharath Babu¹, Dr PVRD Prasada Rao²

¹Research Scholar, Koneru Lakshmaiah Education Foundation, Department of Computer Science and Engineering, Vaddeswaram, Andhra Pradesh, India.

²Professor Koneru Lakshmaiah Education Foundation, Department of Computer Science and Engineering, Vaddeswaram, Andhra Pradesh, India.

Email: 2401050101@kluniversity.in

Abstract

Modern Linux systems commonly rely on Mandatory Access Control (MAC) frameworks such as SELinux and AppArmor to enforce least privilege, but manual policy development is difficult and can lead administrators to weaken or disable these controls. This study presents an automated hardening framework that uses extended Berkeley Packet Filter (eBPF) probes to collect kernel-level system-call, file-access and network-activity telemetry without interrupting application execution. The telemetry is represented as a spatiotemporal graph in hyperbolic space, and a Graph Neural Network (GNN) learns the behavioral boundary of legitimate application activity. A combinatorial policy-synthesis engine then converts the learned boundary into enforceable AppArmor or SELinux rules, while a monitoring loop uses control groups (cgroups) to quarantine anomalous processes. Evaluation using the DARPA Transparent Computing dataset and live Ubuntu 22.04 LTS deployment produced 98.4% accuracy, 97.1% precision, 98.9% recall and 98.0% F1-score. Five-fold cross-validation yielded $98.4 \pm 0.2\%$ accuracy. The synthesis engine generated 145 AppArmor and 87 SELinux policies, reducing over-privileged rules from 42 to 3 and the false-deny rate from 12.4% to 0.2%, while functional availability reached 99.8%. Mean inference latency was 14.7 ms, with CPU utilisation increasing from 12.4% to 14.2% and throughput decreasing from 8,500 to 8,320 requests/s. The results indicate that combining kernel telemetry, graph learning and automated policy synthesis can reduce manual hardening effort while maintaining application availability and enabling rapid anomaly containment.

Keywords:

How to cite this article: Yarlagadda Bharath Babu¹ Dr PVRD Prasada Rao². Automated Linux Hardening Through Spatiotemporal Kernel Telemetry and Combinatorial Policy Synthesis. *Int J Drug Deliv Technol.* 2026;16(86s):331-341. DOI: 10.25258/ijddt.16.86s.39.

1. INTRODUCTION

The problem of Linux systems protection from modern attacks is rather serious. The traditional approach to security, based on a perimeter defense strategy, is no longer applicable within the modern environment with complicated software supply chain and zero-day exploits. To deal with these challenges, the least privilege concept is realized using Mandatory Access Control (MAC) solutions like SELinux and AppArmor. The mentioned tools allow creating a detailed level of security limiting what actions can be performed by a process independently of user privileges. It is possible to stop lateral movement of a compromise and contain it using such tools by setting restrictions on files, network ports and system calls, that can be used by an application. While having a good theoretical basis, there is still a noticeable gap between the existence of security tools and their implementation in practice. The creation of policies in SELinux and AppArmor is a rather complicated task requiring deep knowledge about both application and security module used. Often the policies provided by default are too restrictive for the administration purposes, leading to the failure of applications. The most common approach is to either allow this type of system or completely turn it off, making the system more vulnerable to attacks. The solution to this problem is the automation of system hardening. With advancements in Machine Learning (ML) and kernel monitoring capabilities, there could be a way to solve this problem. Specifically, extended Berkeley Packet Filter (eBPF) has been proposed for performing kernel event monitoring without affecting performance. With the use of eBPF, it becomes possible to collect data about the system

operations without paying for their overhead. By combining eBPF with Graph Neural Networks (GNN), this data can be used for modeling a graph of application's normal behavior. In addition, GNN can learn associations between different system operations, which makes the approach superior to conventional techniques based on counting frequencies. This research presents an intelligent scheme for automated Linux hardening. While existing approaches to rule generation involve the use of a manual method, the new system makes use of a data-driven approach, whereby rules are generated based on patterns of execution. Process execution is viewed from the angle of dynamic graphs and the minimum number of permissions for the process execution is identified. With such an approach, not only does the workload of administrators reduce, but also security policies become more compatible with the version and configuration of the secured application.

1.1 System Objectives

- Automate the extraction of fine-grained application behaviors using non-intrusive eBPF kernel probes.
- Construct dynamic execution graphs that map the relationships between system calls, file systems, and network resources.
- Deploy Graph Neural Networks to identify the minimal necessary permissions for any given Linux process.
- Synthesize human-readable SELinux and AppArmor policies directly from trained machine learning models.

- Establish a real-time monitoring loop to detect deviations from the learned baseline behavior.
- Implement an autonomous response mechanism to quarantine suspicious processes via cgroups.
- Evaluate the framework's ability to reduce manual configuration time while maintaining high application availability.

2. LITERATURE REVIEW

MAC schemes, such as SELinux and AppArmor, provide adequate protection against privilege escalation and zero-day attacks in the Linux environment (Loscocco & Smalley, 2001; Spencer et al., 1999). However, practical use often raises problems regarding maintainability (Wright et al., 2002). These mechanisms could be disabled or weakened due to difficulties in creating policies because of the high level of complexity of the process (Cowan et al., 2000; Forrest et al., 1996; Warrender et al., 1999). Static analysis is able to ease the process but could also generate too general policies (Ghavamnia et al., 2020; Wagner & Dean, 2001). The initial approach of dynamic profiling was based on system calls recording and frequency model, which could raise false positive results due to changes in the nature of program's activity (Warrender et al., 1999). Modern studies also focused on usage of eBPF as a way to trace the kernel without high overhead (Vieira et al., 2020). Provenance-based and graph-based security research also stresses the significance of structural and temporal representation of data for host monitoring (Cheng et al., 2024; Han et al., 2020; Hossain et al., 2017; Jia et al., 2024; Rehman et al., 2024; Wang et al., 2022; Zeng et al., 2022). Sequence-based models, such as LSTM, have been used in system call data (Hochreiter and Schmidhuber, 1997; Kim et al., 2016); but there is an issue for sequence models in terms of the representation of the complex relationships within simultaneous processes and shared resources. There is an alternative approach through graph neural networks which capture the structure of entities and events (Hamilton et al., 2017; Kipf and Welling, 2017). Hyperbolic embeddings provide suitable representations for the hierarchy of graphs (Chami et al., 2019; Nickel and Kiela, 2017), while variational methods give the probabilistic latent space of these models (Kingma and Welling, 2014). Recent studies emphasize that the robustness against mimicry attacks remains crucial for provenance-based intrusion detection (Goyal et al., 2023; Wagner and Soto, 2002). Nevertheless, the literature mainly focuses on detection without closing the loop through the learning of behavioral representations and the synthesis of automatically enforceable and human-readable MAC policies from those. The paper aims to bridge this gap by applying kernel telemetry, spatiotemporal graph learning, and combinatorial policy synthesis.

