Griffin: Real-time network intrusion detection system via ensemble of autoencoder in SDN

Liyan Yang, Yubo Song, Shang Gao,
Aiqun Hu, Bin Xiao, *Senior Member*

***Abstract*—Many efforts have been devoted to the development of efficient Network Intrusion Detection System (NIDS) using machine learning approaches in Software-defined Network (SDN). Unfortunately, existing solutions failed to detect real-time and zero-day attacks due to their limited throughput and prior knowledge-based detection. To this end, we propose Griffin, a NIDS that uses unsupervised machine learning expertise to detect both known and zero-day intrusion attacks in real-time with high accuracy. Specifically, Griffin uses an efficient feature extraction framework to capture the sequential features of the traffic packets. Then, it utilizes cluster analysis to reduce the feature scale to achieve low throughput. Moreover, an ensemble autoencoder is built automatically to further extract features with low complexity and high precision to train the model. We evaluate the accuracy, robustness, and complexity of the system using open datasets. The result shows that Griffin's complexity is about 40% lower, and its accuracy is at most 19% higher than existing NIDS.Additionally, even in the situation with evasion, the Griffin has at most 9% decrease of AUC, which is a good performance compared with other solutions. Furthermore, this paper also utilizes the differential privacy framework during training autoencoders to protect datasets' privacy which is inherent in machine learning approaches.**

***Index Terms*—Software-defined Network (SDN), Network intrusion detection system (NIDS), Autoencoder, Ensemble learning, Differential privacy**

I. Introduction

OVER the past several years, we have witnessed an unprecedented growth of Software-defined Network (*SDN*) applications alongside a vast deployment of SDN-enabled devices [1], [2]. SDN technology is an approach to network management that has characteristics of decoupled network control (i.e., Users can flexibly change the network as needed in SDN.) and network programmability (i.e., SDN enables users to automatically program the network to support applications [3]) to improve network performance and monitoring. Due to these two promising features, SDN is extremely popular in a variety of scenarios ranging from smartphone [4] to the platform for virtual remote office [5], and has dramatically changed the landscape of networking.

Unfortunately, SDN is still threatened by traditional network intrusion attacks in all components in its structure: the data plane, control plane, and application plane. For instance, the former two planes could be compromised by the Distributed Denial-of-service (*DDoS*) attack when a large number of network traffic packets are first received by the switches in the data plane, which then transmit received packets to the controller (in the control plane) where all high bandwidth could use up [6]. Besides, the application plane of SDN is vulnerable to recon attack (e.g., Mirai), in which malware can log and infect the device through scanning the internet for devices when the default username-and-password combo is not changed. On October 21, 2016, a number of essential Internet sites in North America were rendered inaccessible because of a DDoS attack caused by the Mirai botnet, which led to a substantial economic loss.

To mitigate intrusion attacks, a number of network intrusion detection systems (*NIDS*) [7]–[10] have been proposed. Most of these NIDSs leverage machine learning expertise to improve their detection accuracy. In addition, comparing to traditional rule based detection methods [11]–[14], machine learning based methods can also effectively identify zero-day intrusion attacks [15]–[18]. At a high level, these machine learning-based methods require the process of feature extraction, training, and detection based on numerous packets, which need much time and memory. However, identifying some network intrusion attacks (e.g., DDoS attack) in high-performance networks in a short period (nearly the moment the attack appears) is urgent. Unfortunately, the aforementioned defense strategies failed to satisfy the requirements of real-time manner due to the low detection throughput on the process of machine learning algorithms.

Besides, existing solutions, including the aforementioned machine learning-based detection and other rule detection, focus on either the flow-level or packet-level. However, the existing packet-level solutions [19], [20] can easily be evaded by injecting noise from attackers. Therefore, those solutions always fail to detect malicious traffic when attackers conduct such an evasion method. Meanwhile, those packet-level solutions also failed to detect zero-day attacks accurately for the weak robustness. Moreover, the state-of-the-art methods based on traditional rule methods [21]–[24], which analyzed numerous statistical traffic features, always introduce significant detection latency. Therefore, a robust machine learning-based NIDS, which can detect zero-day attacks in a real-time manner, is still missing.

L. Yang, Y. Song, and A. Hu are with the School of Cyber Science and Engineering, and the Frontiers Science Center for Mobile Information Communication and Security, Southeast University, and Purple Mountain Laboratories, Nanjing 211189, China. E-mail: yangliyanyan@gmail.com, songyubo@seu.edu.cn, aqhu@seu.edu.cn.

B. Xiao and S. Gao are with the Department of Computing, The Hong Kong Polytechnic University, Kowloon, Hong Kong. E-mail: csbxiao@comp.polyu.edu.hk, shanggao@comp.polyu.edu.hk.

To advance the state-of-the-art, we present Griffin [1]: a real-time robust network intrusion detection system that can detect zero-day attacks with high accuracy and low latency. Griffin effectively extracts and analyzes the sequential feature of traffic based on traditional flow-level statistic features and appropriate extraction intervals. Specifically, we use the damped function to extract statistical traffic features to precisely illustrate sequential information of traffic packets. Therefore, it is difficult to disturb the detection system by injecting noise packets. Furthermore, we reduce features' scale and feature complexity to achieve high throughput and real-time detection. To achieve this, we first map the similar features into a sub-instance by cluster analysis. Secondly, we further extract the features from these sub-instances by using ensemble autoencoders to learn the information contained in sub-instances. Due to the low feature dimensional and precise feature presentation, Griffin realizes the high accuracy and real-time detection manner.

We evaluate Griffin with the open Datasets (e.g., Mirai, Active Wiretap) to demonstrate its detection effectiveness. The results show that Griffin's complexity is about 40% lower, and its accuracy is at most 19% higher than existing NIDSs. Additionally, even in the situation with evasion, the Griffin has at most 9% decrease of AUC, which is a good performance compared with other solutions. Moreover, this paper also utilizes the Differential Privacy framework during training autoencoders to protect datasets' privacy which is inherent in machine learning approaches. To this end, we add Gaussian noise into parameters in the machine learning training process. After that, attackers cannot deduce the sensitive information from users' datasets. Finally, with the privacy protection mechanisms based on Gaussian Mechanism, the system detection accuracy is up to 97%, which is only a lower 1 percentage than before.

In short, this paper makes the following contributions:

- We propose Griffin: a novel network intrusion detection system in SDN, which can detect zero-day attacks in a real-time manner with robustness, high accuracy, and low latency.
- We develop an efficient feature extraction framework to capture the sequential features of the traffic packets, which paves the way for robustness.
- We develop the cluster analysis and ensemble of autoencoder to capture the precise feature having low dimension, which lays the foundation for the low latency, real-time detection, and high accuracy.
- We perform the Differential Privacy framework during training autoencoders to protect datasets' privacy which is inherent in machine learning approaches. In our model, we achieve the goal of gaining both the best privacy efficiency and excellent accuracy simultaneously, which is a challenging issue in differential privacy framework establishment.
- We compare the Griffin with some traditional algorithms and state-of-art solutions. Our results show that Griffin achieves at most 19% improvement of AUC, while achieving at most 40% improvement of complexity. Even in the situation with evasion, the Griffin has just at most 9% decrease of AUC, which is a good performance compared with other solutions.

[1]This paper is an extended version of "Griffin: An Ensemble of AutoEncoders for Anomaly Traffic Detection in SDN" published in 2020 IEEE Global Communications Conference.

The remainder of the paper is organized as follows: We review related work in Section II and introduce the threat model and design goals in Section III. Section IV presents Griffin's framework, its entire machine learning pipeline. Section V illustrates the design details of the Griffin. Section VI describes the evaluation and experimental results in terms of detection accuracy, latency, and the privacy-preserving mechanism effectiveness. Finally, in Section VII we present our conclusion.

