FNF: Flow-Net Based Fingerprinting

Bo Fu[&], **Yang Xiao**[+], and **Hui Chen**[*]

[&]Cisco, 170 West Tasman Dr., San Jose, CA 95134 USA

[+]Dept. of Computer Science, The Univ. of Alabama, Tuscaloosa, AL 35487-0290 USA

[*]Dept. of Mathematics and Computer Science, Virginia State Univ., Petersburg, VA 23806 USA

***Abstract*—A flow-net technique is a logging methodology that can build comprehensive system and network logs and help track events and event relationships in computer and network systems. In this paper, we propose flow-net based fingerprinting (FNF) to capture the characteristics of a system or network behavior. Furthermore, we propose a fingerprint lookup algorithm to solve the fingerprint matching problem, i.e., given a behavior, to check whether a flow net logging contains the behavior with the same fingerprint. We future apply FNF into detecting the fingerprints of malicious behaviors in computer and network systems. Finally, evaluation results from experiments demonstrate better performance than other schemes.**

Keywords—Flow-net, intrusion detection, fingerprint;

I. INTRODUCTION

Current computer and network systems are subject to many types of intrusion attacks. Intrusion Detection Systems (IDS's) are developed to monitor system and network behaviors aiming to discover signs of intrusions, such as unauthorized access and data modification [1]. Upon detecting suspected intrusions, an IDS typically sends alerts to system and network administrators [2, 3, 4]. The ability to detect the intrusions increases in importance as the computers and networks are increasingly integrated into the systems that we rely on for the reliable and efficient functioning of our society [2]. In an IDS, one or more sensors collect system or activity data called events, which are organized as logs that are typically organized as files or stored in databases. An IDS can search the logs for intrusions [6, 7, 8, 12].

The existing logging schemes of IDS's typically treat the events individually without explicitly considering "relationship" among events. Flow-net is a logging technique to build comprehensive logs, to help track events, and to maintain explicit relationships of events in the logs [17, 20].

Similar to human fingerprints, there are also unique characteristics for activities called fingerprints [22]. Fingerprint will be a powerful tool for intrusion detection and forensics [22].

Typically, a behavior in a computer system or a network may be composed of multiple events. The relationship among the events can be critical in intrusion detection. In this paper, we propose a Flow-Net based Fingerprinting (FNF) method and apply it to detect intrusion by examining both the events and the relationship among the events. Each type of behaviors causes certain events, and the correlation of these events has patterns that can be referred to as a *fingerprint* of the type of behaviors. Intrusions to a computer system or a network can be detected by checking the fingerprints of different types of behaviors appearing in flow-net logs.

In this paper, we propose flow-net based fingerprinting (FNF) to capture the characteristics of a system or network behavior. Furthermore, we propose a fingerprint lookup algorithm to solve the fingerprint matching problem, i.e., given a behavior, to check whether a flow net logging contains the behavior with the same fingerprint. We future apply FNF into detecting the fingerprints of malicious behaviors in computer and network systems. Finally, evaluation results from experiments demonstrate better performance than other schemes.

The rest of this paper is organized as follows. In Section II, we propose FNF. A fingerprint lookup algorithm is proposed in Section III. FNF is applied to intrusion detection in Section IV. Evaluation results are presented in Section V. We conclude our work in Section VI.

II. FLOW-NET BASED FINGERPRINT (FNF)

A. A formal definition of Flow-net

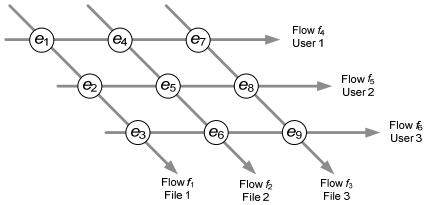

Fig. 1 An example flow-net

A computing system includes many entities (e.g., files, processes, or users). Each entity has a flow. An entity's flow is defines as an event list of all the events in temporal order that are associated with the entity [17]. All the flows form a flow-net. Fig. 1 shows an example of flow-net, in which, all the events associated by user 1 and arranged in temporal order form the user 1's flow, i.e., flow f_4, and all the events executed on file 1 and arranged in temporal order is file 1's flow, i.e., flow f_1. The flows in Fig. 1 contain explicitly the relationships among the events.

We denote a flow-net as a directed graph $fn = (E, R)$ in which E is the set of nodes and R is the set of edges. In the graph, a node e represents an *event*, and a directed edge $r_{e_i,e_j} = r_{i,j} = (e_i, e_j)$ represents the *relationship* that Events e_i and e_j are two consecutive events and e_i occurs before e_j. Relationship $r_{i,j}$, sometimes written as r_{e_i,e_j}, can easily capture the causal relation that e_i triggers e_j or other types of relationship or correlation. A flow consists of a set of events and the relationships of consecutive events, and thus can be represented as $f=\{r_{1,2}, r_{2,3}, \ldots r_{n-1,n}\}$ that shows the relationships of two consecutive events in their temporal order. This notation is also used to describe a part of a flow, starting from event e_1 ending at event e_n. For convenience, in this paper, we use "flow" as an abbreviation of a part of a flow. Each event has certain properties or attributes (i.e., timestamp, event name). Each flow also has certain properties or attributes (i.e., flow name, flow type).

B. Behavior

We refer a behavior in a computer system or a network to as an action of a set of related actions that can be described semantically and can have clearly defined beginning and ending points. A behavior is typically composed of multiple events. Flow-net is designed to record all these events and their relationships. Therefore, we may consider a behavior as a set of

correlated events that are aimed at fulfilling certain purpose or performing a function.