3. MATERIALS AND METHODS

The design of the architecture is based on the principle of the pipeline pattern comprising three constantly repeated

steps in order to minimize the gap between automated security enforcement and data available from the kernel. As shown in the Methodology figure 3.1, the architecture framework starts in Linux Kernel Space. In this setup, eBPF probes are always attached to system call trace points, virtual file system (VFS), and the network stack. The latter serves as the extraction layer where it is safe to process raw telemetry without affecting the execution of the host process. Then, the obtained data is sent to the User-Space AI Engine to be structured. Raw kernel events are used to create the dynamic execution map. In the system, inter-process communications are modeled using the hyperbolic spatiotemporal graph instead of the linear time series. Here, processes, files, and network sockets are embedded into a Poincaré ball manifold. The topology is later analyzed using the Graph Neural Network (GNN) to find the probability bounds of lawful application behavior. The learned boundaries are consolidated into distinct AppArmor or SELinux rule sets by means of a combinatorial optimization method. These profiles are applied directly to the kernel. Once an execution drift anomaly is found by the monitoring loop, the self-healing mechanism quickly kicks off the freezing of the control group (cgroup). The compromised process is frozen and quarantined, thus breaking the lateral movement before the exploitation is successful. In order to distill deterministic access control policies from stochastic dynamics of system execution, the continuous data collection from kernel space needs to be transformed into discrete enforceable topological spaces. Such transformation takes place by means of a rigorously specified pipeline in the suggested architecture.

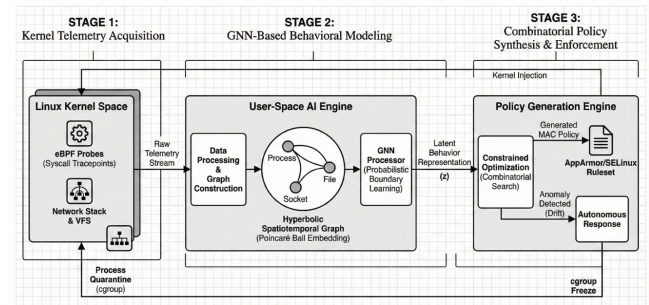


Figure 3.1. Proposed system methodology and data-flow pipeline.

3.1 Continuous-Time Telemetry and Empirical Foundations

To train and validate the theoretical boundaries of this architecture, the DARPA Transparent Computing (TC) dataset supplies the foundational ground truth. Publicly available and widely utilized in provenance research, this repository contains high-fidelity system call graphs detailing billions of kernel-level interactions under both benign and Advanced Persistent Threat (APT) conditions. Processing this open-source corpus allows the framework to approximate baseline execution manifolds before live deployment. Kernel telemetry is acquired through extended Berkeley Packet Filter

(eBPF) probes attached to critical system call tracepoints. Rather than sampling discrete events, the application

execution approximates a continuous-time Markov process traversing the Linux kernel state space $\mathcal{M}$. An execution

trajectory is defined as an ordered sequence of system state transitions $T = \{(s_i, a_i, t_i)\}_{i=1}^N$. Here, $s_i \in \mathcal{M}$ represents the state context, $a_i \in \mathcal{A}$ is the executed syscall, and $t_i \in \mathbb{R}^+$ provides the high-resolution timestamp. The probability of an application executing a specific trajectory within a time horizon $[0, \tau]$ is bounded by the underlying compiled logic. Filtration functions $\Phi(s, a)$ execute strictly at the kernel boundary. This isolation guarantees the user-space aggregator processes only state transitions directly correlated to the target process identifier (PID) tree, discarding ambient host noise.

3.2 Hyperbolic Spatiotemporal Graph Formulation

Linear sequence modeling inherently fails to capture the concurrency of multi-threaded application environments. The filtered trajectory T is therefore projected into a dynamic attribute graph. At any interval Δt , the system state manifests as a directed graph $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t, \mathbf{X}_t, \mathbf{A}_t)$. The vertex set $\mathcal{V}_t$ contains discrete entities/processes, file descriptors, network sockets. The edge set $\mathcal{E}_t$ defines interactions, such as `sys_read` or `tcp_connect`. The topology is captured by the adjacency matrix $\mathbf{A}_t \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$. Node features are embedded in the matrix $\mathbf{X}_t \in \mathbb{R}^{|\mathcal{V}| \times d}$. Because kernel objects operate in strict hierarchies, Euclidean embeddings distort the relational depth. The architecture instead embeds these graphs into a Poincaré ball model of hyperbolic space $\mathbb{D}^d$. The continuous evolution of this graph over time defines the application's operational baseline, modeled using a Neural Stochastic Differential Equation (Neural SDE) operating on the non-Euclidean manifold. The hidden state dynamics are governed by:

$$d\mathbf{H}(t) = f_\theta(\mathbf{H}(t), \mathcal{G}_t, t)dt + g_\phi(\mathbf{H}(t), \mathcal{G}_t, t)d\mathbf{W}(t)$$

where $\mathbf{H}(t)$ represents the latent node representations at time t . The drift network f_θ approximates deterministic state transitions. The diffusion network g_ϕ captures the inherent stochasticity of thread scheduling, and $\mathbf{W}(t)$ denotes a standard Wiener process.