II. Related Work

a) Network intrusion detection system in SDN: Network intrusion detection system in the SDN environment is in the research and exploration stage. In the past several years, there have been many efforts on this work. For example, Granted, Tang et al. used a five-layer neural network to train the model, but due to poor feature selection, the accuracy rate was only 75% [25]. Z. Liu et al. [26] employed SO-BP neural network to classify the traffic, though high accuracy of 95.65%, their algorithm was limited to detecting DDoS attacks. Ayyoob Hamza, Hassan Habibi Gharakheili et al. attempted to translate MUD policies into flow rules that could be enforced using SDN and related the exception behavior to attacks. However, they ignored the latency [27]. Similarly, Atiku Abubakar and Bernardi Pranggono also ignored the latency of the system, though their accuracy was high [28]. Besides, the methods mentioned above lacked a privacy protection mechanism for the training datasets and failed to detect zero-day cyber-attacks in real time.

b) Machine learining-based NIDS: Meanwhile, as an emerging prosperity paradigm, machine learning-based network intrusion detection has been developed. Table. I concludes and compares various machine learning-based intrusion into attack detection methods. As shown in Table. I, machine learning-based methods can detect zero-day attacks with high accuracy. Unfortunately, those methods have low throughput because of their complex machine learning algorithms, leading to high latency and failed to realize real-time detection [23], [29]. Moreover, those machine learning-based methods are easily affected by injected noise traffic packets. Therefore they are not satisfied with the requirement of robustness.

c) Data privacy preserving mechanism in machine learning: Additionally, there also exist users' data privacy leakage problems in machine learning-based methods. For these problems, predecessors have also done much research. Martín Abadi, Andy Chu, et al. proposed a new technique to analyze privacy costs using the differential privacy framework to protect training data privacy. This research protects data privacy in a deep neural network and has a modest cost in training efficiency, software complexity, and model quality [34]. Pathum Chamikara Mahawaga Arachchige, Peter Berto, et al. proposed the LATENT: an algorithm for local differential private. It

TABLE I
COMPARING THE EXISTING MACHINE LEARNING-BASED NETWORK INTRUSION DETECTION METHODS

Category of NIDS	Feature Analysis	Zero-Day Attacks Detection	High Accuracy	Robustness	Real-time Detection
Machine Learning-based NIDS	Signature based analysis [19], [30]	✔	✔	✗	✔
	Payload statistics analysis [20]	✔	✔	✗	✗
	Flow-level statistics analysis [24], [31], [32], [33]	✔	✗	✗	✗
	Application usage statistics analysis [21]–[23]	✔	✔	✗	✗
	Griffin, ensemble autoencoders feature extraction and analysis	✔	✔	✔	✔

protects the privacy of data in the randomization layer before achieving an untrusted service [35]. Also, NhatHai Phan, Xintao Wu, et al. focus on developing a new mechanism to use differential privacy in deep neural networks [36]. According to those researchers, it is practical to utilize differential privacy mechanisms in deep neural networks to resolve its inherent data privacy problem. Therefore their method provides new insight into our scenarios.

Above all, we are supposed to create a robust real-time NIDS in SDN to detect zero-day attacks with high accuracy, low latency, and privacy-preserving mechanisms.

III. THREAT MODEL AND DESIGN GOALS

A. Threat model

The network work intrusion attackers can utilize the configuration in all architecture layers: application layer, control layer, and data layer. They use network intrusion attacks to steal network resources, jeopardize network security, and destroy data privacy. We will illustrate those attacks based on the attack surface following.

a) Application plane: In the application plane, attackers use business productivity software and scripting languages to conduct intrusion attacks. The attackers can easily bypass the users' awareness because those tools are not malware. Actually, in most cases, some business justification enables the attackers to blend in with these tools.

b) Control plane: In the control plane, attackers can use vulnerabilities of protocols (e.g., ARP, TCP, and UDP) to conduct intrusion attacks. For example, they can imitate protocols or send protocol messages to conduct man-in-the-middle attacks and steal the data without the server's permission. Additionally, attackers can even crash the server to interrupt the service.

c) Data plane: In the data plane, when the network supports asymmetric routing, attackers will use multiple routes to achieve the targets. In this case, attackers can evade detection by bypassing the specific network segments and NIDS. On the other hand, the attackers can also create a large number of traffic loads to induce congestion in the network. After that, they can conduct attacks without being detected.

B. Design goals

This paper aims to develop a NIDS named Griffin, which consist of four parts: packet capture, feature extractor, feature mapper and anomaly detector. This system is suspected to be a robust real-time NIDS that can detect zero-day attacks with high accuracy, low latency, and privacy-preserving mechanisms. Remarkably, the Griffin should complete the following three goals, which are not well tackled in the literature.

1) Real-time detection and low latency: In some cyber-attacks (e.g., DDoS attacks), the failure of a web server is instantaneous. Therefore, it is essential to design a real-time network intrusion detection system with low latency. In this paper, we achieve this goal using the technology of training model according to each packet with Hierarchical Clustering Algorithm in the feature mapper. More specifically, the Hierarchical Clustering Algorithm in this part reduces the dimension of the feature sets. Therefore, this method reduces this system latency in an exemplary manner.

2) High accuracy and zero-day detection: To achieve the goal of high accuracy and zero-day detection, we use the efficient feature extractor framework to extract precise features. After that, we use the ensemble of autoencoder to learn the feature of traffic packets in an unsupervised manner. The ensemble of autoencoder captures the precise features of traffic packets from un-labeled datasets, which makes Griffin realize to detect the attacks not present before.

3) Privacy preserving: The training data sets contain user privacy, and some model parameters can be used to reveal user privacy. Therefore, the model is supposed to satisfy user data security. To achieve this goal, we improve the stochastic gradient descent (*SGD*) using Gaussian Mechanism in anomaly detector to protect the intrusion data set in the system from privacy leakage. Specifically, the Gaussian Mechanism adds noise to the parameters in the model, which makes it difficult for attackers to deduce the privacy information in user traffic packet data from the parameters of the model.

IV. OVERVIEW OF GRIFFIN

In this section, we present our network intrusion detection system Griffin for SDN. Griffin achieves high-performance detection by extracting precise flow-level statistic feature sequences. Specifically, it reduces the feature scale and feature complexity via cluster analysis and ensemble of autoencoder to achieve the real-time detection. Since the ensemble of the autoencoder is automatically established in an unsupervised manner, Griffin realizes the zero-day attacks detection. Fig. 1 shows the overview process of Griffin.

a) Packet capturer module.: This module provides the traffic packet feature sequences to the feature extractor module for extracting the flow-level statistic features. This module

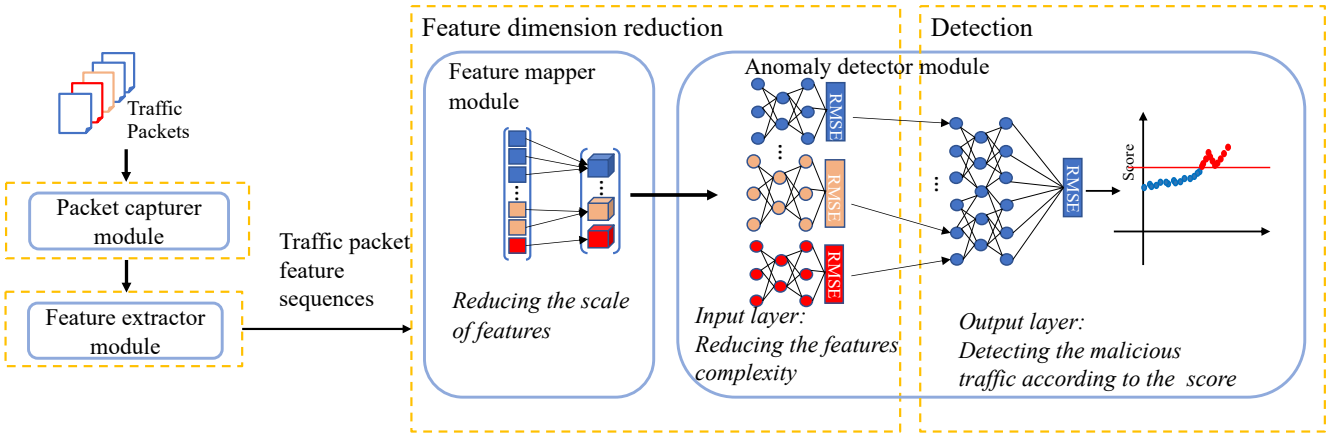


Fig. 1. The process workflow of Griffin

does not capture particular attack-related features and only offers traffic packet feature sequences to facilitate the following processes. More specifically, this module mainly uses the switches and OpenFlow protocol in SDN to capture the sequence of traffic packet features (e.g., source MAC address,source IP address, destination MAC address, IP protocol version, etc.).

b) Feature extractor module.: This module is in the data plane in the SDN, it extracts the flow-level statistic features from the packet feature sequences. More specifically, this module periodically extracts the required statistic feature from the packet capture module with some fixed time intervals. These features with statistic Information are provided for the feature mapper module. However, it is difficult to extract the flow-level statistic features of traffic in high throughput networks in real-time because of the numerous sophisticated and irregular flow patterns [37], [38]. We will show the details of this module in Section V-A.

c) Feature mapper module.: To achieve the goal of real-time detection and high throughput. In the data plane of SDN, we map the packet statistic features from the feature extractor module to reduce the scale of features. To this end, this module maps the features that illustrate traffic packets' particular property into a sub-instance. These sub-instances are provided for the anomaly detector module to facilitate network intrusion detection. We will illustrate the details of the module in Section V-B.

d) Anomaly detector module.: In this module, we utilize the features in sub-instances from feature mapper module to further reduce features complexity and detect the malicious. This module is in the control plane of SDN for the good computational capacity of the controller. In the training mode, the input layer generate s set of Root Mean Square Error (*RMSE*) representing those sub-instances' properties. Additionally, the output layer add a particular noise to those RMSEs to protect the data privacy. In the monitoring mode, we detect the malicious traffic packets according to RMSE (the Score) from the output layer. Griffin can detect various of network intrusion attacks in a good performance. We will elaborate on the anomaly detector in Section V-C.

V. Design Details

In this section, we illustrate the design details of Griffin, i.e., the design of three main parts in Griffin.

