TR-IIS-16-002

Model-Based Anomaly Detection on Network Services

Shun-Wen Hsiao, Yeali S. Sun, Meng Chang Chen, Hui Zhang

June 15, 2016 || Technical Report No. TR-IIS-16-002
http://www.iis.sinica.edu.tw/page/library/TechReport/tr2016/tr16.html

Model-Based Anomaly Detection on Network Services

Shun-Wen Hsiao[1], Yeali S. Sun[2], Meng Chang Chen[1], Hui Zhang[3]

[1] Institute of Information Science, Academia Sinica,
No. 128 Sec. 2, Academia Rd., Taipei 11529, Taiwan
{hsiaom,mcc}@iis.sinica.edu.tw
[2] Department of Information Management, National Taiwan University,
No.1, Sec. 4, Roosevelt Rd., Taipei 10617, Taiwan
sunny@im.ntu.edu.tw
[3] School of Computer Science, Carnegie Mellon University,
5000 Forbes Avenue, Pittsburgh, PA 15213, U.S.A.
hzhang@cs.cmu.edu

Abstract. The key hypothesis to anomaly detection assumes anomalous behaviors are suspicious from a normality point of view. This work provides a new perspective, network service, to model network activity for detecting anomalies. Past models often suffer from lacking of model normality verification, only including particular behavior aspect, and focusing on individual model. To confront them, we propose a framework based on the perspective of multiple network service models. For normality verification, we show how to construct underlying protocol models by static and dynamic approach to guarantee the usability. For the latter two problems, we show how to construct composite service model with protocol interaction and correlation. Classes of attacks, such as worm and botnet, illustrate that our composite model detects the symptom of attack that individual ones cannot. This work provides a promising perspective from network service and a new construction method for service modeling.

Keywords: Intrusion Detection, Modeling, Computer Network Security, Network Services, Principle Component Analysis.

1 Introduction

One of the important problems of network security is that we may not know what form of future attack would be. However, a careful study of hostile traffic reveals the existence of peculiar characteristics that could differentiate it from normal communications across the network [1]. Therefore, several works focus on defining the notion of normality in different ways to detect anomaly. Once the definition of normality is specified, the violation of the normality, or say anomaly, is determined.

The notion of normality is usually provided by a formal model that describes the normal or expected behavioral operations or properties of a subject that we would like to protect. A model can be a description of a particular process or an essential or distinctive attribute or quality of a thing. For an anomaly detection system, the detection process is a measure that allows obtaining the deviation (or likeness) of a given activity with regard to the predefined model. For a model, false alarms may be introduced mainly by incorrect or imprecise modeling, and even unsuitable subject to model.

In the past, user behavior is the initial subject to model, since malicious user is the main threat of computer system in early days (e.g., mainframe user). User behavior model [2] is used as a reference to detect abnormal activity and resource usage. However, user behavior can suddenly change and is usually not well predictable. Hence, program model then draws the attention because of its relatively fixed behavior.

A program is considered harmful only after executing some unexpected system calls [3]. There are two research streams to model program behavior by using system calls: static and dynamic approach. Static model is built by referring to the given binary or source file of a program. All possible sequence of system calls that the binary or source file can produce are analyzed statically and recorded in the normal model (which is usually in a form of state machine [4] or database entry [3]). Any unexpected sequence of system calls that are not specified in the static model are viewed as an attack. Nevertheless, the notion of attack is beyond static analysis of system calls. For attacks such as brute force attack and stepping stone attack, which do not cause harm by executing unexpected system calls, are also security threats. Therefore, dynamic approach of model generating is adopted. Dynamic model is built by referring to a collection of runtime system call sequences that are produced by programs in a normal environment [3] or filter out non-normal execution. The accuracy of a dynamic model depends on the quality of the collected system call sequences, meaning how many it cover all possible normal execution sequences. In reality, it is difficult, through not impossible, to collect them, so that this approach may introduce false alarm due to not enabling to collect rare-seen normal execution.

Rather focusing on programs, there are works looking into monitoring network activities. They are differentiated by different network properties used in constructing their model. Network traffic is the most well studied subject. Traffic volume with attack propagation model [5], the variation of traffic entropy [6] [26], and network contact behavior studies [7] all belong to this category. Usually, statistics is served as a tool to model the normal usage of header field in network layer, transport layer and application layer. The outlier of the statistics model is considered as an anomaly, for example, worm propagation may cause entropy variation. Nevertheless, quantitative measures can be deliberately disguised by sophisticated attacker [8].

Since a network activity is governed by protocols it uses. Some works [9] - [11] construct their normal models based on network protocol specifications (e.g., RFCs). Such model can detect attacks or anomalies going along with protocol standard misuses, such as undefined protocol message (e.g., a TCP packet with SYN and FIN both set) and undefined protocol process (e.g., login again after successfully login). Let's make a metaphor to understand the weakness of this approach. We can view a network activity as composing a paragraph, each sentence is a network protocol execution, and the grammar serves as network protocol standard. Such approach can only detect anomalies by "checking syntactic error" of individual sentence (i.e., protocol execution). The first shortcoming is that, we can make a sentence which is syntactically error-free but not actually accepted in the real world. We point out that purely relying on protocol specification is good but may not be practical. A protocol is usually well designed so that most of the misuses are already defined in specification with an error handling mechanism. Security expert must further specify which messages or executions in the specification are considered anomalous. Second, a paragraph is only meaningful and readable if all the sentences in it are correct and follow a context. Similarly, we point out that anomalies can be founded by considering viewing a communication activity as a whole to collectively examine the normality of the entire underlying protocol executions. Past works only examine the normality of individual network protocols rather than the entire context communication.

There are several works [12] - [14] which derive specific legitimate network properties (commonly referred as behavior signature) by analyzing network activities and show they can detect certain attacks. These models are constructed by security experts with their experiences to known and popular attacks. The security experts extract and generalize the behavior of similar attacks and build a common model. Usually, finding such high quality and useful models is difficult and time-consuming. Such model is suitable for well-studied attacks or attacks having popular behavior patterns.

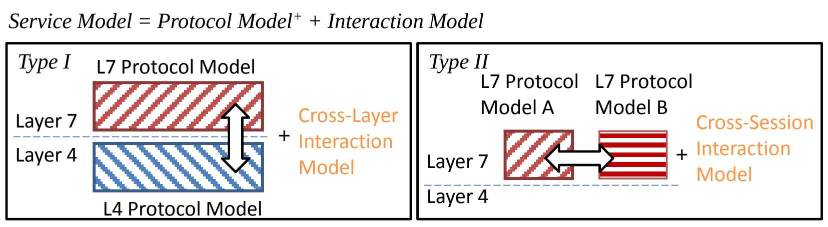
We now consider that a more robust network-based approach is needed and desired. Hence, we provide another perspective to model normal network activity that past works neglect before. Past normal models often suffer from three problems that limit their detection capability: lacking of model normality verification, only including particular behavior aspect, and focusing on individual model. To confront these problems, we propose a framework based on multiple network service models (and their underlying protocol models) to provide another viewpoint to reveal anomalies hidden in the network. For normality verification, we show how underlying protocol models are constructed by combining both static and dynamic approach to guarantee the normality that they describe. Also, the procedure of model construction is specified without human tricks. And for the latter two problems, we propose a method to construct composite model by using multiple service and protocol models as a whole to describe the model of network activity. Like the metaphor of composing a paragraph, we can expect that more anomalies can be revealed by such composite approach.