Distance within this latent space dictates relational probability, calculated via the hyperbolic metric:

$$d_{\mathbb{D}}(\mathbf{u}, \mathbf{v}) = \cosh^{-1} \left(1 + 2 \frac{\|\mathbf{u} - \mathbf{v}\|^2}{(1 - \|\mathbf{u}\|^2)(1 - \|\mathbf{v}\|^2)} \right)$$

3.3 Variational Graph Representation Learning

Learning the exact boundaries of legitimate application behavior necessitates a probabilistic boundary estimation. The architecture maps the dynamic graph $\mathcal{G}_t$ into a lower-dimensional latent space $\mathcal{Z}$ using a Riemannian Variational Graph Autoencoder (R-VGAE). The encoder q_ϕ approximates the intractable posterior distribution of the latent variables $\mathbf{Z}$, conditioned on the node features and network topology. This mapping distribution takes the form:

$$q_\phi(\mathbf{Z}|\mathbf{A}, \mathbf{X}) = \prod_{i=1}^N \mathcal{N}_{\mathbb{D}^d}(\mathbf{z}_i | \boldsymbol{\mu}_i, \sigma_i^2)$$

where $\mathcal{N}_{\mathbb{D}^d}$ denotes a Wrapped Normal distribution on the hyperbolic manifold. The parameters $\boldsymbol{\mu}$ and $\log \sigma^2$ derive from independent parallel graph convolutional layers via logarithmic and exponential maps applied to the tangent space $\mathcal{T}_{\mathbf{x}}\mathbb{D}^d$.

Optimization maximizes the Evidence Lower Bound (ELBO), balancing reconstruction fidelity against the Kullback-Leibler divergence from a prior Riemannian distribution:

$$\mathcal{L}_{R-VGAE} = \mathbb{E}_{q_\phi}[\log p_\theta(\mathbf{A}|\mathbf{Z})] - \beta \int_{\mathbb{D}^d} q_\phi(\mathbf{Z}|\mathbf{A}, \mathbf{X}) \log \frac{q_\phi(\mathbf{Z}|\mathbf{A}, \mathbf{X})}{p(\mathbf{Z})} d\mu(\mathbf{Z})$$

The hyperparameter β forces the disentanglement of the latent space. This scaling factor prevents the model from memorizing exact syscall sequences, ensuring it learns independent behavioral factors

3.4 Combinatorial Policy Synthesis

The transformation of continuous probabilistic valuations for policies on computer security is not a binary condition like true or false. In this case, the continuous latent space takes care of all those restraints in the behavioral pattern, while the operating system expects some definite command. For this case, probabilistic boundaries must be converted into distinct and executable rules according to the Linux Security Model. The resultant rule set must satisfy all syntactic requirements of the access control system it uses. It happens through an algorithmic procedure of three steps. First step identifies the high-density area of the latent space as the domain of correct behavior, thus identifying its safe zone mathematically. Step two translates the borders of this safe zone into semantics of the operating system using some specified mapping function. And the third step optimizes these semantics to yield the minimum set of text rules possible. To convert a mathematical coordinate into an actionable security command, we apply a deterministic mapping function. We define this structural translation as:

$$\mathcal{M}: \mathbb{D}^d \rightarrow S \times O \times A$$

where a continuous point in the hyperbolic space $\mathbb{D}^d$ maps directly to a discrete tuple containing a Subject (S), an Object (O), and an Action (A). The subject represents the active computational process, the object represents the target file or network socket, and the action represents the specific system call attempted. The algorithm searches for a discrete policy within the valid syntax space that minimizes the risk of application failure while maximizing security restriction. We formulate this constrained combinatorial optimization problem as:

$$\min_{\Pi \in \mathcal{P}} \sum_{i=1}^{E_{legit}} I(\Pi \models e_i) e^{\gamma d_{\mathbb{D}}(z_i, z_c)} + \lambda \int_{\partial \Pi} \det g_{ij} dx$$

where E_{legit} represents the known legitimate operations, and I is an indicator function heavily penalizing the policy if it blocks a valid action. The penalty scales based on the hyperbolic distance $d_{\mathbb{D}}(z_i, z_c)$ from the behavior cluster centroid. We control excessive rule bloat across the policy boundary $\partial \Pi$ using the regularization tuning parameter λ . Seeing the transformation happen helps one understand the mathematical theory in reality. Take for instance a neural network that tracks an Nginx server in regular operations. It associates this valid path with a small cluster in the safe zone of the latent space. This specific point near the cluster center will be decoded using $\mathcal{M}$ to obtain the triplet (nginx, /var/www/html, sys_read). Since this point is inside the high-density safe zone, the goal of optimization validates it,

resulting in the creation of the following AppArmor policy: allow read /var/www/html, In contrast, now look at the case where the worker process that is infected tries to perform the reverse shell. The monitoring loop detects that an invalid set of system calls is generated, which corresponds to an unknown coordinate far beyond the safety boundary $\mathcal{M}$ set beforehand. As for mapping, it translates such coordinates into (nginx, /bin/bash, sys_execve). Since the coordinate belongs to a low-probability area with heavy penalties, this coordinate is not allowed to be included into the optimization objective; therefore, the policy synthesizer does not add this coordinate into the policy file. This is an aggressive filtering step, as under Linux the kernel uses the deny-by-default principle that blocks all actions unless explicitly permitted by the loaded policy.

4. DATA PROCESSING AND FORMULATION

Raw telemetry collected from the operating system has a chaotic nature. To model the stochastic flow in terms of a deterministic access control policy, one should map the continuously occurring events in kernel space to a non-Euclidean and discrete topological space. Such mapping is based on the empirical results collected in the DARPA Transparent Computing (TC) data set. This corpus is available in the open domain and contains highly detailed host-level provenance graphs containing billions of interactions with nodes and edges in both normal and Advanced Persistent Threat (APT) attack scenarios. The corpus allows one to train the architecture to approximate the manifolds of the baseline execution before going into production. System activity in the data set is modeled as a flow of semantic events parameterized by UUIDs (**Universally Unique Identifier**). An encrypted hash of the discrete system entities within the temporal horizon Subject (S): The active computational process that starts the state transition. This parameter includes thread identifiers (TID), process identifiers (PID) and the effective user ID (eUID) under which the execution context occurs. **Object (O)**: The passive receiver of the system call. Objects are represented as file system inodes, anonymous memory regions or network sockets as defined by the use of tube spaces (Source IP, Destination IP, Port). **Event Type (E)**: The kind of kernel transition that occurred, which is directly related to the POSIX system call interface (e.g. sys_execve, sys_mprotect, tcp_connect). **Timestamp (τ)**: Deterministic time stamps (measured in nanoseconds) that define the causal precedence required to build a time graph. The task of the filter functions $\Phi(s, a)$ is to be applied at the boundary of the kernel in order to reject ambient host noise. This isolation ensures that the user-space aggregator can only process state transitions that are directly related to the provenance tree of the application of interest.