A. Feature Extractor

The feature extractor aims to achieve the goal of high accuracy with precise features. More specifically, since flow-level-based detection has robustness, we use the flow-level feature extraction window with a damping parameter to create a set of feature vector $\overrightarrow{x} \in \mathrm{R}^{n}$ to describe packet and its channel. For the complex network environment and functions, finding a feature that could describe the packet's connection and the path of traffic is essential. For example, consider a signal DNS packet. The packet may be a request from a server, or it may be with malicious payload or leakage data presenting a sustained significant jitter. As another example, consider the SMTP packet. The packet may be a protocol packet from a mail server, or it may be one of the millions of packets sent in an attempt to cause a mail bomb attack. Therefore, it is essential to select and extract traffic features precisely.

To accurately detect the various attacks, we first select the six validated factors [39] to compose the initial characteristics of packets: packet's source MAC (*Src-MAC*) and IP address (*IP-Addr*), packet's source IP (*Src-IP*), packet's channel between source and destination (*Channel-Src-Des*), packet's source TCP/UDP Socket (*TCP/UDP-Soc*). Secondly, since in many light devices, there exist memory restriction, we use an frame with attenuation function to maintain increasing statistics in duration windows (100ms, 500ms, 1.5s, 10s, and 1min) for practicality [33]. To implement this frame, firstly, we define a attenuation function as [39], [40]:

$$d_{\lambda}(t) = 2^{-\lambda t} \tag{1}$$

where $\lambda > 0$ is the attenuation coefficient and t denotes the duration time since observing S_i last time. In boundless data flow (e.g., time): $S = \{x_1, x_2, x_3, \ldots\}$, where $x_i \in R$. When damping weight is 0, the former temporal incremental statistic will be deleted. Secondly, we define a damping increment statistical tuple as: $TS_{i,\lambda} := (w, FS, SS, SK_{ij}, T_{last})$, where

FS is the sum of features, SS is the sum of squares of features, w is the current weight, and T_{last} denotes the time stamp when update the $TS_{i,\lambda}$ last time. Thirdly, we define SK_{ij} as the sum of the remaining of flow i and flow j, which can be used to calculate two-dimensional statistical analysis. Therefore, the attenuation coefficient can be defined as:

$$\gamma \leftarrow d_\lambda \left(t_{cur} - t_{last}\right) \tag{2}$$

Additionally, we update the attenuation process as follows:

$$TS_{i,\lambda} \leftarrow (\gamma w, \gamma FS, \gamma SS, \gamma SK, T_{cur}) \tag{3}$$

$$TS_{i,\lambda} \leftarrow \left(w+1, FS + x_{cur}, SS + x_i^2, SK_{ij} + r_i r_j, T_{cur}\right) \tag{4}$$

According to those traffic packet statistic features, we could select 23 features displayed in Table. II, with those seven statistical analysis indicators: W_i, $\mu_i = FS/w$, $\sigma_i = \sqrt{\left|\frac{SS}{w} - (\frac{FS}{w})^2\right|}$, $\|S_i, S_j\| = \sqrt{\mu_{s_i}^2 + \mu_{s_j}^2}$, $R_{S_i,S_j} = \sqrt{\left(\sigma_{S_i}^2\right)^2 + \left(\sigma_{S_j}^2\right)^2}$, $Cov_{S_i,S_j} = \frac{SK_{ij}}{w_i + w_j}$ and $P_{S_i,S_j} = \frac{\text{Cov}_{S_i,S_j}}{\sigma_{S_i}\sigma_{S_j}}$. More specifically, the μ_i and σ_i of Src-MAC-IP-Addr, Src-IP, Channel-Src-Des, and TCP/UDP-Soc describe the bandwidth of the outbound traffic. The $\|S_i, S_j\|$, R_{S_i,S_j}, Cov_{S_i,S_j}, and P_{S_i,S_j} of Channel-Src-Des and TCP/UDP-Soc depict the bandwidth of the outbound and inbound traffic together. The W_i of Src-MAC-IP-Addr, Src-IP, Channel-Src-Des, snd TCP/UDP-Soc illustrate the packet rate of the outbound traffic. Additionally, the W_i,μ_i, and σ_i of Channel-Src-Des describe inter-packet delays of the outbound traffic. Finally, we extract final 115 features by using 5 intervals (100ms, 500ms, 1.5s, 10s, and 1min) extract characteristics showing in Table. II.

TABLE II
FEATURES SELECTED

The packet's...	Statistics	Characteristic	Features
Size	μ_i, σ_i	Src-MAC-IP-Addr Src-IP Channel-Src-Des TCP/UDP-Soc	8
Magnitude	$\|S_i, S_j\|$ R_{S_i,S_j} Cov_{S_i,S_j} P_{S_i,S_j}	Channel-Src-Des TCP/UDP-Soc	8
Count	W_i	Src-MAC-IP-Addr Src-IP Channel-Src-Des TCP/UDP-Soc	4
Jitter	W_i, μ_i, σ_i	Channel-Src-Des	3

B. Feature Mapper

The feature mapper aims to reduce the dimension of the feature set to achieve a low latency and real-time detection.Because our dataset is unbunded, we used the incremental Hierarchical Clustering Algorithm to merge $\overrightarrow{x}$'s n features into k smaller sub-instances [41]. More specifically, the correlation distance d_{cor} that is used in this case defined as:

$$d_{\text{cor}}(u,v) = 1 - \frac{(u-\bar{u})\cdot(v-\bar{v})}{\|(u-\bar{u})\|_2 \|(v-\bar{v})\|_2} \tag{5}$$

where $\bar{u}$ is the mean of elements in vector u, $\bar{v}$ is the mean of elements in vector v, $(u-\bar{u})\cdot(v-\bar{v})$ is dot product.

Next, we will calculate the correlation between features in hierarchical clustering. To this end, firstly, we compute the value of sub-instance i in time index t: $c^{(i)} = \sum_{t=0}^{n_t} x_t^{(i)}$, where $\vec{C}$ is n dimensional vector including the sum of every feature and n_t is the number of features in a sub-instance. Secondly, we compute every sub-instance's residual error and quadratic sum residual error: $c_r^{(i)} = \sum_{t=0}^{n_t}\left(x_t^{(i)} - \frac{c^{(i)}}{n_t}\right)$,$c_{rs}^{(i)} = \sum_{t=0}^{n_t}\left(x_t^{(i)} - \frac{c^{(i)}}{n_t}\right)^2$. Therefore, the product between sub-instance's residual errors can be calculated as follow: $[C_{i,j}] = \sum_{t=0}^{n_t}\left(\left(x_t^{(i)} - \frac{(i)}{n_t}\right)\left(x_t^{(j)} - \frac{c^{(j)}}{n_t}\right)\right)$, where C is the partial correlation matrix. After that, we calculate the correlation distance matrix:

$$D = [D_{i,j}] = 1 - \frac{C_{i,j}}{\sqrt{c_{rs}^{(i)}}\sqrt{c_{rs}^{(j)}}} \tag{6}$$

To map the similar features into the same sub-instance, firstly, we initialize the cluster singleton (G_i) for each feature x_i. Secondly, we choose G_i and G_j to make $D(G_i, G_j)$ is minimized among all pairs. After that, G_i and G_j will be merged into a group (i.e., $G_{i,j} = G_i \cup G_j$). And then, we add the $G_{i,j}$ to the list and remove the G_i and G_j from list. We repute this process until there are two groups in the list [42].

Executing this algorithm, we get a dendrogram chart shown in Fig. 2. From the dendrogram, we can extract many different flat partitions, which are presented by black parts. This is a

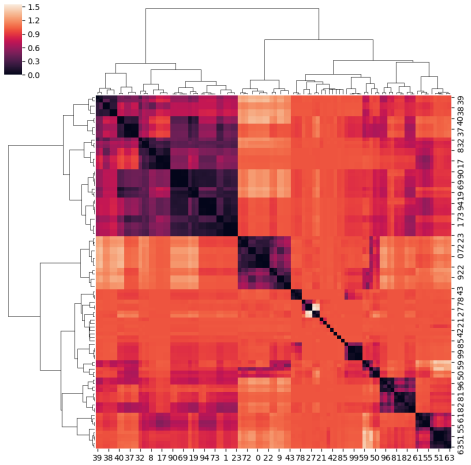


Fig. 2. An example dendrogram of the 100 features clustered together, from 1000000 network packets.

symmetric thermodynamic chart about the oblique symmetry axis, which illustrates the result of mapping n features to k sub-instances. The dark parts in the chart are symmetrical about the diagonal of the square graph, which indicates various cluster methods. Meanwhile, the dark parts illustrate that the distinguished degree of sub-instances is high, beneficial to later

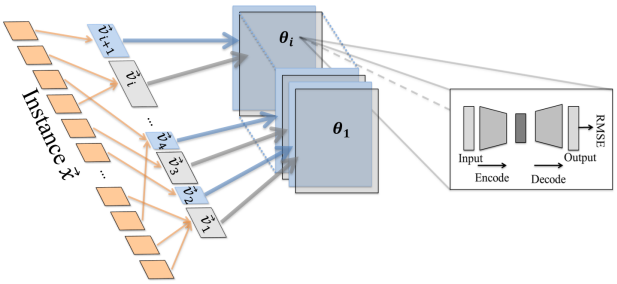


Fig. 3. An illustration of Griffin's anomaly detection algorithm.

detection research on traffic packets. As for the number of sub-instances used in the anomaly detector module, they directly determine the number of autoencoders in the input layer of the anomaly detector module. Therefore, we determine the number of sub-instances by trying every possible value of autoencoders in the input layer of the anomaly detector module. Finally, we get 19 sub-instances to input into the autoencoders in the next module.