Classes of attacks, such as worm, fingerprinting and botnet, are shown as examples to illustrate that the composite model detects symptoms of attacks that individual model cannot. This work provides a promising perspective from network service and a new approach to combine models together as a whole for anomaly detection.

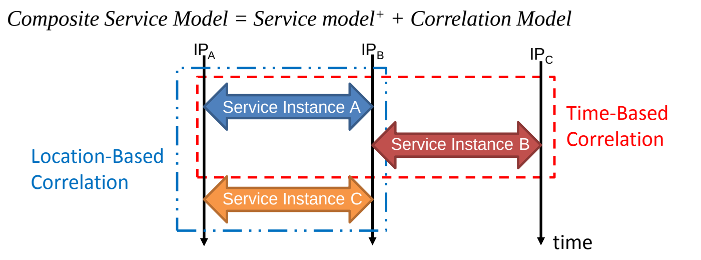
The rest of the paper is organized as follows. The next section gives an overview of our approach. In Section 3, we present the construction of underlying protocol model using both protocol specification and real world traffic trace. In Section 4, we provide the method for connecting multiple protocol models to a service model as well as the method for correlating multiple services models to detect anomalies. In Section 5, we demonstrate the detection capability by real world attacks. Then, we discuss some issues of our works. In Section 7, the related work briefly overviews different approaches taken in the past. Finally, we conclude our work in the last section.

2 System Overview

This section provides an overview and the concept of our approach to generate network-based anomaly detection models based on multiple network services with their underlying protocols.



(a) A service model is composed by multiple protocol models with interaction model.



(b) A network activity is composed by multiple service models with correlation model.

Fig. 1. The proposed composite service model scheme.

There are two stages of model construction: protocol model construction and composite service model construction. In the first stage, the input is the protocol specification and a collection of real world network traffic trace (raw packets), and the output is the normal protocol model. Our construction approach takes the advantages of both static and dynamic approach to reduce the incorrectness and imprecision problem of modeling. We also tackle the issue of model normality verification in this stage.

In the second stage, a network service model is composed by multiple protocol models with two types of interaction model. Together they can describe the normal model of a service operation. Besides, we also observe that in the runtime a network activity is accomplished by several network services. Especially for an attack, multiple network services could be involved to collaboratively accomplish an entire attack, such as host probing, vulnerability probing, service vulnerability exploitation, control and command channel establishment, file transferring. Hence, in addition to single service model, we further introduce correlation model combining multiple service models to describe a complex network activity. The correlation model specifies which instances of the service should be correlated in the runtime to reveal anomalies. Fig. 1 depicts the relationships between each model in the second stage.

Once we obtain the models, we examine our anomaly detection system by launching attacks collected from the Internet in a controlled environment. Then, we identify the deviated network behavior from our models. A deviation can be an undefined process of the model or an abnormal property of service execution. The deviated behavior is further extracted with the status of its related models. Afterwards, they are used to describe an attack scenario, called attack symptom. An attack symptom may consist of a deviation from sole protocol model or it can be a complex execution sequence from multiple service models and their correlation models.

We point out some concepts of our approach in the following subsections, and the detail of notation, model construction and anomaly detection is in Section 3 and 4.

2.1 Protocol Model Construction

Protocol Specification and Finite State Machine. Due to several successful past works [9], [15] and the characteristics of network protocol asymmetry; we choose finite state machine (FSM) as a tool to model a network protocol. Each state of the FSM represents a temporal status of protocol execution, such as waiting for acknowledge, or sending packets. Events in the FSM could be a network event (e.g., receiving network messages) or a host event (e.g., timeout). The definition of states, events, and transitions of the FSM are statically based on the written documents, i.e., protocol specification and open source implementations. Certain specification, such as TCP RFC [16], provides an execution state diagram (the figure 6 in [16]) for reference. At this point, we can build a complete view of protocol execution as precisely as possible. If one state (in the document) has more than one semantic meaning, it should be clearly separated. Take TCP for example, at ESTABLISHED state, its connection could be either "just completing three way handshake" or "just finishing data transfer", which should be separated in the FSM. In our system we set counter of ACK to differentiate them. Similarly, one message having different meanings should be identified as different events. A protocol FSM is then populated.

Execution Path. Network traffic traces are collected in campus network in terms of raw packets. Packets under same communication context (i.e., the same IP address pair, the same protocol number, and the same port number pair) are classified in advanced. The classified packets are then replayed, and populate the events defined in the corresponding FSM to trigger the transition of the FSM. Each instance can plot an execution path in its protocol FSM, and an execution path is considered as the behavior of the protocol execution. By repeating this, we can gather a large amount of execution paths for different protocols to represent the usage of these protocols in our collected traces.

State Vector (SV) and Principle Component Analysis (PCA). Since we have no idea how the usage of a protocol would be like (e.g., all instances traverse only few paths or all go to different paths in the FSM), we take principle component analysis (PCA) as a tool to analyze the behavior of collected execution paths for building protocol model. We introduce the concept of state vector (SV) to transform an execution path (which is a sequence of state transition) to a state vector (which is an indexed vector with each element is either 1 or 0, and each element maps to a specific state in its FSM). The coded state vector can be used by PCA for cluster analysis. Execution paths (or say state vectors) with similar state transition are considered having similar execution behavior, and they are clustered into same groups by PCA. Then, we extract the characteristics of each behavior cluster (also in terms of state vector) to represent different types of protocol usage. In order to guarantee the accuracy of the model, we extract high enough state vectors in each cluster to build the protocol model (see Section 3 for details). The extracted state vectors then transformed back to execution paths and we form the model of protocol by overlaying these execution paths on the protocol FSM. With these steps, we can first understand how many behavior clusters are there in these mixed executions paths, and what they are. Second, we extract behavior characteristics from each cluster instead of from all execution paths, which is more considerable for describing rare-seen behavior. Third, we combine static and dynamic approach for a better description of the protocol model. Semantic meanings of the state from static specification can be preserved, and this model can also reflect the major protocol usage in the real world. Usually, the final protocol model is a subset of the original FSM obtained from the specification. Certain states and paths are discarded by the PCA under a guaranteed coverage rate. Although it reduces the complexity of the final model, yet it introduces false alarms. (As the metaphor we mentioned, in this stage we can check to see if the sentences are syntactic-correct and meaningful.)

2.2 Composite Service Model Construction

Session and Connection. We define a session as an instance of transport layer protocol execution (e.g., TCP connection), and a connection as an instance of application layer protocol execution (e.g., HTTP session). Although sessions and connections are the basis of the network service, yet they are low level activities and do not interact with each others.

