Hindawi
Wireless Communications and Mobile Computing
Volume 2018, Article ID 9809345, 10 pages
https://doi.org/10.1155/2018/9809345

Research Article

An Efficient Security System for Mobile Data Monitoring

Likun Liu, Hongli Zhang, Xiangzhan Yu, Yi Xin, Muhammad Shafiq, and Mengmeng Ge

School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China

Correspondence should be addressed to Xiangzhan Yu; yuxiangzhan@hit.edu.cn

Received 13 February 2018; Accepted 8 May 2018; Published 11 June 2018

Academic Editor: Lu Wang

During the last decade, rapid development of mobile devices and applications has produced a large number of mobile data which hide numerous cyber-attacks. To monitor the mobile data and detect the attacks, NIDS/NIPS plays important role for ISP and enterprise, but now it still faces two challenges, high performance for super large patterns and detection of the latest attacks. High performance is dominated by Deep Packet Inspection (DPI) mechanism, which is the core of security devices. A new TTL attack is just put forward to escape detecting, such that the adversary inserts packet with short TTL to escape from NIDS/NIPS. To address the above-mentioned problems, in this paper, we design a security system to handle the two aspects. For efficient DPI, a new two-step partition of pattern set is demonstrated and discussed, which includes first set-partition and second set-partition. For resisting TTL attacks, we set reasonable TTL threshold and patch TCP protocol stack to detect the attack. Compared with recent produced algorithm, our experiments show better performance and the throughput increased 27% when the number of patterns is $\mathbf{10^6}$. Moreover, the success rate of detection is 100%, and while attack intensity increased, the throughput decreased.

1. Introduction

More and more mobile data, including bad along with good, emerged and congested the network, which brings challenges to improve system performance and attack detection capabilities. In order to improve this situation, ISPs constantly update the security devices and systems, such as NIDS or NIPS which are the front line of defense against cyber-attacks. A central component of NIDS/NIPS is Deep Packet Inspection (DPI) engine, in which the payload of the packets is inspected to detect predefined signatures of malicious information [1]. The vulnerability is exposed while the set of patterns is getting bigger and bigger over 10^6, and pattern matching algorithm can be taken down because of costing too many resources that lie at the center of most DPI engines. Therefore, efficient pattern matching algorithms are challenge for high performance.

In order to overcome the high performance barrier, multicore architectures provide an opportunity to achieve high performance at a relatively low cost. For each core of a conventional multicore with the traditional processors, it is favourable to adapt coarse-gained parallelism [2]. Previous research focused on the partitions of pattern set and mapping the subsets on parallel processors (cores); therefore, the problem is transformed into a scheduling problem. All the works divide the same length patterns as a minimal subset and suppose different combinational algorithms. However, the ideal result is based on the number of same length patterns uniformly distributed. Similar to buckets, scheduling problem depends on the most time-consuming core. If a subset costs too much time, the whole matching is still slow, and the advantages of multicore will be lost.

And on the other hand, emerging new attacks bring new challenges to the existing security system, such as the recent TTL attack that the adversary inserts a spurious packet with wrong sequence number and short TTL, while the short TTL makes maximum probability of the crafting packet reach NIDS, rather than service provider. After TCP control block (TCB) maintained by NIDS receives manipulating packet, the status will be out of synchronization and the TCP stream will be teardown. At the moment, attacker successfully evades NIDS and sends bad data to server.

Table 1: Performance of AC, WM, and SBOM.

Algorithm	Time complexity	Data structure
AC	$O(n)$	Trie
WM	$O\left(\frac{n}{(m-b+1)\times\left(1-\left((m-b+1)\times r\right)/\left(2\times\lvert\Sigma\rvert^{b}\right)\right)}\right)$	Hash Table
SBOM	$O\left(\frac{n\times\log_{\lvert\Sigma\rvert}mr}{m-\log_{\lvert\Sigma\rvert}mr}\right)$	Factor Oracle

Σ denotes an alphabet set, b is the block size in WM, m denotes the minimum length of pattern, and r denotes the number of pattern sets.

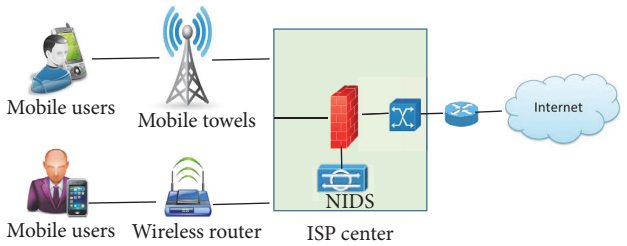

Figure 1: Security system deployment.

In this paper, the main contributions of this work are as follows.

(i) We designed an efficient security system for mobile data monitoring, which is deployed as a bypass to the mobile network, as shown in Figure 1. It is located at the ISP export. The system architecture will be introduced in Section 3.1.

(ii) We propose fined-gained parallel algorithm, a new two-step partition of pattern set. Firstly, we select a better partition from original set-partition and Tan's set-partition. Secondly, we make decision whether or not to split the uneven subsets, and standard deviation is taken as a measure. Finally, divide subsets and map all subsets on cores by AEA algorithm.

(iii) We design a table to record the hops between security system and TCP server, which contains TTL threshold and server hash (ip and port). The table is real-time update through learning TTL value from server packets. When $Server_{TTL} < Table_{TTL}$, the table will be updated. In case of $Client_{TTL} < Table_{TTL}$, and $Client_{seq} + datalen > TCB_{ack}$, drop packet and send alarm.