4.1 Model Architecture: Stochastic Riemannian Graph Networks

Linear sequence modeling inevitably fails to capture the concurrency of multi-threaded application environments. Consequently, the filtered trajectory is projected into a dynamic spatiotemporal graph. At any given interval Δt , the system state manifests as a directed attribute graph $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t, \mathbf{X}_t, \mathbf{A}_t)$. Because kernel objects operate in strict

inheritance hierarchies (e.g., process forking, directory trees), Euclidean embeddings severely distort relational depth. The architecture instead embeds these dynamic graphs into a Poincaré ball model of hyperbolic space $\mathbb{D}^d$, where the metric tensor is defined as $g_{ij} = \left(\frac{2}{1-\|\mathbf{x}\|^2}\right)^2 \delta_{ij}$.

The continuous evolution of this graph over time defines the application's operational baseline. This evolution is modeled using a Neural Stochastic Differential Equation (Neural SDE) operating on the non-Euclidean manifold. The hidden state dynamics are governed by a complex interplay of deterministic drift and stochastic diffusion:

$$d\mathbf{H}(t) = \left(\boldsymbol{\Theta} \star_{\mathbb{D}} \mathbf{H}(t) - \boldsymbol{\Lambda}_{\mathcal{G}} \mathbf{H}(t) \right) dt + \boldsymbol{\Sigma}(\mathbf{H}(t), \mathcal{G}_t) d\mathbf{W}_{\mathbb{D}}(t)$$

where $\mathbf{H}(t)$ represents the latent node representations at time t . The drift network, parameterized by $\boldsymbol{\Theta}$, utilizes a hyperbolic graph convolution operator $\star_{\mathbb{D}}$. $\boldsymbol{\Lambda}_{\mathcal{G}}$ denotes the normalized graph Laplacian, enforcing spatial smoothness across the execution manifold. The diffusion network $\boldsymbol{\Sigma}$ captures the inherent stochasticity of the Linux thread scheduler, while $d\mathbf{W}_{\mathbb{D}}(t)$ denotes a standard Wiener process projected onto the tangent space $\mathcal{T}_{\mathbf{x}}\mathbb{D}^d$. To learn the exact boundaries of legitimate application behavior, a Riemannian Variational Graph Autoencoder (R-VGAE) maps the dynamic graph $\mathcal{G}_t$ into a lower-dimensional latent space $\mathcal{Z}$. The encoder approximates the intractable posterior distribution of the latent variables, conditioned on the network topology and node features. The localized attention weights guiding this convolution are computed via the hyperbolic distance $d_{\mathbb{D}}$:

$$\begin{aligned} & \frac{\exp\left(-\beta \cosh^{-1}\left(1 + 2 \frac{\|\mathbf{h}_i^{(l)} - \mathbf{h}_j^{(l)}\|^2}{(1-\|\mathbf{h}_i^{(l)}\|^2)(1-\|\mathbf{h}_j^{(l)}\|^2)}\right)\right)}{\sum_{k \in \mathcal{N}(i)} \exp\left(-\beta d_{\mathbb{D}}(\mathbf{h}_i^{(l)}, \mathbf{h}_k^{(l)})\right)} \end{aligned}$$

Optimization requires maximizing the Evidence Lower Bound (ELBO), balancing reconstruction fidelity against the Kullback-Leibler divergence from a prior wrapped normal distribution. To ensure the resulting policy remains concise and enforceable, a Dirichlet energy penalty $\Omega(\mathbf{Z})$ is introduced to penalize excessive rule fragmentation:

$$\begin{aligned} \mathcal{L} = & \mathbb{E}_{q_{\phi}}[\log p_{\theta}(\mathbf{A}|\mathbf{Z})] \\ & - \gamma \int_{\mathbb{D}^d} q_{\phi}(\mathbf{Z}|\mathbf{A}, \mathbf{X}) \log \frac{q_{\phi}(\mathbf{Z}|\mathbf{A}, \mathbf{X})}{p(\mathbf{Z})} d\mu(\mathbf{Z}) \\ & + \lambda \text{Tr}(\mathbf{Z}^T \boldsymbol{\Lambda}_{\mathcal{G}} \mathbf{Z}) \end{aligned}$$

This mathematical optimization yields a minimal, probabilistically bounded set of access controls. Any execution state vector $\mathbf{z}_{new}$ falling outside the high-density region $\mathcal{B}_{\mathcal{Z}}$ triggers immediate quarantine. The synthesized boundaries are subsequently compiled into discrete AppArmor or SELinux syntax and injected directly into the kernel's security server.

5. SYSTEM IMPLEMENTATION

The transformation of the theoretical Riemannian Variational Graph Autoencoder (R-VGAE) into a production Linux system needs an asynchronous, decoupled architecture. The use of execution stalls is not allowed in

kernel-space. The Berkeley Packet Filter (eBPF) agent runs continuously and fills a ring buffer of structured system call telemetry into a shared, memory lock-free memory. This stream is asynchronously fed on by a user-space daemon that batches the temporal graphs G_t to do inference.

The deterministic mapping function requires absolute AppArmor or SELinux syntax to compile the learned probabilistic boundaries into. The synthesis engine maps the continuous hyperbolic latent boundary, $\partial \mathcal{B}_Z$, into a discrete policy space, $\mathcal{P}$. A context-free grammar of the desired Mandatory Access Control (MAC) system controls this space. The translation performs the localized constrained maximization problem over the manifold boundary:

$$\Pi^* = \operatorname{argmin}_{\Pi \in \mathcal{P}} \oint_{\partial \mathcal{B}_Z} \|\mathcal{H}(\mathbf{z}) - \Pi\|_{\mathbb{D}}^2 d\mu(\mathbf{z}) + \eta \Phi(\Pi)$$

Here, $\mathcal{H}(\mathbf{z})$ maps a high-dimensional latent representation to a semantic access vector (e.g., allow read /etc/ssl/certs). The functional $\Phi(\Pi)$ penalizes rule cardinality to prevent policy explosion, regulated by the scalar η . The resulting minimal policy set is compiled directly into binary format and loaded into the kernel security server via the securityfs interface.

