MineHunter: A Practical Cryptomining Traffic Detection Algorithm Based on Time Series Tracking

Shize Zhang*
Tsinghua University
Beijing, China

Zhiliang Wang*†
Tsinghua University
Beijing, China

Jiahai Yang*†‡
Tsinghua University
Beijing, China

Xin Cheng*
Tsinghua University
Beijing, China

Xiaoqian Ma
Beijing Wuzi University
Beijing, China

Hui Zhang*
Tsinghua University
Beijing, China

Bo Wang*
Tsinghua University
Beijing, China

Zimu Li*
Tsinghua University
Beijing, China

Jianping Wu*
Tsinghua University
Beijing, China

ABSTRACT

With the development of cryptocurrencies' market, the problem of cryptojacking, which is an unauthorized control of someone else's computer to mine cryptocurrency, has been more and more serious. Existing cryptojacking detection methods require to install anti-virus software on the host or load plug-in in the browser, which are difficult to deploy on enterprise or campus networks with a large number of hosts and servers. To bridge the gap, we propose MineHunter, a practical cryptomining traffic detection algorithm based on time series tracking. Instead of being deployed at the hosts, MineHunter detects the cryptomining traffic at the entrance of enterprise or campus networks. Minehunter has taken into account the challenges faced by the actual deployment environment, including extremely unbalanced datasets, controllable alarms, traffic confusion, and efficiency. The accurate network-level detection is achieved by analyzing the network traffic characteristics of cryptomining and investigating the association between the network flow sequence of cryptomining and the block creation sequence of cryptocurrency. We evaluate our algorithm at the entrance of a large office building in a campus network for a month. The total volumes exceed 28 TeraBytes. Our experimental results show that MineHunter can achieve precision of 97.0% and recall of 99.7%.

CCS CONCEPTS

• **Security and privacy → Intrusion detection systems**.

KEYWORDS

cryptomining; traffic analysis; anomaly detection

ACM Reference Format:
Shize Zhang, Zhiliang Wang, Jiahai Yang, Xin Cheng, Xiaoqian Ma, Hui Zhang, Bo Wang, Zimu Li, and Jianping Wu. 2021. MineHunter: A Practical Cryptomining Traffic Detection Algorithm Based on Time Series Tracking. In *Annual Computer Security Applications Conference (ACSAC '21), December 6–10, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3485832.3485835

1 INTRODUCTION

Due to the anonymity of cryptocurrencies' transactions, cryptocurrencies have been widely used in the dark web [26]. Lee et al. studied more than 27 million dark webpages and found that the monetary volume of cryptocurrency was around 180 million USD [19]. Cryptomining is a process in which transactions for various forms of cryptocurrency are verified and added to the blockchain digital ledger [33]. Furthermore, cryptojacking, the unauthorized use of someone else's computer for cryptomining, has become a popular attack similar to ransomware since 2018 [23]. At present, there are mainly two types of cryptojacking. One is that adversaries compromise popular webservers and embed malicious mining codes in the websites [24][31]. When users browse the websites, they will help the adversaries to mine cryptocurrencies. The detection method for this kind of cryptojacking is commonly to install a plug-in in the user's browser, which analyzes the JavaScript code in the website and the usage of the computing resources [18][3]. However, this method requires the cooperation of users and browser vendors, and it is difficult to deploy on a large scale environment. The other is that adversaries control the users' computers through malware, making them bots for cryptomining [17]. The detecting solution for this kind of cryptojacking is similar to the detection method of malware, mainly by deploying anti-virus software on the host [11]. But with the rapid development of cryptojacking, the compromised hosts are not only manifested by general computers, but also include all the devices with computing power, such as tablets, mobile phones, smart devices, in which the anti-virus software is difficult to deploy effectively [14]. Moreover, the detection performance of this method generally depends on malware detection capabilities of anti-virus software. Only a small number of known malware

*Institute for Network Sciences and Cyberspace, Beijing National Research Center for Information Science and Technology, Tsinghua University, China
†Co-corresponding authors.
‡Peng Cheng Laboratory, Shenzhen, China.

ACSAC '21, December 6–10, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8579-4/21/12…$15.00
https://doi.org/10.1145/3485832.3485835

can be detected, and it is challenging to detect a large number of unknown and variant malware [16].

To overcome these shortcomings of the existing detection methods, we propose MineHunter, a practical cryptomining traffic detection algorithm based on time series tracking. Instead of deploying at the hosts, MineHunter detects the cryptomining traffic at the entrance of enterprise or campus networks by traffic analyzing method, thus it can avoid large overhead. Similar to the network intrusion detection system, the detection algorithm obtains input traffic by port mirroring or optical fiber splitting and does not require the host to deploy any related software, which is more convenient to deploy in the actual network. In addition, the detection algorithm can detect both kinds of cryptojacking, embedding mining JavaScript codes in the website or utilizing the mining malware, which both have a similar pattern in network communication.

Below we summarize the challenges faced by MineHunter.

Challenge 1: Extremely unbalanced datasets. Data imbalance is the core challenge in the field of traffic anomaly detection. Although many network traffic anomaly detection algorithms based on machine learning have been proposed over the years, these algorithms usually require a relatively balanced dataset [39][21]. However, in the actual network environment, the scale of cryptomining traffic is tiny. To solve this challenge, we investigate the association between the network flow sequence of cryptomining and the block creation sequence of cryptocurrency. Using these characteristics, we propose a practical time series anomaly detection algorithm through long-term tracking of network behavior to distinguish a minimal amount of cryptomining traffic from enormous benign traffic. The evaluation results show that our algorithm can achieve 97.0% precision and 99.7% recall in an average data ratio of $1:200,000$ imbalanced datasets.

Challenge 2: Uncontrollable number of alarms. Traditional network traffic anomaly detection algorithms usually have the problem of high false positives and cannot guarantee the specific number of false positives. In addition, traffic volumes in the actual network environment are relatively large. For example, we measure a network with 4096 active addresses, which generates an average of 4.3 million flows per day. Therefore, even with a false alarm rate of 0.1%, 4,300 false alarms will be generated every day, which is unacceptable to network administrators. Our anomaly detection algorithm not only judges whether the flow is a cryptomining flow but also ranks the similarity. Network administrators can process the top n from the ranked table according to their processing capabilities.

Challenge 3: Traffic confusion. With the development of traffic detection algorithms and countermeasure technologies, the ability of adversaries to obfuscate traffic continues to increase. Common obfuscation techniques include adding proxy, load encryption, port replacement, and packet padding. In order to solve the challenges of these countermeasures, our detection algorithm only uses the timestamp of packet and the number of packets in a flow. In the following part, we will introduce that the features we use are based on the working principle of cryptomining, which is difficult to be confused. We evaluate the countermeasures in Section 4.5.

Challenge 4: Online detection. Due to the rapid growth of network bandwidth in the actual network environment, there are strict restrictions on the computational complexity of the detection algorithm. In order to meet the real-time requirements in the actual network environment, we only use two simple features, the timestamp of packet and the number of packets in a flow, and propose a linear complexity anomaly detection algorithm. Our evaluation experiments show that on an ordinary server, the processing power of our algorithm can reach about 350,000 packets per second. According to the average packet length of 1000 bytes, the algorithm on an ordinary server can afford a 2.8 Gbit/s link. Moreover, our algorithm detects each flow separately. Therefore, it can be easily parallelized to improve the overall processing performance of the system.

In summary, the main contributions of this paper are as follows:

- Unlike traditional traffic analysis, we not only analyze the network traffic characteristics of cryptomining, but also associate the flow sequence of cryptomining with the block creation sequence of the cryptocurrency.
- we propose a cryptomining traffic detection algorithm based on time series tracking, which first tries to solve all the above four core challenges.
- We conduct a large-scale evaluation experiment in a campus network environment within one month. The total number of packets in the network traffic exceeds 30 billion. The experimental results show that our algorithm can achieve 97.0% precision and 99.7% recall on the extremely unbalanced dataset outperforming the state-of-art approach based on machine learning.

In the spirit of open source, we make the codes and datasets publicly available at https://github.com/zsz147/MineHunter. We have considered possible ethical issues and data privacy protection, which are explained in detail in Section 4.1.3.

The rest of this paper is organized as follows: In Section 2, we describe the fundamental of cryptomining. In Section 3, we introduce the design of the detection algorithm. In Section 4, we evaluate the detection algorithm at the entrance of a campus network. In Section 5, we discuss the limitations of the algorithm. In Section 6, we introduce the related works about cryptojacking. We conclude the paper in Section 7.

