A Semantics-Aware Approach to the Automated Network Protocol Identification

Xiaochun Yun, *Member, IEEE*, Yipeng Wang, *Member, IEEE*, Yongzheng Zhang, *Member, IEEE*, and Yu Zhou, *Member, IEEE*

***Abstract*—Traffic classification, a mapping of traffic to network applications, is important for a variety of networking and security issues, such as network measurement, network monitoring, as well as the detection of malware activities. In this paper, we propose Securitas, a network trace-based protocol identification system, which exploits the semantic information in protocol message formats. Securitas requires no prior knowledge of protocol specifications. Deeming a protocol as a language between two processes, our approach is based upon the new insight that the n-grams of protocol traces, just like those of natural languages, exhibit highly skewed frequency-rank distribution that can be leveraged in the context of protocol identification. In Securitas, we first extract the statistical protocol message formats by clustering n-grams with the same semantics, and then use the corresponding statistical formats to classify raw network traces. Our tool involves the following key features: 1) applicable to both connection oriented protocols and connection less protocols; 2) suitable for both text and binary protocols; 3) no need to assemble IP packets into TCP or UDP flows; and 4) effective for both long-live flows and short-live flows. We implement Securitas and conduct extensive evaluations on real-world network traces containing both textual and binary protocols. Our experimental results on BitTorrent, CIFS/SMB, DNS, FTP, PPLIVE, SIP, and SMTP traces show that Securitas has the ability to accurately identify the network traces of the target application protocol with an average recall of about 97.4% and an average precision of about 98.4%. Our experimental results prove Securitas is a robust system, and meanwhile displaying a competitive performance in practice.**

***Index Terms*—Latent Dirichlet Allocation, machine learning, network security, protocol identification.**

I. Introduction

A. Background and Motivation

IDENTIFYING network traces associated with different network protocols has many applications in the area of networking and security, such as network measurement, tunnel detection, quality of service (QoS), and intrusion detection and prevention systems (IDSs/IPSs). The protocol message format specifications from network traces are often leveraged for the accurate identification of application protocols. Taking its application in IDSs/IPSs as an example, most IDSs/IPSs parse the contents of network packets to implement their functionalities, where detailed knowledge of the protocol specifications is important. Several application protocol parsers, such as FlowSifter [1], UltraPAC [2], GAPA [3], and binpac [4], have been proposed in prior literature. All these application protocol parsers require protocol specifications in order to generate parsers for the corresponding protocol. However, nowadays, most application protocols (including apps for mobiles) on the Internet are proprietary and generally have no well-documented protocol specifications in public. According to a report of Internet2 NetFlow on backbone traffic in 2007, nearly 50% of Internet traffic belongs to unknown application protocols [5]. Moreover, botnet command and control protocols do not have publicly available protocol specifications from their designers. Motivated by the rise of unknown Internet traffic volume, we devise a novel approach for protocol identification which automates the analysis with minimal manual efforts required.

B. Limitation of Prior Art

In this paper, we conduct protocol identification based on the packet payloads of raw network traces. Previous methods in this field fall into two categories: 1) protocol parsing-based methods, and 2) protocol signature-based (i.e., application fingerprint) methods. To identify flows of application protocols, protocol parsing-based methods infer protocol message formats by analyzing both the executable code and the network traces of the target protocol. However, reverse engineering, which is a major instrument for protocol parsing, is very time-consuming and laborious. Our paper belongs to the second category—protocol signature-based methods.

Protocol signature-based methods conduct the analysis only relying on the study of the payloads of network traces. Such methods typically involve two ways to implement their functionalities, including manual analysis and automatic analysis. Generally, the methods of manual analysis extract application-level signatures by means of heuristics or previous knowledge of the application protocols. Note that manual analysis process is deemed as an unscalable task and their results cannot be taken for granted. In contrast, the methods of automatic analysis apply statistical learning methods on the payloads of network traces

Manuscript received August 03, 2013; revised June 12, 2014 and October 17, 2014; accepted November 19, 2014; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor M. Meo. This work was supported by the National Natural Science Foundation of China under Grants No. 61402472 and No. 61303170, the National High Technology Research and Development Program of China under Grant No. 2013AA014703, and the Knowledge Innovation Program of the Chinese Academy of Sciences under Grant No. XDA06030200. *(Corresponding authors: Yongzheng Zhang and Yipeng Wang.)*

The authors are with the Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China (e-mail: yunxiaochun@cert.org.cn; wangyipeng@iie.ac.cn; zhangyongzheng@iie.ac.cn; zhouyu@iie.ac.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNET.2014.2381230

to carry out the whole process. Therefore, recently most researchers [6]–[14] focused their attention on automatically extracting the statistical signatures for protocol identification.