The paper is organized as follows: Section 2 presents the related work. Section 3 describes our system, fined-gained parallel algorithm, and a method to resist latest attack, followed by some experimental results in Section 4. Finally, we summarize the research in Section 5 with a discussion of the future work.

2. Related Work

For improving performance of NIDS, researchers endeavor to develop multi-core and many-core. Some researchers focused on the architecture, and other researchers devote themselves to the parallelization algorithm research of core component automata. The former pays more attention to system scheduling [3, 4] and cache optimization [5, 6]. The latter is dedicated to efficient pattern matching problem. In this paper, we mainly elaborate the research results of string matching parallelization algorithm.

Similarly, the previous work for string matching algorithm can be categorized into three classes: prefix searching, suffix searching, and factor searching. In prefix search methods, Aho-Corasick algorithm (AC) [7] is the most typified and efficient. The algorithm moved windows by computing the longest common prefix between the text and the patterns. In suffix search methods, the classic algorithm is Wu-Member algorithm (WM) [8], which features a backward searching from right to left within the window. In factor searching methods, the famous Set Backward Oracle Matching (SBOM) [9] is of this kind. It is treated as a combination of prefix searching and suffix searching. The performance of the above classical algorithms is determined by three factors: the size of the alphabet, the minimal length of the patterns, and the number of the patterns (see Table 1). AC performs well on short string, while WM is the opposite. SBOM is the most efficient in practice for long patterns.

The corresponding parallelization algorithm is subdivided into two directions: the text parallelization and pattern sets parallelization. The former cuts the text into multiple subtext which is sent to each core. Around cutting point, the combination of two adjacency subtext boundaries is a big nuisance. So far there is no better way, so the latter got more room for research. We divided the latter into two categories: bit-split and pattern-merge.

Bit-split is suitable for small-scale pattern sets. Hyun-Jin [10] proposed a memory-efficient bit-split deterministic finite automata- (DFAs-) based string matching scheme with multiple string matchers. The target is reducing memory requirements. He used the graph coloring of a unique graph to group iteratively patterns into multiple unique sets. Shervin et al. [11] introduced a pattern grouping algorithm for heterogeneous bit-split string matching architectures. The algorithm is composed of two phases: (1) a seed selection process, which uses a calculation to estimate the correlation between strings, and (2) a seed growing process for mapping strings onto subgroups.

Pattern-merge is applied to large-scale pattern sets; Liu et al. [12] proposed a shortest-path model for the optimal partition finding problem, which is suitable for filtering with the large-scale patterns set. In this approach, the patterns with same length would locate in one subset as a node. The

weight of the edge between any two nodes is the minimum runtime of AC, WM, and SBOM. The optimal partition is finding the shortest-path and consolidated subsets. Tan et al. [2] challenge Liu's optimal partition. They regarded processors as a factor of division and proved the optimal allocation of subsets to processors is NP-hard. $P_1, P_2 \dots P_l$ are subsets; $q_1, q_2 \dots q_s$ are processors. Tan considered two cases, $l \leq s$ and $l > s$, and designed two dynamic programming algorithms to get optimal partition. Two shortcomings are hidden in Tan's strategy. On the one hand, when $l > s$, a Greedy algorithm is applied to scheduling subsets into multicores, but it is easily trapped in the local optimization. A set-partition based genetic algorithm (GA) [13] is proposed to resolve the problem, but it is still easy to fall premature. On the other hand, if some subsets cost too much runtime, according to the strategy by Tan et al., there will be a phenomenon $\min \max_{i=1}^{q}\{T_i\} = \max_{i=1}^{q}\{T_i\}$; this means that the whole runtime always depends on a subset. In this case, neither shortest-path by Liu et al. nor dynamic programming by Tan et al. can achieve the optimal division.

We leverage the idea of Tan et al. [2] to develop a parallel multiple pattern matching algorithm. AEA algorithm, instead of Greedy algorithm, is adopted to schedule subsets and this algorithm can jump out of local optimization and avoid premature. Then standard deviation helps to make decision. On the assumption that standard deviation is greater than tolerable error θ, we further split subsets to satisfy uniformed distribution on each core. In this way, computation cost is minimized.

For attack-resistance, as reported by [14], attacker sent specially crafted packets, especially "insertion" packets. These insertion packets are crafted such that they are ignored by the intended server (nor never reach the server) but are accepted and processed by the NIDS. In insertion packet, the TTL and checksum are manipulated field—a packet with a short TTL value would never reach the intended server and a packet with wrong checksum would be discarded by the server. The basic principle is to destroy the TCB of NIDS. There are three attack points as follows.

(i) **TCB Creation**. A SYN packet, with fake sequence number and short TTL value, is injected while TCP builds a three-way handshake. The insertion packet was firstly sent, followed by a real SYN packet. The NIDS will ignore the real connection because of its "unexpected" sequence number.

(ii) **Data Reassembly**. There are two cases.

(1) Out-of-order data overlapping: for IP fragments, the only difference between insertion packet and real packet is that the former's data is filled with garbage. The insertion packet is sent prior to the real packet. For TCP segments, the order of sending is the opposite. For example, to be sent IP fragment packets are IP_1, IP_2, the sensitive words lie in IP_2, the attacker will insert IP_{2fake} with same offset and length as IP_2 and junk data, and the final order is IP_{2fake}, IP_2, IP_1.