2 THE FUNDAMENTAL OF CRYPTOMINING

2.1 The Basic Principle of Cryptocurrency

A cryptocurrency (or crypto currency) is a digital asset designed to work as a medium of exchange by the technologies of cryptography and blockchain [13]. The working principle of cryptocurrency based on blockchain is shown in Fig. 1. The whole system can be regarded as a billing system. The system uses blockchain technology to record the transaction records between different users in a block by a tree hash algorithm [9]. For example, four transactions are recorded in (Tx0, Tx1, Tx2, Tx3) in Fig. 1. By using tree hash algorithm, all transactions are hashed to Tx_root. Each block contains the timestamp, the hash value of the previous block and a random field nonce. Mining refers to find a particular nonce so that the hash calculation of the entire block meets certain special conditions (for example, 51 consecutive bits 0). The first one to find a nonce that meets the conditions will be rewarded with a certain amount of cryptocurrency. Since the average creation time of the block is fixed, the difficulty of calculating the block hash value increases with the

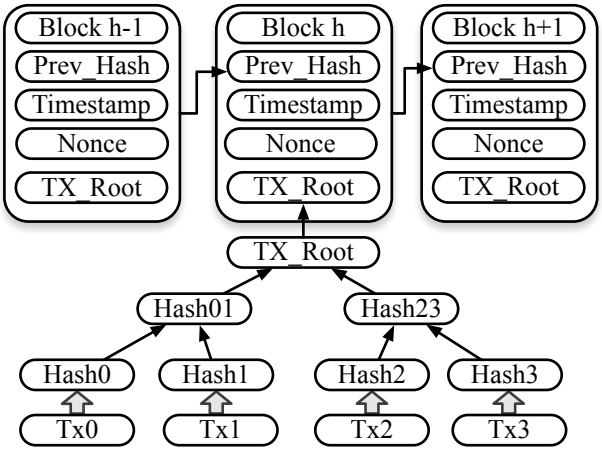

Figure 1: Working principle of cryptocurrency

increasing network computing power. With the continuous increase of the total network computing power, the probability of a single machine digging into cryptocurrency is almost zero. Therefore, the mining pool, a new type of mining method, has been presented [35].

The mining pool gathers a large number of mining machines to increase the computing capacity and the success rate of mining. The revenue is divided according to the calculation power contribution of different mining machines. In actual operation, the mining pool divides a complex computing task into many subtasks and delivers the subtasks to different workers according to the computing power of the mining machine.

2.2 Network Communication of Cryptojacking

Website mining hijacking and malicious mining software have similar network communication. There are three different scenarios, as shown in Fig. 2. The first scenario is the simplest. The adversary directly controls the victim to establish a link with the public mining pool. Since most public mining pools currently have public domain names or IP addresses, this situation can be solved by blacklisting [40]. The second scenario is that the adversary establishes a proxy used to establish a link with the public mining pool. The communication content of the public mining pool is forwarded to the victim by the proxy, in which case the blacklist method is ineffective. However, most of the current public pools use the Stratum protocol, which is a plaintext protocol [27]. In this scenario, the packet content signatures matching method can be used to detect the mining behavior. The third scenario is the most complicated. Compared with the second scenario, the adversary no longer allows the proxy to forward the data from the public mining pool but uses load obfuscation or encryption to escape the detection of content signature matching. The detection method proposed in this paper focuses on the third complex scenario, where the blacklist and the payloads of the packet cannot be used.

Due to the traffic detection challenges and the complex environment of cryptojacking, we must thoroughly analyze the inherent features of the mining communication process. The specific communication process of the mining process is shown in Fig. 3. The mining communication consists of several mining cycles. The beginning of the mining cycle indicates the creation of a new block. The end of the mining cycle indicates that the current block has been dug up, and the next new block is created. At the beginning of each mining cycle, a task packet is sent by the sender, which is a proxy or a public mining pool, and the task packet contains the hash value of the previous block, the creation time of the current block, the hash value of the transaction record, and the target that the miner needs to accomplish. During a mining cycle, the miner finds a nonce that satisfies the condition and submits a result to the sender. The sender will give a response of "ok" or "error" depending on the result. In addition, the sender also sends heartbeat signals at regular intervals.

Note that although the design goal of MineHunter is to meet the complex traffic detection challenges of cryptojacking like the third scenario in the Fig. 2, MineHunter is only used to detect cryptomining behavior in the enterprise or campus at the network level and cannot distinguish whether cryptomining is authorized or not, because authorization is subjective to the user's consciousness and cannot be directly reflected in the communication behavior of cryptomining. But we think this is not a essential issue in our scenario, since MineHunter is devoted as a network security management tool of the network administrator in the enterprise or campus. When cryptomining is confirmed, the network administrator can simply ask the owner of the host to know whether cryptomining is authorized or not.

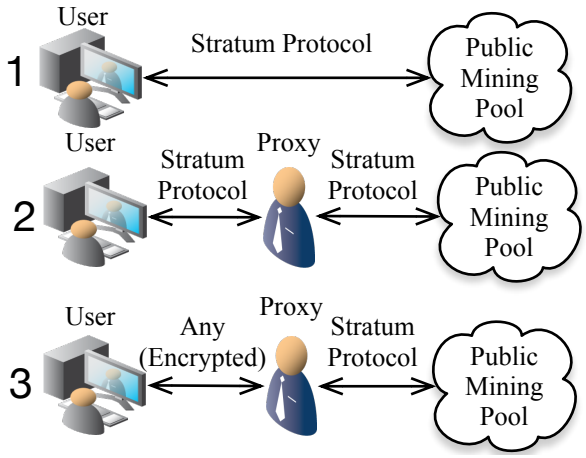

Figure 2: Scenarios of cryptojacking communication

3 DETECTOR DESIGN

3.1 Overview

According to our analysis of mining principles in Section 2.2, mining traffic has two essential characteristics. One is that the time of task packet issued by a proxy or a mining pool is the same as the time when a new block is created. The other is that cryptomining requires a long period of communication. We use these two features to

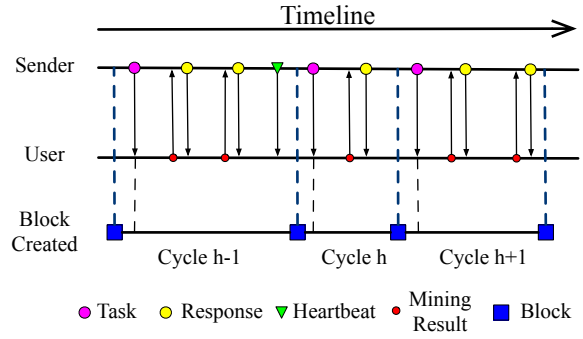

Figure 3: Communication process of the mining

build MineHunter. Its architecture is shown in Fig. 4. MineHunter consists of three parts, flow data acquisition and preprocessing, block creation time series acquisition, and core detection algorithm.

flow data acquisition and preprocessing: MineHunter is designed to deploy on the entrance gateway of a campus or enterprise. Therefore, raw traffic can be collected to the server through port mirroring or optical fiber on the entrance gateway. We aggregate the raw traffic into two-tuple (source-destination address) flow and record the timestamp of each packet. Choosing this flow aggregation method is to prevent adversaries from using port confusion to disperse cryptomining traffic to different ports to avoid detection.

block creation time series acquisition: Since current cryptocurrencies such as Monero, Bitcoin, Ethereum, and Litecoin all adopt a p2p (peer-to-peer) network structure, it is possible to obtain block creation time information in real time by joining the p2p network of different cryptocurrencies. For example, we use monerod [12] to join the p2p network of Monero to collect block creation information. The <Received NOTIFY_NEW_FLUFFY_BLOCK> type record in the running log of the monerod contains the time when each new Monero block is created.

core detection algorithm: Our core detection algorithm contains two parts, local similarity table, and global similarity table. The local similarity table uses the correlation between mining flow and block creation time to calculate the similarity of cryptomining behavior in an interval. The global similarity table uses the long-term communication characteristics of cryptomining to accumulate local similarity results. The output of the algorithm is a ranked table, which records the similarity of each flow belonging to cryptomining. The network administrator can use the threshold or top n in the ranked table as the alert condition.

3.2 Cryptomining Traffic Detection Algorithm

Before introducing our specific detection algorithm, we first formulate our problem and detection target. We first define the raw network traffic as a flow set $F = \{f_1, f_2, ..., f_n\}$, where each f represents a two-tuple (source-destination address) flow. Each flow f consists of a large number of communication packets. According to the different sending time of packets, each flow f can be represented as a time series, $f = \{p_1, p_2, ..., p_m\}$, where $p_i, i = 1, 2, .., m$ means the timestamp of packet. Since cryptomining is a long-term communication behavior, we define a detection time interval $[t_s, t_e]$, and a duration time $t_i = t_e - t_s$. Therefore, our detection target is to calculate the cryptomining similarity of each time sequence f in F within $[t_s, t_e]$. The formulation of the target is $MH(f|[t_s, t_e]) = S$, where MH is MineHunter, $S \in [0, 1]$ means the cryptomining similarity. When S is closer to 1, it means that f is more likely to be a cryptomining flow. Conversely, when S is closer to 0, f is more likely to be a benign flow. By calculating the S of each flow in F, we will obtain a ranked table to generate alerts. Next, we will introduce the core detection algorithm in detail.