Formally, a *behavior* in flow-net *fn=(E, R)* is denoted as a graph $b = (E_b, R_b)$ in which E_b represents the nodes of the graph and is the set of events of the behavior and in which R_b represents the edges of the graph and is the set of the relationships among the events in E. Note that $b=(E_b, R_b)$ is a subgraph of *fn*, i.e., $E_b \subseteq E$ and $R_b \subseteq R$.

C. Fingerprint

A fingerprint represents the pattern of a type of behaviors in computer systems and networks. We extract the fingerprint of the type of behavior aiming to identify all instances of the "establishing TCP connection" behaviors since they belong to a single type of behaviors. For convenience, we simply refer an instance of a type of behaviors as a behavior from this point onward.

Flow-net records many details of an event. The details consist of the values of various attributes of the event, such as timestamp, user name, file name, host name and IP address. The type of behaviors to which an event belongs to is determined not only by the values of the attributes of the event but also its context in a flow-net, i.e., the related events that also belong to the same behavior. Although the attributes and their values of an event are important, the type that a behavior belongs to is of the utmost concern. We assign an *essential type* to each event. For instance, the essential type of a network event can be "*receiving an IP packet*". We only use events' essential types to extract fingerprints of various types of behaviors. For instance, the fingerprint of behavior "establishing TCP connection" can be six related events with essential types of "sending SYN," "receiving SYN", "sending SYN-ACK", "receiving SYN-ACK", "sending ACK", and "receiving ACK." Since the relationships of these events are already preserved in the flow-net, the values of the attributes of the events become non-essential to infer the event relationships. The fingerprint can be expressed by the event relationships and their *essential types* rather than actual values of the attributes.

Denote *e'* be the *abstract event* of event *e*. Abstract event *e'* of event *e* contains the event's essential type and does not contain the values of event *e*'s attributes. For an event *e*, we define a mapping denoted as *proto* from the event *e* to its *abstract event e'*, i.e., *e' = proto*(*e*). Abstract event *e'* of event *e* is a prototype of event *e*. In the following, we use *event prototype* and *abstract event* interchangeably. It is not rare that two events are of the same essential type.

Having obtained abstract events of all events, we replace event *e* by its abstract event *e'* in relationship set *R* to obtain the relationships among abstract events. The definition of flow-net then remains the same except that the flow-net is now on abstract events (i.e., event prototypes). The actual mapping between an event and its abstract event is determined by the specific characteristics of the event. To determine common abstract events for TCP/IP networks, it requires an analysis on TCP/IP protocols.

A behavior's fingerprint is the abstraction of the behavior that is expressed as its abstract events and the relationship among the abstract events, which have a specific pattern.

Given a behavior $b = (E_b, R_b)$, for any event *e* in event set E_b, we obtain its prototype e' (i.e., abstract event e'). The set of all the event prototypes is called *event prototype set* E_b', which is formally defined as $E_b' = \{e' \mid e' = proto(e) \text{ and } e \in E_b\}$. For any relationship $r_{e_i,e_j} \in R_b$, we can easily obtain a corresponding relationship $r_{e_i',e_j'}$ in *b*'s fingerprint since $e_i'=proto(e_i)$, $e_j'=proto(e_j)$, $e_i, e_j \in E_b$, and $e_i', e_j' \in E_b'$. The set of the relationships between event prototypes, denoted as R_b', is called *abstract event relationship set (or event prototype relationship set)*. The aforementioned process implies a straightforward method to get a behavior's fingerprint once a behavior is defined by its events, i.e., to replace each event with its prototype in E_b and R_b to obtain E_b' and R_b'. Given events e_1, e_2, e_3, and e_4, and relations r_{e_1,e_2} and r_{e_3,e_4}, we have two same event prototype relationships $r_{e_1',e_2'} = r_{e_3',e_4'}$ if and only if $proto(e_1) = proto(e_3)$ and $proto(e_2) = proto(e_4)$.

Formally, given behavior $b = (E_b, R_b)$, *b*'s *fingerprint* is denoted as $fp_b = fingerprint(b) = (E_b', R_b')$ in which E_b' is the event prototype set corresponding to the *b*'s event set E_b, R_b' is the relation set corresponding to the *b*'s relation set R_b.

Different behaviors that have the same fingerprint belong to a type of behaviors. In other words, we can use the fingerprint to exactly and uniquely denote a type of behaviors. Given two behaviors b_1 and b_2, the two behaviors have the same fingerprints, i.e., $fingerprint(b_1) \equiv fingerprint(b_2)$ if and only if the following conditions hold,

(1) There is a one-to-one correspondence between the event prototypes of the fingerprint graph, such that two event prototypes are consecutive in $fingerprint(b_1)$ if and only if their corresponding event prototypes are consecutive in $fingerprint(b_2)$. This condition ensures that the two fingerprints/graphs are isomorphic.

(2) For any relationship r_{e_i,e_j} in b_1, we must be able to find a relationship r_{e_k,e_l} in b_2 such that $proto(e_i)=proto(e_k)$ and $proto(e_j)=proto(e_l)$.

(3) For any relationship r_{e_i,e_j} in b_2, we must be able to find a relationship r_{e_k,e_l} in b_1 such that $proto(e_i)=proto(e_k)$ and $proto(e_j)=proto(e_l)$.

Conditions (2) and (3) ensure that the event prototype sets and the event prototype relationship sets in the two fingerprints are identical to each other.

If any one of above three conditions is left out, the fingerprints of behaviors b_1 and b_2 may not be the same. For instance, when $R_{b_2}' = R_{b_1}' \cup \{r_{e_m',e_n'}\}$ where $e_m', e_n' \in R_{b_2}'$ and $e_m', e_n' \notin R_{b_1}'$, condition (2) may hold, but condition (3) does not hold, in which case, the two fingerprints are not the same.