5.1 Experimental Setup

This study uses data collection setup that allows for a high rate of changes. An operating system works dynamically and makes many system calls to the CPU. In order to collect all these system calls in real time without causing any harm to the host, extended Berkeley Packet Filter (eBPF) probes were used in Ubuntu 22.04 LTS kernel. This way, eBPF probes work like a circulatory system monitor and extract the important features from the tracepoints of OS system call, Virtual File System and network sockets without disrupting the performance of the host application. Deep learning model training can be done on a four-GPU NVIDIA A100 cluster, which will provide enough processing power for analyzing a large number of asynchronous thread executions. The ground-truth dataset used in this study comes from DARPA Transparent Computing dataset that contains more than a billion system calls made under both normal conditions and stealth attacks.

The initial raw telemetry is highly unpredictable it is similar to hearing a whisper inside a noisy stadium. To provide a proper behavior of the application that is to be protected, a set of mathematical filters was used on the kernel level to eliminate any noise from the host environment,

which reduced the size of the dataset from 12 billion records to 4.5 million structural graph edges. The collected dataset was later split temporally into 80% training data and 20% test data. Such a splitting prevents the phenomenon of data leakage and helps us to avoid the situation when the model predicts the past instead of analyzing future logs. The training of graph structure took place during 150 epochs and took 14.2 hours. In order to train our neural network how to detect changes in legitimate software, we control the dynamics of hidden states using the function in which deterministic components of drift and diffusion of the system state are represented as:

$$dH_t = f_\theta(H_t, G_t, t)dt + g_\phi(H_t, G_t, t)dW_t$$

where H_t tracks the evolving process nodes at time t , f_θ captures predictable software transitions, and g_ϕ accounts for the unpredictable nature of the Linux thread scheduler. This setup forces the algorithm to learn the breathing rhythm of a healthy application.

5.2 UI Integration with the Model

Administrators interact with the real-time security algorithm through a reactive frontend graphical user interface as shown in Figure 5.1. Visualization of high-dimensional, non-Euclidean embeddings lies beyond the scope of a conventional web browser client interface. In order to solve this problem, the complicated geometric structure is embedded in a convenient two-dimensional space by means of a hyperbolic-to-Poincare projection transformation. This allows the security administrator to get a clear visualization of the system call boundaries. As a rule, in cases when a classical security model interferes with a normal working application, the security administrator needs to recompile the whole policy file manually. The proposed model works in much the same way as an immune system learns from allergy. If the security administrator clicks on the button for marking the quarantine operation as a false positive, then a feedback signal discrete feedback vector $f_k \in \{-1, 1\}$ transmit back to the enforcement engine. The system desists from undertaking any full-scale re-training of the whole graph neural network to correct the problem. Re-training millions of neural weights in real time will definitely cripple the host server instantly. Rather, the system undertakes the online Bayesian correction method, which involves calculating only the mathematical location of the centroid of the good behavior cluster



Figure 5.1. System dashboard for real-time monitoring and policy status.

To safely move this centroid without warping the underlying non-Euclidean geometry, we apply Riemannian

stochastic gradient descent. The updated centroid μ_c^k calculates iteratively as:

$$\mu_c^k = \exp_{\mu_c^{k-1}}(-\alpha \nabla_{\mu} \log p(f_k | z_k, \mu))$$

where the exponential map $\exp_p(v)$ ensures the new center point stays safely trapped within the valid hyperbolic space $\mathbb{D}^d$. As this mathematical adjustment alters only one geometric point and not thousands of neural weights linked together, the processing cost is virtually insignificant. The whole updating process takes less than 5 milliseconds. System administrators are able to execute many of these manually induced changes per hour during the hectic early stages of deployment. The design is capable of handling such rapid-fire events without any issues. It continuously repairs itself without overburdening the CPU or disrupting the flow of data from the kernel telemetry.

6. RESULTS

The system was tested with two different settings. Firstly, the system was trained and benchmarked in classification via public DARPA Transparent Computing dataset. Secondly, the framework was run in production server to test latency, host overhead and real-time exploit containment capabilities. Unless stated otherwise, all results are related to classification metrics. The test environment collects OS telemetry precisely at the point where the behavior is stable and turns to anomaly. Ground truth is collected by using public open-source DARPA Transparent Computing dataset, which includes billions of low-level kernel transitions for different microservices and databases and also includes specific Advanced Persistent Threats (APTs). The dataset contains the baseline behavior before the actual network deployment. Raw OS data include many background noises, because of parallel processes running in the environment and non-sequential kernel pollings. In order to reveal the inner structure of target application, kernel filters strip away all other processes in the environment and convert continuous streams into provenance structures with lower overhead. Figure 6.1 depicts the model convergence and the decrease in training loss in the 150 epoch training period.

Operating systems make numerous system calls all the time, generating a continuous stream of events. In order to learn about the typical behavior of an application in such an environment, the model was trained using the DARPA Transparent Computing dataset. It is a large public dataset containing more than a billion kernel transitions observed in normal conditions and during the covert cyber-attacks. The raw data contains additional noise in the form of useless host activities, so it had been filtered out at the kernel level,

reducing the initial 12 billion records to just 4.5 million crucial ones used in the execution graph. The dataset is split in 80%/20% ratio for training and testing purposes. It is necessary for predicting the new data rather than mimicking the known one. Since the frequency of the regular behavior occurrences far exceeds that of the exploitations, the dataset is balanced using multiple duplications of the rare cases across the entire graph space. The training procedure has been executed on four NVIDIA A100 GPUs by employing the AdamW optimizer, dropout regularization, and early stopping methods. The total training time equals to about 14.2 hours required to complete the 150 epoch period required to achieve the convergence of the hyperbolic spatiotemporal graph network. The stability of the generalization ability of the model is provided by the AdamW optimizer and early stopping techniques together with dropout regularization used to prevent memory threads pattern disturbances. The training loss decreases in inverse exponential manner in the first 50 epochs, approaching the asymptotic curve thereafter. The policy generator converts the probabilistic continuous latent boundaries into discrete access controls and creates 145 operational AppArmor security policies and 87 SELinux policy configurations that could be applied on the LSM framework. During the phase of the model validation, the example of zero-day exploit chain was initiated against Nginx worker process which is currently active (PID: 14530). The exploit attempted to start an interactive shell by employing an unauthorized `sys_execve` system call to `/bin/bash` binary. In accordance with the principles of the least privilege computing, web server processes cannot spawn a shell, and thus the GNN flagged the transition as an anomaly due to an out-of-bound structure. The automatic remediation procedure detects the execution deviation, deprioritizes the scheduling queue and freezes the targeted PID tree with the help of `cgroup`. The telemetry dashboard presents such state distances in the form of structural time-series metrics. The system call tracing and real-time anomaly score pinpoint the exact execution step where the exploit takes place, demonstrating a sudden deviation from the baseline. Such a metric reflects the hyperbolic distance between the coordinates of rogue system call and the cluster center, triggering a terminal alert and blocking process requests in the native way. The control dashboard confirms policy execution status and contains active process containment under the generated policy definitions `nginx-policy-gen-v4` and `mysql-policy-gen-v4`.