3.2.1 Local Similarity Algorithm. According to the above description, our algorithm uses the characteristics of long-term communication of cryptomining. Therefore, the selection of detection duration time t_i is generally relatively large, such as 1 hour, 2 hours, 6 hours, or even 1 day. We will evaluate t_i in Section 4.3. We divide $[t_s, t_e]$ into several intervals according to the time series created by the block. The specific formulation is shown in Fig. 5. We define the block creation time series within $[t_s, t_e]$ as $X = \{x_0, x_1, ..., x_h\}$, where x means the creation time of the block and $(x_0, x_1), (x_1, x_2), ..., (x_{h-1}, x_h)$ means different intervals. The local similarity algorithm is to calculate the cryptomining similarity of the flow in each interval. From the analysis in Section 2, if there is cryptomining in the interval, there will be a task packet created which is close to the left boundary of the interval. Therefore, we define the local interval distance of the flow f in interval k as $e(f^k) = \min_{x_{k-1} \le p < x_k} dis(p, x_{k-1})$ and use $dis(p, x_{k-1})$ to denote the distance between p and x_{k-1}. Since p and x both represent time and have the same measurement dimension, we choose Manhattan-distance[29], which means $dis(p, x_k) = p - x_{k-1}$. For example, in Fig. 5, $e(f^k) = p_{k_1} - x_{k-1}$. Furthermore, we define the local similarity s_l of flow f in interval k as $s_l(f^k) = 1 - \frac{e(f^k)}{x_k - x_{k-1}}$, $s_l \in [0, 1]$. When s_l is closer to 1, it means that f is more likely to be a cryptomining flow in interval k. Finally, we maintain a local similarity table (LST) to store the result of the local similarity of the each flow f in the flow set F. The data structure of the LST is a dictionary, in which the key is the flow name (source-destination address), and the value is the result of the local similarity, like the core detection algorithm model in the Fig. 4. The specific algorithm pseudocode of the naive calculation method is shown in Appendix A.1.

Although this naive algorithm can identify cryptomining in the local interval, it also produces enormous false positives in the following two scenarios. The first is shown in Fig. 6. Some applications, such as video streaming, generate high-frequency and large-scale data communications. These communications will make the naive local similarity algorithm achieve great similarity and even reach 100%. This scenario requires us to consider the influence of packet numbers. The other scenario is shown in Fig. 7. Some applications such as Microsoft system updates will generate periodic heartbeat signals for a long time. This period is generally 30s or 60s. Although this low-frequency noise signal cannot achieve high local similarity in every interval, its long-term communication behavior is similar to cryptomining. Therefore, through the long-term accumulation of local similarity, there may still be false positives.

To solve these two scenarios, we propose a local similarity calculation algorithm based on credible probability estimation. Based

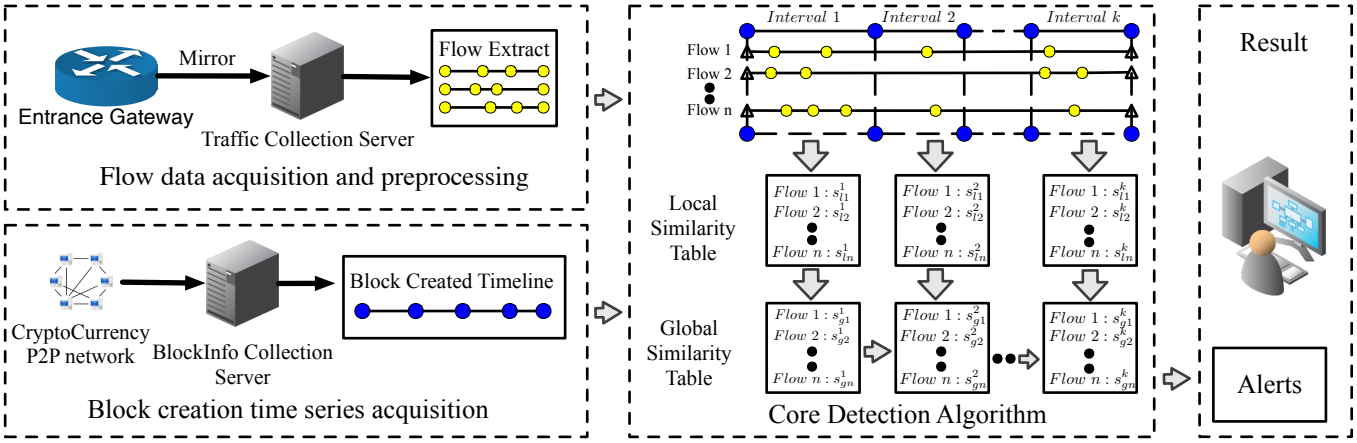

Figure 4: Structure of detection system

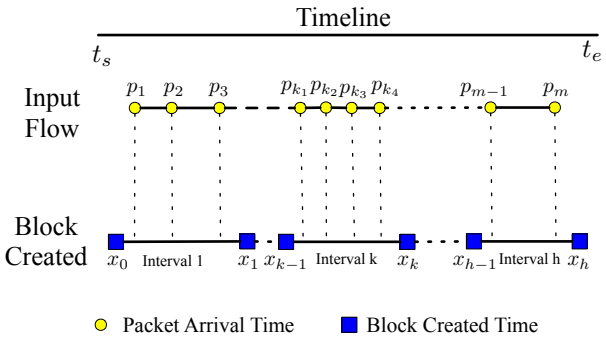

Figure 5: The formulation of local similarity algorithm

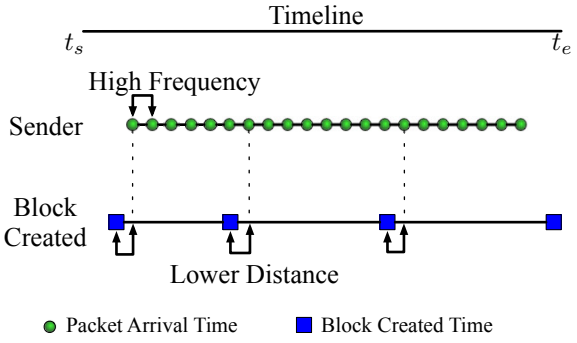

Figure 6: The first false positive scenario of naive algorithm

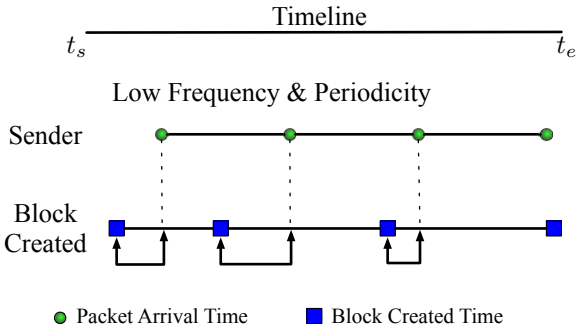

Figure 7: The second false positive scenario of naive algorithm

on the naive local similarity algorithm, we add a credible probability coefficient α belonging to [0,1] to indicate the credibility of the naive algorithm. Therefore, we improve the local similarity calculation as $s_l(f^k) = \alpha * (1 - \frac{e(f^k)}{x_k - x_{k-1}})$. Intuitively, α is used to evaluate whether a flow "intentionally" generates a task packet to obtain a small local interval distance, which is the essential difference between cryptomining and noise. To measure "intention", we analyze the probability of obtaining a certain local interval distance from a random sequence from the perspective of probability. Intuitively, "intention" means that a random event with a small probability occurs frequently. We first define a random sequence in an interval, which refers to a sequence of m packets. Each packet in the sequence has the same probability of appearing at any point in the interval and is independent of each other. Next, we calculate the probability that a random sequence f with m_k packets obtains the local interval distance e_k in the interval of length n_k. $P(e = e_k) = (\frac{n_k - e_k}{n_k})^{m_k} - (\frac{n_k - e_{k-1}}{n_k})^{m_k}$, which is proved in the Appendix B. By calculating the probability of all e_k, we obtain the probability density distribution curve of e_k. When the local interval distance of a flow f is $e(f^k)$, the cumulative distribution probability $P(e \le e(f^k))$ represents the cumulative probability for a random sequence to obtain smaller local interval distance. When $P(e \le e(f^k))$ approaches 0, the probability that a random sequence obtains a smaller local interval distance approaches 0, which means that the flow f obtaining the local interval distance $e(f^k)$ is a small

Table 1: An example of α.

Distance \ # Packets	1	2	5	10	60	120
0	0.992	0.983	0.959	0.920	0.605	0.366
1	0.984	0.967	0.919	0.846	0.365	0.133
2	0.976	0.951	0.881	0.777	0.219	0.048
3	0.968	0.935	0.844	0.713	0.131	0.017
4	0.960	0.919	0.808	0.654	0.078	0.006
5	0.952	0.903	0.773	0.599	0.046	0.002
10	0.912	0.826	0.618	0.382	0.003	0.001
15	0.872	0.751	0.488	0.239	0.001	0.001
20	0.832	0.681	0.382	0.147	0.001	0.001

Table 2: The statistics of background traffic.