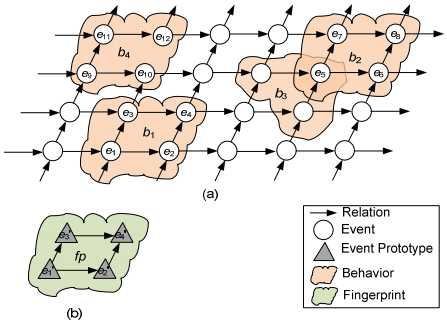

Fig. 2 A behavior and its fingerprint. In this figure, $e_1'=proto(e_1)=proto(e_5)=proto(e_9)$, $e_2'=proto(e_2)=proto(e_6)=proto(e_{10})$, $e_3'=proto(e_3)=proto(e_7)=proto(e_{11})$, $e_4'=proto(e_4)=proto(e_8)=proto(e_{12})$, $r_{e_1',e_2'} = r_{e_5',e_6'}$, $r_{e_2',e_4'} = r_{e_6',e_8'}$, $r_{e_3',e_4'} = r_{e_7',e_8'}$, and $r_{e_1',e_3'} = r_{e_5',e_7'}$. Therefore, Behavior b_1 and b_2 have the same fingerprint, called *fp*.

Fig. 2 shows an example of some behaviors and their fingerprints. In this figure, there are two behaviors, b_1 and b_2, which are composed of different events, and the two graphs

denoted by b_1 and b_2 are isomorphic. Both of the behaviors have the same fingerprint, denoted as *fp*, i.e., *fp* = *fingerprint*(b_1) = *fingerprint*(b_2), because e_1' = *proto*(e_1) =*proto*(e_5), e_2'=*proto*(e_2)=*proto*(e_6), e_3'=*proto*(e_3) = *proto*(e_7), e_4' = *proto*(e_4) = *proto*(e_8), $r_{e_1',e_2'} = r_{e_5',e_6'}$, $r_{e_2',e_4'} = r_{e_6',e_8'}$, $r_{e_3',e_4'} = r_{e_7',e_8'}$, $r_{e_1',e_3'} = r_{e_5',e_7'}$, and there does not exist a relationship in either R_{b_1}' or R_{b_2}' that cannot be found in the event prototype set of the other behavior. However, behavior b_4's fingerprint is not *fp* because $r_{e_2',e_4'} \neq r_{e_{10}',e_{12}'}$ since there is a relationship between two abstract events in the prototype relationship set R_{b_4}' that cannot be found a match in R_{b_1}' (or R_{b_2}'), or vice versa. Behavior b_3's fingerprint is not *fp* either since there are fewer events in b_3 than those in b_1 (or b_2).

Having introduced flow-net fingerprint, we can view a flow-net as a group of fingerprints instead of a collection of individual events. Note that some fingerprints may overlap with each other, such as behaviors b_2 and b_3 in Fig. 1. By introducing the flow-net fingerprint, we can focus on the identification of various types of behaviors without unnecessary costly examination of attributes of the events.

III. FLOW-NET FINGERPRINT MATCHING

Giving a flow-net fingerprint of a type of behaviors, we are interested in identifying the type of behaviors from flow-net logs. We refer the process of identifying occurrences of a type behavior in flow-net logs fingerprint matching.

Since both flow-net fingerprints and flow-nets are organized as directed graphs in which each vertex has a "color", i.e., the type of event that the vertex represents, the fingerprint matching process is to identify from flow-net subgraphs that are of the same graphs of the fingerprints of the given type of behavior, which is in effect a subgraph isomorphism for colored directed graphs. Subgraph isomorphism problem in general is a NP-complete problem [14, 19]. Subgraph isomorphism for colored directed graph can be less computational expensive if certain structure exists in the graph.

We propose a behavior lookup algorithm that identifies the behaviors that match the given fingerprint in the flow-net. In the remainder of this paper, we make the following assumptions: 1) As a log file, the data of flow-net, which includes event information, time stamp, relationship, etc., is stored completely; 2) The stored data of flow-net is reliable without being tampered with.Behavior LookupAlgorithm

We propose a behavior lookup algorithm to solve the **Fingerprint Matching Problem (FMP)**, i.e., given a behavior *b* and a flow-net *fn*, check whether *fn* contains a behavior b_2 such that *fingerprint*(*b*)=*fingerprint*(b_2). To solve this problem, we have to obtain *b*'s fingerprint *fp* first and then check in *fn* to find whether or not there is any behavior with a fingerprint that is *fp* as well. We know that behavior, fingerprint, and flow-net are all denoted as graphs. As discussed above, such problem is in effect a subgraph isomorphism for colored directed graphs. Although general algorithms for solving such graph problem have been proposed, we show that an efficient algorithm exists to solve the problem due to the unique structure of flow-net fingerprints and flow-nets.

We propose an algorithm called *Fingerprint Lookup Algorithm* (FLA)**,** to solve FMP. As shown in Fig. 3, FLA has a function called *contain* that checks whether behavior *b*'s fingerprint is contained in flow-net *fn*. Before executing the function, we first call *getFingerprint* function to obtain *b*'s fingerprint *fp*, and then check if any behavior in *fn* has the same fingerprint as *fp*. If such a behavior exists in *fn*, we say *fp* matches (a part of) *fn*. To check if *fp* matches any part of *fn*, we have to select prototype *e'* of the head event in *fp* where the head event is the event that happens the very first in *fp* or a rare event in *fp*. Then check in *fn* to see whether or not there is an event *e* such that *e'*=*proto*(*e*). After getting *e*, we start a new process that checks whether *fp* matches this part of *fn* by comparing two graphs. Two events or two event prototypes have the relationship denoted by the flow in the flow-net. If two events are consecutive in a flow, then they are neighbors. The *relationship* function checks whether two events or events prototypes are consecutive. In the following match function, we recursively check an event's and event prototype's neighbors. If all pairs of the events/event prototypes have the same relationship, then the fingerprint matches the flow-net.