(2) In-order data overlapping: before sending a real packet, a fake packet filled with same header except TTL and junk data is sent.

(iii) **TCB Teardown**. After finishing TCP three-way handshake, client can send a RST or FIN packet to NIDS; it will make TCB of NIDS teardown.

Zhang et al. [15] enhanced attack techniques and proposed two evasion strategies.

(iv) **Resync + Desync**. The sequence number of a SYN insertion packet is out of the expected receive window of the server. After three-way handshake, the client sends a SYN insertion packet. Subsequently, he sends a 1-byte data packet containing an out-of-window sequence number to desynchronize NIDS, and a real request followed.

(v) **TCB Reversal**. This situation has a prerequisite, which is connection established when NIDS receives SYN/ACK. The client will first send a SYN/ACK insertion packet. It will confuse client and server for NIDS.

Based on these technologies, Zhang et al. made further study and dug out a combination program of TCB Creation + Resync/Desync and TCB Teardown + TCB Reversal. Average successful evasion rate was 95.6% and 96.2%.

The above attack methods are mainly based on two aspects. (1) Insertion packet with small TTL or wrong checksum only reaches NIDS not intended server. (2) Some fields of insertion packet are filled with wrong data or garbage data to make TCB of NIDS teardown. The wrong checksum is always invalid because most NIDS will do check. Therefore, we study solutions to combat TTL attacks. The characteristic of the attack is small TTL value. We only need to find the right threshold. The choice of the threshold is crucial, but not easy because the proper threshold value varies from case to case and can be set by experience. We set it by learning from normal server packets. About the second aspect, we subtly design a small additional receive buffer to handle packets with same or unexpected sequence number. The corresponding TCP state machine will also be upgraded.

3. Efficient Security System

3.1. System Architecture. The security system consists of six modules: (1) traffic capture module, (2) protocol stack module, (3) DPI engine module, (4) Alarm/Log module, (5) set-partition module, and (6) attack detection. The first four modules are the original NIDS modules, while the last two are new modules. System architecture is shown as Figure 2. The analysis of each module is as follows.

(i) Traffic capture: traditional receiving packet is triggered by each packet interrupt; if the packet is too fast, the interrupt is too frequent; CPU always handles interrupt; thus other tasks cannot be scheduled, and the performance will reach the bottleneck. The DPDK technology we use in this system adopts the polling-mode drivers for networking, which makes them receive and send packets within the minimum number of CPU cycles (usually less than 80 cycles). Thus the throughput increases significantly.

(ii) Protocol stack module: this module decodes packets and analyzes protocols of transport and application layer.

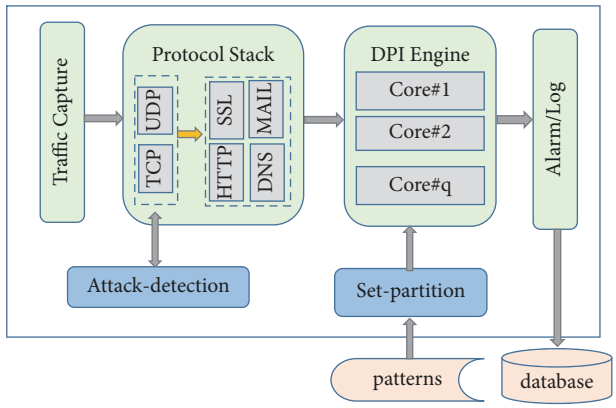

Figure 2: Security system architecture.

In this paper, we focus on TCP because it is the object of attack. TTL value of each TCP packet will be checked, that is, compared with the TTL record table (TRT) in attack detection module. An auxiliary small receive buffer is added to the TCP module for TCP reassembly attack detection.

(iii) DPI engine: DPI is the central component of security system. Its main function is to match the data from protocol stack module with the pattern set. Because this module consumes a large amount of CPU and memory, an efficient parallel matching algorithm is crucial. Each subset maps one automaton, and several automatons are assigned to the same core.

(iv) Alarm/Log: this is output module. If the matching is successful, the module will send an alarm message to the specific device and a log to database.

(v) Set-partition: this module is necessary for large-scale pattern set. An optimal partition algorithm is integrated in this module. Firstly, we get a partition by dynamic programming algorithm and calculate the average point of all the cores. Then AEA is used to schedule subsets to cores. Finally, if the runtime of each core is around the average point, the optimal partition is finished; else a new improved Greedy algorithm we proposed helps to subdivide a subset of the same length. See Section 3.2 for details.

(iv) Attack detection: this module provides common attack and the latest TTL attack detection interface. It can detect the following attacks: (1) common attack: ICMP flood, TCP SYN flood, TCP LAND, UDP flood, and ping of death; (2) the latest TTL attack: TCB Creation with SYN, reassembly out-of-order data, reassembly in-order data, TCB Teardown with RST, TCB Teardown with RST/ACK, TCB Teardown with FIN, TCB Creation + Resync/Desync, and TCB Teardown + TCB Reversal. A TTL table and a new TCP receive buffer aid in completing the detection. See Section 3.3 for details.