Service Model and Interaction Model. We introduce a more complex concept named network service model, which is defined as a set of protocol models with an interaction model for accomplishing a network task. For example, FTP service is not merely executed by one single session. It may include one control session (usually at port 21), more than one data sessions (usually at port 20), as well as all their underlying TCP connections. To relate all these instances in a single service model, we identify two types of interaction models to describe the relationship: cross-session and cross-layer. First, a cross-session interaction model relates two or more sessions by referring to protocol specification. For example, in the FTP specification, the service data sessions are spawned by the control session (with PORT and PASV command). We can easily specify and relate them by packet content inspection technique. Take VoIP application as another example, Real-time Transport Protocol (RTP) carries media stream controlled by Session Initiation Protocol (SIP), and their instances will be related into one service. Second, a cross-layer interaction model relates two instances at different layers (i.e., transport layer and application layer). This idea inherits the design of network protocol stack. An execution of network service should not focus merely at one layer. Viewing them as a whole provides more information to describe the execution of a network service simultaneously at both layers. We point out that the service model can provide another perspective to collectively examine the execution of network service rather than individual protocols. The normality of individual instance does not imply the normality of them as a whole.

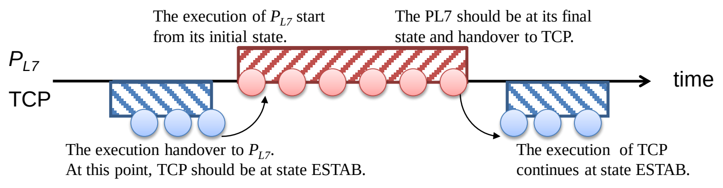
Correlation Model. Besides, we also observe that in the runtime a network activity is accomplished by several network services. For an attack, multiple network services could be involved to collaboratively accomplish the entire attack. However, their relationship may not obvious from the understanding of protocol and service operation logic. We then further identify two correlation models for correlating service instances into a composite one. Correlation models are built according to two other dimensions: time and (cyberspace) location. Research works also notice the importance of finding time-related [17] [18] and location-related [13] [19] relationship of different traffic flows for traffic analysis. Service instances that are executed under certain time constrain are possibly related. We demonstrate two time-related correlation models: execution time and execution duration. The (cyberspace) location in this paper does not indicate only IP address. The port number, protocol, and the role of service (e.g., client, server, and peer) can be served as a basis for the location model. The information can identify different location of a service instance. Once we obtain the instance of composite service, we than can detect anomalous behavior by using these instances. (As the metaphor we mentioned, from thousands of fragmentary sentences, we now understand which of them are actually in the same paragraph and they should be aligned together for checking its context.)

2.3 Anomaly Detection and Attack Symptom

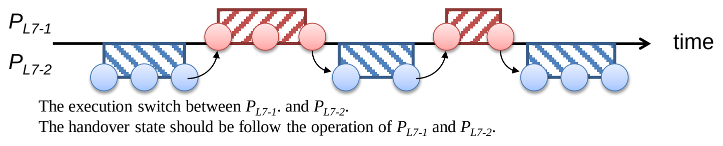
Anomalies. For anomaly detection, network traffic is collected in terms of packets and they are used to detect anomalies to check against the models we constructed. Similar to protocol model construction, packets under same communication context (i.e., the same IP address pair, the same protocol number, and the same port number pair) are classified in advanced. Each instance is then transformed to an execution path. For a protocol model, anomalies are found if the execution path cannot successfully traverse the normal protocol model (an FSM) from the initial state to one of the final states (i.e., complete a normal execution). For a service model, anomalies may be found if the execution paths in it violate the interaction models.

Take cross-layer interaction model for example in Fig. 2, if we view the execution of the service as a whole, the execution of the application layer protocol should be embraced by its execution of transport layer protocol (e.g., TCP in Fig. 2). The execution must be handover at TCP ESTAB (established) state; otherwise such execution is considered as an anomaly. Not only we examine the normality of individual protocols, we also check the interaction between them as a whole. For cross-session interaction model, we take similar approach of checking handover state for two or more application layer protocols. The handover states are determined at model construction stage by referring to the collected network traffic traces. For correlation models, in addition to anomalies found by checking handover state, we can also detect anomalies by unusual time or (cyberspace) location behavior. For example, compromised host that shows abnormal role change can be found, such role change method is usually used by worm attacker or botnet.

Attack Symptom. The anomalies found by protocol models, interaction models and correlation models are then extracted to build the attack symptom. The anomalies are


The execution of P_{L7} start from its initial state.
The PL7 should be at its final state and handover to TCP.
P_{L7}
TCP
time
The execution handover to P_{L7}. At this point, TCP should be at state ESTAB.
The execution of TCP continues at state ESTAB.

(a) The cross-layer interaction model.


P_{L7-1}
P_{L7-2}
time
The execution switch between P_{L7-1}. and P_{L7-2}.
The handover state should be follow the operation of P_{L7-1} and P_{L7-2}.

(b) The cross-session interaction model.

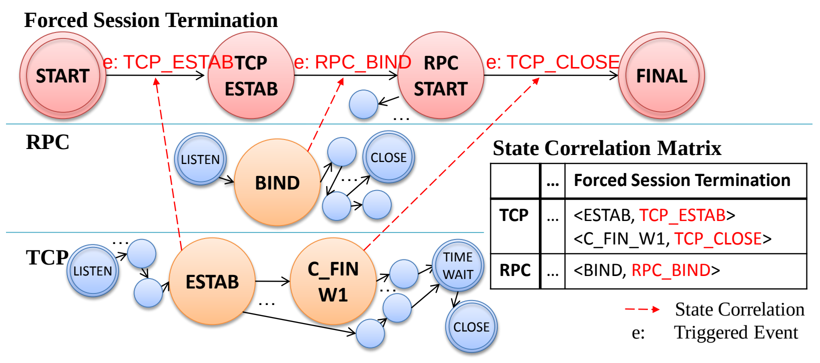
Fig. 2. The example of anomaly found by interaction model.

usually in a form of state machine transition that is not specified in normal FSM or an abnormal property of the execution. The entire service execution (including normal and non-normal) are extracted and aligned with time. Fig. 3 shows an example of an attack symptom (plotted with its underlying protocol FSMs), and it describes an attack scenario – Blaster attack does not normally accomplish the execution of Remote Procedure Call (RPC) service and its underlying TCP. The attack does not normally traverse the RPC FSM to the one of the final states; instead it handovers the execution back to TCP at a non-handover state. Such anomaly, called forced session termination, can only be revealed by viewing RPC and TCP as a whole. We demonstrate more examples in Section 5.

3 Protocol Model Construction and Verification

3.1 The Static Protocol Model and Specification

As we mentioned, once we obtain the protocol specification (possibly a RFC or open source code), the temporal protocol execution status and network/host events are extracted as the states and events of its protocol FSM. A protocol FSM can be denoted as $P_i = <\Sigma_i, S_i, int_i, \delta_i, F_i>$, where Σ_i is the event occurred of the protocol; S_i is a finite, non-empty set of protocol execution state; int_i is an initial state of the protocol which is an element of S_i; δ_i is the state-transition function: δ_i: $S_i \times \Sigma_i \rightarrow S_i$; F_i is the set of final states which is a subset of S_i. Each event of the FSM usually maps to a specific value of header field or a binary/text string in the packet payload. Each state of the FSM represents a temporal status of protocol execution. In this step, we manually construct the model for different protocols in a customized BNF (Backus–Naur Form). Currently, the model of TCP, UDP, ICMP, HTTP, FTP, RPC, SIP, and TFTP are constructed in our system.