C. Anomaly detector

The anomaly detector is the core part of Griffin, which takes the responsibility to achieve the privacy preserving and zero-day attacks detection using the unsupervised module of ensemble of autoencoder. At the high level, the anomaly detector part consists of training and monitoring modes. More specifically, the training mode uses an unsupervised learning model composed of two layers: ensemble layer and output layer. The input of the ensemble layer is the result of the feature mapper, based on which the ensemble layer generates RMSE as the input of the output layer. Meanwhile, the output layer is an autoencoder that considers data privacy and finally generates the anomaly score for detection. Next, we will show the detailed implementation of the anomaly detector.

1) Training mode: As aforementioned, the training mode includes two layer: ensemble layer and output layer. Specifically, let θ denotes an autocoder, let $L^{(1)}$ and $L^{(2)}$ be the ensemble layer and output layer respectively. Therefore, $L^{(1)}$ is the ordered set:

$$L^{(1)} = \{\theta_1, \theta_2, \ldots, \theta_{k'}\} \tag{7}$$

In the ensemble layer, when receiving the first set of mapped instance V from the feature mapper, it uses V as a blueprint initialization architecture for autocoders. The autocoders $\theta_i \in L^{(1)}$ in ensemble layer have three-layer neural network, namely input layer and output layer ($\dim(\overrightarrow{v_i})$), also the hidden layer ($[\beta \cdot \dim(\overrightarrow{v_i})]$, $\beta \in (0, 1]$). The Fig. 3 displays the mapping between $\overrightarrow{v_i} \in V$ and $\overrightarrow{\theta_i} \in L^{(1)}$.

We define $L^{(2)}$ as autocoder θ_0, which has k' input and output neurons and $\lceil k' \cdot \beta \rceil$ internal neutrons. It's input come from 0-1 normalized RMSE error signal of every autoencoder in $L^{(1)}$, which can reduce the complexity of network. Meanwhile, we utilize the evenly distributed random value: $\mathcal{U}\left(\frac{-1}{\dim(\vec{v}_i)}, \frac{1}{\dim(\vec{v}_i)}\right)$ to initialize the weight of autocoder θ_i. And then we use Gaussian Mechanism Stochastic Gradient Descent (SGD_{Gau}) to updated weight iteratively. SGD_{Gau} is a algorithm develop from SGD for privacy preserving of users' traffic packets.

Algorithm 1 Gaussian Mechanism SGD (SGD_{Gau})

Input: Examples: $\{x_1, \ldots, x_N\}$, loss function: $\mathcal{L}(\theta) = \frac{1}{N}\sum_i \mathcal{L}(\theta, x_i)$. Parameters: learning rate η_t, noise scale σ.
Initialize: θ_0 randomly.
Output: θ_T

for $t \in [\mathrm{T}]$ **do**
  Take a random sample L_t with sampling probability $\frac{L}{N}$
  Compute gradient:
  for $\mathrm{i} \in L_t$ **do**
    Compute $g_t(x_i) \leftarrow \nabla_{\theta_t}\mathcal{L}(\theta_t, x_i)$
  Add noise:
  $\tilde{g}_t \leftarrow \frac{1}{L}\sum_i \left(\bar{g}_t(x_i) + N\left(0, \sigma^2\right)\right)$
  Decent:
  $\theta_{t+1} \leftarrow \theta_t - \eta_t \tilde{g}_t$
return θ_T

In the process of training mode, the training data may be stolen by attackers to deduce sensitive Information of users, causing privacy leakage. For this problem, some people try to add noise to the parameters after training. In this way, the whole training process is regarded as a black box, and finally, the data with noise is obtained, which is not vulnerable to linkage attacks. However, this method of adding noise is generally based on a conservative noise, which is often the analysis choice in the worst case, so it is easy to destroy the effectiveness of the learning model. Therefore, we add Gaussian noise to the SGD in the output layer's autoencoder of the training mode. In this way, it is possible to prevent privacy leakage while not destroying the model's utility in this system. The following will describe this method in detail.

Algorithm 1 summarizes the model's training process based on SGD_{Gau} optimizer [34]. The core idea of this algorithm is to train the Griffin system by making the objective function $\mathcal{L}(\theta)$ take the minimum value and update the θ in the network. In each round of training update process of this training algorithm based on SGD_{Gau} optimizer, we first calculate a random subset example $\nabla_{\theta_t}\mathcal{L}(\theta_t, x_i)$, and then add Gaussian noise to protect data privacy. After that, we conduct the calculation in the opposite direction of the noise gradient. Finally, Griffin is trained through the gradient descent algorithm based on the Gaussian Mechanism.

More specifically, in Algorithm 1, the gradient $\mathcal{L}$ is computed by calculating the gradients of the losses for a set of instances and taking the average of them. This average offers an unbiased estimator, whose variance decreases rapidly with the group size. We named such a group Lot. Therefore, we randomly arrange the instances and divide them into appropriately sized groups to use Gaussian Mechanism to protect data privacy. Meanwhile, for analysis, we assume that each batch is formed by selecting each example with independent probability ($p = \frac{Lot}{N}$), where N is the input data set size. Relevant research test indicates that this privacy protection for deep neural networks can be achieved at a modest cost in

Algorithm 2 The back-propagation training algorithm with Gaussian Mechanism SGD optimizer in Griffin.

Procedure: train $\left(L^{(1)}, L^{(2)}, \mathrm{v}\right)$

1: Train ensemble layer
2: $\vec{z} \leftarrow \mathrm{zeros}(k)$ //Initialize the input for $L^{(2)}$
3: **for** θ_i *in* $L^{(1)}$ **do**
4: $\vec{v}_i' = \mathrm{norm}_{0-1}\left(\vec{v}_i\right)$
5: $A_i, \vec{y}_i \leftarrow h_{\theta_i}\left(\vec{v}_i'\right)$ //Forward propagation
6: $Errors_{s_i} \leftarrow b_{\theta_i}\left(\vec{v}_i', \vec{y}_i\right)$ //Backward propagation
7: $\theta_i \leftarrow \mathrm{SGD}\left(A_i,\ Errors_{s_i}\right)$ //Weight update
8: $\vec{z}[i] \leftarrow RMSE\left(\vec{v}_i', \vec{y}_i\right)$ //Set error
End
9: Train output layer
10: $\vec{z}' = \mathrm{norm}_{0-1}(\vec{z})\ A_0,\ \vec{y}_0 \leftarrow h_{\theta_0}\left(\vec{z}'\right)$ //Forward propagation
11: $Errors_{S_0} \leftarrow b_{\theta_0}\left(\vec{Z}', \vec{y}_0\right)$ //Backward propagation
12: $\theta_0 \leftarrow SGD_{Gau}\left(A_0,\ Errors_{S_0}\right)$ //Weight update and add noise
13: **return** $L^{(1)}, L^{(2)}$

software complexity, model quality, and training efficiency so it is practicable to use Gaussian Mechanism in this case [34].

The algorithm for training mode with SGD_{Gau} optimizer on a single instance is presented in Algorithm 2. In the training process of the ensemble layer, the autoencoder neural network is still trained according to the previous description, that is: first, perform 0-1 normalization on the fourth line. Specifically, in this process, each autoencoder must record the maximum and minimum values of each input feature. It is worth mentioning that these maximum and minimum records are only updated during the training mode. Secondly, we execute the forward propagation algorithm and the backward propagation algorithm to update the training parameters in the autoencoder neural network in the ensemble layer, and finally record the Rmse value. Additionally, in the training process of the output layer, a stochastic gradient descent optimizer with Gaussian Mechanism is used. According to the Algorithm 1 aforementioned, Gaussian noise is added to the SGD optimizer, so data privacy is protected. To further explain, as shown in Fig. 4, we will use the SGD_{Gau} in the training process of the output layer.

In conclusion, We will train the training mode by the steps following:

1) Use normal instance x_i in X to train autocoder.
2) Execute: $\mathrm{s} = \mathrm{RMSE}\left(x_i, h_\theta\left(x_i\right)\right)$.
3) Update: if $(s \geq \emptyset)$ then $\emptyset \leftarrow s$ (let $\emptyset$ be threshold, whose initial value is -1).
4) Update the θ through learning the features of x_i.