3.2. High Performance

3.2.1. Preliminary Knowledge. A set of patterns P, where P_i is a subset with same length, is

$$P = \{P_1, P_2, \ldots, P_n\} \tag{1}$$

We denote runtime of a subset as

$$T^i = \min\left\{T^i_{AC}, T^i_{WM}, T^i_{SBOM}\right\}, \quad 1 \le i \le n. \tag{2}$$

An optimal partition (Tan et al. [2]) using dynamic programming algorithm is

$$\begin{aligned} P_{dynamic} &= \{S_1, S_2, \ldots, S_k\}, \\ S_i &= \{P_r, \ldots P_s\}, \quad 1 \le r < s \le n \end{aligned} \tag{3}$$

Comparing runtime,

$$T_{P_{dynamic}} \le T_P \tag{4}$$

3.2.2. Optimal Patterns Set Decomposition and Schedule. The strategy for set-partition includes two steps:

(i) First choice of set-partition.

(ii) Second set-partition decision.

Step 1 (first choice of set-partition).

Problem 1. Given a set of patterns P and c cores, to find its decomposition $P_1, P_2, \ldots, P_l$, and scheduling of mapping subsets to cores,

$$\begin{aligned} \min \quad & \max_{i=1}^{c} \left\{ \sum_{j=1}^{n_i} T^i_j \right\} \\ \text{s.t.} \quad & P_i \cap P_j = \emptyset, \\ & \bigcup_{l}^{i=1} P_i = P, \\ & 1 \le i \ne j \le l. \end{aligned} \tag{5}$$

P_i is with same length, n_i is the number of pattern sets selected by core i, and the running time of each core is $\sum_{j=1}^{n_i} T^i_j$.

Problem 1 is a multiobjective combination problem, which is NP-hard. For the problem, we firstly divide the patterns into subsets and then schedule them. For first set-partition, we compare two methods: original set-partition and Tan's [2] dynamic programming algorithm. Before scheduling, $T_{P_{dynamic}} \le T_P$, but the result may be different after scheduling. We take an opposite example to prove it.

(1) Initialization set $P = \{P_1, P_2, P_3, P_4, P_5\}$

(2) Runtime $T_P = \{1, 1, 6, 7, 8\}$

(3) Tan's method $P_{dynamic} = \{S_1, S_2\} = \{\{P_1, P_2\}, \{P_3, P_4, P_5\}\}$

$$T_{P_{dynamic}} = \{2, 14\} \tag{6}$$

(4) Cores=2, Greedy algorithm to schedule

$$\begin{aligned} SCH_P &= \{\{8,1,1\},\{7,6\}\}, \\ \max_P &= 13 \\ SCH_{P_{dynamic}} &= \{2,14\}, \\ \max_{P_{dynamic}} &= 14 \end{aligned} \tag{7}$$

(5) $\max_{P_{dynamic}} > \max_P$

The result shows that Tan's method is not optimal; the reason is that the partition $P_{dynamic}$ is actually based on one core to get, which is not suffice to show that the multicore scheduling result using Greedy algorithm is optimal. We leverage the result to choose $\min\{\max_{P_{dynamic}}, \max_P\}$ as the first partition.

Schedule. Since Greedy algorithm is easier to fall into local optimization, we design annealing evolution algorithm (AEA) instead.

We set some notations as follows:

Pop: population;

G_{max}: evolution generations;

F: the fitness function.

P_{cross}: cross probability;

$P_{mutation}$: mutation probability;

ε: premature decision sign;

C: temperature coefficient of cooling;

T: annealing initial temperature;

T_{min}: lower temperature limit.

Functions are defined as follows:

(i) The fitness function F: the optimal set-partition problem is to find the maximum; we set fitness function as

$$f(x) = \frac{1}{1 + c - g(x)}, \quad c \geq 0,\ c - g(x) \geq 0 \tag{8}$$

where c is an estimate of the boundary of the objective function.

(ii) P_{cross} and $P_{mutation}$ have a great influence on the convergence of algorithm; thus we use the following formula for adaptive tuning. Assume the average of fitness function is $f'_{avr} = (1/T)\sum_{i=1}^{T} f'_i$, where f'_i is fitness function for individual i at the Gth generation and T is population size at the Gth generation. The fitness of the best individual is f'_{best}. All individuals whose fitness is less than f'_{best} are averaged to get f'_{avr}. Let $\varepsilon = f'_{best} - f'_{avr}$, as ε increased, the algorithm has a premature trend; in order to prevent it, we use the following functions:

$$\begin{aligned} P_{cross} &= 1 + a\frac{-1}{1 + e^{-k_1 * \varepsilon}} \\ P_{mutation} &= b + b\frac{-1}{1 + e^{-k_2 * \varepsilon}} \end{aligned} \tag{9}$$

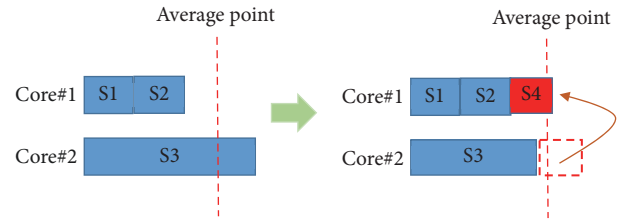

Figure 3: Second set-partition schematic.

where a = 0.4, b = 0.04, $k_1 > 0$, $k_2 > 0$, k_1 *and* k_2 are constant.

Pseudocode is shown in Algorithm 1.