	...	Forced Session Termination
TCP	...	<ESTAB, TCP_ESTAB> <C_FIN_W1, TCP_CLOSE>
RPC	...	<BIND, RPC_BIND>

Fig. 3. Attack symptom: Forced Session Termination.

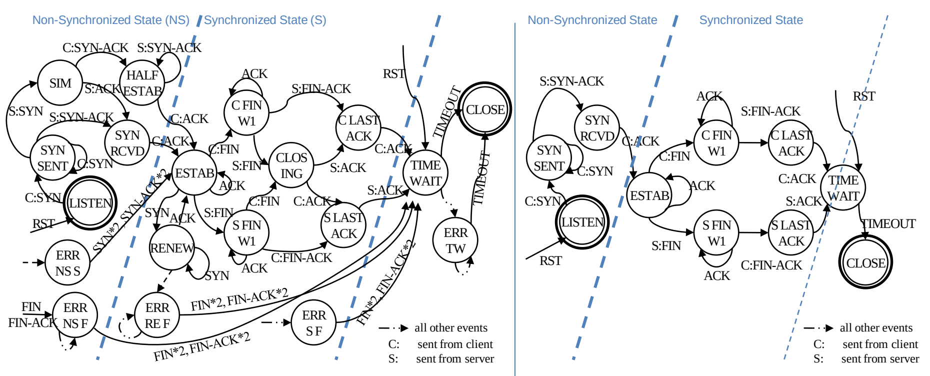
There is a benefit for modeling protocol rather than program. There might be dozens of HTTP servers and clients implemented in the real world; however they are the variants of the reference model from HTTP specification. No matter how programmers implement their servers and clients, the observed behavior of the HTTP protocol should align with the reference model. Although we can expect that the protocol behavior of different implementations may not be exactly the same (proven by [20], [21] or some embedded light weight server/client), yet ideally all of them should still follow (part of) the reference model to accomplish a protocol execution. That is, each implementation should be a subset of the specification. For the reason of efficiency, we model the protocol rather than individual programs.

3.2 The Dynamic Protocol Model and PCA

The model from previous step only serves as a reference that how an execution could be (from the perspective of the specification). Similar to static approach on program behavior [4], such model does not consider the real world usage of a protocol. It may introduce false positive if we take such model for anomaly detection, because allowed execution does not guarantee attack or anomaly free. Hence, we modify the static model by the dynamic approach with real world network traces. If we directly adopt dynamic approach without the static model, the constructed model would be complex as the exact DFA (deterministic finite automaton) in [17], which has high accuracy of describing a behavior but it may lost the human readable information.

Raw IP packets are collected from network as data set D and are classified based on header fields: IP address pair, protocol number, and TCP/UDP port number pair. Protocol events are specified based on the element of Σ_i of the corresponding protocol FSM P_i. According to state transition function δ_i: $S_i \times \Sigma_i \rightarrow S_i$ and observed protocol events in D, we output the execution path as a sequence of state transition $<t_1, t_2, ..., t_k>$, where all transitions belong to δ_i, and for each transition t_k: $s_{k,x} \times e_{k,y} \rightarrow s_{k,z}$, we have $s_{k-1,z} = s_{k,x}$ when $k > 1$, and $s_{1,x}$ is the initial state int_i.

As we mentioned, we analyze the behavior of protocol execution by using state vector rather than execution path. (Because we consider state vector is more suitable to characterize the protocol execution behavior. See next sub-section for more details.) We then transform an execution path to a state vector. In this step, multiple execution



(a) TCP model built based on RFC 793. (b) TCP normal model built by using dynamic approach.

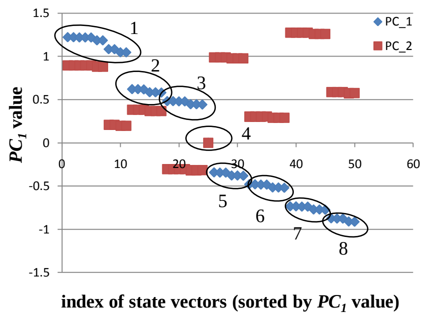
Fig. 4. The TCP FSMs built from different sources.

paths might map to the same state vector, and we consider their behaviors are similar. For P_i, the state vector is a vector having $|S_i|$ elements and the value of its element is either 1 or 0. Each element maps to a specific state of S_i and the value indicates if the state is reached (1) or not reached (0) by the corresponding execution path.

After collecting a set of state vector, we analyze the usage of protocol to build the normal model of this protocol. Briefly, the Principle Component Analysis can output a set of component from a set of features or variables. Let the state in S_i of P_i be the variables we analyze. We input n state vectors (having $|S_i|$ elements or say $|S_i|$ features) to PCA. PCA outputs another $|S_i|$ principle components and each of them is the linear polynomials of the original variables. For example, let $|S_i| = 3$ and the state vector is $<X_1, X_2, X_3>$, by analyzing n state vectors, PCA outputs the result: $<PC_1, PC_2, PC_3>$ where $PC_1 = \alpha * X_1 + \beta * X_2 + \gamma * X_3$. For a state vector instance <1, 0, 1>, its PC_1 value would be $(\alpha*1 + \beta*0 + \gamma*1)$. The variance of PC_1 of all the n instances, V_{PC1}, is the largest (i.e., $V_{PC1} \geqq V_{PC2} \geqq V_{PC3}$). And PC_1 can be viewed as the most important characteristic to describe these n instances. (It is because that a variable with small variance is relatively difficult to distinguish between data points.) And we can expect X_i is an important component if it has significant factor to affect the value of PC_1.

The PCA can also classify different behavior in our case. Since X_i are binary variables (either 1 or 0), the number of possible value of PC_1 (and also PC_2 and PC_3) is limited to $2^{|Si|}$. In the previous example, they are 0, α, β, γ, (α+β), (α+γ), (β+γ), and (α+β+γ). Hence, we can simply classify these n state vectors into $2^{|Si|}$ classes by using PC_1. (The value of V_{PC1}, V_{PC2} and V_{PC3} should be considered to decide how to perform classification. If V_{PC1} is large enough to distinguish the characteristic of different protocol behavior, using PC_1 is good enough; otherwise we should consider using PC_1 and PC_2 to classify these n state vectors. See next sub-section for more details.) We note that, in real word, certain protocol states are mutually exclusive, so that the total number of possible classes might be less than $2^{|Si|}$. Another case occurs when, for example, there is a very small γ in PC_1, so that the PC_1 value of <1, 1, 0> and <1, 1, 1> are quit close to (α+β). Since γ is small and X_3 is considered insignificant, we say <1, 1, 0> and <1, 1, 1> are similar behavior and are classified in the same class.