Duration time	Active host number	Total packet number	Total volumes
1 month	4096	30 billion	28 TBytes
Maximum packets per second	Maximum bits per second	Average flow number per day	Average packet number per day
280533 pps	1.3 Gbit/s	4.7 million	0.9 billion

probability event. Furthermore, when $P(e \leq e(f^k))$ is closer to 0, it means that the flow more likely "intentionally" sends a task packet, that is, the more likely it is a cryptomining flow. Therefore, we define $\alpha = 1 - P(e \leq e(f^k)) = P(e > e(f^k))$, which means that when $P(e \leq e(f^k))$ is closer to 0, α is closer to 1 and the naive local interval similarity $s_l(f^k)$ is more reliable. The specific pseudocode of the algorithm is shown in Appendix A.2. We use a specific example to explain how α distinguishes the cryptomining from the above two types of noise signals. We set an interval with a length of 120 seconds, which is the average creation time of Monero, and analyze the values of α corresponding to the number of different packets and the local interval distance. The results are shown in Table 1. The red part in the table represents α of the cryptomining flow (the number of packets is small, and the local interval distance is small). The green part represents α of high-frequency noise (the number of packets is large, and the local interval distance is small). The yellow part represents α of low-frequency periodic noise (the number of packets is small, and the local interval distance is large). It can be seen that α is small under the two types of noise, while α of the cryptomining flow is close to 1. We will compare the naive algorithm and the improved algorithm in Section 4.3.

3.2.2 Global Similarity Table. Although the local similarity algorithm can calculate the similarity of cryptomining in one local interval, we cannot distinguish between cryptomining flows and benign flows in one interval due to the extremely unbalanced challenge of network traffic. Therefore, we use the long-term communication characteristics of cryptomining to propose a global similarity algorithm. We maintain a global similarity table within $[t_s, t_e]$ as *GST*. The structure of the *GST* is similar to the *LST*, which is also a dictionary where the key is flow name and the value is the result of the global similarity, like the core detection algorithm model in the fig. 4. When the calculation of the local similarity table (*LST*) in each interval is completed, we will update the *GST*. The core of the *GST* is an iterative algorithm. We use the method of addition increment and subtraction decrement. Specifically, we initialize an empty *GST* at the beginning of t_s. As the detection time passes, when an *LST* calculation is completed, we will update *GST* with the current *LST*. The specific update strategy is as follows. We will traverse the *GST* to determine whether each flow in *GST* also appears in *LST*. If a flow is in *GST* and also in *LST*, we will add the result of this flow in the current *LST* to *GST*. On the contrary, if a flow is in *GST* but not in *LST*, it means that this flow has no communication in the current interval, which violates the continuous working principle of cryptomining. Therefore, we penalize the global similarity of this flow. We subtract a penalty term β from the similarity result of this flow in *GST*. Because the maximum value of local similarity is 1, we set $\beta = 1$. Next, we traverse each flow in the *LST* to find flows that do not appear in the *GST*. We will add the result of this flow in the *LST* to *GST*. When all the intervals within $[t_s, t_e]$ are calculated, we will calculate the average interval similarity of each flow in the global similarity table to get the global similarity $s_g(f) = \frac{GST[f]}{n_x}$, in which $GST[f]$ represents the value of f in the *GST* and n_x represents the number of block intervals in the $[t_s, t_e]$. Finally, we sort the $s_g(f)$ of each flow to generate alerts. Network administrators can configure different strategies to generate alarms, for example, by setting a fixed alarm threshold, taking top n, or checking from the head of the table, and stopping checking until a false alarm is found. The specific pseudo code is shown in Appendix A.3.

4 EVALUATION AND ANALYSIS

4.1 Data

Since there is no public cryptomining traffic dataset, we construct a mixed dataset by ourselves to evaluate the algorithm. The mixed dataset includes two parts: background and cryptomining traffic. In real scenarios, cryptomining traffic volumes are much smaller than benign traffic volumes. To make the environment conform to the real scene, we use large-scale background traffic and a small amount of cryptomining traffic.

4.1.1 Background Traffic. To verify the performance of our algorithm in actual scenarios, we collect background traffic from the entrance gateway of a large office building in a campus network by using port mirroring. The simple statistics of background traffic are shown in Table 2. The total volumes exceed 28 TeraBytes, and the number of total packets reaches about 30 billion. The number of flows is relatively stable within a month, which fluctuates at 4.7 million per day. Since the scale of background traffic is enormous, determining that all flows are truly benign is impossible. We remove the cryptomining flow in the background traffic based on plaintext load filtering method (details in Appendix C). Despite this, some encrypted cryptomining sessions may still in the background traffic. We further discuss the possible biases after showing the evaluation results.

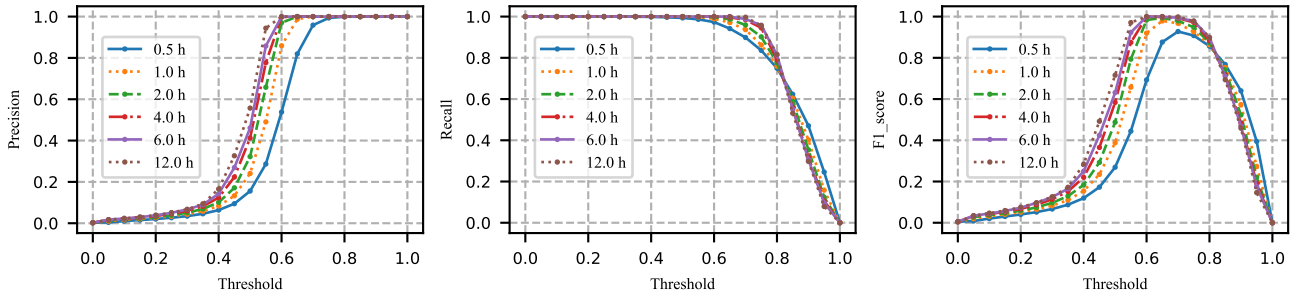

Figure 8: Overall evaluation results of MineHunter

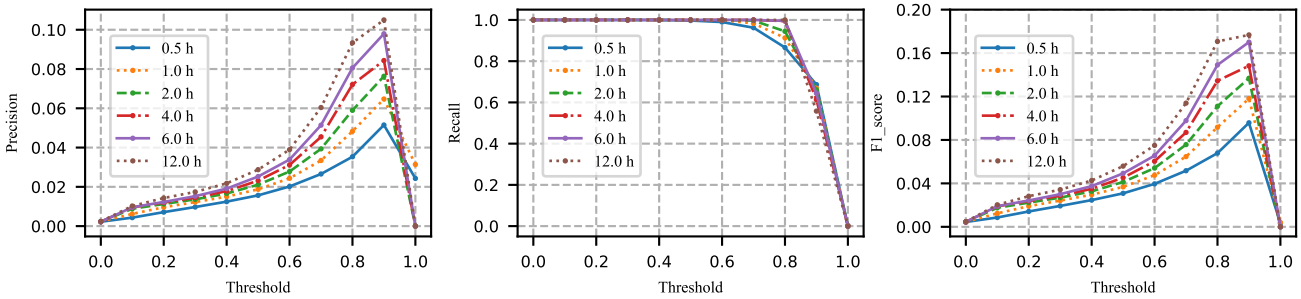

Figure 9: Evaluation results of MineHunter-Naive

4.1.2 Cryptomining Traffic. Previous measurements indicated that the most popular cryptocurrency used for cryptojacking is Monero since it has no special requirements for hardware [18], hence our main evaluation experiment is for Monero. We evaluate the scalability of other cryptocurrencies in Section 4.7. We mainly collect cryptomining traffic from popular mining pools of Monero. We follow two principles for the selection of mining pools. One is to select the mining pool with high computing power according to [20]. The other is to select the mining pool that supports TLS communication to verify the algorithm's ability to detect encrypted traffic. According to these two principles, we selected a total of 21 mining pool nodes, covering nearly 70% of the total computing power of Monero from the statistic in [20]. The specific list of mining pool nodes is shown in Appendix D. We collect the cryptomining traffic from all 21 mining pool nodes through TLS protocol and XMRig, one of the most used tools by cryptojacking malware [37][25]. The duration time is the same as background traffic. The IP address of the mining pool nodes is fixed, hence we collected 21 two-tuple (source-destination address) flows from 21 mining pool nodes respectively. Since the average number of flows is 4.7 million per day in background traffic, the average data imbalance ratio is $21 : 4,700,000 \approx 1 : 200,000$ in one day. We merge the cryptomining traffic and background traffic in chronological order through mergecap[30] and replay the merged dataset for detection.

4.1.3 Ethical Considerations. Since background traffic used in this paper comes from a real network environment, to protect the private information of users in the campus network, we anonymize the IP addresses belonging to the campus network in the traffic through Crypto-PAn, a public IP address anonymization tool [38]. Furthermore, we remove the payload of packets in the background traffic and leave only the packet header. Therefore, all background traffic used in our experiments is in accordance with the policies defined by our institution.

4.2 Evaluation Metric

Since we use an extremely unbalanced dataset, TPR (true positive rate) and FPR (false positive rate) cannot effectively evaluate the performance of our algorithm. For example, in our scenario, the average imbalance ratio of the data set is $1 : 200,000$. A seemingly excellent algorithm that the TPR reaches 100% and the FPR reaches 0.1% will generate $200,000 * 0.1\% = 200$ false positives, which is 200 times the real cryptomining samples. Hence, TPR and FPR may be misleading in the scenario of extremely unbalanced dataset. Instead, we choose precision, recall, and f1-score as evaluation metrics.