For every normal packet, the training mode executes forward propagation. More specifically, for every autoencoder, the second layer ($l^{(2)}$) is activated by the weight ($W^{(1)}$) of the first layer ($l^{(1)}$). After that, the weight ($W^{(2)}$) of the second layer activates the third layer ($l^{(3)}$). To gain the $a^{(i+1)}$ from $l^{(i)}$, we transfer the $a^{(i)}$ to $l^{(i+1)}$: $a^{(i+1)} = f\left(W^{(i)} \cdot a^{(i)} + b^{(i)}\right)$. After that, we use the activation function called *sigmoid*: $f(\vec{x}) = \frac{1}{1+e^{\vec{x}}}$. We define the last output as: $\overrightarrow{y'} = a^{(L)}$, and

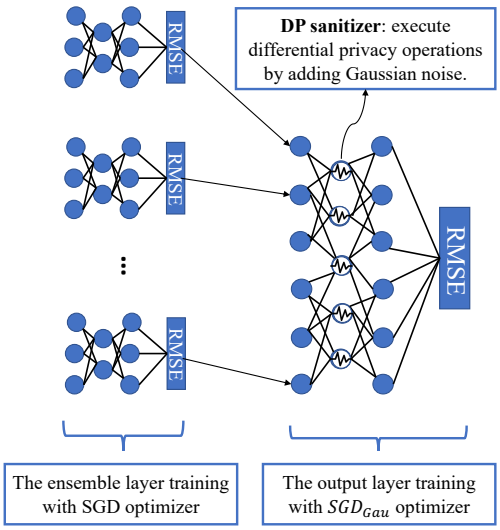


Fig. 4. An illustration of Griffin's output layer with SGD_{Gau}

let h be the forward propagation from input $\vec{v}_i$ to output $\overrightarrow{y'}$. Therefore, the hole function can be defined as:

$$h_\theta\left(x_i\right) = \overrightarrow{y'} \tag{8}$$

In the back propagation process, we use SGD. For this algorithm, the error between actual value and expected value would be stored. In this paper, the largest iteration number is 1, so we only learn once from each instance to maintain the online process.

2) Monitoring mode: As mentioned before, the monitoring mode aim to consider the abnormality of the instances and generate a anomaly score to make difference between benign traffic and malicious traffic. Specifically, the autocoder trained in X can recreate instance with same data distribution as X. Therefore, if a instance is in-conformity to the features learned from X, the RMSE would be high between recreated features and original features. Moreover, it is practical to regard RMSE as the anomaly score to determine whether the traffic is malicious. In conclusion, the monitoring mode can be executing below:

Execute: $s = RMSE\left(\vec{x}, h_\theta(\vec{x})\right)$.
Judgment: if $(s \geq \emptyset\beta)$ then alert.

In this process, the autoencoder does not update any internal parameters. Instead, it performs forward propagation through the entire network and returns the RMSE of $L^{(2)}$, which is the anomaly score $s \in [0, \infty)$. The higher the score, the more likely the system is to be attacked by network intrusion.

VI. Experimental Evaluation

In this section, we provide an evaluation to Griffin on its detection effectiveness. Specifically, we mainly test and evaluate Griffin's detection latency, accuracy, and privacy-preserving mechanism. As the fundamental technique of our evaluation, we first explain how our system set up, and then describe the detail of this evaluation.

TABLE III
THE DATASET USED TO EVALUATE GRIFFIN

Attack Type	Attack Name	Attack descriptions	Packets
Man in the middle (MITM)	ARP MitM	MITM attacks are designed to intercept normal network traffic, tamper with and sniff data, without the knowledge of the two communication parties	2504267
	Video Injection		2472400
	Active Wiretap		4554000
Reconnaissance	Fuzzing	The attackers though canning and fuzzing to obtain users' information.	2244139
	OS Scanning		1697000
Malicious Malware	Mirai	The attackers through malicious malware to intrude service.	764137
Denial of service	SYN DoS	Interferes with a server to make it less available.	2771276
	SSL Renegotiation		6084492
	SSDP Flood		4077266

A. Experimental Setup

1) Experimental Environment: Griffin is an NIDS, which conduct in SDN. Therefore, we use the Mininet platfrom to execute the experimental evaluation. As for the hardware condition, we use the core of Intel(R) Core(TM) i7-8550U CPU and the memory of 8G.

2) Datasets: The datasets used in this evaluation are the open Datasets, whose details are shown in Table. III.

3) Baselines: To evaluate the accuracy, robustness, and latency of Griffin, we used the following methods to establish baselines.

a) Signature-based method.: We used the state-of-art NIDS, Suricata [30]. It filters malicious packets by building various rules aiming for different network attacks characteristics. During the evaluation, we installed the Suricata version 6.0.4 and implemented the open-source Suricata based on the same hardware environment and datasets as Griffin.

b) Flow-level clustering.: We used the state-of-art data stream extraction, named pcStream2 [33]. It is an updated version of pcStream, using the Just-In-Time principal component analysis (PCA) to monitor the situation of processing streams. For a fair comparison, we used the same hardware condition and datasets as Griffin.

c) Traditional classification methods with same features with Griffin.: We used some traditional classification methods–Support Vector Machine and Random Forest to demonstrate the improvement of our method. Those methods (noted as F_SVM and F_RF) use the same features as Griffin, so it is fair to compare F_SVM and F_RF with Griffin.

4) Methics: We measure the detection latency and use the AUC to evaluate the detection accuracy. Moreover, we also use the change of AUC to evaluate the robustness. Finally, we use the loss value and true positive rate to evaluate the effectiveness of privacy preserving.

B. Sub-instance Selection

To improve the accuracy and reduce the latency of the Griffin, we perform an importance evaluation to 19 sub-instances (obtained from feature mapper) based on Random Forest. After that, we remove some sub-instance with low importance. Specifically, we order all sub-instances based on their features' importance using Random Forest. When constructing a decision tree according to the sub-instances, we get node segmentation effect metric by using the Gini Index and calculating the Gini Importance to indicate the feature importance. Moreover, the Gini Index is defined as [43]:

$$\mathrm{Gini(p)} = \sum_{q=1}^{Q} p_q \left(1 - p_q\right) = 1 - \sum_{q=1}^{Q} p_q^2 \tag{9}$$

where q denotes q-th class and p_q denotes the sample weight of q-th class. In this case, the importance of feature X_j in node m, namely the variable quantity of Gini Index in the node m, can be defined as:

$$VIM_{jm}^{(Gini)} = GI_m - GI_l - GI_r \tag{10}$$

where GI_l and GI_r denote Gini Index of new nodes after branching. If the node of feature X_j in decision tree i is in set M, then the importance of X_j in i-th tree $VIM_{ij}^{(Gini)}$ is defined as:

$$VIM_{ij}^{(Gini)} = \sum_{m \in M} VIM_{jm}^{(Gini)} \tag{11}$$

If there are n trees totally, then the sum of them defined as:

$$VIM_{j}^{(Gini)} = \sum_{i=1}^{n} VIM_{ij}^{(Gini)} \tag{12}$$

Finally, a normalization process to importance scores is made as following:

$$VIM_j = \frac{VIM_j}{\sum_{i=1}^{c} VIM_i} \tag{13}$$

where the denominator is the sum of all feature gains, and the numerator is the Gini index of feature j. After utilizing the method above to evaluate the importance of every sub-instance, we get the Fig. 5. Obviously, the importance of sub-instance 1 and sub-instance 5 is negligible. Therefore we can remove them. Therefore, we get 17 autoencoders to deploy in the ensemble layer in anomaly detector.

C. Detection Latency

The main idea in this paper to reduce the system latency is to create a detection model with low complexity, because algorithm complexity is always the main factor restricting the latency. More specifically, we use the feature mapper and ensemble layer to reduce the dimension of features. Therefore, we will first measure the CPU usage to evaluate the system complexity to demonstrate our method's effectiveness. We compare it with the other two SDNs (*F-SVM* and *F-RF*). The result showing in Fig. 6 illustrates that the CPU usage of

TABLE IV
DETECTION ACCURACY OF GRIFFIN AND BASELINES ON 9 ATTACKS

Methods	Griffin	Suricata	pcStream2	F_SVM	F_RF
Metrics	AUC				
ARP MitM	0.8048	-	0.7219	0.7452	0.6512
Video Injection	0.8237	-	0.5816	0.6718	0.6139
Active Wiretap	0.9188	-	0.7413	0.9281	0.7634
Fuzzing	0.9999	-	0.9999	0.9761	0.9599
OS Scan	0.9281	-	0.7513	0.7517	0.7212
Mirai	0.9795	-	0.8612	0.9516	0.8319
SYN DoS	0.9328	1.0000	0.6299	0.9078	0.7091
SSL Renegotiation	0.9412	1.0000	0.8815	0.9119	0.8577
SSDP Flood	0.9999	-	0.9971	0.9876	0.8674

The "-" notes that the corresponding NIDS does not have capacity to detect the attack or its AUC is below 0.5.

Griffin is around 45%. However, the CPU usage of the other two systems is around 85%, which is around twice Griffin's. Therefore, we conclude that the complexity of Griffin is low.