AEA mixes GA algorithm and SA algorithm. GA is the main process. While GA converges, SA furthermore changes the initialization population to overcome the premature and GA runs again, until SA is convergent. Since GA can get optimal set-partition with high probabilities, SA should be quickly annealed, so the temperature coefficient of cooling should be chosen to be larger to speed up the convergence. Similarly, the parameter G_{max} of GA should be set larger too, because if it is small, SA may not be working or annealing times are few, the reasonable value should be near SA convergence point.

Step 2 (second set-partition decision). This part describes a metric that reflects the result of second set-partition decision and how to split subsets.

Measure: standard deviation: it is used to quantify the amount of variation or dispersion of a set of data values.

$$s = \sqrt{\frac{1}{c-1}\sum_{i=1}^{c}\left(T_{c_i} - \overline{T}\right)^2} \leq \theta, \quad 0 < i < c,\ 0 < \theta < 1 \tag{10}$$

where θ is tolerable error.

If $s > \theta$, it shows a large deviation; at this time, a second set-partition is necessary. According to the experiment, it is often caused by such subsets with long length, large size, and WM and SBOM algorithm. A Greedy strategy is applied to subdivide the subsets. As shown in Figure 3, in the core#2 with $\max_{i=1}^{c}\{T_i\}$, S_3 is cut into $\{S_3, S_4\}$ and the new S_4 is assigned to the core#1 with $\min_{i=1}^{c}\{T_i\}$. After the division and reorganization, we need to recalculate s and judge the convergence condition ($s \leq \theta$). If convergence condition is not satisfied, subdivision is continued.

For how to cut a subset, we choose the number of patterns as a measure and find it in relation to running time. We chose 20 sets of experimental data and adopt least-square method for data fitting. The result shows that the relation is linear; that is, the running time for WM and SBOM increases with the number increase, so we divide subset according to the ratio of time

$$N_{S_{new}} = \frac{T_{max} - \overline{T}}{T_{max}} N_S \tag{11}$$

(1) Initialize the $Pop_0 = N, P_{cross} = p_{c0}, P_{mutation} = p_{m0}$, $G = 1$
(2) FOR each generation t *in* M DO
(3) FOR each individual p_i *in* Pop_t DO
(4) calculate the fitness F_i
(5) ENDFOR
(6) IF $G \geq G_{max}$, THEN
(7) find the optimal set-partition.
(8) return
(9) ENDIF
(10) calculate ε_t, p_c, p_m
(11) IF $\varepsilon_t \geq \varepsilon$, THEN
(12) calculate p_{ct}, p_{mt}
(13) crossover operation with probability p_{ct}
(14) mutation operation with probability p_{mt}
(15) ELSE
(16) crossover operation with probability p_{c0}
(17) mutation operation with probability p_{m0}
(18) ENDIF
(19) generate next population Pop_{t+1}
(20) IF $pop_{t+1-n} = pop_{t+1}$, THEN
(21) switch (f'_{best}, f'_{best-1}) as new Pop_0
(22) annealing, $T = CT$, $t = 0$
(23) IF $T \leq T_{min}$
(24) stop and return set-partition.
(25) ENDIF
(26) GOTO 2
(27) ENDIF
(28) ENDFOR

Algorithm 1: Annealing evolution algorithm.

(1) calculate standard deviation s
(2) IF $s > \theta$ THEN
(3) find core i with $\max_{i=1}^{c}\{T_i\}$, core j with $\min_{j=1}^{c}\{T_j\}$
(4) find the subset S_k *with* $\max T_{c_i}^{k}$
(5) split $S_k = \{S_{k1}, S_{k2}\}$, move S_{k2} to core j
(6) recalculate T_i, T_j *and* $\overline{T}$
(7) GOTO 1
(8) ELSE
(9) find the optimal set-partition, return
(10) ENDIF

Algorithm 2: Greedy algorithm to split subsets.

where $N_{S_{new}}$ is the pattern number of new subset which will be moved to the core with T_{min} and N_S is the pattern number of the subset with $\max_{i=1}^{m}\{T_{max}^{i}\}$.

Pseudocode is shown in Algorithm 2.

3.3. Attack-Resistance. Attack detection module includes common attack detection and TTL attack detection. For common attack, which includes ICMP flood, TCP SYN flood, TCP LAND, UDP flood, and ping of death, we set thresholds to prevent attacks as other NIDS. TTL attack detection is the focus of our research. Many attacks against security system are borrowing TTL value and the package with short TTL will never reach the intended server but security system (see Figure 4).

Table 2

Server identify (hash with ip and port)	TTL threshold

Through the analysis of TTL attack characteristics, we assume that servers are credible and design a TTL table to record the minimum hops between security system and server. The record hops are TTL threshold to prevent attacks. The table fields are shown in Table 2.