Once we classify these state vectors into different classes, we then designate one or more representative state vectors (RSV) in each class as references to model the nor-



1. LISTEN → SYN_SENT → SYN_RCVD → ESTAB → S_FIN_W1 → S_LAST_ACK → TIME_WAIT → CLOSE
2. LISTEN → SYN_SENT → SYN_RCVD → ESTAB → S_FIN_W1 → TIME_WAIT → CLOSE
3. LISTEN → SYN_SENT → SYN_RCVD → ESTAB → S_FIN_W1 → CLOSE
4. LISTEN → SYN_SENT → SYN_RCVD → ESTAB → TIME_WAIT → CLOSE
5. LISTEN → SYN_SENT → SYN_RCVD → ESTAB → C_FIN_W1 → C_LAST_ACK → TIME_WAIT → CLOSE
6. LISTEN → SYN_SENT → SYN_RCVD → ESTAB → C_FIN_W1 → TIME_WAIT → CLOSE
7. LISTEN → SYN_SENT → SYN_RCVD → ESTAB → C_FIN_W1 → TIME_WAIT → CLOSE
8. LISTEN → SYN_SENT → SYN_RCVD → ESTAB → C_FIN_W1 → CLOSE

PC_2 = -0.0002*SYN_RCVD - 0.3035*S_FIN_W1 + 0.5133*S_LACK - 0.6942*TIME_WAIT + 0.3035*C_FIN_W1 + 0.2847*C_LACK + 0.00008*RENEW - 0.0003*ERR_NS_S

Fig. 5. The cluster phenomena of TCP behavior

mal usage of this class. State vectors in the same class are considered having similar execution behavior, and they can only be differentiated by states that are considered relatively insignificant. For each class, we first rank the state vectors by the number of execution path instances it presents. (As we mentioned, a state vector may represent multiple execution paths in the original data set). When we take the number of execution path into account, we can then make the judgment whether a state vector is a RSV or not in a class, or even we need to split the class into two sub-classes since the behavior in it is considered quite different. For each class, we examine the element of state vector one by one. If the values of PC_i of all state vectors in a class are the same (all zeros or ones), then this element of RSV is assigned with this value. If the value of PC_i of all state vectors are not the same (having both 0 and 1, such as PC_3 of <1, 1, 0> and <1, 1, 1>), we split this class into two sub-class if the none of each part dominates the execution paths count (say exceeding r% of the total count, we set it to 95% in our experiment) and continue checking next PC_i of each sub-class. Otherwise, if one of the parts has dominant count, we simply take its value as RSV's value and discard the state vectors less than 1 - r%. Iteratively, each (sub-)class has one RSV to present it usage. We then collect all selected RSVs of each (sub-)class and find out their corresponding execution paths in data set. The states and the transitions of these execution paths are then used to construct the normal model of P_i. As we can expect, the states and transitions from RSVs are less than S_i and δ_i. However, these subset of states and subset of transitions represent the normal usage of protocol. Since some of the allowed execution paths are not seen in the real world and might even be an anomaly. We denote the normal protocol FSM as $P^*_i = <\Sigma^*_i, S^*_i, int_i, \delta^*_i, F^*_i>$, where Σ^*_i is the subset of Σ_i, which contains only events are considered normal. S^*_i, δ^*_i and F^*_i are the subset of ${}^*{}_i$, δ_i and F_i, respectively. Without special notice, we simply use P_i to represent the normal model in the following sections.

There is an important reason why we classify behavior before RSV selection. In this step, we would like to model the different behavior. If we select state vectors (according to their execution path count) without classification, state vectors that have small execution path count could possibly be discarded from the model. However, the aggregation of them may create a class of protocol execution that cannot be neglect. Hence, we classify behavior before RSV selection.

3.3 PCA Verification Results

From static analysis, we construct a TCP FSM based on the diagram in TCP RFC specification [16]. There are 19 states and 43 transitions in this static model. For constructing the protocol combining dynamic approach, we collect raw packets in campus network on a core router using Wireshark [22]. The total packet capture length is 200 minutes and it contains 131,499 TCP execution paths. Within these execution paths, there are only 50 identical state vectors. The state vector with highest execution path count (75,230) passes the following states: LISTEN, SYN_SENT, SYN_RCVD, ESTAB, DATA, S_FIN_W1, S_LACK, TIME_WAIT, and CLOSE, which looks like a standard TCP execution written in the textbook. Due to loops in the FSM, there are actually 68 types of execution paths of this state vector. The execution path count of top five state vectors are 75,230, 19,359, 15,074, 14,368, and 3321. All the other counts of 45 state vectors are less than 1200.

The result of PCA is that PC_1 = 0.0002*SYN_RCVD + 0.4814*S_FIN_W1 + 0.6011*S_LACK + 0.1387*TIME_WAIT - 0.4814*C_FIN_W1 - 0.3934*C_LACK + 0.0004*RENEW - 0.0029*ERR_NS_S. The PC_1 value of the state vector which has highest execution path count is 1.2212. The V_{PC1} is 0.4556 (58.76% of total variance), V_{PC2} is 0.2260 (29.14% of total variance), and V_{PC3} is 0.053 (6.89% of total variance). We plot all 50 state vectors with their value of PC_1 and PC_2 in Fig. 5. Although mathematically there could be 2^{19} distinct value of PC_i, yet we can see that there exists only 50 distinct state vectors and they can be roughly classified into 5 classes by PC_1: above 1.0, around 0.5, around 0.0, around -0.5, and between -0.5 and -1.0. (Actually, PC_2 also classifies theses state vectors into the exact same 5 classes.) When we take PC_1 and PC_2 into account, the aggregate variance would be 87.90% (the total variance of all PC_i is 100%), and we consider it is high enough using PC_1 and PC_2 to characterize the behavior of TCP. We then perform the algorithm to find RSVs in each class.

In the first class (PC_1 value is above 1.0), there are 12 similar state vectors. Out of 19 states, 14 states are all zeros or all ones among these 12 state vectors. While considering split these 12 state vectors into several sub-classes using the 5 contradictory states, we find that there is always a dominant count of execution paths (exceed 95% of total count) for a value of a state, so we do not split this class. For the second class, there 13 state vectors. For state TIME_WAIT, 43% of the execution paths pass it (i.e., the value of this element is 1) and the rest do not. Clearly, we split this class into two parts. For the fourth and fifth classes, similar situation exists. Finally, we have 8 (sub)-classes and their corresponding RSVs. The 8 RSVs are then used for constructing the final model of TCP which is shown in Fig. 4. The semantic meanings of these 8 TCP behaviors can be restored by studying its execution path (also in Fig. 5). We feed the original data set to the final TCP protocol model, and it covers 98.47% of the total execution paths, which indicates the model can describe the behavior characteristics quite well.

Due to page limit, we only show another one of our supported protocols – HTTP – here. By using static analysis, the HTTP FSM has 9 states 18 transitions. In a traces having only HTTP traffic, 96,136 execution paths and 11 state vectors are identified. Among them, 11 (sub-)classes of HTTP behavior are specified. And the result of dynamic analysis reduces the FSM to a HTTP normal model with 6 states and 9 transitions. The final model covers 99.98% of the execution paths in the data set.

4 Service Model Construction and Anomaly Detection

4.1 The Construction of Interaction Model

Once we have the model of individual protocols; we then consider the relationship between two protocols to see if they collaboratively accomplish a service. Two types of interaction models are shown in Fig. 1. The cross-layer interaction model is used by describing the relation between one transport layer protocol, P_{L4}, and one application layer protocol, P_{L7}. Inheriting to network protocol stack, the execution of the P_{L7} should be embraced by its execution of P_{L4}. (See Fig. 2.) We introduce a concept of handover state to describe the embrace relationship between P_{L4} and P_{L7}. A handover state s_{ho} in S_{L4} would continue looping while the execution handovers to its P_{L7}. The execution of P_{L7} must start at its initial state int_{L7} and complete the operation at one of the final states F_{L7} of P_{L7}. Then the execution would handover back to P_{L4} again and continues state transiting to next state from s_{ho}. The ESTAB state of TCP is one of such handover state. In order to specify how these handover occur, we mark an "OUT" at state s_{ho} indicating the execution can switch out to P_{L7}. The state int_{L7} is marked as an "IN" state since it can continue the execution. Similar, all the states in F_{L7} are marked as an "OUT" state, and the state s_{ho} is also an "IN" state. The handover information can help us to judge if the execution between P_{L4} and P_{L7} follow the cross-layer interaction model.

