Generative Intrusion Detection and Prevention on Data Stream

HyungBin Seo and MyungKeun Yoon, *Kookmin University*

https://www.usenix.org/conference/usenixsecurity23/presentation/seo

This paper is included in the Proceedings of the 32nd USENIX Security Symposium.

August 9–11, 2023 • Anaheim, CA, USA

978-1-939133-37-3

Generative Intrusion Detection and Prevention on Data Stream

HyungBin Seo
Kookmin University

MyungKeun Yoon
Kookmin University

Abstract

Data arrive in a stream, for example, network packets, emails, or malicious files, and ideally they should be investigated for cybersecurity. The current best practice would be to check if each data includes any suspicious signatures, or simply strings, which were obtained a priori by elaborate manual analysis in previous cyberattack cases. Unfortunately, unknown attacks, called zero-day attacks, cannot be timely detected in this way because no signature is available yet. To tackle this problem, recent studies have presented high-speed methods that can extract frequent substrings from the data stream and use them as attack signatures because the frequently-occurred signatures are often related with attacks; unfortunately, more benign signatures are extracted than malicious ones, especially when there is no attack in most of the time. This causes both a tremendous number of false-positives and extra human interventions to remove benign signatures. In this paper, we design a new streaming algorithm that can first identify a frequent *group* of signatures appearing together at the same time from data streams. Using this frequent *signature-group* instead of frequently-occurred individual signatures, the new scheme achieves a high detection accuracy by mitigating the false-positive problem with only a small fixed amount of memory and a constant number of hash operations, which has not been achieved by any previous work. This improvement comes from a new method for summarizing similar data with a fixed amount of memory, called a *minHashed virtual vector*, which allows us to automatically identify a frequent group of signatures with each data read only once. We perform exhaustive experiments on different private and open datasets, to verify both the practical effectiveness and the experimental reproducibility of the new scheme.

1 Introduction

One of the most important tasks in cybersecurity is to collect and analyze data that come from a range of security products, networking components, servers, and even user endpoints. The data can be security events or alerts triggered by monitoring devices, network packets, emails, suspicious files, etc., which are continuously generated. These data are often collected by a security operations center (SOC) where security analysts work for real-time monitoring and manual analysis for critical data [10, 25, 30, 56]. Recent studies reveal that security alerts overwhelm human resources by their large volumes [10, 16, 25, 30, 45, 56]. For example, more than million alerts are generated per day while only dozens of people work for alert analysis at best. Therefore, intelligent and automatic data analysis tools are essential for this industry.

Attack detection still relies on signature matching after a signature, a simple string or a regular expression, is manually written by experts. The problem is that attackers use more zero-day attacks, vulnerabilities and exploits only known to attackers [61], and the signature-based detection may not effectively mitigate the attacks any longer. This motivates a lot of machine learning (ML) approaches for anomaly detection in cybersecurity. However, ML inherently causes false-positives, non-attacks mistakenly considered as attacks [50]. As the sheer volume of security data meets ML-based detection, a tremendous number of false-positives may be generated. We need a new detection and prevention method that has both the advantages of the precise detection of signature-based methods and the zero-day attack detection of ML-based methods. This motivates us to study a new automatic signature-generation method to mitigate zero-day attacks.

Existing Solutions. The best practices of cyberattack detection and prevention can be categorized into two types, misuse-detection and anomaly-detection [50]; in general, misuse-detection uses known signatures representing specific attacks while anomaly-detection relies on machine learning (ML). Therefore, misuse-detection is generally known to be unable to detect a zero-day attack. Although anomaly-detection is known to mitigate zero-day attacks promptly without human intervention, the false detection problem, also known as false-positive and alert fatigue, still seems challenging [10, 25, 50, 56]. Besides, because ML models hardly provide enough explanation or evidence about their decision, security operators are often reluctant to trust and deploy them in production settings [17]. We summarize the different types of intrusion detection and prevention schemes in Table 1;

- **Manual rule composition**: Detection rules, or signatures, are manually composed by security experts after new attacks are analyzed with enough time and human efforts [43, 49, 53]. This is practically adopted by most security products. Because signatures are specifically defined, detection evidence is also provided. However, this is not appropriate for detecting zero-day attacks because no signatures are known yet.
- **Signature generation**: Signatures are automatically generated to detect a certain type of zero-day attacks that are accompanied with a large number of similar data in a short period of time [9, 39, 48]. Such attacks may include worms, distributed denial of service (DDoS) attacks, etc. This automatic generation is useful, but existing methods find only the most frequently-occurred substrings from a dataset. This causes a large number of false-positives and requires additional human resources.
- **Supervised learning**: When a dataset includes both attack and normal cases with labels, an ML model can be trained in a supervised way [19, 52]. The model can detect not only the attacks seen during the training phase but also their variants [51]. Because attack variants can evade signature-based detection, supervised learning helps to detect simple zero-day attacks to some degree. However, data labeling requires extra costs and human resources.
- **Unsupervised learning**: Even when there are no labels for the dataset, an ML model can be trained in an unsupervised way [54, 62]. In general, the training dataset includes only normal cases without attacks. Then, the model can identify data somewhat similar to something previously seen. For a given test data, if the model concludes that the data was not seen a priori, it can be considered as a new data and potentially a zero-day attack [61]. This inherently raises false-positives [50] and requires extra efforts to refine the training dataset where no attack should be included.

Table 1: Different Types of Intrusion Detection and Prevention

Type		Accuracy	Automation	Zero-day
Misuse-detection (signature)	Manual rule composition [49, 53]	High	Low	Low
	Signature generation [9, 48]	Medium	Medium	High
	Signature-group generation [GIPS]	***High***	***High***	***High***
Anomaly-detection (ML)	Supervised learning [19, 52]	Medium	Medium	Medium
	Unsupervised learning [54, 62]	Low	Medium	High

In this paper, we present a new detection and prevention method to mitigate a certain type of zero-day attacks that include similar data redundantly. The new scheme can identify a group of similar data from a large volume of data streams and automatically extract a group of signatures appearing at the same time over the identified group. The signature-group, instead of single signature, significantly reduces false-positives. Because both the identification of similar data group and the generation of signature group are automatically done without human intervention over data streams, we call the new scheme *Generative Intrusion detection and Prevention on data Stream (GIPS)*. The new scheme achieves a high detection accuracy with a small fixed amount of memory and a constant number of hash operations, which has not been achieved by any previous work. This improvement comes from a new summarizing method for similar data in a fixed amount of memory, called a *minHashed virtual vector*, which allows us to identify a group of similar data with each read only once. The contribution of this paper can be summarized as follows:

- We present GIPS to mitigate zero-day attacks of repetitive content. Because GIPS uses a signature-group instead of a single signature, the accuracy is much better than a state-of-the-art (SOTA) [9] and other previous work [39, 48].
- Motivated by the minHash theory [14], a new data structure to summarize similar groups from data streams is designed. This requires only a fixed amount of memory and a fixed number of hash operations, which runs faster than clustering algorithms by orders of magnitude.
- A range of data types can be covered by GIPS, for example alerts, logs, packets, emails, files, etc. We evaluate GIPS with different datasets, including real network packets collected from an ISP, and some public datasets for reproducible experiments.

The rest of the paper is organized as follows. We introduce the problem and motivation in Section 2. We present the new scheme in Section 3 and the experimental results in Section 4. Sections 5 and 6 cover related work and conclusions.

2 Problem and Design Goals

2.1 Problem

We study the challenging detection and prevention problem of zero-day attacks [61]. We assume that security monitoring tools or products such as intrusion detection and prevention system (IDPS) [1], endpoint detection and response (EDR), security information and event management (SIEM) [56], are deployed, but the detection signature of the zero-day attack is not known yet.

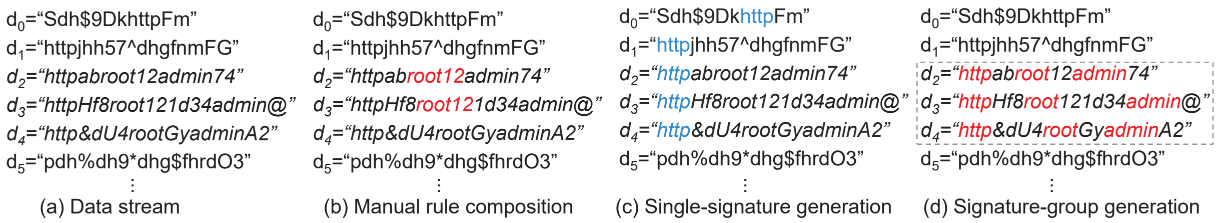


Figure 1: Manual rule composition [43, 49], single-signature generation [9], and signature-group generation of GIPS.

Persistent zero-day attack: In this paper, we assume that an attacker attempts a zero-day attack by repeatedly sending network packets of similar attack commands, emails with the same phishing links or similar malware files attached, which we call a *persistent zero-day attack*. Although the attacker repeats to send similar data, their amount can be relatively small, compared with the normal ones. We define an attack data ratio, r_a, to be the ratio of the zero-day attack data to the all data. For example, if IDPS monitors 10 similar packets from the zero-day attack and 90 normal packets from benign sessions at the same time, r_a becomes 10/100.

The difference between the persistent zero-day attack and the existing worm/DDoS-based zero-day attack of previous work [9,24,35,39,48] is that r_a of the latter is close to 1. Therefore, most data would be associated with the worm/DDoS-based attack, and therefore a simple detection and prevention strategy can mitigate the attack [48]. For example, the single-signature generation strategy of Table 1 works when r_a is close to 1. However, a proper signature cannot be generated with a small r_a. Therefore, we first need to separate a group of attack-related data of high similarity from normal ones, which is another challenging problem. In this paper, we present GIPS to solve the persistent zero-day attack problem.

Data analysis tool: GIPS not only mitigates persistent zero-day attacks but also can be used as a general data-analysis tool to identify similar data groups from data streams and to extract a group of signatures per group. We expect GIPS can process any datasets such as network packets, emails, malware files, alerts and logs from security devices. To the best knowledge of ours, this is the first streaming algorithm for similarity-based grouping on Jaccard index [58] and signature-group generation. We expect GIPS can be practically deployed in the security industry because it can provide manifest signatures as an evidence for cyberattacks. We observe that recent deep-learning models show good performance metrics at the laboratory level; however, they are less popularly used in practice because no clear evidence is provided, called a black box problem, restricting their usage for cybersecurity [17].

2.2 Design Goals

The design goals of GIPS can be summarized as follows:

- **Big-group identification**: GIPS should be able to identify a large group of similar data. When a new data arrives, the number of all previous data that are similar to the new data should be estimated. For the new data, the ratio of the number of its similar data to the number of all data is denoted as r_b; if r_b is bigger than a predefined threshold, we call the group of the similar data including the new data as a *big-group*.
- **Signature-group generation**: A group of common signatures instead of a single common signature should be extracted from a big group. This signature-group reduces false-positives when used for misuse-detection.
- **Streaming algorithm**: GIPS should process data streams as a streaming algorithm with only a small fixed-size memory and a constant number of hash operations.
- **Automatic processing**: Because human interventions cause much higher costs and longer delayed responses, the new scheme should minimize human interventions.
- **Various data types**: The evidence of zero-day attacks can be observed from different datasets, packets, files, emails, etc., and GIPS can be applied to any data type.