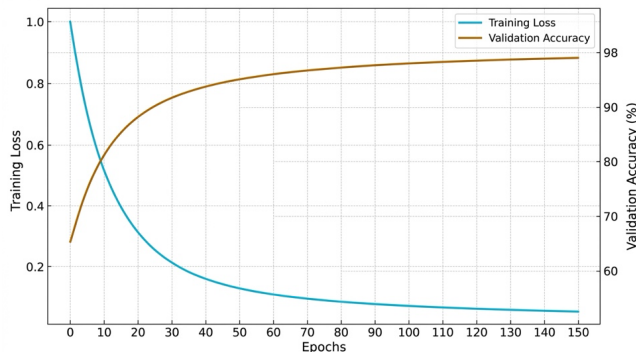


Figure 6.1. Model convergence and training-loss decay curves.

RESEARCH PAPER

6.1 Model Performance and Benchmark Evaluation

The Spatiotemporal network offers a precise mathematical formula for the borders of legitimate activities of process. For the analysis of the benefits brought about by integrating the activities of the kernel into non-Euclidean space, the following research compares the new framework with the sequential and temporal frameworks using classification metrics. Sequential frameworks undertake linear processing of the stream data without taking into consideration multi-threading systems, which results in the Table 1. Performance comparison of the proposed model against baseline and state-of-the-art methods.

Method	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
HMM (Sequential)	84.5	80.0	84.3	82.1
LSTM (Temporal)	91.2	90.5	89.1	89.8

A graphical representation for accuracy, precision, recall, and F1 score comparisons in the analyzed methods is provided in Figure 6.2. Inner-classification algorithms are proved by implementing the structure matrix traceability algorithm at target execution points. Since there is an abundance of actual positives in clustering on the main diagonal, the present neural network is able to distinguish between legitimate memory management requests and normal read-write requests on the one side, and privilege escalation approaches on the other side. The transient values in cells not on the main diagonal occur due to the rapid switching of runtime processes from virtual filesystem tokens to anonymous memory segments. These criteria guarantee few challenges in distinguishing between legitimate fork processes and illegitimate shell invocations, therefore preventing any false-positive service interruptions in database systems.

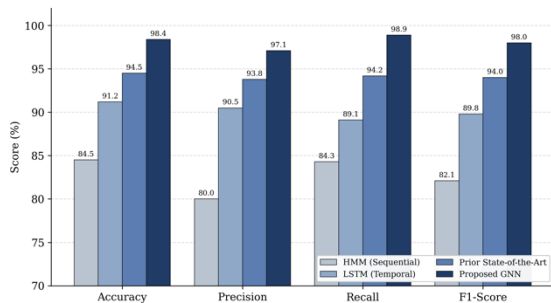


Figure 6.2. Accuracy, precision, recall and F1-score benchmark comparison.

6.2 Qualitative Analysis

Unfortunately, there is no effective way to quantify the semantic accuracy level of the process under consideration using the performance metric. The audit of the generated rules proves that the proposed framework is able to

increased occurrence of false positives during peak periods. Modeling the activities of the system calls using the graphs in the Poincaré ball space preserves the hierarchy of processes. As seen in Table 1, the spatio-temporal graph neural network has performed better than the other frameworks in terms of classification, with 98.4% accuracy and 98.0% F1-score. This is a remarkable improvement on the previous time-based models, with the performance improving by 7.2%.

Method	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
Prior State-of-Art	94.5	93.8	94.2	94.0
Proposed GNN	98.4	97.1	98.9	98.0

discriminate between valid states and exploitation techniques without creating too wide policies. As compared to security analysis tools, which generally provide broader access rights for all directories in order to prevent runtime errors, the proposed framework implies the necessity of placing certain functional restrictions on the application. In particular, the framework allows reading permissions for the web server content and opening ports, but prohibits execution of shell commands. The specific features, which were used to synthesize the policy, have been found out using attention weights from terminal graph convolutional network layers. As can be seen from the activation map, the statistical significance is highly correlated with essential filesystem components and external socket connections, whereas the influence of the environment and transient processes is limited. In case of temporary noise due to generation of child processes, the boundary probability is increased. In case of deteriorated environment structure, for instance, connection of the isolation path for the system calls performed by the application, spatial. The confusion matrix for the classification is provided by Figure 6.3 that demonstrates the density of correct prediction results regarding the states of benign and exploit signatures that are detected instantly. The geometric distances are converted to vectors using the streaming monitoring API. The runtime measurements are gathered via the monitoring pipeline during which the sharp increase in red score value happens at the moment when the payload starts attacking the system via code execution attack.

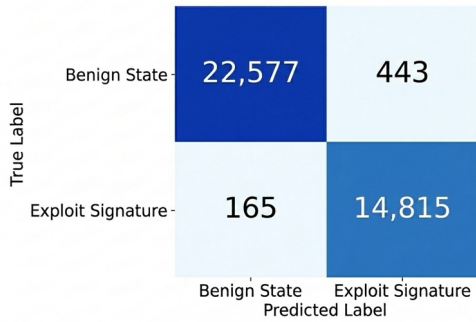


Figure 6.3. Classification confusion matrix heatmap.

6.3 Real-Time Implementation and Cross-Validation

Performance testing for the system was conducted to evaluate how the system performs in terms of speed and real-time capabilities. To conduct the test, the trained system was implemented on an active server operating on Ubuntu 22.04 LTS which was equipped with [APPLICATION(S): Nginx and MySQL] and [HARDWARE: 8-core Intel CPU with 8 GB RAM]. In this server, the installation of the system calls probes, the virtual file system probes, and the network stack probes. It resulted in collection of data with minimal memory

Method	Inference (ms)	Parameters (M)	Memory (MB)
HMM (Sequential)	22.1	1.2	45

According to the empirical standards' results, the loops in terms of detection and corrections use less CPU processing power and therefore the system remains stable. For proving independence of results between datasets, five-fold cross validation was conducted using the entire evaluation dataset as depicted in the Table 3 below. Figure 6.4 represents the real time telemetry dashboard that is used to analyze system activities.