We may let the algorithm returns *true* if and only if the flow-net contains exactly a given fingerprint, an input parameter of the algorithm. However, sometimes a behavior may slightly change its details without changing its basic characteristics. For example, a Denial of Service Attack (DoS Attack) may include a different number of service requests, and this leads to different fingerprints for different occurrences of the same DoS attack. Therefore, in order to detect whether a behavior's fingerprint is contained in a flow-net, we have to tolerate a minor difference between the given fingerprint and that of the behavior in the flow-net to detect the type of behaviors with minor variations.

For Fingerprint *fp*, we count the number of its event prototypes and relationships. We denote the number as *count*(*fp*). For behavior *b*, we count the number of its events and relationships. We denote this number as *count*(*b*). Assuming that Fingerprint fp_1 and fp_2 are similar, we count the number of events and relationship in fn_2 that are different than those in fp_1, and denote this *difference value* as *diff*(fp_1, fp_2). We define *difference ratio dr* as *dr*(fp_2, fp_1)=*diff*(fp_1, fp_2)/*count*(fp_1). Actually, in the definitions of the difference value and the difference ratio above, we can replace the fingerprint with a behavior and do not change the meaning of the definitions. Therefore, assuming that fingerprint *fp* and the fingerprint of behavior *b* are similar, we also have the *difference ratio dr* as *dr*(*b*, *fp*)= *diff*(*fp*, *b*)/*count*(*fp*).

Fig. 2 shows an example for calculating the difference ratio. In Fingerprint *fp* shown in Fig. 2(b), there are 4 event prototypes and 4 relationships among these events. Therefore, *count*(*fp*)=8. We have e_1'=*proto*(e_9), e_2'=*proto*(e_{10}), e_3'=*proto*(e_{11}), e_4'=*proto*(e_{12}), and thus, Fingerprint *fp*'s event prototypes are same as behavior b_4's events' prototypes. Also, the only difference between *fp* and b_4 is that b_4 has one relationship less than *fp* so that *diff*(*fp*, b_4)=1. Therefore, we have *dr*(b_4, *fp*) = *diff*(*fp*, b_4)/*count*(*fp*) = 0.125. Note that the value of the difference ratio must always be in the interval [0, 1]. A value of 0 of the difference ratio implies that the given behavior matches the given fingerprint exactly.

Based upon the above definitions, we define the **Proportional Fingerprint Matching Problem (PFMP)** as follows, given behavior *b*, flow-net *fn*, and a real number *d* ($0 \le d \le 1$), check whether or not *fn* contains a behavior b_2 such that $dr(b_2, fingerprint(b)) \le d$. The real number *d* is a threshold called *difference ratio limitation*.

Fingerprint Lookup Algorithm (FLA):
Input: Behavior *b*, Flow-net *fn=(E, R)*, double *d*
Output: Boolean value

```
contain(Behavior b, Flow-net fn, double d)
{  Fingerprint fp = getFingerprint(b)
  nfp = |E| + |R|
  efp = getHeadEvent(fp)
  For (efn ∈ E) {
    if (efp == proto(efn)) {
      n = matchCount(efp, efn)
      If (n/nfp ≥ 1 - d) return true
      Else return false}}}

matchCount(Event Prototype efp, Event efn)
{  n = 0
  Xfn ={xfn | isNeighbor(efn, xfn) = true
             AND !xfn.visited}
  Xfp ={xfp | isNeighbor(efp, xfp) = true
             AND !xfp.visited}
  For (xfn ∈ Xfn) {
    xfn' = proto(efn)
    If (xfn' ∈ Xfp) {
      n = n + 1
      xfn.visited = true
      xfp.visited = true
      if (relationship(efn, xfn)
            == relationship (efp, xfp){
        n = n + 1
        n = n + matchCount(x', x)}}}
  return n}
```

Fig. 3 Flow-net Fingerprint Lookup Algorithm (FLA).

As shown in Fig. 3, FLA is capable of solving PFMP. The input to the *contain* function are *b*, *fn*, and *d*, and the output is a Boolean value that implies whether the behavior *b*'s fingerprint is contained in the flow-net *fn* within the *difference* ratio limitation *d*.

In FLA, when the threshold *d* is set as 0, i.e., no variation of the type of behavior is allowed, it solves FMP; when d is great than 0, it solves PFMP. We use FLA to solve PFMP, i.e., FMP is a special case of PFMP when *d* is set to 0.

The introduction of the difference ratio limitation *d* in FLA allows us to look up an intrusion even when the intrusion changes its details slightly. The use of the difference ratio limitation makes FLA more powerful because it is not rare for an intrusion to change its behavior slightly every time. We have indicated that FMP is a subgraph isomorphism problem for colored directed graph. Existing solutions of the problem do not support slight changes of the given subgraph. Therefore, the existing algorithms that solve the subgraph isomorphism problem cannot solve PFMP that allows a slight variation of a type of behavior. This is the motivation behind the proposed difference ratio of two fingerprints and the FLA.

In FLA, the key is to find an event prototype and then recursively check if its relationship with neighbors match the flow-net. Therefore, we should find a suitable event prototype *e'* for the *contain* function to work efficiently. For example, if we want to check whether or not a TCP three-way handshake exists in a flow-net, we should use the event prototype "sending SYN-ACK" as the starting event for prototype *e'* since this event prototype is not as common as other kinds of event prototypes in the flow-net. Some events in the flow-net are rare. Rare events are used as starting point of the looking up process in the flow-net. By using a rare event, the PFMP or FMP can be solved more efficiently using FLA.