We show the difference between SOTA and GIPS in Fig. 1. We assume that data continuously arrives in a stream as in Fig. 1 (a). We denote each data as d_i, here network packets or IDPS events including suspicious packets, where only d_2, d_3, and d_4 include similar attack signatures. If the attack is not a zero-day and its detection signature is already known as "root12", security operators can configure security devices to detect and prevent the attack as in Fig. 1 (b). However, if this is a zero-day attack, the attack cannot be detected despite the repeated similar contents of d_2, d_3, and d_4. Even with the known attack signature, d_4 evades the detection.

To mitigate this kind of zero-day attacks, automatic signature-generation schemes have been presented [9, 24, 28, 29, 31, 32, 35, 39, 46, 48, 65]. However, they find only the frequent words or tokens as in Fig. 1 (c). In this case, "http" is the most frequent word; if this is automatically used as the attack signature, a huge number of false positives would occur. Two solutions have been proposed by previous work; first, the frequent signature should be reviewed again by security experts. If the signature is benign, it is filtered out like stop-words in the literature of natural language processing [9]. Second, only attack-related data should be collected heuristically, and the signature is extracted from the collected data group [29]. For example, if we were able to make a group of d_2, d_3, and d_4 a priori, the most frequent word would be "http", "root", and "admin". However, this grouping heuristic from endless data streams is another challenging problem, which may require extra human interventions. Both solutions require extra human resources and delayed processing time, which may prevent their practical use in SOCs.

In this paper, we present GIPS as a new signature generation scheme that first automatically makes a group of similar data from data streams and then generates a group of signatures from the data group as shown in Fig. 1 (d).

3 GIPS: Generative Intrusion Detection and Prevention on Data Stream

When streaming data continue to arrive, GIPS first identifies a big-group of similar data, and then extracts a group of signatures from the big-group. There are four components of GIPS, Minhashed Virtual-Vector (MV2), Jaccard-Index Grouping (JIG), Signature-Group Generation (SG2), and Automatic WhiteListing (AWL); the first two components are for the big-group identification, and the others for the signature generation. Fig. 2 shows the overall process of GIPS with four components. Frequently used notations are summarized in Table 2.

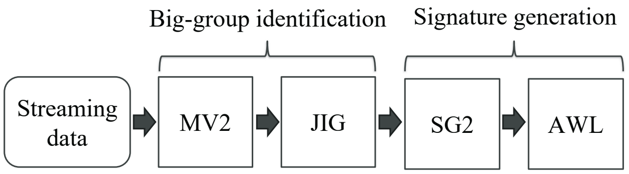


Figure 2: Overall process of GIPS.

Table 2: Notations

d_i	i^{th} data from a data stream.
$CDC(d_i)$	Set of chunks derived from d_i by a CDC algorithm [64]. $CDC(d_i) = d_i$ when d_i is of a set type.
$j(d_i, d_j)$	Jaccard index between $CDC(d_i)$ and $CDC(d_j)$.
$B_i[m]$	Bitmap representing d_i.
$MV[m]$	Integer array summarizing a data stream.
θ_J	Threshold for the similarity between two data.
θ_B	Threshold for the big-data identification.
$\theta_{C,i}$	Threshold for the big-counter of d_i over $MV[m]$.

3.1 MV2: Minhashed Virtual-Vector

When a new data arrives, a vector is generated to represent it at the MV2 step. In this paper, we transform each data into a set. If the original data type is already a set, for example, a set of ASCII strings excerpted from a malware file [11], we use the data as it is; otherwise, for example, a network packet of a byte sequence, usually less than 1,600 bytes at the transport layer, we first parse it into multiple words or tokens, and the collection becomes a set. If necessary, we apply a content-defined chunking (CDC) algorithm to transform any byte sequence into a set [26, 64][1]. In this paper, we denote the i^{th} data from the data stream as d_i, and its set as $CDC(d_i)$. When d_i is inherently a set, d_i is equal to $CDC(d_i)$.

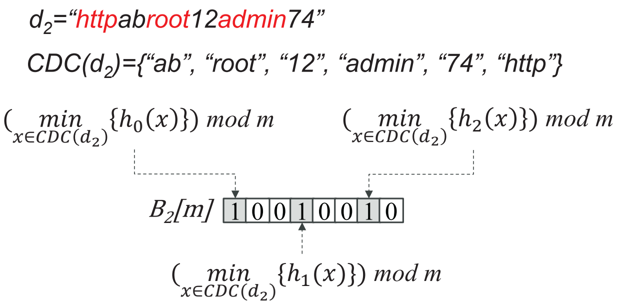

d2="httpabroot12admin74"
CDC(d2)={"ab", "root", "12", "admin", "74", "http"}
$(\min_{x\in CDC(d_2)}\{h_0(x)\})\ mod\ m$
$(\min_{x\in CDC(d_2)}\{h_2(x)\})\ mod\ m$
B2[m]
1 0 0 1 0 0 1 0
$(\min_{x\in CDC(d_2)}\{h_1(x)\})\ mod\ m$

Figure 3: Bitmap encoding of $d_i = d_2$ on minHashed virtual vector where $k = 3$ bits are turned on.

The advantage of MV2 is that each set-type data is encoded into a bitmap with only k-representative numbers that are computed by the minHash algorithm [14]. Broder invented minHash to measure the resemblance of documents as the Jaccard index [58], which are widely used in data analysis and internet search [15, 33, 34]. The Jaccard index measures the similarity between two sets as the ratio of the size of intersection to the size of union, which ranges from 0 to 1. We use the minHash with k-hash functions [14], and $h_j(\cdot)$ is the j^{th} hash function, $0 \le j \le k-1$. We denote the bitmap of size m for d_i as $B_i[m]$, initialized as '0', and the r^{th} bit as $B_i[r]$. For given d_i, we compute $h_j(x)$ for every $x \in CDC(d_i)$ and select the minimum value for the bitmap encoding of d_i and $h_j(\cdot)$. This repeats for every hash function; therefore k bits are turned into '1' for d_i if there is no hash collision. This encoding for d_i can be formally expressed as follows and the bitmap encoding is shown in Fig. 3:

$$B_i[(min_{x\in CDC(d_i)} h_j(x))\ mod\ m] := 1,\ 0 \le j \le k-1. \qquad (1)$$

We can compute the resemblance of d_i and d_j by comparing their bitmaps. When m is enough bigger than k, the Jaccard index between them, $j(d_i, d_j)$ can be estimated as follow [27]:

$$j(d_i, d_j) = \frac{|CDC(d_i) \cap CDC(d_j)|}{|CDC(d_i) \cup CDC(d_j)|} \approx \frac{\sum_{r=0}^{m-1}(B_i[r] \wedge B_j[r])}{\sum_{r=0}^{m-1}(B_i[r] \vee B_j[r])} \qquad (2)$$

where '$\vee$' and '$\wedge$' are bitwise OR and AND operators.

There are two advantages to encode each data into the MV2 vector. First, any data can be encoded with only k-positions in the bitmap irrespective of the original set size of the data, which requires only a small fixed size of memory. In this

[1] We use the AE chunking algorithm with the default window size of four, which generates chunks of the average length of 6.88(=4×1.72) bytes [64].

paper, we set the default values of m and k to 16,384(=2^{14}) and 64 respectively, which can be adjusted depending on the operational environment. Actually, we do not need to use a new bitmap for every data, which is explained in the next section. We can reuse one bitmap for all the stream data. Second, similarity between two data can be accurately estimated by comparing their bitmaps.

Big-group: For d_i, we define its *similar-group*, $sg(d_i)$, as the collection of d_i and d_j for $0 \leq j \leq i-1$ where $j(d_i, d_j)$ is larger than a predefined threshold of θ_J:

$$sg(d_i) = \{d_j | j(d_i, d_j) > \theta_J\} \cup \{d_i\}. \tag{3}$$

If the ratio of $|sg(d_i)|$ to i, a *big-group-ratio* of d_i, denoted as $b_r(d_i)$, is larger than a predefined threshold of θ_B, $sg(d_i)$ is said to be a *big-group* in terms of d_i. Therefore, if inequality (4) is true, we say that d_i is a member of a big-group:

$$b_r(d_i) = \frac{|sg(d_i)|}{i} > \theta_B. \tag{4}$$

If we keep all of the $B_j[m]$ bitmaps for $0 \leq j \leq i$, it is easy to check if $sg(d_i)$ is a big-group by pair-wise comparisons of d_i and d_j. However, its computation requires both the time and space complexities of $O(i)$ because all the previous data, or their $B_j[m]$ bitmaps at least, should be stored for $0 \leq j \leq i-1$. This may be practically impossible for streaming data. A better approach should be required, and we present jaccard-index grouping (JIG) in this paper.

3.2 JIG: Jaccard-Index Grouping

We focus on detecting a persistent zero-day attack that repeats to generate similar attack data during a certain monitoring period, for example, several minutes, hours, days, or even weeks. We assume that the monitoring period includes n data in total, enumerated as [d_0, ..., d_i, ..., d_{n-1}]. Each data can be processed only once when it arrives. We assume that d_i just arrives and its $B_i[m]$ is generated as explained previously.

In this chapter, we present JIG that determines whether $sg(d_i)$ is a big-group. If then, d_i is stored in a separate space for the next process of SG2. The most important role of JIG is to keep the summary of the data stream with a small fixed memory and a limited number of hash operations. This summary data structure enables us to estimate equation 4 although we do not know $sg(d_i)$.

To the best of our knowledge, JIG is the first streaming algorithm that can identify groups of similar data with a small fixed memory and a constant number of hash operations, or both time and space complexities of $O(1)$. The JIG module consists of two steps, the similarity summarization and the membership-checking, which are explained in detail.

3.2.1 Similarity Summarization

The idea of JIG is to accumulate all of the $B_j[m]$ bitmaps for $0 \leq j \leq i-1$ into a counter array of size m. We denote this counter array as $MV[m]$ and $MV[r]$ is the value of its r^{th} counter, which can be formally defined as follows:

$$MV[r] = \sum_{j=0}^{i} B_j[r],\ 0 \leq r \leq m-1. \tag{5}$$

The role of $MV[m]$ is to summarize the data stream after each data is represented as MV2. For d_i, every $d_j \in sg(d_i)$, $0 \leq j \leq i-1$, shares at least $\frac{\theta_J \times (|CDC(d_i)| + |CDC(d_j)|}{1+\theta_J}$ elements in common with d_i. Therefore, those counters that are related with $sg(d_i)$ must have grown greater than others especially when $j(d_i, d_j)$ and $b_r(d_i)$ are close to one.