The mode of cross-session interaction is similar to cross-layer interaction model. It describes the relation of two related application protocols, such as SIP and RTP, and FTP command (TCP port 21) and FTP data (TCP port 20). The relationship may not as simple as TCP and the application layer protocol. Fig. 2 shows an example that the execution of $P_{L7\text{-}1}$ and $P_{L7\text{-}2}$ is interleaved and FTP command and data may quite fit this relationship. The "IN" and "OUT" state could be specified during static analysis by network expert (although most of the protocol specifications do not discuss such relationship), and dynamic analysis approach is another alternative. However, the data collection and techniques for specifying "IN" and "OUT" state by dynamic analysis may need further studies. In our paper, we simply look into the network traces and observe the handover situation of two application layer protocols that we consider they are related.

With the help of handover state, we can conceptually connect two protocol FSMs as one, and use such model for anomaly detection. In our experiment, we find an anomalous protocol execution that handovers at non-"IN" or non-"OUT" state.

4.2 The Construction of Correlation Model

We notice that an intrusive activity may involve more than one network services that may be considered irrelative from the perspective of specification. However, for example, attack may use $P_{L7\text{-}1}$ to compromise a vulnerable victim and use $P_{L7\text{-}2}$ to deliver malware. Hence, we introduce two correlation models for correlating two network services from different perspectives: time-related model and location-related model.

For the time-related model, we define two instances are related if | $P_{L7\text{-}1}$.*start_time* – $P_{L7\text{-}2}$.*start_time* | < T_S and | $P_{L7\text{-}1}$.*duration* – $P_{L7\text{-}2}$.*duration* | < T_D, where T_S and T_D are the predefined thresholds. The *start_time* is a timestamp that a P_{L7} starts exchanging the application layer message, and *duration* is the variable recording the total time spent while executing P_{L7}.

For the location-related model, we define two instances are related if one of the following constrains are satisfied: $P_{L7\text{-}1}$.*dstIP* = $P_{L7\text{-}2}$.*dstIP* (i.e., these two instances are running on the same host), or $P_{L7\text{-}1}$.*dstPort* = $P_{L7\text{-}2}$.*dstPort* (i.e., these two instances are connected by others using same destination port; they are probably network servers and provide same network service), or $P_{L7\text{-}1}$.*srcIP* = $P_{L7\text{-}2}$.*dstIP* & $P_{L7\text{-}1}$.*dstIP* = $P_{L7\text{-}2}$.*srcIP* (i.e., these two instances are running between two hosts and each host initiates one instance; they might provide peer to peer service or one of the service is spawn by another service).

More rules can be added to describe different relationship model in time and location domain. In Section 5, we will also demonstrate how these correlation models are used to detect anomalies on time and location domain.

4.3 Finding Anomalies

For single protocol FSM, an execution path is considered anomaly if the execution path does not accepted by its corresponding normal model (i.e., a deviation is found). Let P_i be the model constructed by static approach, and P^*_i is the normal model constructed by our dynamic approach. For an execution path $<t_1, t_2, \ldots, t_k, \ldots>$ of P_i (all t_k are transitions that belong to δ_i), if there exists a transition t_k: $s_{k,x} \times e_{k,y} \rightarrow s_{k,z}$, where t_k is in δ_i but not in δ^*_i (i.e., in the specification but not in our normal model), we consider $e_{k,y}$ is an anomaly event, and $s_{k,x}$ and $S_{k,z}$ are the anomaly states. Such anomaly states and event will be recorded for constructing attack symptom.

Usually, there are three type of anomaly that causes such anomaly transition: event that is not unknown or cannot be identified in a packet, event that should not be sent at current state, and timeout event caused by unknown communication issue.

For finding anomaly with the interaction model, we check if the execution handovers between two protocols at state marked "IN" and "OUT" or not. If two protocol instances handover from inappropriate state s_x to s_z via event e_y, these states and event are then used for attack symptom construction. For finding anomaly with correlation model, we can detect time and location anomalies, for example a host changes its role from a server to a client of its previous client. Usually, a network server (that does not run P2P service) does not act like a client. Take this example to be more specifically. In our experiment we detect such behavior with time and location information as follow. For two protocol instances, $P_{L7\text{-}1}$ and $P_{L7\text{-}2}$, that are considered time related ($P_{L7\text{-}1}$.*start_time* < $P_{L7\text{-}2}$.*start_time*, $P_{L7\text{-}2}$.*start_time* - $P_{L7\text{-}1}$.*start_time* < T_S, and | $P_{L7\text{-}2}$.*duration* - $P_{L7\text{-}1}$.*duration* | < T_D.), if $P_{L7\text{-}1}$.*dstIP* = $P_{L7\text{-}2}$.*srcIP* and $P_{L7\text{-}2}$.*dstIP* = $P_{L7\text{-}1}$.*srcIP*, then we say that the $P_{L7\text{-}1}$.*dstIP* is the server of first instance and it then becomes the client of the second instance. We record the event of $P_{L7\text{-}1}$ and $P_{L7\text{-}2}$ (that can identify the role of server and client) for constructing attack symptom.

Attacks such as botnet (establishing control and command channels), worm (exploiting procedure and downloading malware file), and stepping stone (a series of telnet connections) typically contain several protocol execution instances that have some sort of time and location relationships between them. We anticipate our approach can capture these behavior characteristics and use for anomaly detection. More experiment result on real world attacks are illustrate in Section 5.

4.4 The Construction of Attack Symptom

The deviations found in the FSM models, interaction models and correlation models are collected and aligned with the time they occur. The state and event that cause the deviation are connected together to describe a serious of changes when intrusive activity occurs. An attack symptom is also represented in a form of state machine. Each state of the attack symptom represents a temporal status of intrusive activity. Its event is created by the change of underlying protocols. The transitions of an attack symptom may lead the status of current protocol executions to an anomaly state. An attack symptom may contain several events and states of different protocols; hence we record the relationship in a correlation matrix *R*. Let AS_j be the attack symptom we currently focus on. The value of the matrix element $R_{i,j}$ is either null or a two-tuple value <state *s* of P_i or event σ of P_i, a list of <event *e* of AS_j>>, which indicates that the reach of state *m* of P_i or the occurrence of event σ would trigger the creation of (one or more) event *e* to the attack symptom AS_j. This matrix records how attack symptoms transit based on the state change and event occurrence of underlying FSMs. In Fig. 3., an example of $R_{i,j}$ is <state "BINDING" of RPC FSM, event "RPC-BINDING" of attack symptom "Forced session termination">), which means the reach of state "BINDING" of RPC will trigger the creation of an event "RPC-BINDING" to the "Forced session termination" attack symptom.