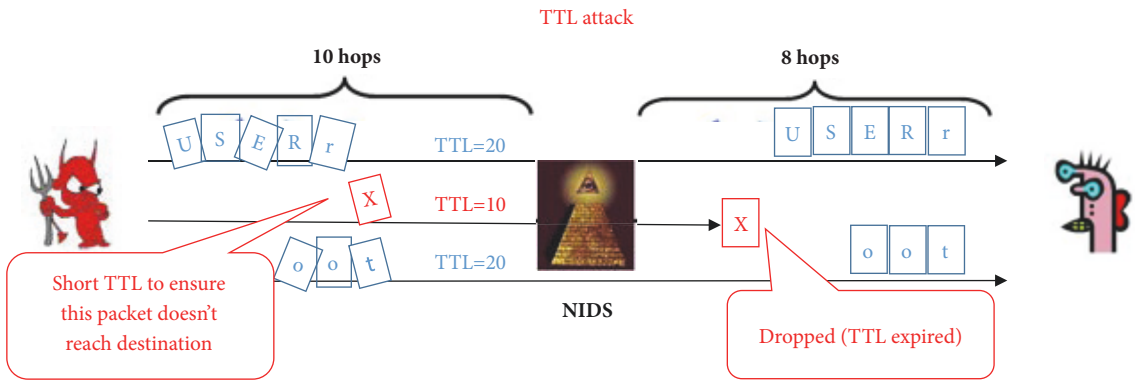

Figure 4: TTL attack.

Table 3: The relationship between defence type and attack method.

Control packet	TCB Creation, TCB Teardown, TCB Reversal, Resync + Desync
Data packet	Data Reassembly: out-of-order data overlapping and in-order data overlapping Data Desync

TTL threshold is learned from server packets. A packet is sent from different system, and the TTL value is different, but the common ground is that the initial value is 2^n, or $2^n - 1$, generally is one of $base[64, 128, 255]$. The security system computes hops as follows:

$$hops = \min_{i=1}^{3} \{|base_i - serverpkt_{TTL}|\} \quad (12)$$

All the IP packets from server need to check the table. If $hops < Table_{TTL}$, the table will be updated with $hops$. In case of $Client_{TTL} < Table_{TTL}$, we suspect that it is an attack packet, but this is not sufficient to prove it; there are two reasons:

(i) Dynamic changes in the network topology and the minimum hops between security system and server may be changed. If $hops_{real} > Table_{TTL}$, the attack packet with TTL value in $[Table_{TTL}, hops_{real}]$ will be missed.

(ii) In addition to setting short TTL, the attacks will operate other fields to complete attacks.

The latest TTL attack [15] combines short TTL with a variety of strategies. The heart of the strategies is to destroy or trick the TCP connection state of security system. To break the heart, we have defense in two aspects: control packet and data packet. Table 3 shows the relationship between the two aspects and attack methods.

Control packet: the attackers aim to destroy the TCB by sending insertion control packet. In order to tear the intention, we build a TCB link for same 4-tuple, so security system will not drop control packets while multiple SYNs or SYN/ACKs are coming. Each TCB has a timer which is updated by new packet. When time is out, the attacked TCB will be released, while the normal one is kept. In addition, a limited number of links are set to prevent continuous attacks. The TCB link will make TCB Creation, TCB Reversal, and Resync + Desync fail. For TCB Teardown, when insertion RST/FIN packet comes, the TCB will not be immediately released and reset the expiration time. During the time, if there is still coming packet, we can make sure it is an attack and keep the TCB.

Data packet: the essence of the attack is that the attacker generates junk data to protect sensitive data to escape security system. Attack techniques regard fraud as purpose, and attack features are forging data packet with same or incorrect control information. In this situation, security system can easily be misled and lose control of data. For same control information (Data Reassembly), we design an auxiliary small receive buffer for TCP reassembly attack detection. When the same sequence number packets come, the first is recorded in normal TCP receive buffer, and the second is recorded in the auxiliary buffer. We do not identify the authenticity of the data, because it costs too much time to parse protocol. TTL check should be a good helper to make decision that the one with short TTL will be abandoned. For incorrect control information (Data Desync), if the sequence number is out of window, there may be two situations: it is an attack packet or the system did not capture the previous packet. The packet is kept in auxiliary buffer and we determine which one by identifying the next sequence number from client and ACK number from server. If $Seq_{out} + Data_{out} = ACK_{server}$, this indicates that server has correctly received the packet, it is not an attack, and the window of security system will be updated. If $Seq_{next} + Data_{next} = ACK_{server}$, obviously, the next packet is right, and the one in auxiliary buffer is an attack packet.

In general, the security system detects the latest attack in two steps: (1) check TTL, (2) check control packet and control information in data packet. Experiments show that security system can effectively detect such attack.

Table 4: Contrast for original sets and Tan's subsets. Runtime is in milliseconds.

Thread Number	Original Greedy	Original AEA	Tan's Greedy	Tan's AEA
1	2955	2915	2935	2913
2	2902	2910	2890	2890
3	2893	2902	2833	2833
4	2870	2893	2810	2832

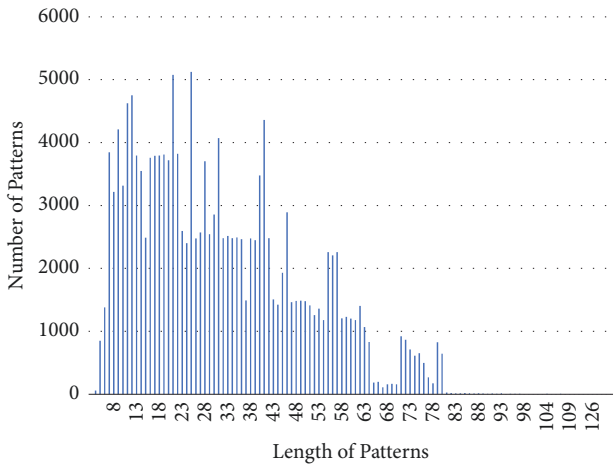

Figure 5: The number of patterns with different length.