Big-counter: For d_i, we define those counters from $MV[m]$ to be *big-counters* that are greater than threshold $\theta_{C,i}$. The other counters are *non-big-group* counters. We explain later how $\theta_{C,i}$ is computed from d_i and $MV[m]$ that should serve as a boundary line between big-counters and non-big-counters.

3.2.2 Membership Checking

The main goal of JIG is to determine whether $sg(d_i)$ is a big-group, or the membership checking of d_i to any big-group. Because this should be repeated for every incoming data, both the time and space complexities should be small.

When d_i arrives, we generate $B_i[m]$ with k-bits turned on, and $MV[m]$ is updated with $B_i[m]$ by equation 5. We know the k-indexes of '1' bits from $B_i[m]$, $\{(min_{x \in CDC(d_i)} h_j(x)) \ mod \ m \mid 0 \leq j \leq k-1\}$. The k-counters of d_i are denoted as $kc(d_i)$, which becomes as follows:

$$kc(d_i) = \{MV[(min_{x \in CDC(d_i)} h_j(x)) \ mod \ m] \mid 0 \leq j \leq k-1\}. \tag{6}$$

We select the big-counters from $kc(d_i)$ as follows, denoted as $bc(d_i)$:

$$bc(d_i) = \{x | x \in kc(d_i),\ x > \theta_{C,i}\}. \tag{7}$$

The idea of JIG is simple; if d_i is a member of a big-group and the members of the same big-group have appeared enough, many counters from $kc(d_i)$ would be much larger than other randomly-chosen counters from $MV[m]$. Especially, if θ_J and θ_B are close to one, most counters from $kc(d_i)$ are much bigger than others, or big-counters.

After $MV[m]$ is updated with d_i, we compute the ratio of the number of big-counters from $kc(d_i)$ to k, denoted as $b_k(d_i)$. If this ratio is larger than threshold θ_J, we assume that d_i is a member of a big-group as follows:

$$b_k(d_i) = \frac{|bc(d_i)|}{k} > \theta_J. \tag{8}$$

We emphasize that equation 8 practically plays the role of equation 4. Because we do not save the previous stream data, we cannot directly use equation 4. On the contrary,

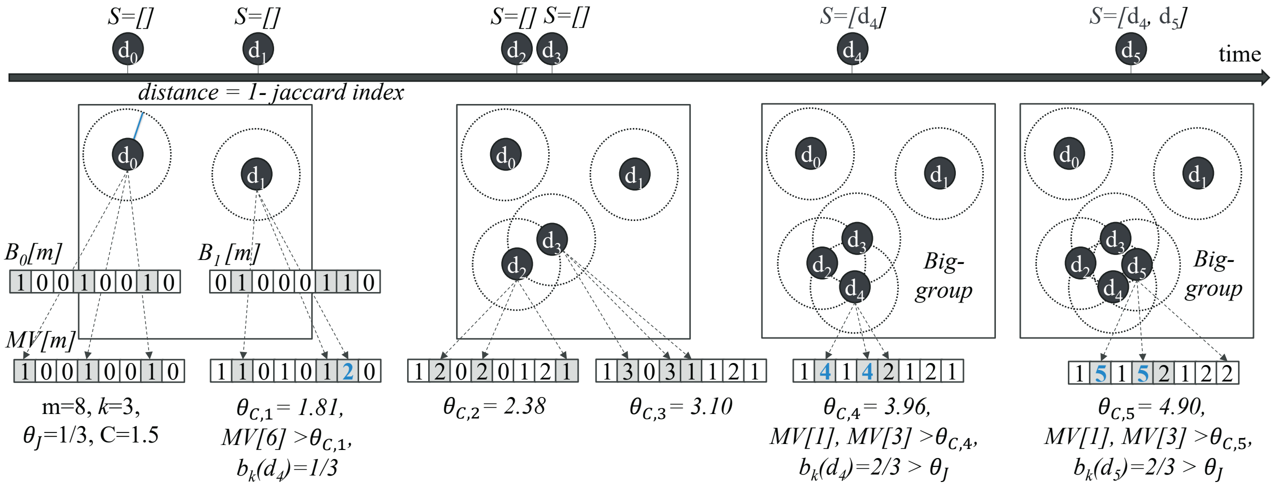


Figure 4: Big-group identification of Jaccard-index grouping (JIG) from data streams.

$MV[m]$ keeps the summary of the previous data and their big-group relations, and therefore we can use equation 8 with the observed statistics from $MV[m]$ and $B_i[m]$.

Fig. 4 shows how JIG works on a data stream where $m = 8$, $k = 3$, and $\theta_J = 1/3$. As time goes, six stream data arrive from d_0 to d_5 where $d_2 \sim d_5$ make a big-group with $j(d_5,d_2) = (d_5,d_3) = (d_5,d_4) = 2/3$. A big-group is identified twice when d_4 and d_5 arrive respectively; two counters of $kc(d_4)$ and $kc(d_5)$, $MV[1]$ and $MV[3]$, become greater than $\theta_{C,4}$ and $\theta_{C,5}$ respectively. We explain how $\theta_{C,i}$ is computed in the next section. Because $b_k(d_4)$ and $b_k(d_5)$ are $2/3$, larger than θ_J, satisfying inequality 8, we conclude that d_4 and d_5 are a member of a big-group. We store members of any big-group into a special space, S, for the next step of SG2.

3.2.3 IORA: Iterative Outlier Removal Algorithm

When d_i arrives, distinguishing between big-counters and non-big-counters is essential for the identification of a big-group. In this paper, we present a heuristic algorithm, *Iterative Outlier Removal Algorithm (IORA)*, to find $bc(d_i)$ for d_i. We observe that there are only a small number of big-counters because the number of big-groups is small, even zero for a usual monitoring period, and m is larger than k by orders of magnitude. In IORA, there is an outlier list (OL) to keep big-counters, which is initialized empty. The list is iteratively enlarged with new big-counters.

The main idea of IORA is that the distribution of non-big-counters of $MV[m]$ follows a normal distribution while big-counters are deviated from that distribution. For simplicity, we assume that there is only one big-group of the big-group ratio of θ_B. When we consider a data that is not included by the big-group, $MV[m]$ is updated $i \times (1 - \theta_B) \times k$ times at random indexes. On the contrary, the k-counters are persistently updated by the big-group members. Because we assume that a majority of the data are not a member of a big-group and m is greater than k by orders of magnitude, most counters of $MV[m]$ remain non-big-counters. This implies that most $MV[r]$ would follow the binomial distribution of $B(n \times (1 - \theta_B) \times k, 1/m)$, which can be approximated as the following normal distribution:

$$N(n \times (1-\theta_B) \times k \times 1/m, n \times (1-\theta_B) \times k \times 1/m \times (1-1/m)) \tag{9}$$

On the contrary, only k-counters of $MV[m]$ would become big-counters and their values are much greater than those of the non-big-counters, depending on θ_B and θ_J.

We present a simple and iterative outlier removal strategy for IORA, which consists of two iterative steps. When d_i arrives, two steps of IORA are iteratively repeated with OL expanded; in the first step, we compute the average and variance of $MV[m]$ except the counters included in OL as follows:

$$\begin{aligned} \mu_i &= \frac{\sum_{r=0,\ mv[r] \notin OL}^{m-1} MV[r]}{m - |OL|} \\ \sigma_i^2 &= \frac{\sum_{r=0,\ mv[r] \notin OL}^{m-1} (MV[r])^2 - \mu_i^2}{m - |OL|}. \end{aligned} \tag{10}$$

In the second step, we define the outlier by using μ and σ. We consider $MV[r]$ as an outlier if $MV[r] > \mu_i + c \times \sigma_i$; c is a tuning parameter, and we set $c = 6$ as a default value in this paper, motivated by the six sigma of a standard deviation [60]. We set $\theta_{C,i}$ to $\mu_i + c \times \sigma_i$. If a new outlier is found, we insert it into OL and repeat the first and second steps; otherwise, we finish IORA, and the counters in OL become big-counters.

3.3 SG2: Signature-Group Generation

The final goal of GIPS is to automatically generate a group of signatures. Although researchers have presented some methods for packets [9, 29, 32, 39, 48], these methods work for only those datasets that consist of very similar contents with each other, i.e., very high values of θ_J and θ_B. If we use them for datasets that include not only similar contents but also dissimilar contents, no meaningful results are obtained. For example, the SOTA method of [9], called *triple-heavy-hitter (THH)*, extracts most frequently-occurred substrings as signatures; however, these signatures often raise only false-positives without a human intervention, which will be shown in our experiments in the next section.

On the contrary, GIPS first identifies a big-group of similar contents with JIG. Therefore, we only need to apply any SOTA method of signature generation to each of the identified big-group. Then, a signature-group for each big-group can be automatically generated. In this paper, we choose THH because this is based on a streaming algorithm. However, any other method can be used instead.

The THH method includes three heavy-hitter modules that can find the most frequently-occurred substrings from a dataset, here a big-group. Because JIG has already saved big-group members into S as shown in Fig. 4, we only need to apply THH to each of the big groups. The first heavy-hitter module of THH keeps how many times a basic n-gram appears while the second module counts each of the frequent variable-sized substrings. They work for data streams with the time and space complexities of $O(1)$. Readers who want to know more about THH are referred to [9]. The THH method is basically designed for string data. Therefore, if d_i is of the byte-sequence type, for example, packets or texts, we could apply THH directly to S. If d_i is originally of the set type, we just select those elements that appear the most from S.

Fig. 5 shows the SG2 step where S already includes two big-groups of $[d_4, d_5, d_7]$ and $[d_8, d_9]$. These big-groups were identified by the previous JIG step. In this example, THH excerpts a group-signature of $\{“http”, “root”, “admin”\}$ from the first big-group and $\{“ST”, “UVW”\}$ from the second one.

Whenever we find big-group members during the JIG process, we put them into a separate space, S, as shown in Fig. 4. If a big-group exists in the data stream, S would include a number of member data. In this sense, JIG collects some members of the big-group that are similar to each other. The size of S does not need to be large; when S includes enough data, we can immediately run SG2.

In general, S includes only members from the same big-group. However, more than one big-group can exist as in Fig. 5. In this case, we may apply a clustering algorithm first to S, which easily separates different big-groups. Because S does not include a large number of data and the distance among different big-groups tend to be large enough, the clustering runs fast with only small computing resources. In this paper, we use DBSCAN because this does not require a hyper-parameter about the number of clusters. This property is very useful for a practical tool like GIPS. The default parameters for DBSCAN are ε=0.4 and min_samples=5 [2]. We use up to the largest five clusters after SG2 finishes.

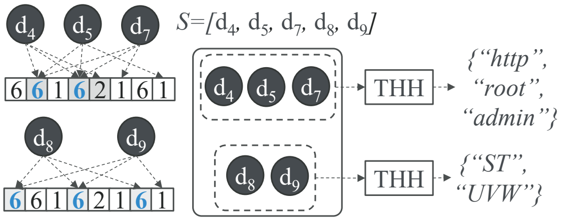


Figure 5: Signature-group generation of GIPS.