We can improve the performance of the algorithm by adding some assisting data structure in the flow-net. When generating a new part of the flow-net, we check the new generated fingerprints on this flow-net in real time. We store the name of the fingerprint, such as TCP connection and file transfer, in the assisting data structure. Therefore, when we want to check if a behavior's fingerprint is in the flow-net, we can simply check the assisting data structure. This revised algorithm saves time than the first one but costs more space to maintain the assisting data structure.

IV. FNF BASED INTRUSION DETECTION

A. FNS based IDS

Flow-net stores all the behaviors including their events and the relationships between events. Intrusions are harmful behaviors to computer systems and networks. To look up an attack, it is fundamentally to look up a type of behavior.

We assume that a host's or a network's behaviors are all stored in the flow-net, and we can look up the attack's fingerprint in the flow-net in order to detect the attack. In other words, detecting an attack is equal to looking up this attack's fingerprint in the flow-net. the FNF-based IDS uses FLA to look up intrusions from flow-net logs using the fingerprints of the intrusion behaviors. Fingerprints may overlap, and furthermore, one fingerprint may contain another fingerprint. Looking up each fingerprint by FLA is an independent work regardless of the overlap situation of the fingerprint.

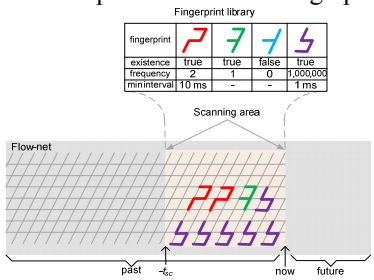

Fig. 4 An illustration of FNF based IDS. In the fingerprint library, the colorful patterns are actually visual notations of the fingerprints. The scanning area is from the time $-t_{sc}$ to now. We only detect attacks in the scanning area of the flow-net.

Fig. 4 shows an illustration of the FNF based IDS. We store the attacks' fingerprints in a fingerprint library. Our goal is to detect these target attacks in the flow-net.

For each target attack, the FNF based IDS stores the following values, a) the target intrusion's fingerprint, b) whether the fingerprint of the intrusion exists in the flow-net, c) how many times the fingerprint appears in the flow-net, and d) the minimum time interval between two occurrences of the type of behavior represented by the fingerprints in the flow-net.

The values (fingerprint, existence, frequency, and minimum interval) in the fingerprint library as shown in Fig. 4 denote the aforementioned properties and can be used for setting a threshold of the intrusion alert.

Although Flow-net stores all behaviors of a host, in a real-time intrusion detection, we do not need to detect all the intrusions taking place in the entire history. In Fig. 4, the flow-net stores the behaviors in current time and the past, and the future of the flow-net is empty since flow-net does not predict the future. In a real-time intrusion detection, we may only need to detect intrusions in a small time window in the past. However, for other purposes, such as log auditing, we may detect the behaviors in a very large time window in the past. No matter how "old" the behaviors that we aim to detect, the time window between the oldest time from which we start to detect intrusions and the present time is called the scanning window. The part of flow-net that consists of events happening during the scanning window is called the scanning area. In other words, we only detect intrusions in the scanning area of the flow-net. For real-time intrusion detection, the scanning window and scanning area should be small, and therefore, the overhead

caused by the flow-net logging will not lower the performance of the FNF based IDS.

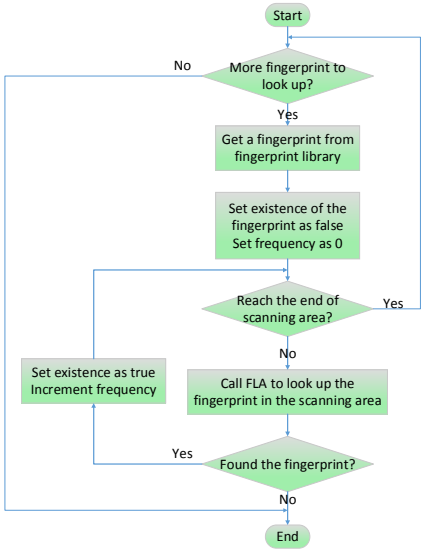

Fig. 5 The flow chart of the FNF based IDS

Fig. 5 shows a flow chart of the FNF based IDS. According to Fig. 5, we first select one fingerprint from the fingerprint library and then use FLA to look up the fingerprint from the scanning area of the flow-net. We will set the existence field of this fingerprint as true or false in the fingerprint library based on whether or not this fingerprint is found. If it is found, we also increment the frequency of the occurrences of the fingerprint. We continue to look up the fingerprint until we reach the end of the scanning area. If multiple occurrences of the fingerprint are found, we record the minimum time interval among the occurrence; otherwise, we record the interval as infinity. We then look up the next fingerprint in the fingerprint library.

Therefore, the FNF based IDS is able to detect intrusions that have unique fingerprints. The fingerprint library can be used for showing whether an intrusion appears in the flow-net, and thus can be used for triggering the intrusion alert to the administrator.

B. FNF based IDS *for Anomaly Detection*

An intrusion has its unique fingerprint that can be detected by the FNF based IDS scheme. However, some attacks, such as Denial-of-Service attack (DoS attack), use normal behaviors but repeat for a great number of times to drain the victim's resource.

In Fig. 5, the fingerprint can be a representative for a type of normal behavior. The frequency field in the fingerprint library records how many times the type of behavior that a fingerprint represents appears in the flow-net, and the minimum interval field records the minimum interval between two occurrences of the same type of behavior.