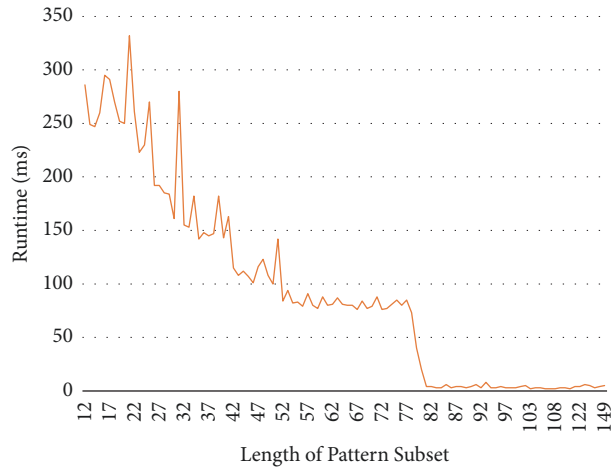

Figure 6: Runtime of each subset with the same length.

4. Experimental Results

4.1. Experimental Environment. We use a system with Intel Core(TM) i7-6700HQ CPU 2.60 GHz, quad-core, where each core has two hardware threads, 32KB L1 data cache (per core), 256KB L2 cache (per core), and 8MB L3 cache (shared among cores), 16G Memory. The system runs Linux CentOS 7, and DPDK is installed to capture packets. Our program runs with 8 threads in parallel, one thread is used to capture packet and preprocess, three threads are used to parse in parallel the application protocol, the remaining four threads work for parallel matching. In addition, we have prepared two windows computers for sending packets, which install CSNAS tool.

The pattern set consists of 10^6 URL strings; there are four sources: (1) extracting from the Snort rules, (2) selecting from URL blacklist, (3) parsing URL of traffic which is captured at the export of the HIT network center, and (4) generating URL randomly according to the length ratio from URL of traffic. Duplicate strings were eliminated. The input text consists of 5×10^5 strings; half is randomly extracted from the pattern set and another half is randomly generated. As Figure 5 shows, the number of patterns is not uniformly distributed. There are 115 different lengths, the shortest is 3, and the longest is 149.

4.2. High Performance Experiment. In this section, all the runtimes represent the algorithm execution time and the input text is introduced in Section 4.1. We first divide the pattern set into subsets by length and choose the best runtime from AC, WM, and SBOM for each subset. The experiment shows that when the length > 10, WM and SBOM performs more efficiently. Figure 6 shows the runtime of each subset with same length; while being combined with Figure 5, it can be drawn that the larger the subset, the longer the runtime.

After dynamic programming, the number of subsets is 21. Then we schedule them into four threads, respectively, by the Greedy algorithm and the AEA algorithm.

According to Table 4, we can see Tan's algorithm yielded better results. We analyze that there are two reasons: (1) in Tan's algorithm, the pattern with length less than 10 is merged into a subset, which is using AC. For small subset, the runtime of AC is not affected by the number of patterns. (2) The number is very small for length greater than 78, and after merging these subsets, the number of hash tables is reduced from 41 pairs to 1 pair that reduce memory usage. In contract to the Greedy and AEA, no matter original subsets or Tan's subsets, AEA is prior to Greedy.

After first set-partition, we will conduct second set-partition experiment. We choose Tan's AEA as input. The results are shown in Table 5; it is obvious that the second set-partition is relatively uniform, and the max runtime is less than the first.

Above we have tested the matching effect of parallel algorithm, then we will start the traffic test. The traffic is captured at the export of the HIT network centre, and we got two files in size 4.2G, 5.1G. Two windows computers are used to send packets. We chose CSNAS tool to send packets and the

Table 5: Contrast for first and second set-partition. Runtime is in milliseconds.

Thread Number	1	2	3	4
First	2913	2890	2833	2832
Second	2875	2868	2864	2861

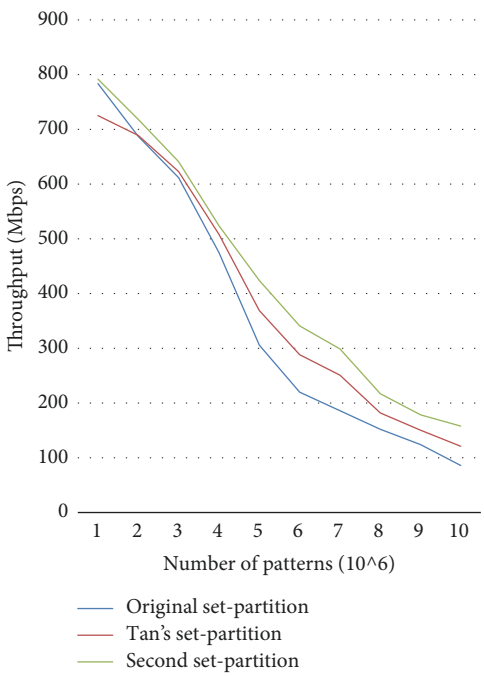

Figure 7: Throughput for different set-partitions.

Table 6: Throughput for different attack intensity.

%	10	20	30	50	80	100
Throughput (Mbps)	154	147	136	96	79	72

speed of sending is limited at 300-400Mbps, and the mode is set cycle.