After testing the system complexity, we will evaluate the latency of the Griffin. In this part, Mininet is selected as the platform to build SDN and we use Ryu as the controller of SDN. The fat-tree structure is selected as the experimental network topology. Because, on the one hand, the fat-tree structure is widely used in the data center network, which is the most common application scenario of SDN. On the other hand, data centers are the focus of attackers. After that, we use the iperf (a tool in Mininet) for function expansion and introduce the collected network traffic datasets into Mininet to conduct the experiment [44]

The latency defined here is the time it takes for the system to detect an attack from the moment of attack. More specifically, let t_{delay} be the system latency, $t_{feature}$ denotes the time it takes for the feature extractor and the feature mapper in switches of the data plane. What is more, t_{test} and $t_{message}$ denote the time taken by the anomaly detector and messaging to switches, respectively. Therefore, the t_{delay} is the summary of those factors: $t_{\text{delay}} = t_{\text{feature}} + t_{\text{test}} + t_{\text{message}}$. We use several widespread attack datasets to conduct the system latency experiment. Finally, we get the result showing in Fig. 7, which is a destiny figure. The peaks of different datasets

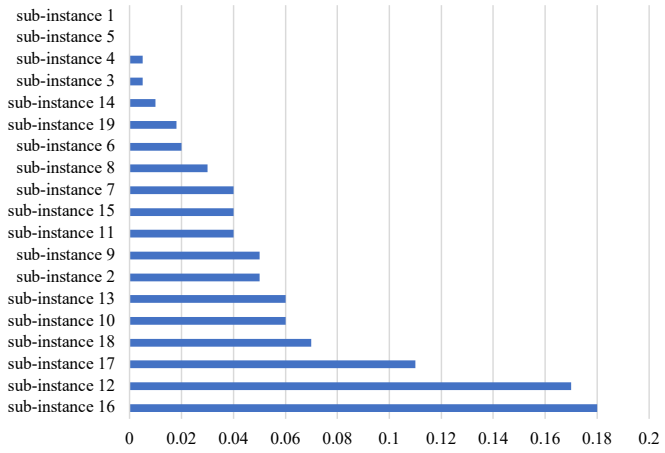


Fig. 5. Feature selection based on random forest

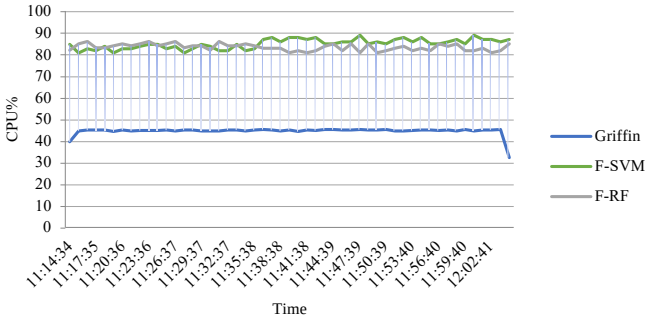


Fig. 6. Compare the Griffin's CPU usage with ANN-SVM NIDS and ANN-RF NIDS.

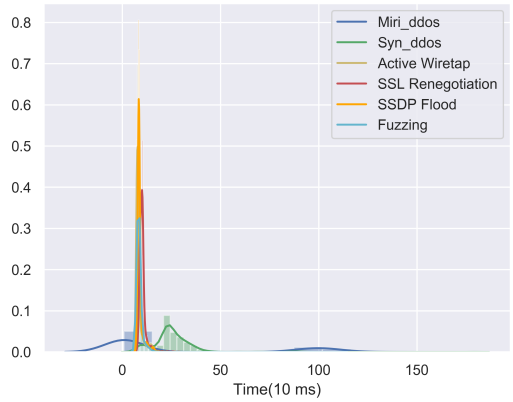


Fig. 7. Time delay of system with multiple test datasets

appear at the same point of 0.1s, which means the system's latency for all attacks concentrates at 0.1s. Moreover, in some particular cases, the latency is not 0.1s, but not more than 1.5s. These results indicate that the Griffin has low latency.

D. Detection Accuracy

In this experiment, we evaluate the detection accuracy of five different NIDSs by measuring the Area Under Curve (*AUC*). The NIDS tested here are declared in VI-A, so we will not depict them redundantly. As for AUC, it is defined as the area enclosed by the coordinate axis under the Receiver Operating Characteristic Curve (*ROC Curve*). The value of this area will not be greater than 1. Since the ROC curve is generally above the line y=x, the value of AUC ranges

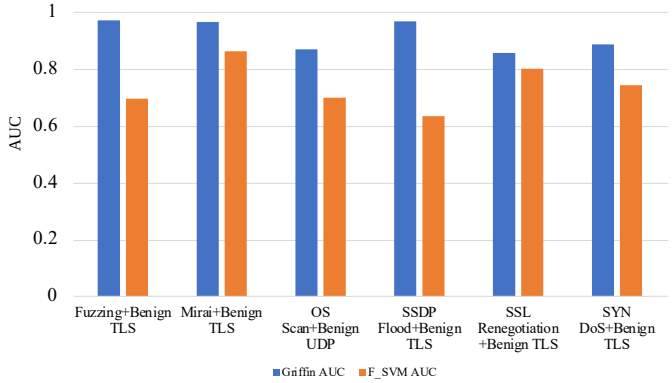


Fig. 8. Detection accuracy with various evasion strategies.

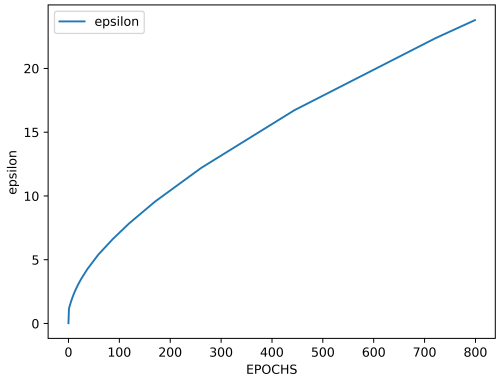


Fig. 9. Diagram of epoches and epsilon

between 0.5 and 1. The closer the AUC is to 1.0, the higher the authenticity of the detection method; when it is equal to 0.5, the authenticity is the lowest and the detection is similar to random guess.

As shown in Table IV, the Griffin outperforms among the five NIDSs in terms of detection accuracy. Firstly, Griffin's average *AUC* value is 0.9254, which is 16%, 6%, and 19% higher than pcStream2, F_SVM, and F_RF respectively. As for Suricata, we found its performance is good in SYN Dos and SSL Renegotiation detection, but it can not detect other attacks. Additionally, although the *AUC* value of Griffin in ARP MitM cases is 0.8048 which is not an excellent value, it is higher than other NIDSs' *AUC* value in the same case. What's more, Griffin has an excellent performance in detecting Fuzzing, Mirai, and SSDP Flood, whose *AUC* values are 0.99, 0.98, and 0.99 respectively.

E. Robustness of Detection

To evaluate the robustness of Griffin, we assume that the attackers know the existence of NIDS. Therefore, they will try to evade malicious traffic detection, i.e., injecting benign traffic into malicious traffic. In the experiment, we simulate the evasion processes by injecting benign TSL and UDP traffic into five malicious traffic – Fuzzing, Mirai, OS Scan, SSDP Flood, SSL Renegotiation, and SYN DoS. The reason why we use TSL and UDP traffic is that those two kinds of traffic occupy a large proportion of benign network traffic.

More specifically, in our experiment, we mix the benign traffic and malicious traffic with the proportion of 1:1, 2:1, 3:1, 4:1, 5:1, 6:1, 7:1, and 8:1 for a large amount of benign traffic (the proportion bigger than 8:1) will reduce the effectiveness of attacks. And we calculate the average of those experiments' results, which are displayed in Figure 8. We compare Griffin with F_SVM, because the F_SVM uses features generated by Griffin and has good performance in the previous test. According to Figure 8, Griffin has at most 9% of AUC value decrease. However, the F_SVM has 36% AUC decrease. For example, the Fuzzing mixed with benign TLS can lead to 28.7% decrease in AUC value in F_SVM. Therefore, we conclude that Griffin's accuracy has a small influence on the various evasion strategies.

F. Privacy Preserving

In order to protect the data privacy of traffic packets, we conduct SGD_{Gau} optimizer in anomaly detector's training mode aforementioned. Therefore, we will experiment with the effectiveness of this method and its impact on the prediction of the system based on the Mirai dataset.

1) Privacy Preserving Setup: As described in Algorithm 2, we add Gaussian Mechanism to the SGD optimizer in the training mode's output layer. We will compare the prediction accuracy of the system before and after using the Gaussian Mechanism to research whether SGD_{Gau} will impact the performance of Griffin.

In this section, we will set up some concrete parameter values of this privacy-preserving method. Firstly, the overall privacy loss (ε, δ) is calculated from the noise level, the sampling ratio of each lot $q = L/N$ (so each epoch consists of $1/q$ batches), and the number of epochs E (so the number of steps is $T = E/q$). After that, we determine the target $\delta = 10^{-5}$ in our experiments. Additionally, we get Fig. 9 through computing the value of ε as a function of the training epoch E. When E = 400, the value is 15.635. Therefore, we achieve (15.635,10^{-5})-differential privacy. Based on the parameters set above, we will explore the training loss of the output layer of the anomaly detection and the impact of some essential parameters on detection.