4.3 Overall Evaluation Results

According to the previous introduction, our algorithm faces four core challenges, which we will evaluate separately. First, we evaluate the algorithm detection performance under extremely unbalanced data scenarios (Challenge 1). According to the description in Section 3.2, our algorithm needs to set a detection time range t_i, we choose $t_i = 0.5h, 1h, 2h, 6h, 12h$ respectively. Meanwhile, we segment the original data in the time dimension according to the detection time range to obtain more detection cases. For example, if t_i is set to $0.5h$, we will obtain a total of 21 $(nodes) * 2 * 24\ (hours) * 32\ (days) \approx 30,000$ detection cases. Besides, we evaluate the impact of different alarm threshold selections. The overall evaluation

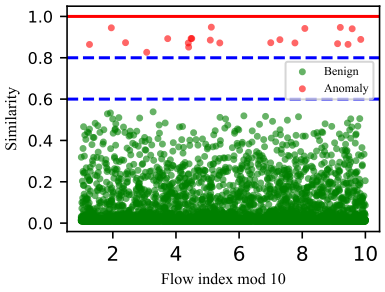

Figure 10: Case study

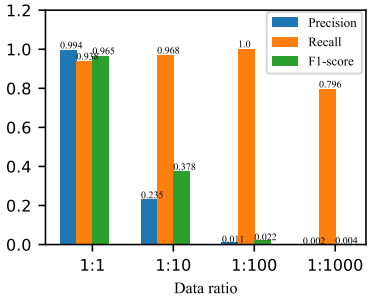

Figure 11: Performance evaluation of [4]

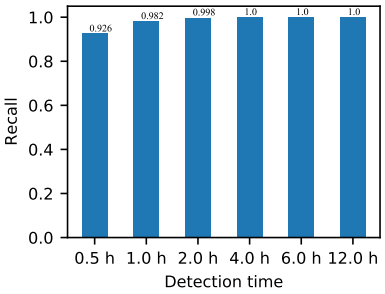

Figure 12: Recall under only one false positive alarm method

results of the algorithm are shown in Fig. 8. As the threshold increases, the precision of the algorithm gradually increases, and the recall gradually decreases. It can be seen from the f1-score sub-plot that the threshold setting between $[0.6, 0.8]$ is appropriate. In addition, as the detection time increases, the overall performance of the algorithm gradually improves. However, the longer detection time is set, the longer detection delay, and the longer duration of cryptomining is required. The detection time of $2h$ is relatively suitable, which is trade-off between the performance of algorithm and the detection delay. And when the threshold is set to 0.6, the precision of the algorithm is 97.0%; recall is 99.7%; f1-score is 0.983.

Since there may be some encrypted cryptomining sessions in the background traffic, the overall evaluation performance of MineHunter may have biases, including two parts. One is that the precision may be underestimated since some false positives may be the encrypted cryptomining in the background traffic. The other is that the recall may be overrated since there may be still some cryptomining in the background traffic not found. To further explain the rationality of the recall, we use MineHunter to detect the cryptomining flows in the background traffic labeled by plaintext load filtering method. The evaluation results show that MineHunter can find all these cryptomining flows (details in Appendix C).

We use a specific case to illustrate the effectiveness of the algorithm intuitively. We set the detection time to $2h$ and select the case of 2020-10-24 16:00:00-18:00:00, which is the peak period of network usage. We select the top $10,000$ from the global similarity table, and the result is shown in Fig. 10. Each point in the figure represents a flow detection result. It can be seen that there is a clear dividing line between cryptomining flows and benign flows. The similarities of cryptomining flows are above 0.8, and the results of benign flows are below 0.6. Moreover, most of the detection results of benign flows are below 0.3, which shows that the detection algorithm effectively filters noise. Therefore, it can be seen that when the threshold is set between $[0.6, 0.8]$, the algorithm can effectively achieve high-precision detection performance under unbalanced datasets.

To further indicate the effectiveness of the algorithm, we compare two different algorithms. The first is MineHunter-naive, which uses the naive distance detection algorithm introduced in Section 3.2.1. The second is a state-of-art algorithm based on machine learning [4].

The performance evaluation results of MineHunter-naive are shown in Fig. 9. It can be seen from the recall plot that both the naive distance algorithm and the improved algorithm can effectively identify the cryptomining traffic. However, it can be seen from the precision plot that the precision of the naive algorithm is less than 10%, which shows that there are many false positives. It can also be seen that MineHunter based on the probability model effectively filters the noise signal by adding the credibility coefficient α.

Maurantonio C et al. tries to use machine learning algorithms to detect cryptomining traffic and convert cryptomining detection into a binary classification problem [4]. Since the ratio of positive and negative samples in the dataset has a great influence on the performance of the machine learning algorithm, we evaluate the performance of algorithm under different ratios, as shown in Fig. 11. With a data ratio of $1:1$, the machine learning algorithm has good performance. However, as the proportion of data imbalance increases, the performance of the machine learning algorithm decreases significantly. When the imbalance ratio reaches $1:1000$, the precision of the algorithm is even less than 1%. Therefore, the machine learning algorithm may be only suitable for scenarios where the ratio of cryptomining traffic to benign traffic is comparable, which cannot meet the requirements of Challenge 1.

4.4 Alarm Condition Evaluation

In this section, we evaluate the ability of MineHunter to solve uncontrollable alarms (Challenge 2). According to the previous description, the output of our algorithm is a ranked table. Therefore, in addition to using the threshold as an alarm judgment condition (evaluated in Section 4.3), network administrators can also check from the beginning of the ranked table and stop until a false alarm is discovered. In this way, network administrators only need to deal with one false positive for each algorithm result. For example, if the algorithm's detection time is set to $2h$, network administrator only needs to deal with $24\ (hours)/2 = 12$ false positives every day. Under this controllable false alarm setting, the recall of the algorithm is a core performance evaluation metric, as shown in Fig. 12. Under different detection time settings, the recall of the algorithm is higher than 90%, which shows that the cryptomining flows are in the top position in ranked table. In addition, when the detection time is set to 2h, the algorithm's recall of 99.8% is slightly better than the result of 99.7% in Section 4.3. Therefore, MineHunter can still achieve high detection performance under the controllable alarm setting.

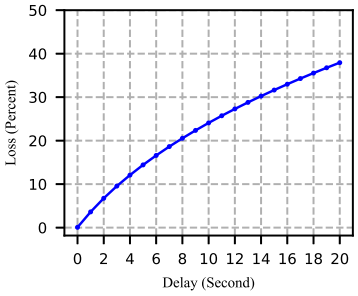

Figure 13: Loss evaluation

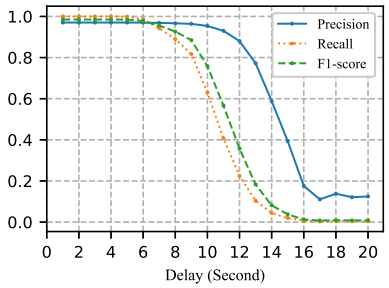

Figure 14: Latency evaluation

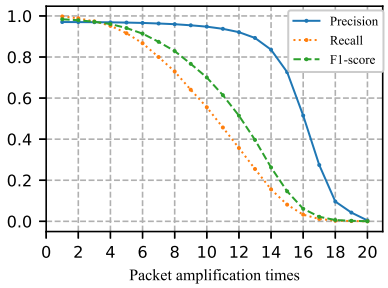

Figure 15: Amplification evaluation

4.5 Anti-adversarial Evaluation

In this section, we evaluate the ability of MineHunter to solve the traffic confusion (Challenge 3). At present, common traffic obfuscation methods used by adversaries include adding proxy, load encryption, port replacement, and packet padding. MineHunter only uses the characteristics of timestamp and number of packets and is not interfered by these common obfuscation methods, hence MineHunter can obviously resist those. We evaluate the countermeasures used by adversaries specially against our detection algorithm under the "white box" condition, which means that adversaries know all details of the algorithm, including the internal structure, design, parameters, etc.

4.5.1 Packet Delay. Our algorithm is constructed by using the correlation between task packets and the time series of block creation. Therefore, adversaries can resist our detection algorithm by delaying the delivery time of task packets. However, delaying task packets will reduce the effective computing time of victim hosts, which will lead to a decline in adversaries' profit. We evaluate the impact of different delay times on the loss of Monero mining profit, as shown in Fig. 13. In addition, we evaluate the impact of packet delay on our algorithm detection performance ($t_i = 2h, threshold = 0.6$), as shown in Fig. 14. It can be seen from Fig. 14 that when the delay time is less than 10s, the overall performance of the algorithm is less affected. When the delay time is higher than 10s, the algorithm performance begins to decrease significantly. Therefore, our algorithm can effectively resist packet delays under 10s. It can be seen from Fig. 13 that when the delay time is 10s, adversaries will reduce the profit by 24.1%. However, adversaries can bypass our detection algorithm by setting a larger delay, which will also lose a larger profit.