The experiment compared the throughput of original set-partition, Tan's set-partition, and two-step set-partition, as shown in Figure 7. As the pattern set increases, the throughput of all methods decreases. The second set-partition we proposed performs best, such that the throughput increased 27% than Tan's when the number of patterns is 10^6.

4.3. Attack Detection Experiment. We constructed a total 700 of TCP attack streams; the method is randomly extracted from two capture files and inserted the attack packet. There are 100 for each attack type. The third computer is used to send attack packets with CSNAS tool. Attack intensity is adjustable by limiting the sending speed of three computers.

The result of attack detection experiment is shown in Table 6, and the success rate of detection is 100%, as attack intensity increases, the throughput decreases.

5. Conclusions

In order to monitor the mobile data in ISP or enterprise, we design a security system to detect malicious information and attacks. We demonstrate the architecture of the entire system and give solutions of high performance and anti-attack. For high performance, we propose a new two-step partition of pattern set. In first set-partition, the best set-partition is chosen from several algorithms, and AEA schedule algorithm replaces Greedy algorithm. In second set-partition, standard deviation is taken as a measure, in order to make each core with similar running time, we propose an equilibrium cut strategy to reorganize subsets. For anti-attack, the security system is based on TTL check, TCB link is added against control packet attacks, and an auxiliary small receive buffer is added against data packet fraud. In the final stage, we, respectively, perform our experiments from the above two aspects and the results show that the security system has high performance and anti-attack capacity.

In the future, we plan to push forward the work in two aspects: on one hand, we will focus on resilience to algorithmic complexity attacks. On the other hand, according to the detected attacks, we will focus on mining the intention of the attackers and classifying the attackers.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported in part by National Key Research and Development Program of China with Grant no. 2016QY05X1000 and no. 2016QY01W0100.

References

[1] Y. Afek, A. Bremler-Barr, Y. Harchol, D. Hay, and Y. Koral, "Making DPI Engines Resilient to Algorithmic Complexity Attacks," *IEEE/ACM Transactions on Networking*, vol. 24, no. 6, pp. 3262–3275, 2016.

[2] G.-M. Tan, P. Liu, D.-B. Bu, and Y.-B. Liu, "Revisiting multiple pattern matching algorithms for multi-core architecture," *Journal of Computer Science and Technology*, vol. 26, no. 5, pp. 866–874, 2011.

[3] H. Jiang, G. Zhang, G. Xie, K. Salamatian, and L. Mathy, "Scalable high-performance parallel design for Network Intrusion Detection Systems on many-core processors," in *Proceedings of the 9th ACM/IEEE Symposium on Architectures for Networking and Communications Systems, ANCS 2013*, pp. 137–146, San Jose, CA, USA, October 2013.

[4] J. Haiyang, X. Gaogang, and S. Kave, "Load Balancing by Ruleset Partition for Parallel IDS on Muti-Core Processors," in *Proceedings of the 22th International Conference on Computer Communications and Networks*, Nassau, Bahamas, 2013.

[5] M. Jamshed, J. Lee, S. Moon et al., "Kargus: A highly-scalable software-based intrusion detection system," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security, CCS 2012*, pp. 317–328, NC, USA, October 2012.

[6] J. Nam, M. Jamshed, B. Choi, D. Han, and K. Park, "Scaling the performance of network intrusion detection with many-core processors," in *Proceedings of the 2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pp. 191-192, Oakland, CA, USA, May 2015.

[7] A. V. Aho and M. J. Corasick, "Efficient string matching: an aid to bibliographic search," *Communications of the ACM*, vol. 18, pp. 333–340, 1975.

[8] S. Wu and U. Manber, "A fast algorithm for multi-pattern searching," Technical Report TR-94-17, 1994.

[9] C. W. Beate, "A string matching algorithm fast on the average," in *Proceedings of the 6th Colloquium on Automata, Languages and Programmin*, pp. 118–132, Graz, Austria.

[10] H. Kim, "Memory-efficient parallel string matching scheme using distributed pattern grouping without matching vectors," *IEEE Electronics Letters*, vol. 52, no. 13, pp. 1124–1126, 2016.

[11] S. Vakili, J. P. Langlois, B. Boughzala, and Y. Savaria, "Memory-Efficient String Matching for Intrusion Detection Systems using a High-Precision Pattern Grouping Algorithm," in *Proceedings of the the 2016 Symposium*, pp. 37–42, Santa Clara, California, USA, March 2016.

[12] L. Ping, L. Yan, and T. Jian, "A partition-based efficient algorithm for large scale multiple-strings matching," in *In Proceedings of the 12th ACM Internet Conference. String Processing and Information Retrieval*, pp. 399–404, Buenos Aires,Argentina, 2005.

[13] J. Liu, F. Li, and G. Sun, "A parallel algorithm of multiple string matching based on set-partition in multi-core architecture," *International Journal of Security and Its Applications*, vol. 10, no. 4, pp. 267–278, 2016.

[14] K. Sheharbano, J. Mobin, and D. A. V. Philip, "Towards Illuminating a Censorship Monitors Model to Faicilitate Evasion," in *Proceedings of the In Proceedings of the 3th USENIX Workshop on Free and Open Communications on the Internet*, Washington, D.C. USA, 2013.

[15] W. Zhongjie, C. Yue, and Q. C. S. Zhiyun, "A Closer Look at Evading Stateful Internet Censorship," in *In Proceedings of the 17th ACM Internet Measurement Conference*, London, UK, 2017.