3.4 AWL: Automatic WhiteListing

We argue that GIPS can enhance the detection of zero-day attacks because GIPS automatically identifies big-groups and then extract signatures from the big-groups. However, a big-group does not always mean an attack signal. Therefore, we present an additional automatic filtering step that refines the signatures one more time. This is a big improvement, compared with the previous work [9, 29], because human interventions and heuristics are less required in GIPS.

We call this additional automatic filtering step as Automatic WhiteListing (AWL), which utilizes both GIPS and THH together. This THH is called a *global THH* to distinguish it from the THH of SG2. We apply GIPS and the global THH to the same data stream as follows: when d_i arrives, JIG determines if this is a member of a big-group. If then, d_i is saved into S; otherwise, the global THH processes d_i. This enables the global THH to keep frequently-occurred signatures that do not frequently appear in big-groups. Finally, after GIPS generates signature-groups from the big-groups at the SG2 step, we subtract the global THH signatures from each of the GIPS signature-groups by considering them as sets. This AWL step removes those signatures that may cause false-positives without any human intervention.

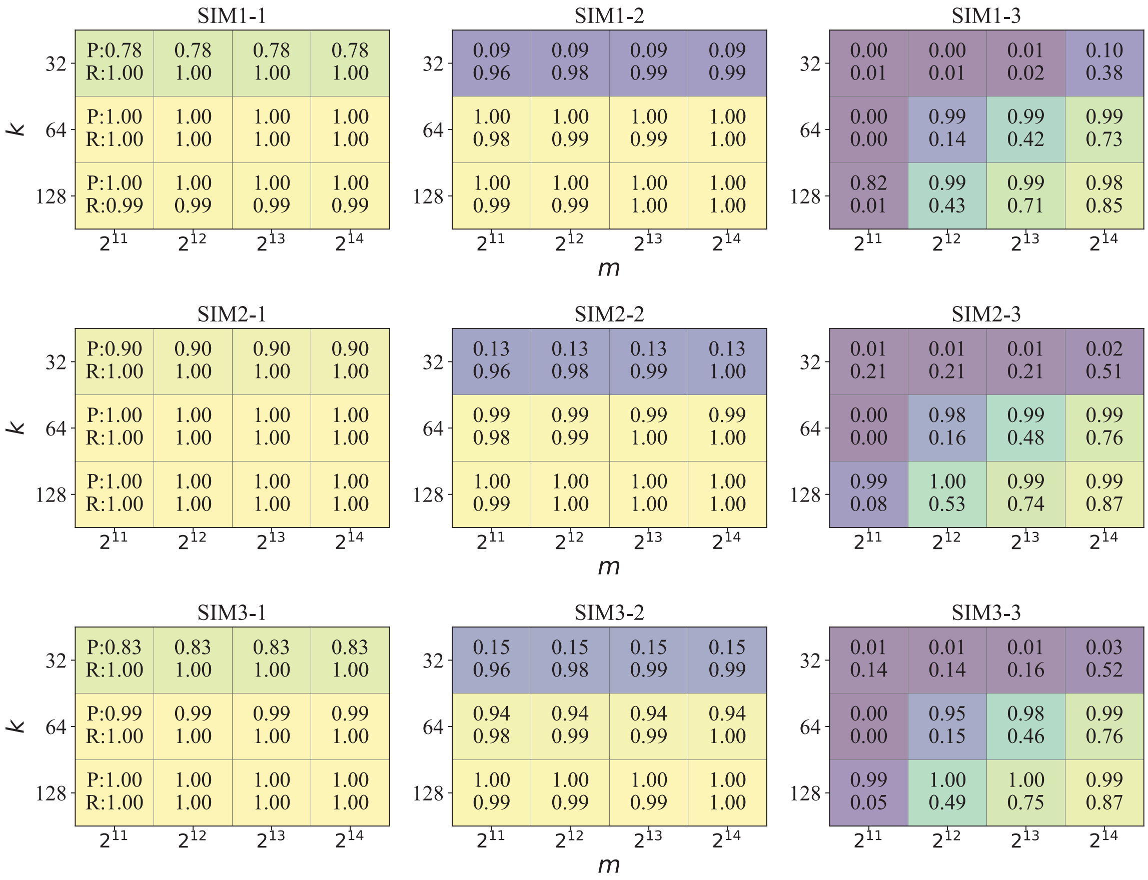
4 Experiments

We evaluate GIPS through extensive experiments with four different datasets, one private and three public datasets. We confirm that GIPS enhances the detection accuracy of persistent zero-day attacks several times higher than previous schemes of THH [9], Earlybird [48], and Polygraph [39].

4.1 Experimental Setup and Dataset

We use four different datasets, a simulated dataset [7], a dataset of malicious and benign internet-of-things (IoT) packets [42], a dataset of intrusion detection evaluation dataset [47], and a dataset of suspicious packets captured by an internet service provider (ISP). The first three datasets are public on the Internet. Table 3 summarizes the datasets, and its extended version is in Appendix A.

Simulated dataset (SIM): We first generate simulated datasets where each data is a random string of length 600



SIM1-1

k	2^{11}	2^{12}	2^{13}	2^{14}
32	P:0.78 R:1.00	0.78 1.00	0.78 1.00	0.78 1.00
64	P:1.00 R:1.00	1.00 1.00	1.00 1.00	1.00 1.00
128	P:1.00 R:0.99	1.00 0.99	1.00 0.99	1.00 0.99

SIM1-2

k	2^{11}	2^{12}	2^{13}	2^{14}
32	0.09 0.96	0.09 0.98	0.09 0.99	0.09 0.99
64	1.00 0.98	1.00 0.99	1.00 0.99	1.00 1.00
128	1.00 0.99	1.00 0.99	1.00 1.00	1.00 1.00

SIM1-3

k	2^{11}	2^{12}	2^{13}	2^{14}
32	0.00 0.01	0.00 0.01	0.01 0.02	0.10 0.38
64	0.00 0.00	0.99 0.14	0.99 0.42	0.99 0.73
128	0.82 0.01	0.99 0.43	0.99 0.71	0.98 0.85

SIM2-1

k	2^{11}	2^{12}	2^{13}	2^{14}
32	P:0.90 R:1.00	0.90 1.00	0.90 1.00	0.90 1.00
64	P:1.00 R:1.00	1.00 1.00	1.00 1.00	1.00 1.00
128	P:1.00 R:1.00	1.00 1.00	1.00 1.00	1.00 1.00

SIM2-2

k	2^{11}	2^{12}	2^{13}	2^{14}
32	0.13 0.96	0.13 0.98	0.13 0.99	0.13 1.00
64	0.99 0.98	0.99 0.99	0.99 1.00	0.99 1.00
128	1.00 0.99	1.00 1.00	1.00 1.00	1.00 1.00

SIM2-3

k	2^{11}	2^{12}	2^{13}	2^{14}
32	0.01 0.21	0.01 0.21	0.01 0.21	0.02 0.51
64	0.00 0.00	0.98 0.16	0.99 0.48	0.99 0.76
128	0.99 0.08	1.00 0.53	0.99 0.74	0.99 0.87

SIM3-1

k	2^{11}	2^{12}	2^{13}	2^{14}
32	P:0.83 R:1.00	0.83 1.00	0.83 1.00	0.83 1.00
64	P:0.99 R:1.00	0.99 1.00	0.99 1.00	0.99 1.00
128	P:1.00 R:1.00	1.00 1.00	1.00 1.00	1.00 1.00

SIM3-2

k	2^{11}	2^{12}	2^{13}	2^{14}
32	0.15 0.96	0.15 0.98	0.15 0.99	0.15 0.99
64	0.94 0.98	0.94 0.99	0.94 0.99	0.94 1.00
128	1.00 0.99	1.00 0.99	1.00 0.99	1.00 1.00

SIM3-3

k	2^{11}	2^{12}	2^{13}	2^{14}
32	0.01 0.14	0.01 0.14	0.01 0.16	0.03 0.52
64	0.00 0.00	0.95 0.15	0.98 0.46	0.99 0.76
128	0.99 0.05	1.00 0.49	1.00 0.75	0.99 0.87

Figure 6: Precision (P) and Recall (R) of JIG for SIM datasets with different big-group ratios, k, and m.

bytes, and 1,000,000 data are generated in total. Some of the data are deliberately generated to include the same substring of length 450 bytes in common, forming a big-group to play the role of a zero-day attack, which is the *Attack* column in Table 3. The other data, the *Normal* column, include a short substring of length 50 bytes in common that would cause the most frequently occurred substring over the whole dataset.

There are nine different datasets for SIM, three with one big-group (SIM1), three with two big-groups (SIM2), and three with three big-groups (SIM3). Each subgroup has the ratio of the attack data of a big-group to the total data, configured with 0.1, 0.01, and 0.001 respectively, as shown from SIM1-1 to SIM3-3 in Table 3.

Because these SIM datasets do not reflect the real-world data distribution, we use them only to measure the big-group identification of GIPS. The data are labeled perfectly during the generation time, which enables us to exactly measure the metrics of *precision* and *recall*; the first is the ratio of the number of true-positives to the sum of the true-positives and false-positives while the second is the ratio of the number of true-positives to the sum of true-positives and false-negatives [59]. We also use F1-Score that is a harmonic mean of precision and recall. The simulated datasets can be download from [7].

IoT dataset (IoT): A labeled dataset with benign and malicious IoT network traffic is open to anyone and can be downloaded from the Internet [42], called the IoT23 dataset. This open dataset is provided with 20 different datasets that include not only malicious packets but also benign packets. The datasets are related with Kenjiro, IRCBot, and Mirai bot attacks. Each dataset is saved as a distinct file; in this paper, we selected only those files larger than 1 GB in size. Only eight files were selected, denoted as IoT1∼8 as in Table 3. We do not use empty packets.

The label of IoT23 is assigned per flow, either attack or normal [2]. The numbers of attack or benign packets for IoT1∼8 are summarized in Tables 3 and 8. In this paper, a packet is also assigned a label; if a packet belongs to an attack flow, its label becomes attack; otherwise, the packet label becomes normal. If any packet from an attack flow is detected, we consider the attack is detected, or a true-positive; otherwise, we consider a false-negative occurs. If any packet from a normal flow is detected, we consider a false-positive occurs; otherwise, we consider a true-negative occurs.

[2] Various attack labels such as DDoS, port scan, C&C, etc. [42] are simply considered as *attacks* in this paper.