We propose a new method to denote the repeated appearance of a type of behavior. We use the star symbol (*) to denote that an event prototype and a relationship appear many times. Assume that we have a type of behavior's fingerprint $fp_1 = \left(\{e_1', e_2'\}, \{r_{e_1',e_2'}\}\right)$, and if this type of behavior appears many times, we can denote it as $fp_1* = \left(\{e_1'*, e_2'*\}, \{r_{e_2',e_3'}*\}\right)$. A DoS attack may be composed by repetition of several different regular behaviors, such as fp_1, fp_2, etc., and we can denote the DoS attack by $fp_{DoS} = fp_1* \cup fp_2* \cup \cdots$.

Shown in Fig. 5, after looking up fingerprints in the scanning area, we update the fingerprint library and get the existence frequency, and minimum interval of each fingerprint. The intrusion alert may be triggered by checking the frequency and minimum interval fields in the fingerprint library. If the fingerprint is for an intrusion, the algorithm in Fig. 5 detects intrusions; if the fingerprint is for normal behavior and a type of normal behavior repeats too many times and too frequently, an alert may be triggered by a detected anomaly.

C. Applications of FNF based IDS on TCP/IP Attacks

A three-way handshake for the TCP connection establishment is normally composed by six consecutive SYN and ACK events. Flow-net is able to record not only these six events but also their relationship. The pattern of the events and the relationship between the events in the three-way handshake can be referred to as the fingerprint of the three-way handshake. Therefore, flow-net is able to record the network behaviors and we can use the FNF based IDS to detect TCP/IP attacks using the fingerprint.

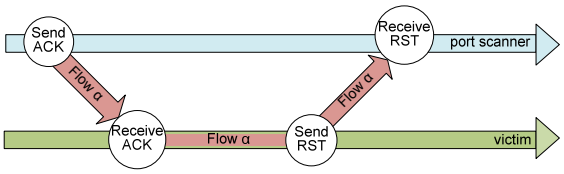

Fig. 6 Flow-net of window scanning

Port scanning may be considered as an attack that are attempted too frequently and too many in a short period of time. Port scanning sends certain TCP or UDP segments to certain port addresses on a host with the goal of prying the vulnerabilities of the host [15, 16, 18]. As a type of port scanning, window scanning is conducted by sending an ACK segment from the attacker to the victim host. After receiving the ACK segment, the victim responds by an RST segment which contains a window size field. If window size equals 0, then the attacker is aware that the port is closed; else the port is open.

Window scanning takes place only when the ACK and RST events are not related with other segment sending events. Fig. 6 shows the flow-net of a window scanning and its fingerprint can be denoted as follows.

Event e_1="send ACK", e_2="receive ACK", e_3="send RST", e_4="receive RST".

$$fp_{window\ scanning} = \left(\{e_1', e_2', e_3', e_4'\}, \{r_{e_1',e_2'}, r_{e_2',e_3'}, r_{e_3',e_4'}\}\right).$$

The events e_2 and e_3 along with the relation r_{e_2,e_3} on the victim is a significant implication of the attack. Then the aforementioned fingerprint or the part of the fingerprint on the victim side can be added to the fingerprint library for the FNF based IDS. Similarly, we can add other TCP/IP attacks' fingerprints to the fingerprint library of the FNF based IDS for intrusion detection. This example is very specific for TCP/IP intrusion that may result to a misleading consideration about the victim's cooperation.

V. EVALUATION

We developed simulation programs in Java programming language to test the FNF based IDS.

A. Comparison of FNF based IDS and log based IDS

We evaluate the time to find a known intrusion from flow-net logs using flow-net based fingerprint and that from non-relational logs. The time is the indicator how quickly the FNF based IDS and the traditional log based IDS can identify an intrusion. We assume that there are in total N events in either the flow-net log or the non-relational log. We assume that an attack is composed of n (n<<N) events. The objective is to

measure the time needed to find all n evens in either the flow-net log or the non-relational log. We denote these n events as e_1, e_2, …, e_n. In the non-relational log, the relationships among the N events are not maintained explicitly and the N events are stored in the order when they were written to the log. In the flow-net log, the relationships among the N events are explicitly maintained. Relationships in the flow-net log are stored as references (i.e., addresses) to each other and the references allow traverse to the related event directly using its reference.

Since the n events of the intrusion in the non-relational log are not necessarily next to each other and there is no direct reference among the events, we have to search sequentially among all the N events for the n events. On the contrary, the n events are referenced to each other in the flow-net log, when one event of the n events is found and the rest of the n-1 events can be located using the references among them, i.e., we only need to trace along the relationships to locate all the n events of the attack in the flow-net log. As shown in Fig. 7, each event has two related events, and we only need to check these two events to determine which one of them is part of the intrusion.

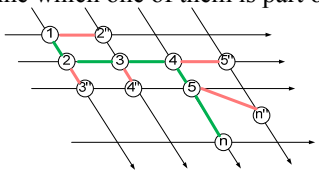

Fig. 7 Event tracing by logging and fingerprint.

We assume that a) in the non-relational log, it takes t_e time to locate next event in the log and b) in the flow-net, it takes the same amount of time to locate a related event. We aim to compare the time of locating all the n events of the attack in both of the logs starting from event e_1 in both logs.

In the non-relational log, it takes $(N-1)$ searches to locate the first event in the worst case scenario since we have to traverse all the N events to find out the n events. Similarly, if any event among the n-1 event stills need $(N-1)$ searches, the cost is $T_l = [(N-1) + (N-2) + (N-3) \ldots + (N-n)]t_e = (2N-1-n)nt_e/2$. In the flow-net log, because the log stores the events' relationships using references of the events, we can trace along the intrusion's n events without looking up all the N events after the first event is located. However, it takes $(N-1)$ searches to locate the first event in the worst case as well. In case of Fig. 7, we need to trace both neighbors of an event, and it takes $T_f = (N-1)t_e + 2(n-1)t_e = (N-2n-3)t_e$ to locate all the n events of the attack.