2) Implementation of SGD with Gaussian Mechanism: Using the parameters mentioned above and a low Gaussian noise scale to conduct experiments, we obtain the loss curve. Fig. 10 depicts the loss values under different epoch with six different datasets: Mirai(*loss_mirai*), SYN DDoS (*loss_syn*), SSL Renegotiation (*loss_ssl*), SSDP Flood (*loss_ssdp*), Fuzzing (*loss_fuzzing*) and Active Wiretap (*loss_active*). No matter which datasets are tested, the loss values are small, which is close to 0. That is to say, the training with SGD_{Gau} optimizer is successful in gaining a detection model. Moreover, this system can preserve privacy without losing the data set's original training features.

According to the Algorithm 1 aforementioned, we question that the increase in the Gaussian noise scale in the SGD optimizer may affect the effect of Griffin system training,

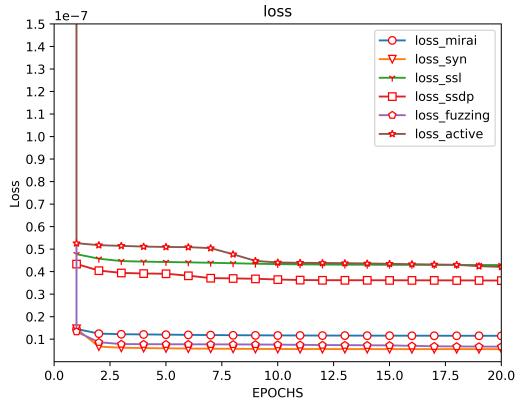


Fig. 10. The loss values under different epoches.

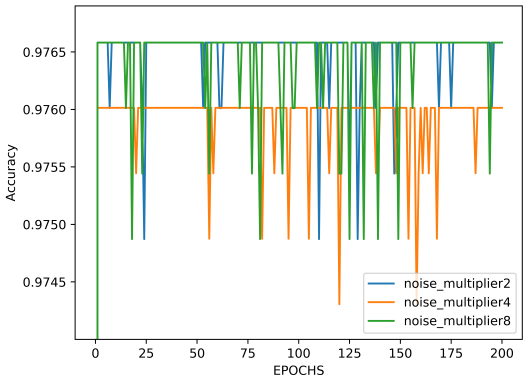


Fig. 11. The detection accuracy under different noise scale.

including system accuracy and privacy protection efficiency. Therefore it is necessary to conduct an analysis experiment and gain the balance between the system accuracy and privacy protection efficiency. To analyze this problem, We use the noise scale parameter as an independent variable and the prediction accuracy of the system as a dependent variable to evaluate the system's performance. As shown in Fig. 11, no matter which noise scales are selected to evaluate the system, the production accuracy is always around 97%—comparing with the production accuracy of the system without Gaussian Mechanism before (98%). Therefore, we conclude that the noise scale of Gaussian Mechanism in this system does not affect the prediction accuracy of the Griffin. That means we can ensure the best privacy efficiency and system accuracy.

Finally, we get Fig. 12, which illustrates the final process – threshold selection (here we take the Mirai intrusion dataset for example).To find a threshold that results in an acceptable True Positive Rate (TPR), False Positive Rate (FPR), and False Negative Rate (FNR), firstly, we get the RMSE values for the training set (all benign samples). After that, we make a comparison between the malicious samples and benign samples distributions. Finally, we select the threshold by using the RMSE value where the probability of getting a higher benign value lower than our requirement. For the Mirai intrusion dataset, we set the threshold in detection mode as 100. Therefore, the scattered red dots indicate a network attack. In addition, from the scattered point distribution in the figure, we can also make a detailed analysis of the attacker's behavior. For example, the gray part at the beginning of the image illustrates that the attacker is opening the attack channel. Moreover, the black scattered dots indicate that the attacker is scanning the intrusion object. At the orange-red part, the attacker is invading the intrusion object and finally launches a DDoS attack.

VII. Conclusion

In this paper, we develop Griffin, a real-time NIDS in SDN that utilizes statistic features of traffic packets via an ensemble of autoencoder neural networks to enable zero-day detection.

Griffin consists of four parts: packet capture, feature extractor, feature mapper, and anomaly detector. In particular, we first use the feature extraction window with a damping parameter in the feature extractor to extract the flow-level statistic features, which can precisely illustrate sequential information of the traffic packet. Therefore, the Griffin can detect zero-day attacks and achieve robustness. Secondly, in the feature mapper, we use the Hierarchical Clustering Algorithm to map the features into sub-instances to reduce the dimension of features. Furthermore, we create an ensemble of autoencoder neural networks to reduce the feature complexity. Therefore, Griffin achieves the goal of real-time detection and low latency. Finally, we add noise to parameters in the anomaly detector to protect the dataset privacy in the training period. Specifically, we use Gaussian Mechanism to add noise into the parameters in the training period, making it difficult for attackers to deduce users' privacy information from the model's parameters.In our model, we achieve the goal of gaining both the best privacy efficiency and excellent accuracy simultaneously, which is a challenging issue in differential privacy framework establishment.

Extensive experimental evaluations and statistical analyses validated the effectiveness and privacy protection of Griffin. We compare the Griffin with some traditional algorithms and state-of-art solutions. Our results show that Griffin achieves at most 19% improvement of AUC, while achieving at most 40% improvement of complexity. Even in the situation with evasion, the Griffin has just at most 9% decrease of AUC, which is a good performance compared with other solutions. Finally, we use the Gaussian mechanism in the SGD optimizer in model training. According to the test, the system prediction accuracy is 97%, which is only a lower 1 percentage than before. It proves that this data privacy protection mechanism does not affect the system detection accuracy.

Acknowledgments

This work is supported in part by National Key R&D Program of China under Grant Nos. 2020YFE0200600. Yubo Song is the corresponding author. This work is partially supported by the Fundamental Research Funds for the Central Universities 2242022k30007.

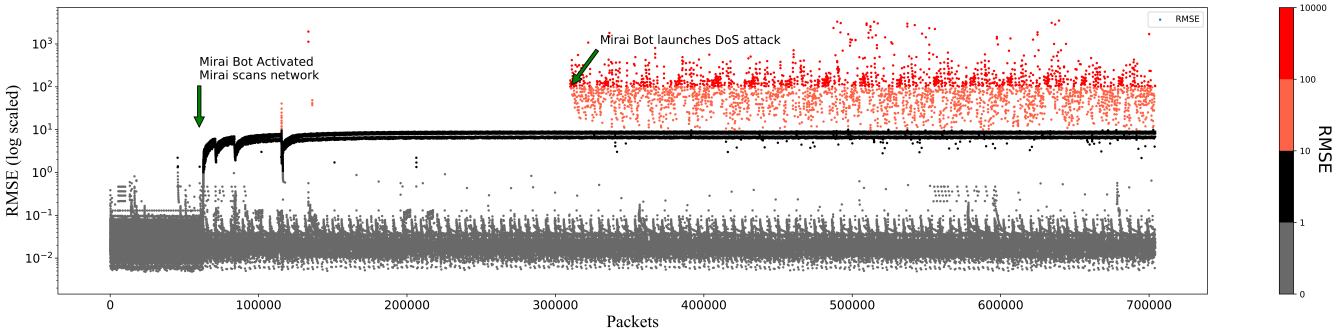


Fig. 12. The result diagram of privacy preserving NIDS based on Mirai dataset

REFERENCES

[1] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010, pp. 1–6.

[2] J. H. Cox, J. Chung, S. Donovan, J. Ivey, R. J. Clark, G. Riley, and H. L. Owen, "Advancing software-defined networks: A survey," *IEEE Access*, vol. 5, pp. 25 487–25 526, 2017.

[3] S. Studios, "Sdn automation, programmability, and programmable networks," 2014, https://www.sdxcentral.com/artificial-intelligence/automation/definitions/programmability-network-automation-sdn-networks/.

[4] K. Pentikousis, Y. Wang, and W. Hu, "Mobileflow: Toward software-defined mobile networks," *IEEE Communications magazine*, vol. 51, no. 7, pp. 44–53, 2013.

[5] M. Haranas, "16 hot networking products putting the sizzle in sd-wan," 2016, https://www.crn.com/slide-shows/networking/300082325/16-hot-networking-products-putting-the-sizzle-in-sd-wan.htm/2.

[6] S. Sezer, S. Scott-Hayward, P. K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, "Are we ready for sdn? implementation challenges for software-defined networks," *IEEE Communications Magazine*, vol. 51, no. 7, pp. 36–43, 2013.

[7] P. Devan and N. Khare, "An efficient xgboost–dnn-based classification model for network intrusion detection system," *Neural Computing and Applications*, pp. 1–16, 2020.

[8] H. Wang, Z. Cao, and B. Hong, "A network intrusion detection system based on convolutional neural network," *Journal of Intelligent & Fuzzy Systems*, vol. 38, no. 6, pp. 7623–7637, 2020.

[9] S. Al and M. Dener, "Stl-hdl: A new hybrid network intrusion detection system for imbalanced dataset on big data environment," *Computers & Security*, vol. 110, p. 102435, 2021.