Although the output of our constructed attack symptom can be used as a reference to characterize the attack behavior for rule-based intrusion detection system; however, providing fixed rules may limit the detection capability. Rather we provide several normal execution models including protocol execution, protocol interaction, and service correlation for anomaly detection, which we believe is a promising alternative for detecting intrusion.

5. Experiment

We collect several real word attacks and replay them in our system to demonstrate the abnormal behavior captured by our prototype system. Single attacks — Blaster, Sasser, Code Red, Code Red II, Nimda, Welchia, Slammer — are collected from the Internet and are used for evaluation, and we also use a well-known attack trace — DARPA 1999 Intrusion Detection Evaluation Data Set [23] — and a general trace collected in campus network for false positive analysis. Single attacks are launched in a private LAN with multiple hosts having unpatched operating systems. Network packets are monitored and analyzed for detecting any deviations. The campus trace is collected in the core router of a campus network lasting 7 hours. It is stored in a 378 GB file containing 45 million TCP and UDP flows and 680 million packets.

5.1 Attack Symptom Found by Protocol Model

When examine the protocol execution of Sasser, we find a malicious TCP execution that is not aligned with our normal TCP model. Sasser would first initiate a TCP connection (at port 445) to its victim. Only six TCP packets are exchanged between them: SYN, SYN-ACK, ACK, FIN, FIN-ACK, ACK. This TCP instance occurs before Sasser tries to attack SMB-RPC service, and we make a reasonable guess that it is a probing connection. Since normal application may not establish a TCP without sending application layer message, our TCP normal model can detect such an anomaly. We notice that such behavior is allowed in TCP specification, but it is malicious while considering the TCP operation of real world practice.

Similarly, the HTTP normal model can find that the attack Code Red II sends an HTTP packet with a non-existing event (a packet with 1 byte "\x00"). Attack Nimda replies an HTTP error message that does not in our HTTP model, but in the specification. For another example, the attack Blaster has an execution path (sending messages with wrong order) that is not in our RPC normal model while preceding the vulnerability exploitation. Even some fingerprinting tools, such as nmap and nessus, deliberately send packets with abnormal headers or contents (e.g., Sending a TCP packet with SIN and FIN flags are both set, or sending a HTTP GET message to a HTTP client) to probe the information of the target. Such behavior can be detected by our protocol models.

Although we understand that such approach of protocol model checking is not new, yet we demonstrate that using model built based on static approach (i.e., specification) may not suitable. With combining dynamic and static approach, we can truly understand the behavior of protocol execution.

5.2 Attack Symptom Found by Service (Interaction) Model

The attack Blaster and it variants Welchia both encounter a problem of forced session termination. In order to successfully exploit the buffer overflow vulnerability of RPC service, their attacks would send a packet with a long message trying to overflow the buffer of RPC service. After receiving such long data, the execution right of the victim would be taken away by the attacker. The original RPC service hangs there and cannot response anymore, and the underlying TCP connection would timeout and sends a RST message to close the TCP session. Hence, the RPC service is forcedly terminated, which is not follow our cross-layer interaction model, and we consider it is an abnormal activity. In our normal model of RPC, a RPC instance should handover back to it TCP at one of the final states, such as RESPONSE.

The attack Sasser has similar attack symptom that runs on SMB-RPC and TCP. The attack Code Red and Code Red II also have a hanged HTTP that is forced terminated by TCP RST. These attacks all perform similar buffer overflow exploitation. Since the original service is buffer overflowed, even the attacker can now execute any program at the victim host, but the compromised service cannot be recovered, so that the attacker cannot avoid being detected by trying being normal. Such attack symptom is not only occurs in TCP and its application layer protocol. For UDP connection, the attack Slammer also encounters similar timeout event after the attacker perform a SQL injection.

All above attacks can be detected by checking the handover state of two protocol instances. We believe they can reveal more malicious behavior than single protocol models.

5.3 Attack Symptom Found by Composite (Correlation) Model

In the attacks that we collected, worm-like attack often initiates another channel for malware downing, such as, Blaster, Nimda, Sasser and Wehchia. Other attacks may initiate another communication channel back to the attackers, such as most of the botnet. Although such behavior is not part of service exploitation, yet most of the attackers want to do more than just service exploitation. They may setup a communication channel to control the victim to perform further attacks. Basically, this following behavior performed by attacker does not align with the design of original service. For example, a RPC service would not act like a FTP or TFPT client of its pervious RPC client, but Welchia, Sasser and Blaster do so in order to communicate with the attacker. The botnet client also establishes C&C channel (control and command channel) to the bot master to obtain instructions.

For a stepping stone attack, the attacker may connect to a host and then connect to another host via the first host. In order to hide itself from be detected, the attacker may take several hops to its final target and launch the attack so that back tracking is difficult. However, if we can obtain these network traces, with the help of time relationship (the staring time of the previous instance and the latter instance should be no longer than 10 seconds) and IP connected (one host is the destination of one instance and is the source of another instance), we can correlate instances that are related for detecting anomaly. We can than generate a path all the way from the victim to the attacker, such as [19].

These attack symptoms rely on the correlation in time and location dimension to figure out the abnormal relationship between each network service. We believe that such design is suitable to confront the complex network environment. We anticipate that the time and space relationship can be more complicated to confront sophisticated attacks. For stealth attack that possess incubation time can possibly not correlated with the related instances. Such studies could be the future works for this research.

5.4 False Alarm Analysis

The DARPA 1999 dataset is used in this subsection. We build our normal protocol models (only for HTTP and TCP) with the training dataset (lasting one week long) in DARPA dataset. Then the DARPA testing data (also one week long) is fed into our system with these normal protocol models. Within the testing data, there are 80 attacks but 18 of them are console-based attacks (telnet or SSH) and 46 of them are attacks that not target on networks. In these “network” attacks, the network is simply a carrier to deliver their attack. The network or the network services are not the target so that they are not our focus. (We will discuss these types of attack in the discussion section.) There are 16 attacks in DARPA's trace that fall into the category that we focus to detect. Within these 16 attacks, ten of them have at least one protocol execution deviation on HTTP or TCP operations that relate to its attack procedure. And the other six are attacks the target on SMTP and IMAP. Unlink signature-based IDS, we do not need to generate attack signature for each attack and its variants. Although some research has been studied the vulnerability-based signature [15] [25], yet they still need time to collect attacks, generated efficient vulnerability-based signature and deliver them. We focus on the network service operation and their relationships for detecting anomaly. In total 323, 841 execution paths in the dataset, we have 464 alerts that are actually not belong to an attack. Hence, the false positive rate is about 0.14%, which is considered acceptable.

In the campus trace, there are total 1,008,534 attack symptoms are identified. Specifically, 289,476 of them are TCP port sweep and 717,170 of them are TCP SYN flooding. Both of them are quite common in today's network; however it still needs further study to understand why and what applications send these traffic. The rest of the attack symptoms are mainly the errors occurred at application layer (~81%), and ~14% deviations are caused by TCP. The rest of the attack symptoms are found by interaction models and correlation models (~4%). Application layer errors could be sending out of order messages or error messages.