4.5.2 Packet Amplification. Our algorithm uses the number of packets to calculate the credibility coefficient α. Therefore, adversaries can bypass our detection algorithm by increasing the number of packets to disguise the cryptomining flow as a high-frequency noise. However, expanding packets also consumes more bandwidth of the proxy server controlled by the adversary. We evaluate the impact of packet amplification on the performance of the algorithm ($t_i = 2h, threshold = 0.6$), as shown in Fig. 15. When packet amplification is less than 10 times, precision of the algorithm does not decline significantly, and recall of the algorithm gradually drops to 60%. This is because the credibility coefficient α of some interval with a small length is rapidly reduced due to packets amplification, which results in that these intervals cannot be effectively detected. However, since cryptomining is a long-term communication behavior, it can still be covered by multiple detections in the time dimension under the premise of high precision. Therefore, the algorithm can effectively combat packet amplification by 10 times. When the packet amplification exceeds 10 times, the performance of our algorithm will be greatly reduced.

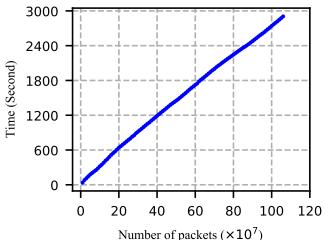

Figure 16: Running time of the algorithm

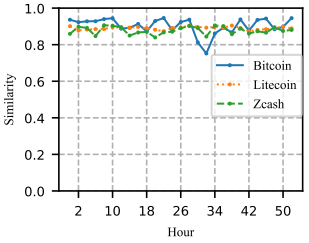

Figure 17: Similarities of different cryptocurrencies

4.6 Efficiency Evaluation

In this section, we evaluate the operating efficiency of the algorithm (Challenge 4). We implement the detection algorithm on an ordinary server (CPU: 2.1 GHz Intel Xeon Silver 4110, Memory: 64GB DDR4) through java language. The running time evaluation result of the algorithm is shown in Fig. 16. The running time of the algorithm increases linearly with the increase of the number of input packets, which means that the processing speed of the algorithm is stable, about 350,000 pps. According to the average packet length of 1000 bytes, the algorithm on an ordinary server can afford a 2.8 Gbit/s link. Furthermore, because the algorithm is based on the detection of each flow, the algorithm can realize parallel processing by configuring the flow hash algorithm on the collection switch.

4.7 Scalability

In this section, we evaluate the performance of MineHunter in the scenarios of different cryptocurrencies and websites with embedded cryptomining scripts to verify the scalability of the algorithm. According to the latest measurement results [25], Monero is the most discussed cryptocurrency in underground forums. Others include Bitcoin, Zcash, Litecoin, and Ethereum. We select three

Table 3: The evaluation results of mining service.

Mining Service	Cryptocurrency	Protocol	Proxy IP	Similarity
CryptoLoot[6]	Uplexa	TLSv1.2	45.79.218.212	0.80
Crypto Webminer[7]	Sumokoin	TLSv1.2	185.163.119.151	0.78
Monerominer.rock[22]	Masari	TLSv1.2	157.230.173.68	0.93

cryptocurrencies (Bitcoin, Zcash, and Litecoin) for evaluation except for Ethereum. This is because Ethereum has high hardware requirements for the mining host (it must have a graphics card with more than 4GB of video memory). It is difficult to run on ordinary hosts, which is not conducive to cryptojacking. We collect Bitcoin traffic from ViaBTC mining pool [34]; Litecoin traffic from f2pool mining pool [8]; Zcash traffic from coinMine.pl [5], lasting about two days. Meanwhile, we separately join the p2p network of three currencies to obtain the block creation time series. We set the same parameters as the Monero detection algorithm, $t_i = 2h, threshold = 0.6$. We also divide multiple detection cases according to the time dimension, which is the same as the Monero detection evaluation. The detection results are shown in Fig. 17. The minimum detection similarity of the three cryptocurrencies is higher than 0.7 at different time, which shows that MineHunter can be effectively applied to different types of cryptocurrencies. Moreover, because the detection of each currency is independent of each other, network administrators can configure the detection of multiple currencies at the same time.

To evaluate whether MineHunter can detect websites with embedded mining code from the perspective of network traffic, we build a web server and respectively run the 3 popular mining services found in the previous studies [2][18]. We use a client browser to access these mining services to respectively capture the traffic within 2 hours. We set the same parameters as the Monero detection algorithm, $t_i = 2h, threshold = 0.6$, and the detection results are shown in Table 3. It can be seen that the similarities of different cryptocurrencies are higher than the threshold, which shows that all of these mining services can be effectively detected by MineHunter. Furthermore, these mining services all use the TLSv1.2 protocol for encrypted communication to avoid payload detection algorithms. In addition, we query these proxy IPs in the AbuseIPDB [1]. Only 45.79.218.212 was recorded in the database in May 2019. However, the credibility of abuse is still 0%. 185.163.119.151 and 157.230.173.68 do not appear in the abuse database, and these IPs are all used by third-party service providers, which can be easily changed. Therefore, it can be seen that the blacklist method is difficult to effectively detect these proxy IPs.

5 LIMITATIONS AND DISCUSSIONS

It can be seen from the analysis in Section 4.3 that the input sequence requires a certain length of time. The evaluation result shows that the detection time of 2 hours is a suitable parameter. Therefore, MineHunter cannot detect immediately when the cryptomining behavior occurs, and the input sequence needs to be accumulated for a period of time. However, the profits of cryptomining are proportional to mining time, and cryptomining is a long-term communication behavior. MineHunter can effectively detect the mining sequence but with a little delay.

MineHunter requires that the packets in a flow are complete without sampling. However, MineHunter does not require the payload of packets and only needs the timestamp in packet header, which means only 4 bytes need to be stored in memory per packet. For example, storing 120 minutes of full packet just including the timestamp of packet header for a 10 Gbit/s link only needs about 36 GB memory space, which is not a harsh restriction.

The evaluation in Section 4.5.1 and 4.5.2 shows that the algorithm has considerable adversarial capabilities under "white box" conditions. However, adversaries can still bypass our algorithm by losing some profits. This is one of the limitations of our method. We will take these two countermeasures as our future work.

6 RELATED WORK

The research on cryptojacking detection is divided into two types: mining JavaScript detection in webpage and mining malware detection on the host. At present, a large number of research works focus on the detection of web mining scripts. Rüth et al. conducted a large-scale measurement analysis of the mining tools on the webpage [28]. The analysis showed that the most common mining tool, coinhive, contributed nearly 1.18% of Monero.

Konoth et al. proposed a new detection tool Minesweeper, which identifies whether a website contains mining code [18]. Minesweeper analyzes the features of JavaScript code from common web mining tools (like coinhive, crpytoNoter, NFWebMiner) and functions in the wasm module containing cryptographic operations (as reflected by XOR, shift, and rotate operations). Minesweeper can be deployed as a plug-in in the browser. Similarly, Hong et al. proposed CMTracker, a behavioral analysis-based detector used to detect mining JavaScript and related domain names [15]. The CMTracker mainly utilizes the mining behavior to require a large number of powers to calculate the hash value and the characteristics of the periodic execution of the mining script. However, the deployment of these tools requires authorization from browser vendors or users.

Pastrana et al. focused on the development of cryptomining malware on the host [25]. The analysis showed that these malware generated millions of earning, and nearly 4.3% of Monero was generated by malware-controlled hosts. Soviany S et al. proposed a machine learning-based approach to detect Android malicious mining software, which uses a layered detector to detect Android malware including malicious mining software [32]. This detection method can only be used for some known malware detection and is also ineffective for some variants of known malware. Gangwal et al. leveraged magnetic side-channel to detect cryptomining [10]. This approach is limited by the physical environment and difficult to deploy on large enterprises.

Maurantonio C et al. first try to use machine learning-based algorithms to detect mining traffic [4]. They use the interarrival time and packet size as features to train a random forest model. The algorithm has achieved good results on the balanced dataset. However, in the actual large-scale unbalanced data scenario, the algorithm has apparent limitations.

7 CONCLUSION

In this work, we associate the network flow sequence of cryptomining with the block creation sequence of the cryptocurrency

and propose MineHunter, a practical cryptomining traffic detection algorithm, which can be deployed at the entrance of enterprise or campus networks. Our algorithm has attempted to solve the four core challenges faced in the actual network environment, including extremely unbalanced datasets, controllable alarms, traffic confusion, and efficiency.

ACKNOWLEDGMENTS

This work is supported by the National Key Research and Development Program of China under Grant No.2018YFB1800204.