[10] Z. Ahmad, A. Shahid Khan, C. Wai Shiang, J. Abdullah, and F. Ahmad, "Network intrusion detection system: A systematic study of machine learning and deep learning approaches," *Transactions on Emerging Telecommunications Technologies*, vol. 32, no. 1, p. e4150, 2021.

[11] K. Borders, J. Springer, and M. Burnside, "Chimera: A declarative language for streaming network traffic analysis," in *21st {USENIX} Security Symposium ({USENIX} Security 12)*, 2012, pp. 365–379.

[12] M. A. Jamshed, J. Lee, S. Moon, I. Yun, D. Kim, S. Lee, Y. Yi, and K. Park, "Kargus: a highly-scalable software-based intrusion detection system," in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 317–328.

[13] H. Li, H. Hu, G. Gu, G.-J. Ahn, and F. Zhang, "vnids: Towards elastic security with safe and efficient virtualization of network intrusion detection systems," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 17–34.

[14] J. Nam, M. Jamshed, B. Choi, D. Han, and K. Park, "Haetae: Scaling the performance of network intrusion detection with many-core processors," in *International Symposium on Recent Advances in Intrusion Detection*. Springer, 2015, pp. 89–110.

[15] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Communications surveys & tutorials*, vol. 18, no. 2, pp. 1153–1176, 2015.

[16] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, pp. 1–58, 2009.

[17] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *computers & security*, vol. 28, no. 1-2, pp. 18–28, 2009.

[18] R. Tang, Z. Yang, Z. Li, W. Meng, H. Wang, Q. Li, Y. Sun, D. Pei, T. Wei, Y. Xu *et al.*, "Zerowall: Detecting zero-day web attacks through encoder-decoder recurrent neural networks," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 2479–2488.

[19] B. J. Radford, L. M. Apolonio, A. J. Trias, and J. A. Simpson, "Network traffic anomaly detection using recurrent neural networks," *arXiv preprint arXiv:1803.10769*, 2018.

[20] K. Wang and S. J. Stolfo, "Anomalous payload-based network intrusion detection," in *International workshop on recent advances in intrusion detection*. Springer, 2004, pp. 203–222.

[21] K. Bartos, M. Sofka, and V. Franc, "Optimized invariant representation of network traffic for detecting unseen malware variants," in *25th {USENIX} Security Symposium ({USENIX} Security 16)*, 2016, pp. 807–822.

[22] L. Invernizzi, S. Miskovic, R. Torres, C. Kruegel, S. Saha, G. Vigna, S.-J. Lee, and M. Mellia, "Nazca: Detecting malware distribution in large-scale networks." in *NDSS*, vol. 14. Citeseer, 2014, pp. 23–26.

[23] T. Nelms, R. Perdisci, M. Antonakakis, and M. Ahamad, "Webwitness: Investigating, categorizing, and mitigating malware download paths," in *24th {USENIX} Security Symposium ({USENIX} Security 15)*, 2015, pp. 1025–1040.

[24] J. Zheng, Q. Li, G. Gu, J. Cao, D. K. Yau, and J. Wu, "Realtime ddos defense using cots sdn switches via adaptive correlation analysis," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 7, pp. 1838–1853, 2018.

[25] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, "Deep learning approach for network intrusion detection in software defined networking," in *2016 International Conference on Wireless Networks and Mobile Communications (WINCOM)*. IEEE, 2016, pp. 258–263.

[26] Z. Liu, Y. He, W. Wang, and B. Zhang, "Ddos attack detection scheme based on entropy and pso-bp neural network in sdn," *China Communications*, vol. 16, no. 7, pp. 144–155, 2019.

[27] A. Hamza, H. H. Gharakheili, and V. Sivaraman, "Combining mud policies with sdn for iot intrusion detection," in *Proceedings of the 2018 Workshop on IoT Security and Privacy*, 2018, pp. 1–7.

[28] A. Abubakar and B. Pranggono, "Machine learning based intrusion detection system for software defined networks," in *2017 seventh international conference on emerging security technologies (EST)*. IEEE, 2017, pp. 138–143.

[29] M. Du, Z. Chen, C. Liu, R. Oak, and D. Song, "Lifelong anomaly detection through unlearning," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 1283–1297.

[30] OISF, "Suricata's thriving global community," [EB/OL], https://suricata.io/ Accessed 18 November ,2021.

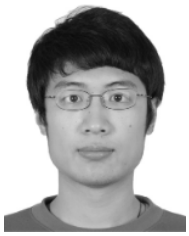
[31] L. Bilge, D. Balzarotti, W. Robertson, E. Kirda, and C. Kruegel, "Disclosure: detecting botnet command and control servers through large-scale netflow analysis," in *Proceedings of the 28th Annual Computer Security Applications Conference*, 2012, pp. 129–138.

[32] C.-Y. Lin and S. Nadjm-Tehrani, "Timing patterns and correlations in spontaneous {SCADA} traffic for anomaly detection," in *22nd Inter-*

national Symposium on Research in Attacks, Intrusions and Defenses ({RAID} 2019), 2019, pp. 73–88.
[33] Y. Mirsky, T. Halpern, R. Upadhyay, S. Toledo, and Y. Elovici, "Enhanced situation space mining for data streams," in *Proceedings of the Symposium on Applied Computing*, 2017, pp. 842–849.
[34] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 308–318.
[35] M. A. P. Chamikara, P. Bertok, I. Khalil, D. Liu, S. Camtepe, and M. Atiquzzaman, "Local differential privacy for deep learning," *arXiv preprint arXiv:1908.02997*, 2019.
[36] N. Phan, X. Wu, H. Hu, and D. Dou, "Adaptive laplace mechanism: Differential privacy preservation in deep learning," in *2017 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2017, pp. 385–394.
[37] H. Dreger, A. Feldmann, V. Paxson, and R. Sommer, "Operational experiences with high-volume network intrusion detection," in *Proceedings of the 11th ACM conference on Computer and communications security*, 2004, pp. 2–11.
[38] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *2010 IEEE symposium on security and privacy*. IEEE, 2010, pp. 305–316.
[39] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: an ensemble of autoencoders for online network intrusion detection," *arXiv preprint arXiv:1802.09089*, 2018.
[40] L. Yang, Y. Song, S. Gao, B. Xiao, and A. Hu, "Griffin: An ensemble of autoencoders for anomaly traffic detection in sdn," in *GLOBECOM 2020-2020 IEEE Global Communications Conference*. IEEE, 2020, pp. 1–6.
[41] F. Murtagh and P. Contreras, "Methods of hierarchical clustering," *arXiv preprint arXiv:1105.0121*, 2011.
[42] F. Nielsen, "Hierarchical clustering," in *Introduction to HPC with MPI for Data Science*. Springer, 2016, pp. 195–211.
[43] M. Kaur, "An approach for sentiment analysis using gini index with random forest classification," in *International Conference On Computational Vision and Bio Inspired Computing*. Springer, 2019, pp. 541–554.
[44] Mininet, "Mininet," [EB/OL], http://mininet.org/ 2021.

Liyan Yang received the B.Sc. degree in information srcurity, from Xi Dian University, China. She is the first-year postgraduate student in School of Cyber Science and Engineering, Southeast University.

Yubo Song received the B.Sc. degree in electric engineering, the M.Sc. degree in communication andinformation system, and the Ph.D. degree in signaland information processing from Southeast University, China. He joined the School of Cyber Scienceand Engineering, Southeast University, as a Lecture after graduation. He is currently an Associate Professor and the Vice Director of the School of Cyber Science and Engineering, Southeast University. He has published more than 30 technical articles in international journals and conferences. His research interests include mobile security, wireless network security, and security protocol analysis.



Shang Gao received the B.S. degree from Hangzhou Dianzi University, China, in 2010, the M.E. degree from Southeast University, China, in 2014, and the Ph.D. degree from the Hong Kong Polytechnic University, Hong Kong, in 2019. He is currently a Researcher Assistant Professor of computing department in the Hong Kong Polytechnic Universiry, Hong Kong. His research interests include information security, network security, software-defined networks, blockchain security, and applied cryptography.

Aiqun Hu received the B.Sc. (Eng.), M.Eng.Sc., and Ph.D. degrees from Southeast University in 1987, 1990, and 1993, respectively.

He was invited as a Post-Doctoral Research Fellow with The University of Hong Kong from 1997 to 1998 and a TCT Fellow with Nanyang Technological University in 2006. His research interests include data transmission and secure communication technology. He has published two books and over 100 technical papers in wireless communications field.

Bin Xiao received the B.Sc. and M.Sc. degrees in electronics engineering from Fudan University, China, and the Ph.D. degree in computer science from The University of Texas at Dallas, USA. He is currently an Associate Professor with the Department of Computing, The Hong Kong Polytechnic University. He has published more than 100 technical articles in international top journals and conferences. His research interests include distributed wireless systems, net-work security, software-defined networks (SDN), and blockchain. He is an ACM member. He is a recipient of the Best Paper Award of the international conference IEEE/IFIP EUC-2011. He is currently an Associate Editor of the Journal of Parallel and Distributed Computing (JPDC) and Security and Communication Networks (SCN).