Table 3: Summary of Experimental Datasets

No.	Dataset	Total (A)	Normal (B)	Attack (C)	Ratio (C/A)
1	SIM1-1	1,000,000	900,000	100,000	0.1000
2	SIM1-2	1,000,000	990,000	10,000	0.0100
3	SIM1-3	1,000,000	999,000	1,000	0.0010
4	SIM2-1	1,000,000	800,000	200,000	0.2000
5	SIM2-2	1,000,000	980,000	20,000	0.0200
6	SIM2-3	1,000,000	998,000	2,000	0.0020
7	SIM3-1	1,000,000	700,000	300,000	0.3000
8	SIM3-2	1,000,000	970,000	30,000	0.0300
9	SIM3-3	1,000,000	997,000	3,000	0.0030
10	IoT1	54,716	54,699	17	0.0003
11	IoT2	4,686	4,658	28	0.0060
12	IoT3	55,412	47,578	7,834	0.0141
13	IoT4	14,845,292	72,776	14,772,516	0.9951
14	IoT5	1,307,003	1,848	1,305,155	0.9986
15	IoT6	10,122	8,938	1,184	0.1170
16	IoT7	11,925	9,485	2,440	0.2046
17	IoT8	4,644	4,554	90	0.0194
18	IDS1	1,302,148	1,294,939	7,209	0.0055
19	IDS2	119,919	118,095	1,824	0.0152
20	IDS3	23,229	23,217	12	0.0005
21	IDS4	3,699,243	3,671,387	27,856	0.0075
22	IDS5	579,004	531,716	47,288	0.0817
23	IDS6	880,347	850,803	29,544	0.0336
24	IDS7	472,750	377,399	95,351	0.2017
25	IDS8	861,441	861,286	155	0.0002
26	ISP1	50,416	39,182	11,234	0.2228
27	ISP2	9,327	1,863	7,464	0.8002
28	ISP3	18,180	55	18,125	0.9970

IDS dataset (IDS): We use the CICIDS2017 dataset [47], a public dataset for intrusion detection evaluation, which includes benign/attack packets and flows collected for 5 days. There are 14 attack types in the dataset; we selected 8 attack types of 1) *Web Attack - Brute Force (bruteforce)*, 2) *Web Attack - XSS*, 3) *Web Attack - SQL injection*, 4) *FTP-patator*, 5) *SSH-patator*, 6) *Infiltration*, 7) *DDoS*, and 8) *Port Scan* for our experiments because they include packets with meaningful application payloads [47]. We denote them as IDS1∼8; the time period while each attack occurs is available from [18]. For example, we use all packets for the *bruteforce* attack from 9:20 to 10:00 on July 6, 2017 in the dataset where both normal and attack packets exist.

Although the CICIDS2017 dataset is widely used, there are two critical issues [20]; first, some flows are incorrectly split, and the correctly labeled flows are available from [5]. In this paper, we use the corrected version for experiments. Second, different data types have different numbers of data, or imbalanced distribution. Therefore, different attack types should not be aggregated when the accuracy-related performance is measured. In this paper, we separately evaluate each attack type of IDS1∼8 for the right experiments.

ISP dataset (ISP): This private dataset was provided by an ISP in South Korea for only academic purposes. The dataset includes three different subsets, denoted as ISP1∼3 in Table 3. Each dataset was carefully reviewed by security experts, and each packet is manually labeled as either attack or normal. Each of ISP1∼3 was collected with packets during a short period of time when the network-based intrusion prevention system observed suspicious activities. Therefore, the attack ratio is higher than that of other datasets as shown in Table 3.

Table 4: Precision and Recall for IoT datasets.

	GIPS		THH		Earlybird		Polygraph	
	Pre.	Rec.	Pre.	Rec.	Pre.	Rec.	Pre.	Rec.
IoT1	0.14	0.50	0.00	0.00	0.00	0.00	0.00	0.00
IoT2	0.06	1.00	0.01	1.00	0.22	1.00	0.00	0.00
IoT3	0.99	0.40	0.40	0.98	0.99	0.40	0.99	0.40
IoT4	1.00	1.00	1.00	1.00	0.00	0.00	n.a.	n.a.
IoT5	1.00	0.15	1.00	0.08	0.00	0.00	n.a.	n.a.
IoT6	1.00	0.82	0.06	0.94	1.00	0.18	0.00	0.00
IoT7	1.00	0.99	0.87	0.99	0.00	0.00	1.00	0.99
IoT8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Rival schemes: We compare GIPS with three rival schemes of THH [9], Earlybird [48], and Polygraph [39], against each of the datasets from Table 3. All rival schemes were designed to extract common string signatures from a given dataset. We do not include machine learning schemes because they cannot extract signatures. Both THH and Earlybird are based on streaming algorithms; however, Polygraph relies on suffix-tree, which requires more processing power and memory space than GIPS by orders of magnitude. Although Earlybird is based on a compact sketch algorithm and a fast hash table, the number of keys may significantly increase.

Parameter configuration: The parameters of the rival schemes are configured the same as their papers, THH [9], Earlybird [48], and Polygraph [39]. We changed only a few parameters to obtain optimal performance; 1) THH: we set the minimum n-gram size to 4 instead of 8 to catch even short strings as well as long ones for signatures [9]; otherwise, THH could not identify signatures. The number of signatures for attack detection is an important parameter for THH. When the number increases, more attack packets can be detected, but false-positives also increase. In this paper, we use the number of signatures for THH when the biggest F1 score is obtained. 2) Earlybird: the dispersion threshold is set to 5 instead of 30 to generate signatures even when the number of distinct source or destination IP addresses related with signatures is smaller than or equal to 5 [48]; otherwise, Earlybird could not identify signatures. 3) Polygraph: we tested it many times with a different K, a repetitive threshold, from 100 to 1,000, and then we selected the longest substring as a signature. Because Polygraph becomes slow and consumes lots of memory with a large dataset, we used this simple selection scheme that was recommended by the paper [39]. Finally, we use the default parameters for GIPS as $k = 64$, $m = 2^{14}$, $\varepsilon = 0.4$, and the number of largest clusters to be five.

All the software of GIPS and rival schemes were implemented in Python 3.10 and Scikit-learn. All experiments were conducted on the same machine that has a 3.0 GHz 13th

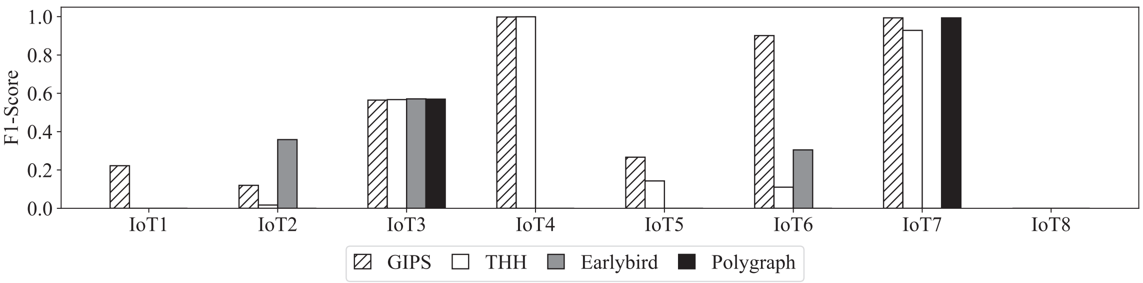


Figure 7: F1-Score of GIPS, THH [9], Earlybird [48], and Polygraph [39] for IoT datasets.

Gen Intel(R) Core(TM) i9-13900K CPU, 128 GB RAM, and Windows 11 Enterprise.

4.2 Experimental Results for SIM Dataset

We use the nine simulated datasets, from SIM1-1 to SIM3-3, for the first set of experiments. Fig. 6 shows the experimental results as heatmaps to measure the big-group identification of JIG. In general, as k, m, and $b_r(d_n)$ become larger, both precision and recall become higher. We confirm that the overall performance of JIG is almost perfect unless k and m are too small. The experimental results show the excellent performance of JIG. However, if k is configured too large, the minHash computation would take more processing time.

4.3 Experimental Results for IoT Dataset

We use the open IoT23 datasets [42] for the second set of experiments. Because we do not know the ground-truth for big-group identification, we compare GIPS and its rival schemes in terms of precision, recall, and F1-score for attack detection.

For a given scheme, each experiment consists of two steps; we first extract attack signatures with the scheme, and then detect attack data with the signatures. Therefore, for each of the IoT datasets, attack signatures are generated by each scheme respectively, and then the same dataset is searched with the signatures to check if attack data are accurately detected.

Fig. 7 and Table 4 compare GIPS and three rival schemes where GIPS overwhelms others. We explain interesting results; first, both the precision and recall of GIPS are significantly higher than others except IoT2 and IoT8. When GIPS works better than others, a meaningful group-signature is often generated. For example, GIPS excerpted “PONG", “root", “chmod", etc. as signatures from IoT6. It is interesting that GIPS and Polygraph successfully excerpted “arch armv7\n" from IoT7, but this string is too small to be caught by Earlybird. For IoT8, static and repeated substrings are hardly found, which prevents all schemes from detecting attacks.

It is interesting that a rival scheme works as good as GIPS for some cases; first, THH works as good as GIPS for IoT4. After analyzing the IoT4 dataset, we find that the ratio of the attack data is abnormally as high as 0.9951, shown in Table 3. This means that almost all the data are attack, and THH works as good as GIPS. Second, Earlybird works better than GIPS for IoT2. We find that a long string repeatedly appears in IoT2, which can be caught by GIPS, THH, and Earlybird. However, some shorter strings from normal packets also appear frequently, which may degrade the performance of THH. For GIPS, AWL can handle this case.

Actually, the abnormal condition of the extreme high ratio of the attack data was assumed by all rival schemes of THH, Earlybird, and Polygraph. When the attack ratio becomes small, for example less than 0.1, the performance of them often becomes lower than that of GIPS. This exactly matches our argument that GIPS is able to not only generate signatures but also find big-groups. Although the attack ratio seems to affect the performance of GIPS and its rival schemes, this is not the only reason for good or bad performance. For example, some datasets show strong repetitiveness while others do not. For repetitiveness, some datasets include a string of a long static sequence while others include a group of short strings.

Next, we show that GIPS can work stably with a different θ_J and ε, which are shown in Figs. 8 and 9, respectively.

4.4 Experimental Results for IDS Dataset

For the third set of experiments, we use the open dataset of CICIDS2017 [47]. Fig. 10 and Table 5 compare GIPS and three rival schemes where GIPS overwhelms others. When GIPS works better than others, a meaningful group-signature is often generated. For example, GIPS excerpted “/dv/vulnerabilities" and “Cookie: security=low" as signatures from IDS2, and “GET / HTTP/1.0\r\n\r\n\r\n" from IDS7.

However, GIPS does not work at all for IDS3, IDS5, and IDS8; We find that IDS3 includes a small number of attack packets and IDS5 includes SSH packets that would be encrypted. In both cases, GIPS cannot perform well. Polygraph and Earlybird do not work as good as GIPS. For IDS8, static and repeated substrings are hardly found, which prevents all schemes from detecting attacks.

Polygraph outperforms GIPS for IDS2. The dataset analysis reveals that Polygraph finds a long string that coincidentally appears in attack packets; a real attack string was not identified by Polygraph, but GIPS identified this correctly.