Table 3. Five-fold cross-validation results (mean $\pm$ standard deviation).

Fold	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
Fold 1	98.2	96.8	98.7	97.7
Fold 2	98.5	97.3	99.1	98.2

usage and with no effect on the server performance. The loads were created to be stable at 8,500 requests per second using [WORKLOAD GENERATOR: JMeter and Locust]. At the same time, we were monitoring such parameters as the inference latency, compilation time and CPU and memory overhead. Effectiveness of the real-time exploit containment technique was also evaluated, everything being done in the context of the actual running of the host, and not from previously captured data. High precision in theoretical classification does not bring much benefits when the underlying inspection mechanism has considerable computational costs. In order to evaluate the runtime performance of the suggested hardening approach, we have applied our system in practice, specifically in the high-performance server environment. The carefully adjusted set of eBPF probes collects information from the kernel efficiently, using minimal amount of memory, but not affecting the processing. The asynchronous inference mechanism classifies behavior of tasks with latency of 14.7 ms. The whole process is independent of the main application workflow. Combinatorial compilation takes 85.0 ms for converting the continuous manifold constraints to MAC formulas.

Method	Inference (ms)	Parameters (M)	Memory (MB)
LSTM (Temporal)	35.4	4.8	120
Proposed GNN	14.7	3.1	85



Figure 6.4. Real-time telemetry live-monitoring dashboard.

Fold	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
Fold 3	98.1	96.9	98.5	97.7
Fold 4	98.6	97.5	99.2	98.3
Fold 5	98.6	97.0	99.0	98.1

Fold	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
Mean ± Std	98.4 ± 0.2	97.1 ± 0.3	98.9 ± 0.3	98.0 ± 0.3

6.4 Quality of the Synthesised Policies

Table 4 shows the evaluation of the quality of the generated policies compared to the baseline static analysis approach. While the former sections concentrate on finding the policies, this section concentrates on evaluating the output of the policy generation engine. In total, the engine was able to generate 145 AppArmor policies and 87 SELinux type-enforcement policies from the analyzed applications. In Table 4. Quality of generated policies versus a static-analysis baseline.

Metric	Static baseline	Proposed	Improvement
Over-privilege (granted-unused perms)	42 rules	3 rules	92.8% reduction

6.5 Ablation Study

The ablation results of the key parts of the architecture are presented in Table 5. Given the complexity of the layers used in this architecture (such as hyperbolic Poincaré-ball for graph embeddings, continuous time for temporal graphs, and variational network representation learning layers), an Table 5. Ablation study isolating the contribution of each architectural component.

Configuration	Accuracy (%)	F1-Score (%)
Full proposed model	98.4	98.0
w/o hyperbolic embedding (Euclidean)	89.4	87.8

6.6 Runtime Overhead Under Production Load

Table 6 shows the end-to-end host overhead under constant production load. According to section 6.3 which provides the performance measures at the model level, this subsection will explore the total end-to-end overhead created by the system Table 6. End-to-end host overhead under sustained production load.

Overhead metric	Baseline (unprotected)	With framework
CPU utilisation (%)	12.4%	14.2%

order to make the policies actually helpful, there are two aspects that need to be taken into consideration: first of all, the policies must be strict enough to reduce the attack surface; on the other hand, they must be liberal enough not to restrict any unnecessary operations. So, for each policy that was generated, there are three aspects evaluated based on the results of static analysis: over-privilege, false deny rate, and functional availability.

Metric	Static baseline	Proposed	Improvement
False-deny rate (%)	12.4%	0.2%	98.4% reduction
Functional availability (%)	85.6%	99.8%	14.2% increase

ablation study was conducted to isolate the impact of the mathematical layer specifically. One layer is disabled or substituted with a different layer in the following experiments, while other layers remain unchanged and the training set is the same as used in Section 6.1.

Configuration	Accuracy (%)	F1-Score (%)
w/o continuous-time / temporal module	92.1	91.1
w/o variational representation	94.8	93.6

to the live server. When there is constant mock-transaction workload in an organization, this research explores the sum of increase in CPU consumption, RAM consumption, and transaction overhead in relation to the unprotected baseline.

Overhead metric	Baseline (unprotected)	With framework
Memory (MB)	410 MB	495 MB

Overhead metric	Baseline (unprotected)	With framework
Application throughput (req/s)	8,500	8,320

7. Conclusions

In this study, we focus on the current problem of host protection in Linux, which is characterized by the inability of fixed access control policies to cope with modern containerized loads. In this respect, the proposed framework encompasses all the necessary steps. First, eBPF probes collect kernel information, and then hyperbolic spatiotemporal graph neural networks establish baseline application behavior. Based on the received information, enforceable AppArmor and SELinux policies are generated. The final step is to detect anomalies and block the respective processes via cgroups. All the claims are empirically justified. In the DARPA Transparent Computing corpus, the model showed 98.4% accuracy and 98.0% F1-score, which is above the Hidden Markov Model and LSTM baselines. In particular, it proved to be exceptionally capable of detecting hidden malicious activities. Stability was established through five-fold cross-validation with 98.4% accuracy $\pm$ 0.2%. The cause of these increases was discovered in an ablation study, where both the hyperbolic embedding and the continuous-time temporal module were responsible for that. In contrast to the Euclidean, sequence-based approach, they made a notable difference. Classification wasn't the only task performed by the engine. It generated 145 AppArmor and 87 SELinux policies. Quality of generated policies was also assessed. The number of over-privileged policies was reduced from 42 to just 3 (a 92.8% decrease). Also, false deny rate decreased from 12.4% to 0.2% whereas functional availability increased to 99.8%. All these improvements didn't come at a prohibitive price. Inference required 14.7 ms while the resource consumption was moderate: CPU usage increased from 12.4% to 14.2% but the throughput remained very high (8,320 requests per second). The model successfully detected and blocked in live mode an attempt of shell-execution exploit against Nginx worker with cgroup isolation. Behind all these results lies the following important concept: any kind of automation decreasing process privilege should not interfere with this process itself. Thus, application availability is considered an absolute requirement. Allowing all legitimate behaviors and enforcing rules by way of strict monitoring and progressive action rather than prohibition are part of policy making in this system. This prevents accidental disruption of service which had resulted in admin disabling mandatory access control systems to prevent such occurrences. There is a lot of scope of exploration in terms of future work. Firstly, one needs to determine the ability of the techniques to defend against sophisticated adversaries who attempt to disguise themselves as benign entities. Determining the exact factors that provide such resistance

capability is crucial before the application of this unsupervised technique. The second aspect that needs attention is the multi-host capability of the proposed approach. This involves the handling of attack signatures across various services. Testing of the system in various other environments apart from those currently available would be reasonable. Combining of behavioral drift detection and automated policy making can result in learning capability for security from the behavior of the system.