We notice that sometimes bad or incorrect implementation of software may also cause sending error messages or a crash program may not be able to complete the entire protocol execution. They could possible one of the reasons that we observe such attack symptom. P2P programs may also send lots of TCP SYN packets for finding files. These applications are the possible false positives.

6. Discussion

Network service model proposed in this paper is not used to replace all the other models (such as user model, program model, or traffic model). Every model has its pros and cons, and can collaboratively form the defense alliance to detect anomalous from different perspective. Here, we provide a promising perspective from network service to detect anomalies that may neglect by other models.

As we discuss in the previous section, several types of attacks do reveal anomalies when we examine the execution of network services. However, there are still certain type of attack does not fall into these categories, such as Cross Site Script (XSS), Cross Site Request Forgery (CSRF), malicious programs executed by human (e.g., Trajan in e-mail), and exchanging malicious files via instance message software or P2P software. Their infection mechanism does not involve network service operation. Network itself if merely a carrier to deliver the malicious intention of the attackers. They infect victims by malicious content carrying in normal network service execution. Research works that focus on malicious content analysis could possibly deal with these attacks. Our proposed approach can detect them only after infected host acts abnormally from the perspective of network service operation. However, this type of attacks is beyond our research focus.

There are works, such as [27] [28], that focus on event correlation approach to detect intrusions; especially data mining techniques are widely used. The benefit of this approach can decrease the time and effort of human involvement and can automatically capture the behavior of intrusion. However, we anticipate that a large amount of mining data is needed, and most of them focus on misuse detection system rather than anomaly detection system.

7. Conclusion

In this paper, we demonstrate the feasibility of using protocol execution model to reveal even more anomalies that past may not find. The combination of static and dynamic approach for constructing protocol models is more appropriate to describe the real world usage of protocols than models built based on protocol specification. The Principle Component Analysis can help us to classify different protocol execution behaviors and figure out the important states in the traces. We also demonstrate that the interaction models and correlation models can provide more delicate descriptions between multiple protocol instances. The proposed composite service model can reveal anomalies in classes of real world attacks with reasonable false positive rate. Unlike rule-based intrusion detection systems that only rely on signatures, we provide an alternative for anomaly detection that focuses on the essential part of network communication—network services. We anticipate that future attacks would be more and more sophisticated, and the involved protocol instances would be diverse. From the perspective of service execution and considering the relationships between them would be a promising approach to describe the characteristics of malicious behavior. In this paper, we propose a method using a more formal way to solve the problem of constructing normal models and correlating their relationships.

Acknowledgments. This work was partially supported by the iCAST project sponsored by the National Science Council, Taiwan, under the Grant No. NSC97-2745-P-001-001.

References

1. Juan M. Estévez-Tapiador, Pedro Garcia-Teodoro, Jesús E. Díaz-Verdejo: Anomaly detection methods in wired networks: a survey and taxonomy. Computer Communications 27(16): 1569-1584 (2004)
2. Dorothy E. Denning: An Intrusion-Detection Model. IEEE Symposium on Security and Privacy 1986: 118-133
3. Stephanie Forrest, Steven A. Hofmeyr, Anil Somayaji, Thomas A. Longstaff: A Sense of Self for Unix Processes. IEEE Symposium on Security and Privacy 1996: 120-128
4. David Wagner, Drew Dean: Intrusion Detection via Static Analysis. IEEE Symposium on Security and Privacy 2001: 156-
5. Cliff Changchun Zou, Weibo Gong, Donald F. Towsley, Lixin Gao: The monitoring and early detection of internet worms. IEEE/ACM Trans. Netw. 13(5): 961-974 (2005)

6. George Nychis, Vyas Sekar, David G. Andersen, Hyong Kim, Hui Zhang: An empirical evaluation of entropy-based traffic anomaly detection. Internet Measurement Comference 2008: 151-156
7. Ruoming Pang, Vinod Yegneswaran, Paul Barford, Vern Paxson, Larry L. Peterson: Characteristics of internet background radiation. Internet Measurement Conference 2004: 27-40
8. Stuart Staniford, Vern Paxson, Nicholas Weaver: How to Own the Internet in Your Spare Time. USENIX Security Symposium 2002: 149-167
9. J. Treurniet: A Finite State Machine Algorithm for Detecting TCP Anomalies. Technical report, Defence R&D Canada, 2005
10. Juan M. Estévez-Tapiador, Pedro Garcia-Teodoro, Jesús E. Díaz-Verdejo: Stochastic Protocol Modeling for Anomaly Based Network Intrusion Detection. IWIA 2003: 3-12
11. InSeon Yoo, Ulrich Ultes-Nitsche: Towards Run-Time Protocol Anomaly Detection and Verification. ICETE (2) 2004: 299-304
12. Giovanni Vigna, Richard A. Kemmerer: NetSTAT: A Network-based Intrusion Detection System. Journal of Computer Security 7(1): (1999)
13. D. R. Ellis, J. G. Aiken, K. S. Attwood, and S. D. Tenaglia: A Behavioral Approach to Worm Detection. Proc. of the 2004 ACM Workshop on Rapid Malcode (WORM): 43-53.
14. R. Sekar, Ajay K. Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang, S. Zhou: Specification-based anomaly detection: a new approach for detecting network intrusions. ACM Conference on Computer and Communications Security 2002: 265-274
15. Helen J. Wang, Chuanxiong Guo, D.R. Simon, Alf Zugenmaier: Shield: vulnerability-driven network filters for preventing known vulnerability exploits. SIGCOMM 2004: 193-204
16. J. Postel: Transmission Control Protocol. IETF RFC 793, Sept. 1981.
17. Jayanthkumar Kannan, Jaeyeon Jung, Vern Paxson, Can Emre Koksal: Semi-automated discovery of application session structure. Internet Measurement Conference 2006: 119-132
18. Peng Ning, Y. Cui, D.S. Reeves: Constructing attack scenarios through correlation of intrusion alerts. ACM Conference on Computer and Communications Security 2002: 245-254
19. Yinglian Xie, Vyas Sekar, D.A. Maltz, Michael Reiter, Hui Zhang: Worm Origin Identification Using Random Moonwalks. IEEE Symposium on Security and Privacy 2005: 242-256
20. Gordon Lyon : Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning. Nmap Project (2009)
21. Nessus. [Online]. Available: http://www.nessus.org/
22. Wireshark. [Online]. Available: http:// www.wireshark.org /
23. DARPA 1999 Intrusion Detection Evaluation Data Set. [Online]. Available: http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/data/1999data.html/
24. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The Physiology of the Grid: an Open Grid Services for Distributed Systems Integration. Technical report, Global Grid Forum (2002)
25. David Brumley, James Newsome, Dawn Xiaodong Song, Hao Wang, Somesh Jha: Towards Automatic Generation of Vulnerability-Based Signatures. IEEE Symposium on Security and Privacy 2006: 2-16
26. Arno Wagner, Bernhard Plattner: Entropy Based Worm and Anomaly Detection in Fast IP Networks. WETICE 2005: 172-177
27. Wenke Lee, Salvatore J. Stolfo, Kui W. Mok: A Data Mining Framework for Building Intrusion Detection Models. IEEE Symposium on Security and Privacy 1999: 120-132
28. Peng Ning, Dingbang Xu: Learning attack strategies from intrusion alerts. ACM Conference on Computer and Communications Security 2003: 200-209