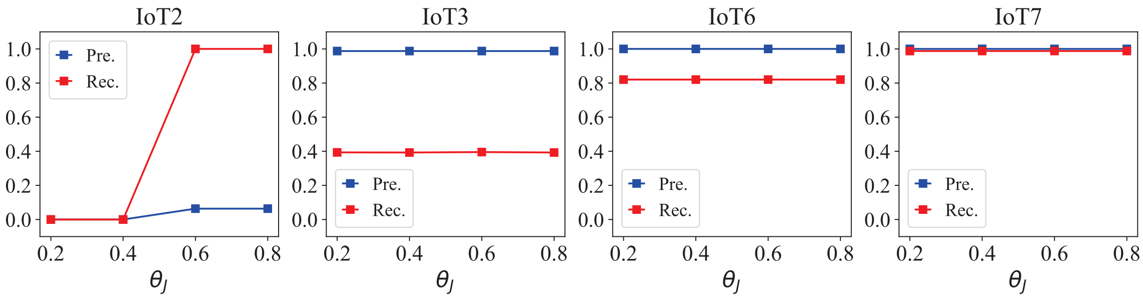


Figure 8: GIPS for IoT datasets with different θ_J.

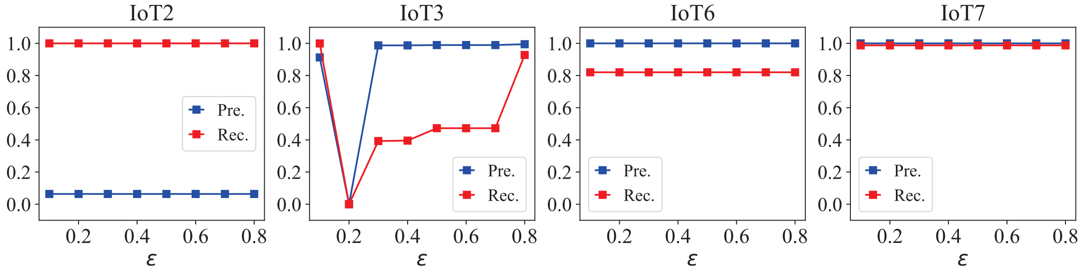


Figure 9: GIPS for IoT datasets with different ε.

Although the recall of THH is close to 1 for IDS2, IDS3, IDS5, and IDS6, its precision is close to 0. This means that THH just generates signatures that appear in most packets.

Table 5: Precision and Recall for IDS datasets.

	GIPS		THH		Earlybird		Polygraph	
	Pre.	Rec.	Pre.	Rec.	Pre.	Rec.	Pre.	Rec.
IDS1	0.99	0.99	0.00	0.77	n.a.	n.a.	n.a.	n.a.
IDS2	0.08	1.00	0.00	1.00	0.00	0.00	1.00	0.96
IDS3	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00
IDS4	0.71	1.00	0.00	0.00	n.a.	n.a.	n.a.	n.a.
IDS5	0.00	0.00	0.08	0.99	n.a.	n.a.	n.a.	n.a.
IDS6	1.00	0.40	0.00	1.00	n.a.	n.a.	n.a.	n.a.
IDS7	0.99	1.00	0.00	0.00	n.a.	n.a.	n.a.	n.a.
IDS8	0.00	0.00	0.00	0.26	n.a.	n.a.	n.a.	n.a.

4.5 Experimental Results for ISP Dataset

Because the ISP dataset was manually analyzed and labeled by security experts, we know the exact data label for attack vs normal. The experiments are performed in the same way as the IoT and IDS datasets.

Fig. 11 and Table 6 compare GIPS and three rival schemes where GIPS overwhelms others. An interesting observation is that the performance of THH is as good as that of SG2 in ISP3. We find that the ratio of the attack data are abnormally high, 0.9970, as shown in Table 3.

Table 6: Precision and Recall for ISP datasets.

	GIPS		THH		Earlybird		Polygraph	
	Pre.	Rec.	Pre.	Rec.	Pre.	Rec.	Pre.	Rec.
ISP1	0.84	0.90	0.78	0.70	0.00	0.00	1.00	0.09
ISP2	1.00	1.00	1.00	1.00	0.88	0.67	1.00	0.15
ISP3	1.00	1.00	1.00	1.00	1.00	0.19	1.00	0.07

Analyzing the signatures generated by GIPS from the ISP datasets, we find that all of the three datasets really include attack evidences as shown in Table 7. For ISP1, GIPS identified two big-groups that include attack signatures respectively as shown in Table 7; in Spring Cloud Function versions 3.1.6, 3.2.2 and older unsupported versions, a user can provide a specially crafted routing-expression that may result in remote code execution and access to local resources [40]. The second signature is related with a remote code execution vulnerability of Huawei HG532 [37]. For the ISP2 dataset, GIPS again identified a big-group that is related with a CISCO switch vulnerability [36]. Finally, GIPS identifies a big-group related with DHDiscover reflection attacks [41].

Table 7: GIPS Signatures and Attacks for ISP Datasets

Dataset	Signature	Related Attack
ISP1	*getRuntime().exec(”touch /tmp/test.txt”), /bin/busybox*	CVE-2022-22963 [40] CVE-2017-17215 [37]
ISP2	0002736c0000ff	CVE-2010-1574 [36]
ISP3	\x00\x00\x00*DHIP*\x00... ”*Port*” : 37777,”*RemoteVideo*...	Reflection attack [41]

4.6 Discussion

Dataset privacy: Four different types of datasets are used for experiments. Three public datasets of SIM, IoT, and IDS have no privacy issues. We emphasize that even the private dataset of ISP also has no privacy issue; the ISP dataset consists of a limited number of packets from its network-based intrusion prevention system during a short period of time when the system detected such suspicious activities as port scanning or DDoS attacks. No specific IP addresses were targeted or monitored. Above all, all IP addresses were completely de-

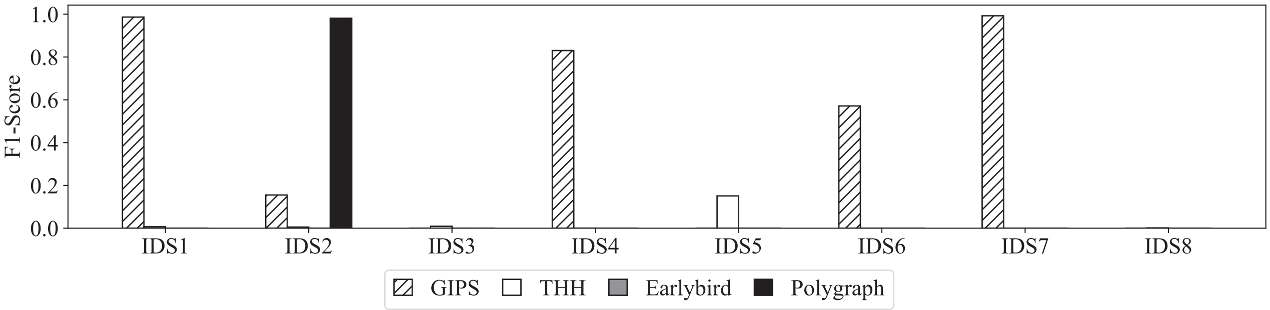


Figure 10: F1-Score of GIPS, THH [9], Earlybird [48], and Polygraph [39] for IDS datasets.

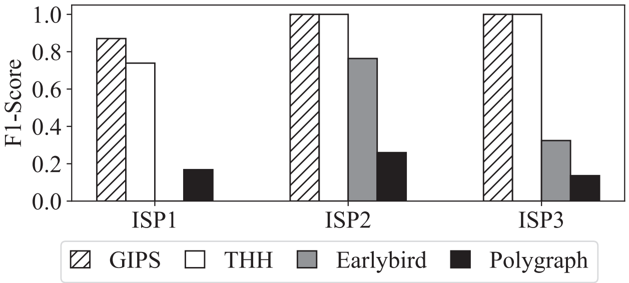


Figure 11: F1-Score of GIPS, THH [9], Earlybird [48], and Polygraph [39] for ISP datasets.

identified to prevent any privacy risks before the experiments were performed.

Computing resources: We design GIPS as a streaming algorithm, and therefore GIPS should work with less computing resources as THH does. Fig. 12 shows the memory space and processing time for GIPS, THH, Earlybird, and Polygraph, respectively. Because Polygraph was not designed as a streaming algorithm, Polygraph requires more computing resources than others. For example, Polygraph did not finish its process for the large datasets of IoT5, IDS1, and IDS5, while using up all the available memory space. We observe that GIPS requires more memory space than THH because GIPS stores identified big-group data in S.

Limitations: The advantage of GIPS is to identify big-groups first and then generate signature-groups as a streaming algorithm. However, GIPS has some limitations and we discuss them in details.

First, GIPS cannot work on encrypted data, which is clearly shown in the experimental result of IDS5 in Fig. 10. Actually, other rival schemes have the same limitation for any encrypted data. Because more network traffic is now encrypted [23], GIPS may need decryption boxes that can obtain plain packets from encrypted ones [21]. Then, GIPS can work properly again, but extra costs are required for the boxes. If GIPS is applied to server-side data such as Web Application Firewall (WAF) [8,54], Endpoint Detection and Response (EDR), non-encrypted data would be available to GIPS.

Second, GIPS can mitigate only those persistent zero-day attacks that have repetitive contents. For example, GIPS works almost perfect for the ISP datasets where suspicious packets were collected. If packets have been randomly captured from backbone lines, GIPS would have not generated useful signatures. However, we emphasize that previous schemes can catch a simple and long string only when the attack ratio is close to 1.0 [9,39,48]; simple worm attacks can only be mitigated. Actually, we show that the previous schemes could not effectively work for our experimental datasets, but GIPS can generate group-signatures against a various range of attack ratios from 0.001 to 0.997. Because security experts need to detect attacks with a solid ground, or a signature if any, they are willing to collect datasets for GIPS and to save time and cost by verifying the GIPS-provided information first instead of a large volume of raw data.

Third, GIPS and existing anomaly detection tools are complementary to each other; when GIPS cannot process encrypted data, anomaly detection tools may detect suspicious activities. For example, FAIL2BAN can detect repetitive login trials to a SSH daemon by scanning server-side log files [6]. This anomaly detection tool can also update firewall rules to reject suspicious IP addresses. Security practitioners want to use GIPS and FAIL2BAN for different purposes.

Fourth, GIPS cannot generate group-signatures in real time. There is a time delay from the collection of attack data to the generation of group-signatures. Generally, the signatures need to be verified by security experts, and the security devices are configured with the verified signatures. All these processes delay attack prevention. However, the time delay would increase by orders of magnitude if GIPS is not used.

5 Related work

Automatic Signature Generation. Automatic signature generation has been intensively studied since the Internet worm and DDoS attacks were popularly launched. Most studies assumed that collecting suspicious data and extracting signatures from homogeneous data of similar contents are separate tasks [9,24,35,39,48]. Although THH was designed to process network traffic mixed with normal and attack packets, human interventions are required to reduce false-positives [9].