REFERENCES

[1] AbuseIPDB. 2021. AbuseIPDB. https://www.abuseipdb.com/
[2] Hugo LJ Bijmans, Tim M Booij, and Christian Doerr. 2019. Inadvertently Making Cyber Criminals Rich: A Comprehensive Study of Cryptojacking Campaigns at Internet Scale. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*. 1627–1644.
[3] Hugo LJ Bijmans, Tim M Booij, and Christian Doerr. 2019. Just the Tip of the Iceberg: Internet-Scale Exploitation of Routers for Cryptojacking. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. ACM, London United Kingdom, 449–464.
[4] Maurantonio Caprolu, Simone Raponi, Gabriele Oligeri, and Roberto Di Pietro. 2020. Cryptomining Makes Noise: a Machine Learning Approach for Cryptojacking Detection. arXiv:1910.09272 [cs.CR]
[5] CoinMine.pl. 2016. Zcash ANONYMOUS MINING. https://www2.coinmine.pl/zec/index.php
[6] Crypto-loot.org. 2018. Cryptoloot. https://crypto-loot.org/
[7] Crypto-webminer.com. 2021. Crypto Webminer - Mining in your Browser with Webmining technology. https://www.crypto-webminer.com/
[8] F2pool.com. 2021. F2pool mining pool. https://www.f2pool.com/
[9] JAKE FRANKENFIELD. 2020. Merkle Root(Cryptocurrency). https://www.investopedia.com/terms/m/merkle-root-cryptocurrency.asp
[10] A. Gangwal and M. Conti. 2020. Cryptomining Cannot Change Its Spots: Detecting Covert Cryptomining Using Magnetic Side-Channel. *IEEE Transactions on Information Forensics and Security* 15 (2020), 1630–1639. https://doi.org/10.1109/TIFS.2019.2945171
[11] Moses Garuba, Chunmei Liu, and Alicia Washington. 2008. A Comparative Analysis of Anti-Malware Software, Patch Management, and Host-Based Firewalls in Preventing Malware Infections on Client Computers. In *Fifth International Conference on Information Technology: New Generations (itng 2008)*. IEEE, Vegas, Nevada, 628–632. https://doi.org/10.1109/ITNG.2008.233
[12] Getmonero.org. n.d.. MONERO. https://www.getmonero.org/resources/user-guides/vps_run_node.html
[13] Andy Greenberg. 2011. Crypto Currency. https://www.forbes.com/forbes/2011/0509/technology-psilocybin-bitcoins-gavin-andresen-crypto-currency.html#5f63487a353e
[14] Lester Hio. 2018. Cybercriminals now cryptojacking mobile phones. https://www.straitstimes.com/tech/cybercriminals-now-cryptojacking-mobile-phones
[15] Geng Hong, Zhemin Yang, Sen Yang, Lei Zhang, Yuhong Nan, Zhibo Zhang, Min Yang, Yuan Zhang, Zhiyun Qian, and Haixin Duan. 2018. How You Get Shot in the Back: A Systematical Study about Cryptojacking in the Real World. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (Toronto, Canada) *(CCS '18)*. Association for Computing Machinery, New York, NY, USA, 1701–1713. https://doi.org/10.1145/3243734.3243840
[16] Dhilung Kirat, Giovanni Vigna, and Christopher Kruegel. 2014. BareCloud: Bare-metal Analysis-based Evasive Malware Detection. In *23rd USENIX Security Symposium (USENIX Security 14)*. USENIX Association, San Diego, CA, 287–301. https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/kirat
[17] Radhesh Krishnan Konoth, Rolf van Wegberg, Veelasha Moonsamy, and Herbert Bos. 2019. Malicious cryptocurrency miners: Status and Outlook. arXiv:1901.10794 [cs.CY]
[18] Radhesh Krishnan Konoth, Emanuele Vineti, Veelasha Moonsamy, Martina Lindorfer, Christopher Kruegel, Herbert Bos, and Giovanni Vigna. 2018. MineSweeper: An In-Depth Look into Drive-by Cryptocurrency Mining and Its Defense *(CCS '18)*. Association for Computing Machinery, New York, NY, USA, 1714–1730. https://doi.org/10.1145/3243734.3243858
[19] Seunghyeon Lee, Changhoon Yoon, Heedo Kang, Yeonkeun Kim, Yongdae Kim, Dongsu Han, Sooel Son, and Seungwon Shin. 2019. Cybercriminal minds: an investigative study of cryptocurrency abuses in the Dark Web. In *Network and Distributed System Security Symposium*. Internet Society, San Diego, California, 1–15.
[20] MiningPoolStats. [n.d.]. MiningPoolStats. https://miningpoolstats.stream/monero
[21] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. 2018. Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection. arXiv:1802.09089 [cs.CR]
[22] MoneroMiner.Rocks. 2020. Browser Based Web Mining. https://monerominer.rocks/
[23] Michael Nadeau. 2020. What is cryptojacking? How to prevent, detect, and recover from it. https://www.csoonline.com/article/3253572/what-is-cryptojacking-how-to-prevent-detect-and-recover-from-it.html
[24] Lindsey O'Donnell. 2018. Cryptojacking Campaign Exploits Drupal Bug, Over 400 Websites Attacked. https://threatpost.com/cryptojacking-campaign-exploits-drupal-bug-over-400-websites-attacked/131733/
[25] Sergio Pastrana and Guillermo Suarez-Tangil. 2019. A First Look at the Crypto-Mining Malware Ecosystem: A Decade of Unrestricted Wealth. In *Proceedings of the Internet Measurement Conference* (Amsterdam, Netherlands) *(IMC '19)*. Association for Computing Machinery, New York, NY, USA, 73–86. https://doi.org/10.1145/3355369.3355576
[26] Sergio Pastrana, Daniel Thomas, Alice Hutchings, and Richard Clayton. 2018. CrimeBB: Enabling Cybercrime Research on Underground Forums at Scale, In Proceedings of the 2018 World Wide Web Conference. *WWW '18: Proceedings of the 2018 World Wide Web Conference*, 1845–1854. https://doi.org/10.1145/3178876.3186178
[27] Ruben Recabarren and Bogdan Carbunar. 2017. Hardening stratum, the bitcoin pool mining protocol. *Proceedings on Privacy Enhancing Technologies* 2017, 3 (2017), 57–74.
[28] Jan Rüth, Torsten Zimmermann, Konrad Wolsing, and Oliver Hohlfeld. 2018. Digging into Browser-Based Crypto Mining. In *Proceedings of the Internet Measurement Conference 2018* (Boston, MA, USA) *(IMC '18)*. ACM, New York, NY, USA, 70–76. https://doi.org/10.1145/3278532.3278539
[29] Steven Salzberg. 1991. Distance metrics for instance-based learning. In *International Symposium on Methodologies for Intelligent Systems*. Springer, 399–408.
[30] Bill Guyton Scott Renfro. n.d.. mergecap - Merges two or more capture files into one. https://www.wireshark.org/docs/man-pages/mergecap.html
[31] DENIS SINEGUBKO. 2017. Hacked Websites Mine Cryptocurrencies. https://blog.sucuri.net/2017/09/hacked-websites-mine-crypocurrencies.html
[32] Sorin Soviany, Andrei Scheianu, George Suciu, Alexandru Vulpe, Octavian Fratu, and Cristiana Istrate. 2018. Android Malware Detection and Crypto-Mining Recognition Methodology with Machine Learning. In *2018 IEEE 16th International Conference on Embedded and Ubiquitous Computing (EUC)*. IEEE, 14–21.
[33] Forrest Stroud. n.d.. cryptocurrency mining. https://www.webopedia.com/TERM/C/cryptocurrency-mining.html
[34] Viabtc.com. 2021. Viabtc mining pool. https://www.viabtc.com/
[35] Wikipedia. 2020. Mining pool. https://en.wikipedia.org/wiki/Mining_pool
[36] Foudge Xmrig, SChernykh. 2021. Stratum mining protocol. https://github.com/xmrig/xmrig-proxy/blob/master/doc/STRATUM.md
[37] Foudge Xmrig, SChernykh. 2021. XMRig. https://github.com/xmrig/xmrig
[38] Jun Xu, Jinliang Fan, Mostafa H Ammar, and Sue B Moon. 2002. Prefix-preserving ip address anonymization: Measurement-based security evaluation and a new cryptography-based scheme. In *10th IEEE International Conference on Network Protocols, 2002. Proceedings.* IEEE, 280–289.
[39] Yuqi Yu, Hanbing Yan, Hongchao Guan, and Hao Zhou. 2018. DeepHTTP: Semantics-Structure Model with Attention for Anomalous HTTP Traffic Detection and Pattern Mining. arXiv:1810.12751 [cs.CR]
[40] ZeroDot1. 2019. CoinBlockerLists. https://zerodot1.gitlab.io/CoinBlockerLists Web/

A ALGORITHM PSEUDOCODE

A.1 Local Similarity Table-Naive

Algorithm 1 shows the specific calculation process of naive local similarity table mentioned in 3.2.1. We first compute the naive local similarity for each flow f in the flow set F separately (on line 6). Then we calculate the local interval distance of the flow f (on lines 7-10). Finally, we calculate the local interval similarity of the flow f and add the result to the local similarity table (on lines 11-12).

A.2 Local Similarity Table-Improved

Algorithm 2 shows the specific calculation process of the improved local similarity table mentioned in Section 3.2.1. Compared to Algorithm 1, we add a credible probability coefficient α (on lines 12-17).