In the aforementioned case, let $T_f = T_l$, then $n = N + \frac{3}{2} \pm 2\sqrt{N^2 + N + \frac{33}{4}}$. This result means when $n < N + \frac{3}{2} + 2\sqrt{N^2 + N + \frac{33}{4}}$, we have $T_f < T_l$. Apparently, $\frac{3}{2} + 2\sqrt{N^2 + N + \frac{33}{4}} > 0$. Therefore, it is ensured $T_f < T_l$ when $n<N$. Since typically $n<<N$, it takes significantly less time using flow-net logs than using non-relation logging.

B. Performance Analysis of FNF based IDS

We analyze the computational performance of the FNF based IDS, which calls the FLA algorithm repeatedly in order to find all intrusions that match the corresponding fingerprints in the fingerprint library. One execution of the FLA algorithm is to find behaviors that match a fingerprint in the library. Therefore, in the FNF based IDS, the FLA algorithm is executed in the loop by which we search behaviors in flow-net logs matching all fingerprints in the fingerprint library as shown in Fig. 5. The FLA algorithm solves PFMP that is essential in the intrusion detection. Apparently, the difference ratio limitation d affects the time cost of the FLA. We look for an optimal value of d. The FLA algorithm that takes a difference ratio limitation d as the parameter is executed in the loop until we get all the matched behaviors in the flow-net. The time cost for the FNF based IDS are composed by two parts: a) the time for executing the FLA algorithm in the loop and b) the time for updating the fingerprint library and handling and storing the behaviors that match the fingerprint. The value of d affects both of these two time costs.

On one hand, We have to check more event prototypes and relationships in the FLA. Therefore, as d decreases, the time cost for checking the behaviors increases. For a Behavior b in the flow-net fn, the time cost for checking if b matches Fingerprint fp is denoted by $t_b(d, fp)$. As d decreases, $t_b(d, fp)$ increases. The total time cost for checking all the behaviors is $\sum_{b \in fn} t_b(d, fp)$.

On the other hand, as d increases, an increased number of behaviors match the input fingerprint, and therefore, the time cost handling and storing all the matched behaviors in Flow-net fn is larger. We assume that the time cost for handling and storing one behavior is a constant value T_{const}, and the number of behaviors that match the Fingerprint fp is denoted as $n_{match}(d, fp)$. As d increases, $n_{match}(d, fp)$ increases. The total time cost for handling and storing all matched behaviors is $n_{match}(d, fp) \cdot T_{const}$.

Therefore, in the FNF based IDS, the total time cost for getting all the matched behaviors with a given difference ratio d in Flow-net fn is $T_{allmatch}(d, fp, fn) = \sum_{b \in fn} t_b(d, fp) + n_{match}(d, fp) \cdot T_{const}$. We are looking for a value d that minimizes $T_{allmatch}$. In other words,

$$d = \arg\min_d \{T_{allmatch}(d, fp, fn)\}$$

$$= \arg\min_d \left\{ \sum_{b \in fn} t_b(d, fp) + n_{match}(d, fp) \cdot T_{const} \right\}.$$

C. Performance simulation of the FNF based IDSscheme

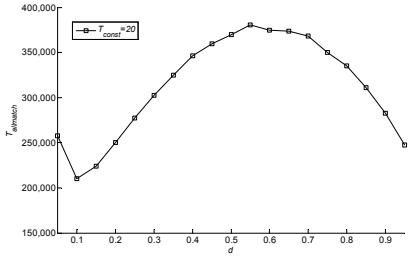

Fig. 8 The time for matching all fingerprints when T_{const}=20

Given a fingerprint and a difference ratio limitation d, the previous subsection shows the calculation of the total time called $T_{allmatch}$, for getting all the matching behaviors by comparing to the fingerprints in the fingerprint library. The difference ratio reflects the difference between the fingerprint and the matched behaviors. $T_{allmatch}$ is affected by both d and the time cost for handling and storing one matched behavior, T_{const}. We simulate a flow-net with 10000 behavior types. We provide a fingerprint library and get all the behaviors with fingerprints that have a different ratio less than d when compared to the given fingerprints. Then, we process the matched fingerprints

by storing them in a different location. The total time of these two parts of time cost is denoted as $T_{allmatch}$.

Given a value of d and a given fingerprint, we can look up the behaviors that are similar to the fingerprint. We want to know an optimal value of d such that the value of $T_{allmatch}$ is minimum, and this means that that we are looking for an optimal value of d that saves time for behavior matching and handling.

In Figs 8 and 9, $T_{allmatch}$ is the y-axis and the value of d is the x-axis. In Fig. 8, we set T_{const}=20, and it shows that when d=*0.1*, we have the smallest value of $T_{allmatch}$. As shown in Fig. 8, a very small value of d (such as 0.05) causes a large value of $T_{allmatch}$ because a smaller d requires a longer time for fingerprint matching as explained in the previous subsection. The value of $T_{allmatch}$ gets larger after d=0.1 because a larger d causes more numbers of matched fingerprints. Thus, the handling time is longer.

However, when d is approaching 1, the value of $T_{allmatch}$ is decreasing because only a small amount of time is spent to compare the given fingerprint and a behavior. The small amount of time is enough to make sure that their similarity is more than *1-d*. Therefore, when d is close to *1*, the time for fingerprint matching is very little although the time for behavior handling is very large. Thus, in this situation, the total time, $T_{allmatch}$, is relatively small. This is why the value of $T_{allmatch}$ drops down when d approaches *1* in Fig. 8.