Seminal work of the automatic signature generation for network packets found only a fixed-length signature [48]. Find-

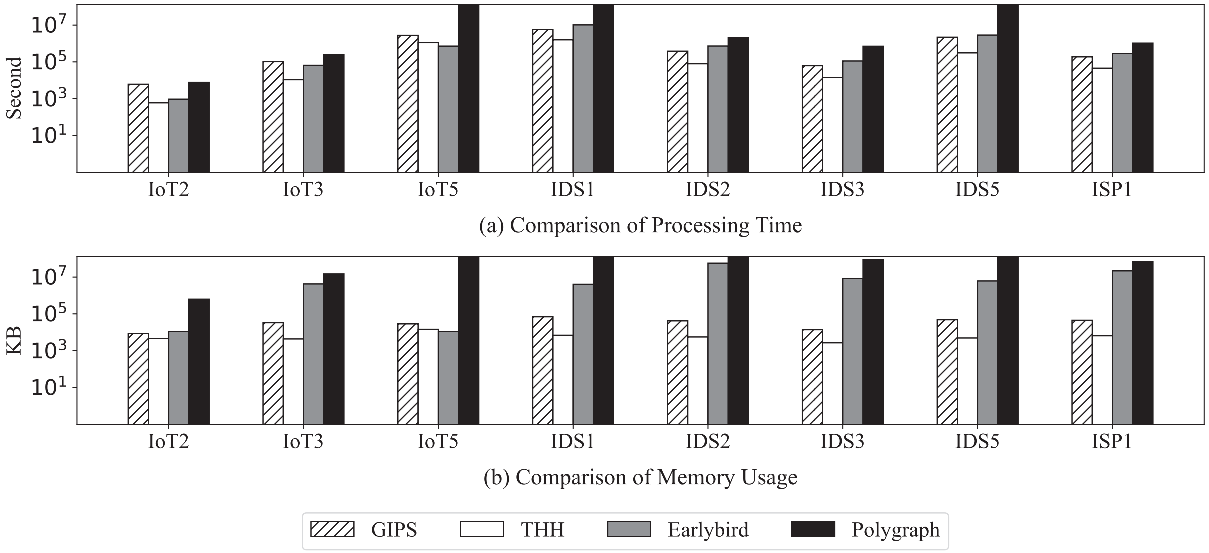


Figure 12: Comparison of processing time and memory usage for GIPS, THH [9], Earlybird [48], and Polygraph [39].

ing variable-length signatures is a more challenging problem; Kreibich and Crowcroft adopted suffix-tree algorithm and pattern matching to tackle the problem [31]. Although suffix-tree can process packets to find longest common substrings within packet payloads at high speed, the space complexity is not scalable for a large amount of packets. Kim and Karp adopted a content-based payload partitioning method [38] to extract variable-length common signatures [29], which is similar to the CDC algorithm [64] used in GIPS. However, their content-based method predefined the average signature size, which reduced the flexibility in generating common signatures. On the contrary, GIPS can produce a group of signatures where each signature can be variable-sized.

Afek et al. presented a tool for zero-day attack signature extraction [9], called THH, which extracts most frequently-occurred substrings from legitimate traffic and most frequently-occurred substrings from traffic mixed with legitimate and attack packets.The authors argued that set intersection would leave attack-related substrings, or signatures, which can be used to detect attack packets. However, human interventions are required to keep legitimate traffic datasets and distinguish between false-positive substrings and true-positive ones. The difference is that GIPS first identifies big-groups and then extract common substrings from each of the big-group. Actually, GIPS uses THH as a substring extraction module after a big-group is identified.

Reducing False-Positives. A false-positive problem, also known as alert fatigue, is a challenging problem for security monitoring during the last decades [13, 16, 22, 25, 44]. Writing precise detection rules for intrusion detection is a very difficult task, and practical systems prefer general rules that can cover a range of related threats instead of a specific exploit. However, general rules may cause a significant number of false-positives [43, 44, 49]. Recent threat detection products provide a tuning method to reduce false-positives [3,4], or SOCs have their own practice cycles for determining and fixing false-positives [30]. In this paper, we present the first signature-group generation method that minimizes false-positives while mitigating a zero-day attack. Signatures that may cause a large number of false-positives are automatically removed in GIPS.

Network Intrusion Detection and Prevention. A network-based IDS inspects packets to find cyber attacks and suspicious activities. The IDS has played a pivotal role in cybersecurity over the past decades because it can protect multiple servers and endpoint systems at gateways [43, 49, 53, 55, 63]. An IPS is an active protection system that not only identifies threats but also blocks or remediates the threat [1]. Both IDS and IPS are called IDPS in this paper.

Although a network IDPS is still one of the most important security systems, there are two serious challenges; first, as more network packets are encrypted, IDPSs cannot look up attack signatures. To tackle this problem, a decryption box can be deployed to obtain plain packets [21], or anomaly detection can also be used for encrypted packets [12, 57]. Second, too many false-positives are generated, resulting alert fatigue [16, 22, 25]. In this paper, we present GIPS that can find zero-day attacks from a range of datasets including IDPS alerts, network packets, emails, etc., with few false-positives.

6 Conclusion

In this paper, we presented a new zero-day attack detection and prevention method that first identifies big-groups of similar contents from data streams and then generate signature-groups for each of the big-group. To the best of our knowledge, this is the first streaming algorithm that can identify big-groups based on minHash and then automatically extract robust signatures, meaning few false-positives.

Acknowledgments

This work was partly supported by National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No.2023R1A2C1004653, Generative Security for Zero-Day Defense, 80%) and by Institute of Information & Communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (No.IITP-2020-0-01826, ICT Challenge and Advanced Network of Human Resources Development (HRD) (ICAN), 20%).

References

[1] Intrusion detection system (ids) vs intrusion prevention system (ips). https://www.checkpoint.com/cyber-hub/network-security/what-is-an-intrusion-detection-system-ids/ids-vs-ips/, 2021. [Online; accessed 5-Feb-2023].

[2] sklearn.cluster.dbscan. https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html, 2021. [Online; accessed 5-Feb-2023].

[3] Tuning false positives. https://www.ibm.com/docs/en/qsip/7.4?topic=performance-tuning-false-positives, 2021. [Online; accessed 5-Feb-2023].

[4] Tuning intrusion policies using rules. https://www.cisco.com/c/en/us/td/docs/security/firepower/70/configuration/guide/fpmc-config-guide-v70/tuning_intrusion_policies_using_rules.html, 2021. [Online; accessed 5-Feb-2023].

[5] Extended documentation of the wtmc paper. https://intrusion-detection.distrinet-research.be/WTMC2021/extended_doc.html, 2022. [Online; accessed 5-Feb-2023].

[6] Fail2ban. https://www.fail2ban.org/wiki/index.php/Main_Page, 2023. [Online; accessed 5-Feb-2023].

[7] Sim dataset. https://drive.google.com/file/d/1ppFEUSiEFCpIojoLgXzk7KrivJpRsTbu/view?usp=share_link, 2023. [Online; accessed 5-Feb-2023].

[8] Ssl visibility. https://techdocs.broadcom.com/us/en/symantec-security-software/web-and-network-security/ssl-visibility/5-4/sslv_overview.html, 2023. [Online; accessed 5-Feb-2023].

[9] Yehuda Afek, Anat Bremler-Barr, and Shir Landau Feibish. Zero-day signature extraction for high-volume attacks. *IEEE/ACM Transactions on Networking*, 27(2):691–706, 2019.

[10] B. Alahmadi, L. Axon, and I. Martinovic. 99% false positives: A qualitative study of soc analysts' perspectives on security alarms. In *USENIX Security Symposium*, 2022.

[11] Hyrum S Anderson and Phil Roth. Ember: an open dataset for training static pe malware machine learning models. *arXiv preprint arXiv:1804.04637*, 2018.

[12] E. Areström and N. Carlsson. Early online classification of encrypted traffic streams using multi-fractal features. In *IEEE INFOCOM 2019 - Workshops (INFOCOM WKSHPS)*, pages 84–89, 2019.

[13] Stefan Axelsson. The base-rate fallacy and its implications for the difficulty of intrusion detection. In *ACM CCS*, 1999.

[14] A. Z. Broder. On the resemblance and containment of documents. In *Proceedings of Compression and Complexity of SEQUENCES 1997 (Cat. No.97TB100171)*, pages 21–29, 1997.

[15] Andrei Z Broder, Moses Charikar, Alan M Frieze, and Michael Mitzenmacher. Min-wise independent permutations. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 327–336, 1998.

[16] Kevin Broughton. Automated incident response: Respond to every alert. https://swimlane.com/blog/automated-incident-response-respond-every-alert/, 2017. [Online; accessed 5-Feb-2023].

[17] Anna L Buczak and Erhan Guven. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications surveys & tutorials*, 18(2):1153–1176, 2015.

[18] CIC. Intrusion detection evaluation dataset (cic-ids2017). https://www.unb.ca/cic/datasets/ids-2017.html, 2017. [Online; accessed 5-Feb-2023].

[19] Alec F. Diallo and Paul Patras. Adaptive clustering-based malicious traffic classification at the network edge. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, pages 1–10, 2021.

[20] Gints Engelen, Vera Rimmer, and Wouter Joosen. Troubleshooting an intrusion detection dataset: the cicids2017 case study. In *2021 IEEE Security and Privacy Workshops (SPW)*, pages 7–12, 2021.

[21] J. Fan, C. Guan, K. Ren, Y. Cui, and C. Qiao. Spabox: Safeguarding privacy during deep packet inspection at a middlebox. *IEEE/ACM Transactions on Networking*, 25(6):3753–3766, 2017.

[22] FireEye. The numbers game: How many alerts is too many to handle? https://www.fireeye.com/offers/rpt-idc-numbers-game-special-report.html, 2014. [Online; accessed 5-Feb-2023].

[23] Chuanpu Fu, Qi Li, and Ke Xu. Detecting unknown encrypted malicious traffic in real time via flow interaction graph analysis. In *NDSS*, 2023.

[24] Kent Griffin, Scott Schneider, Xin Hu, and Tzi-cker Chiueh. Automatic generation of string signatures for malware detection. In *RAID*, volume 5758, pages 101–120. Springer, 2009.

[25] Wajih Ul Hassan, Shengjian Guo, Ding Li, Zhengzhang Chen, Kangkook Jee, Zhichun Li, and Adam Bates. Nodoze: Combatting threat alert fatigue with automated provenance triage. In *NDSS*, 2019.

[26] JunNyung Hur, Hahoon Jeon, Hyeon Gy Shon, Young Jae Kim, and MyungKeun Yoon. Finding critical files from a packet. *IEEE INFOCOM 2021*, 2021.

[27] Jiyong Jang, David Brumley, and Shobha Venkataraman. Bitshred: Feature hashing malware for scalable triage and semantic analysis. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, page 309–320, 2011.