Algorithm 1 Local Similarity Table-Naive

Input:
The interval $k = (x_{k-1}, x_k)$ of block sequence;
The flow set, F;
Output:
The local similarity table of F in the interval (x_{k-1}, x_k);
function LST-Naive(x_{k-1}, x_k, F)
    Initialize an empty local similarity table as *LST*, a *dict* in which *key* is the flow name and *value* is the local interval similarity.
    for each f in F **do**
        Initialize the local interval distance as $e_l = x_k - x_{k-1}$;
        for each p in f **do**
            if p in (x_{k-1}, x_k) **then**
                $e_l \leftarrow min\ (e_l, p - x_{k-1})$
        $s_l \leftarrow 1 - \frac{e_l}{x_k - x_{k-1}}$
        LST.add(f,s_l)
    return *LST*;

Algorithm 2 Local Similarity Table-Improved

Input:
The interval $k = (x_{k-1}, x_k)$ of block sequence;
The flow set, F;
Output:
The local similarity table of F in the interval (x_{k-1}, x_k);
function LST-Improved(x_{k-1}, x_k, F)
    Initialize an empty local similarity table as *LST*, a *dict* in which *key* is the flow name and *value* is the local interval similarity.
    for each f in F **do**
        Initialize the local interval distance as $e_l = x_k - x_{k-1}$; Initialize the packet count as $m_l = 0$
        for each p in f **do**
            if p in (x_{k-1}, x_k) **then**
                $m_l \leftarrow m_l + 1$
                $e_l \leftarrow min\ (e_l, p - x_{k-1})$
        Initialize the cumulative distribution probability $CDP = 0$; Set the probability as P; Set the credible probability coefficient as α.
        for $i = 0$ *to* e_l **do**
            $P \leftarrow (\frac{x_k - x_{k-1} - i}{x_k - x_{k-1}})^{m_l} - (\frac{x_k - x_{k-1} - i - 1}{x_k - x_{k-1}})^{m_l}$
            $CDP \leftarrow CDP + P$
        $\alpha \leftarrow 1 - CDP$
        $s_l \leftarrow \alpha * (1 - \frac{e_l}{x_k - x_{k-1}})$
        LST.add(f,s_l)
    return *LST*;

A.3 Global Similarity Table

Algorithm 3 shows the specific calculation process of the global similarity table mentioned in Section 3.2.2. To generate the final similarity result, we first measure the local similarity in each interval by using the LST-Improved method (on lines 7-9). Then we update the *GST* by the strategy mentioned in Section 3.2.2 (on lines 10-21).

Algorithm 3 Global Similarity Table

Input:
The detection time interval, $[t_s, t_e]$;
The flow set, F;
The time series of created block, X;
Output:
The global similarity table of F in the interval $[t_s, t_e]$;
function GST-Improved(t_s, t_e, F, X)
    Initialize an empty global similarity table as *GST*, a *dict* in which *key* is the flow name and *value* is the global interval similarity.
    for each x_i in X **do**
        if $t_s < x_i$ *and* $x_{i+1} < t_e$ **then**
            $LST \leftarrow$LST-Improved(x_i, x_{i+1}, F);
            for each flow f in *GST* **do**
                if f in *LST* **then**
                    $v_s \leftarrow GST.get(f)$
                    $v_l \leftarrow LST.get(f)$
                    $GST.set(f, v_s + v_f)$
                else
                    $v_s \leftarrow GST.get(f)$
                    $GST.set(f, v_s - 1)$
            for each flow f in *LST* **do**
                if f not in *GST* **then**
                    $v_l \leftarrow LST.get(f)$
                    $GST.add(f, v_l)$
    return *GST*;

B PROOF OF THE PROBABILITY OF THE LOCAL INTERVAL DISTANCE

we define a random sequence f with m_k packets obtains the local interval distance e_k in the interval of length n_k.

Lemma B.1. $P(e = e_k) = (\frac{n_k - e_k}{n_k})^{m_k} - (\frac{n_k - e_k - 1}{n_k})^{m_k}$

Proof. First, we consider the case of $e = 0$, which means at least one packet appears at the beginning of the interval. Therefore, the inverse situation $e \neq 0$ is $e > 0$, which means all packets appear at points other than the beginning of the interval. Since we assume that each packet is independent, the probability that a packet appears outside the starting point is $\frac{n_k - 1}{n_k}$ and the probability of m_k packets is $(\frac{n_k - 1}{n_k})^{m_k}$. Hence,

$$P(e = 0) = 1 - P(e \neq 0) = 1 - (\frac{n_k - 1}{n_k})^{m_k} \quad (1)$$

Second, we consider the case of $e = 1$, which means at least one packet appears at the 1 second from the beginning of the interval and no packet appears at the beginning of the interval. Therefore, the inverse situation $e \neq 1$ is $e > 1$ or $e = 0$. $e > 1$ means that all packets are more than 1 second away from the beginning of the

```
{
  "jsonrpc": "2.0",
  "method": "job",
  "params": {
    "blob": "0707d5efb9d6057e95a35...",
    "job_id": "4BiGm3/RgGQzgkTI/xV0smdA+EGZ",
    "target": "b88d0600"
  }
}
```

Figure 18: The example of job message in the Stratum protocol

Table 4: Evaluation result of the cryptomining flows in the background traffic.

Flow (anonymized)	Mining pool	Maximum Similarity
Flow 1	45.9.148.117	0.97
Flow 2	45.9.148.125	0.93
Flow 3	45.9.148.129	0.84
Flow 4	45.9.148.117	0.79
Flow 5	45.9.148.125	0.81
Flow 6	45.9.148.129	0.78
Flow 7	45.9.148.117	0.86
Flow 8	45.9.148.117	0.93
Flow 9	94.130.165.87	0.98

Table 5: Detailed Monero cryptomining pool nodes.

Domain name	IP nodes
sg.minerxmr.com	139.99.62.196, 139.99.68.128
pool.minexmr.com	37.59.44.193, 94.130.164.163, 37.59.54.205, 37.59.43.131
fr.minexmr.com	37.59.54.205, 37.59.44.193
de.minexmr.com	94.130.164.163, 88.99.193.240 94.130.165.85
ca.minexmr.com	158.69.25.77, 158.69.25.71 158.69.25.62
us-west.minexmr.com	147.135.37.31, 51.81.245.40
pool.supportxmr.com	91.121.140.167, 94.23.23.52
web.xmrpool.eu	54.37.7.208
monerohash.com	107.191.99.221, 107.191.99.95
xmr.2miners.com	51.89.96.41
monero.hashvault.pro	131.153.76.130
moneroocean.stream	18.180.72.219
xmr.crypto-pool.fr	163.172.226.137

interval. Hence, $P(e > 1) = (\frac{n_k-2}{n_k})^{m_k}$. Hence,

$$\begin{aligned} P(e=1) &= 1 - P(e=0) - P(e>1) \\ &= 1 - (1 - (\frac{n_k-1}{n_k})^{m_k}) - (\frac{n_k-2}{n_k})^{m_k} \\ &= (\frac{n_k-1}{n_k})^{m_k} - (\frac{n_k-2}{n_k})^{m_k} \end{aligned} \tag{2}$$

Similarity,

$$\begin{aligned} P(e=2) &= 1 - P(e=0) - P(e=1) - P(e>2) \\ &= 1 - (1 - (\frac{n_k-1}{n_k})^{m_k}) - ((\frac{n_k-1}{n_k})^{m_k} - (\frac{n_k-2}{n_k})^{m_k}) - (\frac{n_k-3}{n_k})^{m_k} \\ &= (\frac{n_k-2}{n_k})^{m_k} - (\frac{n_k-3}{n_k})^{m_k} \end{aligned} \tag{3}$$

Finally,

$$\begin{aligned} P(e=e_k) &= 1 - P(e=0) - P(e=1) - \ldots - P(e=e_k-1) - P(e>e_k) \\ &= 1 - (1 - (\frac{n_k-1}{n_k})^{m_k}) - ((\frac{n_k-1}{n_k})^{m_k} - (\frac{n_k-2}{n_k})^{m_k}) - \ldots \\ &- (\frac{n_k-e_k-1}{n_k})^{m_k} \\ &= (\frac{n_k-e_k}{n_k})^{m_k} - (\frac{n_k-e_k-1}{n_k})^{m_k} \end{aligned} \tag{4}$$

□

C DETAILED PAYLOAD FILTERING METHOD

According to the scenarios described in Section 2.2, the cryptomining generally uses the Stratum protocol. Therefore, we extract keywords from the payload of the Stratum protocol as filter conditions. The payload of Stratum protocol is encoded by JSON-RPC messages, and the general encoding form of mining jobs is shown in the Fig. 18 [36]. We extract the ASCII codes of "blob" and "target" as the conditions for load filtering. After filtering, we manually verify the results. In this way, we find 9 flows containing cryptomining behavior in the background traffic, including 4 confirmed mining pool addresses (45.9.148.125, 45.9.148.117, 45.9.148.129, 94.130.165.87).

We supplement that to further verify the capabilities of MineHunter, we use MineHunter to detect these 9 flows and the similarity calculation results are shown in Table 4. Considering some flows with long-term mining behaviors, covering multiple windows (we set the detection window to 2 hours), we select the maximum similarity of each detection window as the final result of a flow, since a cryptomining flow can be effectively discovered as long as it is detected in any window. MineHunter can effectively detect all plaintext labeled cryptomining flows (the maximum similarity greater than the set threshold 0.6), which can indirectly verify the recall of MineHunter.

D DETAILED MONERO CRYPTOMINING POOL NODES

See Table 5.