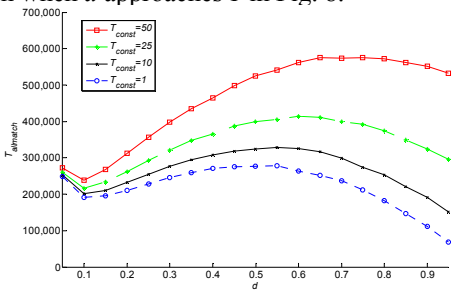

Fig. 9 The time for matching all fingerprints when T_{const} varies

Fig. 9 shows the same trend of $T_{allmatch}$, corresponding to different values of T_{const}. An extremely small value of T_{const} makes $T_{allmatch}$ drops down quickly, and an extremely large value of T_{const} makes $T_{allmatch}$ drop down slowly as d approaches to *1*. These simulation results show that our design and analysis in the previous sections are reliable.

VI. CONCLUSION

In this paper, we proposed a formal definition of flow-net, based upon which we propose an accurate description of behaviors in computer and network systems. Each type of behavior has its unique fingerprint. We propose a Flow-net based Fingerprint (FNF). Furthermore, we proposed a fingerprint lookup algorithm to solve the fingerprint matching problem. We future applied FNF into detecting the fingerprints of malicious behaviors in computer and network systems. Finally, evaluation results from experiments demonstrate better performance than other schemes.

REFERENCES

[1] V. Marinova-Boncheva, "A Short Survey of Intrusion Detection Systems," Problems of Engineering Cybernetics And Robotics, 2007.

[2] A. Jones, and R. Sielken, "Computer System Intrusion Detection: A Survey," Technical Report, 1999.

[3] S. Axelsson, "Intrusion Detection Systems: A Survey and Taxonomy," Technical Report, 2000.

[4] K. Scarfone and P. Mell, "Guide to Intrusion Detection and Prevention Systems (IDPS)," Computer Security Resource Center. 2007.

[5] K. Wang, G. Cretu, and S. J. Stolfo, "Anomalous Payload-Based Network Intrusion Detection," Recent Advances in Intrusion Detection. Springer Berlin. 2011.

[6] M. Sebring, E. Shellhouse, M. Hanna, and R. Whitehurst, "Expert Systems in Intrusion Detection: A Case Study," In Proceedings of the 11th National Computer Security Conference, pp. 74–81, Baltimore, Maryland, 1988.

[7] K. Jackson, D. DuBois, and C. Stallings, "An Expert System Application for Network Intrusion Detection," In Proceedings of the 14th National Computer Security Conference, pp. 215–225, Washington, D.C., 1991.

[8] D. Anderson, T. Frivold, and A. Valdes, "Next-generation intrusion-detection expert system (NIDES)," Technical Report SRI-CSL-95-07, Computer Science Laboratory, SRI International, 1995.

[9] K. Ilgun , R. Kemmerer, and P. Porras, "State Transition Analysis: A Rule-Based Intrusion Detection Approach," IEEE Transactions on Software Engineering, Vol. 21, pp. 181–199, 1995.

[10] S. Eckmann1, G. Vigna1, and R. Kemmerer, "STATL: An attack language for state-based intrusion detection," Journal of Computer Security, Vol. 10, No. 1–2, pp. 71–103, 2002.

[11] C. Michael and A. Ghosh, "Simple, State-Based Approaches to Program-Based Anomaly Detection," ACM Transactions on Information and System Security, Vol. 5, No. 3, pp. 203–237, August 2002.

[12] F. Sailhan and J. Bourgeois, "Log-Based Distributed Intrusion Detection for Hybrid Networks," Proceedings of the 4th Annual Workshop on Cyber Security And Information Intelligence Research: Developing Strategies to Meet the Cyber Security And Information Intelligence Challenges Ahead, pp. 1–3, 2008.

[13] P. Porras and R. Kemmerer, "Penetration State Transition Analysis: A Rule-Based Intrusion Detection Approach," Proceedings of the Eighth Annual Computer Security Applications Conference, December 1992.

[14] J. Ullmann, "An algorithm for subgraph isomorphism", Journal of the ACM, Vol. 23, pp. 31–42, 1976.

[15] W. Du, "Attack Lab: Attacks on TCP/IP Protocols", Laboratory for Computer Security Education, 2010. Available: http://www.cis.syr.edu/~wedu/seed/Labs/Attacks_TCPIP/TCPIP.pdf

[16] M. Tanase, "IP Spoofing: An Introduction," 2003. Available: http://www.symantec.com/connect/articles/ip-spoofing-introduction

[17] Y. Xiao, "Flow-Net Methodology for Accountability in Wireless Networks," IEEE Network, Vol. 23, No. 5, Sept./Oct. 2009, pp. 30-37.

[18] R. Shirey, "Internet Security Glossary," RFC2828. Available: http://tools.ietf.org/html/rfc2828

[19] S. A. Cook, "The complexity of theorem-proving procedures", Proc. 3rd ACM Symposium on Theory of Computing, pp. 151–158, 1971.

[20] Y. Xiao, K. Meng, and D. Takahashi, "Accountability using Flow-net: Design, Implementation, and Performance Evaluation," (Wiley Journal of) Security and Communication Networks, Vol.5, NO. 1, pp. 29–49, Jan. 2012.

[21] S. Lim and A. Jones, "Network Anomaly Detection System: The State of Art of Network Behaviour Analysis," Proceedings of the 2008 International Conference on Convergence and Hybrid Information Technology (ICHIT '08), pp. 459-465, 28-30 Aug. 2008.

[22] D. Takahashi, Y. Xiao, Y. Zhang, P. Chatzimisios, and H.-H. Chen, "IEEE 802.11 User Fingerprinting and Its Applications," (Elsevier) Computers and Mathematics with Applications, Vol. 60, No. 2, July 2010, pp. 307-318.