[28] Jeffrey O Kephart. Automatic extraction of computer virus signatures. In *Proc. 4th Virus Bulletin International Conference, Abingdon, England, 1994*, pages 178–184, 1994.

[29] Hyang-Ah Kim and Brad Karp. Autograph: Toward automated, distributed worm signature detection. In *USENIX security symposium*, volume 286. San Diego, CA, 2004.

[30] Faris Bugra Kokulu, Ananta Soneji, Tiffany Bao, Yan Shoshitaishvili, Ziming Zhao, Adam Doupé, and Gail-Joon Ahn. Matched and mismatched socs: A qualitative study on security operations center issues. In *ACM CCS*, 2019.

[31] Christian Kreibich and Jon Crowcroft. Honeycomb: creating intrusion detection signatures using honeypots. *ACM SIGCOMM computer communication review*, 34(1):51–56, 2004.

[32] Suchul Lee, Sungho Kim, Sungil Lee, Jaehyuk Choi, Hanjun Yoon, Dohoon Lee, and Jun-Rak Lee. Largen: automatic signature generation for malwares using latent dirichlet allocation. *IEEE Transactions on Dependable and Secure Computing*, 15(5):771–783, 2016.

[33] Ping Li and Christian König. b-bit minwise hashing. In *Proceedings of the 19th international conference on World wide web*, pages 671–680, 2010.

[34] Ping Li, Art Owen, and Cun-Hui Zhang. One permutation hashing. *Advances in Neural Information Processing Systems*, 25, 2012.

[35] Zhichun Li, Manan Sanghi, Yan Chen, Ming-Yang Kao, and Brian Chavez. Hamsa: Fast signature generation for zero-day polymorphic worms with provable attack resilience. In *2006 IEEE Symposium on Security and Privacy (S&P'06)*, pages 15–pp. IEEE, 2006.

[36] MITRE. Cve-2010-1574. https://cve.mitre.org/cgi-bin/cvename.cgi?name=2010-1574, 2023. [Online; accessed 5-Feb-2023].

[37] MITRE. Cve-2017-17215. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-17215, 2023. [Online; accessed 5-Feb-2023].

[38] Athicha Muthitacharoen, Benjie Chen, and David Mazières. A low-bandwidth network file system. *SIGOPS Oper. Syst. Rev.*, 35(5):174–187, oct 2001.

[39] James Newsome, Brad Karp, and Dawn Song. Polygraph: Automatically generating signatures for polymorphic worms. In *2005 IEEE Symposium on Security and Privacy (S&P'05)*, pages 226–241. IEEE, 2005.

[40] NIST. Cve-2022-22963 detail. https://nvd.nist.gov/vuln/detail/CVE-2022-22963, 2023. [Online; accessed 5-Feb-2023].

[41] NSFOCUS. Dhdiscover reflection attacks can magnify nearly 200 times of the attack 1. https://nsfocusglobal.com/dhdiscover-reflection-attacks-can-magnify-nearly-200-times-of-the-attack-1/, 2023. [Online; accessed 5-Feb-2023].

[42] A Parmisano, Sebastian Garcia, and Maria Jose Erquiaga. A labeled dataset with malicious and benign iot network traffic. *Stratosphere Laboratory: Praha, Czech Republic*, 2020.

[43] Vern Paxson. Bro: A system for detecting network intruders in real-time. *Computer Networks*, 31:2435–2463, Dec. 1999.

[44] T. Pietraszek. Using adaptive alert classification to reduce false positives in intrusion detection. In *RAID*, 2004.

[45] Kevin Prince. 9 ways to eliminate siem false positives. https://www.connectwise.com/blog/cybersecurity/9-ways-to-eliminate-siem-false-positives/, 2021. [Online; accessed 5-Feb-2023].

[46] M Zubair Rafique and Juan Caballero. Firma: Malware clustering and network signature generation with mixed network behaviors. In *Research in Attacks, Intrusions, and Defenses: 16th International Symposium, RAID 2013, Rodney Bay, St. Lucia, October 23-25, 2013. Proceedings 16*, pages 144–163. Springer, 2013.

[47] Iman Sharafaldin, Arash Habibi Lashkari, and Ali Ghorbani. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. *ICISSP*, 1:108–116, 2018.

[48] Sumeet Singh, Cristian Estan, George Varghese, and Stefan Savage. Automated worm fingerprinting. In *OSDI*, volume 4, pages 4–4, 2004.

[49] Snort. Snort - network intrusion detection & prevention system. https://www.snort.org, 2021. [Online; accessed 5-Feb-2023].

[50] Robin Sommer and Vern Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *2010 IEEE symposium on security and privacy*, pages 305–316. IEEE, 2010.

[51] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2020 (4th Ed.).

[52] Jakapan Suaboot, Adil Fahad, Zahir Tari, John Grundy, Abdun Naser Mahmood, Abdulmohsen Almalawi, Albert Y. Zomaya, and Khalil Drira. A taxonomy of supervised learning for idss in scada environments. *ACM Computing Surveys*, 53(2), april 2020.

[53] Suricata. Open source ids / ips / nsm engine. https://suricata.io/, 2021. [Online; accessed 5-Feb-2023].

[54] Ruming Tang, Z. Yang, Zeyan Li, Weibin Meng, Haixin Wang, Q. Li, Yongqian Sun, Dan Pei, Tao Wei, Yanfei Xu, and Y. Liu. Zerowall: Detecting zero-day web attacks through encoder-decoder recurrent neural networks. In *IEEE INFOCOM*, pages 2479–2488, 2020.

[55] Lionel Nganyewou Tidjon, M. Frappier, and Amel Mammar. Intrusion detection systems: A cross-domain overview. *IEEE Communications Surveys & Tutorials*, 21:3639–3681, 2019.

[56] Thijs van Ede, Hojjat Aghakhani, Noah Spahn, Riccardo Bortolameotti, Marco Cova, Andrea Continella, Maarten van Steen, Andreas Peter, Christopher Kruegel, and Giovanni Vigna. DEEPCASE: Semi-Supervised Contextual Analysis of Security Events. In *IEEE Symposium on Security and Privacy*, May 2022.

[57] Thijs van Ede, Riccardo Bortolameotti, Andrea Continella, Jingjing Ren, Daniel J. Dubois, Martina Lindorfer, David Choffness, Maarten van Steen, and Andreas Peter. FlowPrint: Semi-Supervised Mobile-App Fingerprinting on Encrypted Network Traffic. In *NDSS*, 2020.

[58] Wikipedia. Jaccard index. https://en.wikipedia.org/wiki/Jaccard_index, 2022. [Online; accessed 5-Feb-2023].

[59] Wikipedia. Precision and recall. https://en.wikipedia.org/wiki/Precision_and_recall, 2022. [Online; accessed 5-Feb-2023].

[60] Wikipedia. Six sigma. https://en.wikipedia.org/wiki/Six_Sigma, 2022. [Online; accessed 5-Feb-2023].

[61] Wikipedia. Zero-day (computing). https://en.wikipedia.org/wiki/Zero-day_(computing), 2022. [Online; accessed 5-Feb-2023].

[62] Zhen Yang, Xiaodong Liu, Tong Li, Di Wu, Jinjiang Wang, Yunwei Zhao, and Han Han. A systematic literature review of methods and datasets for anomaly-based network intrusion detection. *Computers & Security*, 116:102675, 2022.

[63] Zeek. An open source network security monitoring tool. https://zeek.org, 2021. [Online; accessed 5-Feb-2023].

[64] Y. Zhang, H. Jiang, D. Feng, W. Xia, M. Fu, F. Huang, and Y. Zhou. Ae: An asymmetric extremum content defined chunking algorithm for fast and bandwidth-efficient data deduplication. In *IEEE INFOCOM*, pages 1337–1345, 2015.

[65] Hanxun Zhou, Yeshuai Hu, Xinlin Yang, Hong Pan, Wei Guo, and Cliff C Zou. A worm detection system based on deep learning. *IEEE Access*, 8:205444–205454, 2020.

A Datasets

Table 8: Experimental Datasets

No.	Dataset	Total (A)	Normal (B)	Attack (C)	Ratio (A/C)	Open	Comment
1	SIM1-1	1,000,000	900,000	100,000	0.1000	public	One big-group with $b_r(d_n) = 0.1$
2	SIM1-2	1,000,000	990,000	10,000	0.0100	public	One big-group with $b_r(d_n) = 0.01$
3	SIM1-3	1,000,000	999,000	1,000	0.0010	public	One big-group with $b_r(d_n) = 0.001$
4	SIM2-1	1,000,000	800,000	200,000	0.2000	public	Two big-groups with $b_r(d_n) = 0.1$
5	SIM2-2	1,000,000	980,000	20,000	0.0200	public	Two big-groups with $b_r(d_n) = 0.01$
6	SIM2-3	1,000,000	998,000	2,000	0.0020	public	Two big-groups with $b_r(d_n) = 0.001$
7	SIM3-1	1,000,000	700,000	300,000	0.3000	public	Three big-groups with $b_r(d_n) = 0.1$
8	SIM3-2	1,000,000	970,000	30,000	0.0300	public	Three big-groups with $b_r(d_n) = 0.01$
9	SIM3-3	1,000,000	997,000	3,000	0.0030	public	Three big-groups with $b_r(d_n) = 0.001$
10	IoT1	54,716	54,699	17	0.0003	public	Original set number 17, Kenjiro
11	IoT2	4,686	4,658	28	0.0060	public	Original set number 33, Kenjiro
12	IoT3	55,412	47,578	7,834	0.0141	public	Original set number 39, IRCBot
13	IoT4	14,845,292	72,776	14,772,516	0.9951	public	Original set number 43, Mirai
14	IoT5	1,307,003	1,848	1,305,155	0.9986	public	Original set number 44, Mirai
15	IoT6	10,122	8,938	1,184	0.1170	public	Original set number 48, Mirai
16	IoT7	11,925	9,485	2,440	0.2046	public	Original set number 49, Mirai
17	IoT8	4,644	4,554	90	0.0194	public	Original set number 52, Mirai
18	IDS1	1,302,148	1,294,939	7,209	0.0055	public	Web attack - bruteforce
19	IDS2	119,919	118,095	1,824	0.0152	public	Web attack - XSS
20	IDS3	23,229	23,217	12	0.0005	public	Web attack - SQL Injection
21	IDS4	3,699,243	3,671,387	27,856	0.0075	public	FTP-Patator
22	IDS5	579,004	531,716	47,288	0.0817	public	SSH-Patator
23	IDS6	880,347	850,803	29,544	0.0336	public	Infiltration
24	IDS7	472,750	377,399	95,351	0.2017	public	DDoS
25	IDS8	861,441	861,286	155	0.0002	public	Port Scan
26	ISP1	50,416	39,182	11,234	0.2228	private	Suspicious packets captured by an ISP
27	ISP2	9,327	1,863	7,464	0.8002	private	Suspicious packets captured by an ISP
28	ISP3	18,180	55	18,125	0.9970	private	Suspicious packets captured by an ISP