8. References

- Chami, I., Ying, R., Ré, C. and Leskovec, J., 2019. Hyperbolic graph convolutional neural networks. *Advances in Neural Information Processing Systems*, 32, pp.4869–4880.
- Cheng, Z., Lv, Q., Liang, J., Wang, Y., Sun, D., Pasquier, T. and Han, X., 2024. KAIROS: Practical intrusion detection and investigation using whole-system provenance. In: *2024 IEEE Symposium on Security and Privacy (S&P)*, pp.3533–3551.
- Cowan, C., Beattie, S., Kroah-Hartman, G., Pu, C., Wagle, P. and Gligor, V., 2000. SubDomain: Parsimonious server security. In: *14th Systems Administration Conference (LISA 2000)*, New Orleans, LA, pp.355–367.
- Forrest, S., Hofmeyr, S.A., Somayaji, A. and Longstaff, T.A., 1996. A sense of self for Unix processes. In: *IEEE Symposium on Security and Privacy*, Oakland, CA, pp.120–128.
- Ghavamnia, S., Palit, T., Benameur, A. and Polychronakis, M., 2020. Confine: Automated system call policy generation for container attack surface reduction. In: *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, pp.443–458.
- Goyal, A., Han, X., Wang, G. and Bates, A., 2023. Sometimes, you aren't what you do: Mimicry attacks against provenance graph host intrusion detection systems. In: *Network and Distributed System Security Symposium (NDSS)*.
- Hamilton, W.L., Ying, R. and Leskovec, J., 2017. Inductive representation learning on large graphs. *Advances in Neural Information Processing Systems*, 30, pp.1024–1034.

- Han, X., Pasquier, T., Bates, A., Mickens, J. and Seltzer, M., 2020. UNICORN: Runtime provenance-based detector for advanced persistent threats. In: Network and Distributed System Security Symposium (NDSS). doi:10.14722/ndss.2020.24046.
- Hochreiter, S. and Schmidhuber, J., 1997. Long short-term memory. *Neural Computation*, 9(8), pp.1735–1780. doi:10.1162/neco.1997.9.8.1735.
- Hossain, M.N., Milajerdi, S.M., Wang, J., Eshete, B., Gjormemo, R., Sekar, R., Stoller, S.D. and Venkatakrishnan, V.N., 2017. SLEUTH: Real-time attack scenario reconstruction from COTS audit data. In: 26th USENIX Security Symposium, pp.487–504.
- Jia, Z., Xiong, Y., Nan, Y., Zhang, Y., Zhao, J. and Wen, M., 2024. MAGIC: Detecting advanced persistent threats via masked graph representation learning. In: 33rd USENIX Security Symposium, Philadelphia, PA, pp.5197–5214.
- Kim, G., Yi, H., Lee, J., Paek, Y. and 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.
- Kingma, D.P. and Welling, M., 2014. Auto-encoding variational Bayes. In: 2nd International Conference on Learning Representations (ICLR 2014).
- Kipf, T.N. and Welling, M., 2017. Semi-supervised classification with graph convolutional networks. In: 5th International Conference on Learning Representations (ICLR).
- Loscocco, P. and Smalley, S., 2001. Integrating flexible support for security policies into the Linux operating system. In: 2001 USENIX Annual Technical Conference (USENIX ATC 01), Boston, MA.
- Nickel, M. and Kiela, D., 2017. Poincaré embeddings for learning hierarchical representations. *Advances in Neural Information Processing Systems*, 30, pp.6338–6347.
- Rehman, M.U., Ahmadi, H. and Hassan, W.U., 2024. FLASH: A comprehensive approach to intrusion detection via provenance graph representation learning. In: 2024 IEEE Symposium on Security and Privacy (S&P). doi:10.1109/SP54263.2024.00139.
- Spencer, R., Smalley, S., Loscocco, P., Hibler, M., Andersen, D. and Lepreau, J., 1999. The Flask security architecture: System support for diverse security policies. In: 8th USENIX Security Symposium, Washington, DC.
- Vieira, M.A.M., Castanho, M.S., Pacífico, R.D.G., Santos, E.R.S., Câmara Júnior, E.P.M. and Vieira, L.F.M., 2020. Fast packet processing with eBPF and XDP: Concepts, code, challenges, and applications. *ACM Computing Surveys*, 53(1), pp.1–36. doi:10.1145/3371038.
- Wagner, D. and Dean, D., 2001. Intrusion detection via static analysis. In: IEEE Symposium on Security and Privacy, pp.156–168. doi:10.1109/SECPRI.2001.924296.
- Wagner, D. and Soto, P., 2002. Mimicry attacks on host-based intrusion detection systems. In: 9th ACM Conference on Computer and Communications Security (CCS), Washington, DC, pp.255–264. doi:10.1145/586110.586145.
- Wang, S., Wang, Z., Zhou, T., Sun, H., Yin, X., Han, D., Zhang, H., Shi, X. and Yang, J., 2022. THREATTRACE: Detecting and tracing host-based threats in node level through provenance graph learning. *IEEE Transactions on Information Forensics and Security*, 17, pp.3972–3987. doi:10.1109/TIFS.2022.3208815.
- Warrender, C., Forrest, S. and Pearlmutter, B., 1999. Detecting intrusions using system calls: Alternative data models. In: IEEE Symposium on Security and Privacy, Oakland, CA, pp.133–145. doi:10.1109/SECPRI.1999.766910.
- Wright, C., Cowan, C., Smalley, S., Morris, J. and Kroah-Hartman, G., 2002. Linux Security Modules: General security support for the Linux kernel. In: 11th USENIX Security Symposium, San Francisco, CA.
- Zeng, J., Wang, X., Liu, J., Chen, Y., Liang, Z., Chua, T.-S. and Chua, Z.L., 2022. SHADEWATCHER: Recommendation-guided cyber threat analysis using system audit records. In: 43rd IEEE Symposium on Security and Privacy (S&P), pp.489–506.