Previous works on automatic processes for protocol signature construction do not exploit the semantic information provided in the payloads of network traces, namely the correlations among grammatical elements in the packet payloads. Note that they miss the opportunity of achieving more accurate classification results with fewer features. Consequently, there is a need to exploit the semantic information in protocol messages, and meanwhile make few assumptions on protocols when performing signature-based inference.

C. Proposed Approach

In this paper, we propose a semantics-aware classification system, Securitas,[1] that takes network traces as input and effectively identifies the network traces of the target application protocol from mixed Internet traffic. In practice, we observe a similarity between application protocols and natural languages, as a protocol can be regarded as a language for two processes to communicate. Securitas is based upon our new insight that the n-grams of protocol traces, just like those of natural languages, exhibit highly skewed frequency-rank distribution that can be leveraged for accurate protocol identification. Our proposed approach performs statistical inference methods and machine learning techniques on the syntax and semantics of observed network traces.

We proposed ProDecoder [15], a system for automatically reverse engineering unknown protocols from network traces. Securitas differs from ProDecoder in terms of goals—ProDecoder aims at inferring protocol message formats, whereas Securitas aims at identifying application protocols. To this end, the processing procedures of the two systems are very different. Securitas generally involves three major phases: (I) Modeling phase; (II) Training phase; and (III) Classification phase. To classify a raw packet, Securitas first builds statistical characterization for each packet payload, and then leverages a supervised classifier to assign a proper class to it. Furthermore, to describe the complete structure of an application protocol, ProDecoder uses the whole number of bytes of each packet payload for format inference. In Securitas, we employ the first l bytes of a packet for protocol identification, where l is 16. In this paper, we want to present the first attempt that leverages natural language processing technology to conduct accurate protocol identification.

D. Novelty and Advantages of Our Approach

Securitas is inspired from natural language processing literature, and it is focused on the use of n-gram models and Latent Dirichlet Allocation to extract statistical protocol signatures from the payloads of network traces. Thus, our system distinguishes itself from prior work in this field on the following important aspects, to be discussed here. First, in practice, a protocol keyword generally involves several correlated n-grams to specify a protocol command. Our proposed approach is able to find such correlation among n-grams, and then group correlated n-grams together to form a protocol keyword. For example, the set of 3-grams {RCP, CPT, PT_, T_T, _TO, TO:} altogether indicates the recipient of the message in SMTP protocol. In addition, Securitas is also able to make distinctions to the same n-grams that may express different semantic information. In contrast, simply counting the occurrences of n-grams ignores the contextual information of each n-gram, and thus cannot make such distinctions. Take the SMTP communication session in Fig. 1 as an example. We notice that, in our example, the same 3-grams "354" for different occurrences have different semantic meanings. Thus, they should be clustered into different keywords.

[1]Securitas was the goddess of security and stability.

```
1 S: MAIL FROM:<JQP@HOSTY.ARPA>
2 R: 250 ok
3 S: RCPT TO:<JOE@HOSTW.ARPA>
4 R: 250 ok
5 S: DATA
6 R: 354 send the mail data, end with .
7 S: Date: 23 Oct 81 11:22:33;
     From: JQP@HOSTY.ARPA
     To: JOE@HOSTW.ARPA
     Subject: Meeting Address Changes to Avenue 354
     ...
     .
8 R: 250 ok
```

Fig. 1. Example SMTP communication session. Letter "S" indicates the packet from the e-mail sender, and letter "R" indicates the packet from the e-mail receiver.

E. Key Contributions

In this paper, we propose Securitas, a robust and multidisciplinary approach to network trace-based application protocol identification. We make four key contributions in this paper as follows.

- Securitas is applicable to both connection-oriented protocols (such as TCP) and connection-less protocols (such as UDP).
- Securitas is based on the statistical characterization of packet payloads, and it assumes no knowledge on protocol specifications, such as delimiters. Accordingly, Securitas can be applied to both textual and binary protocols.
- As a packet-based solution to protocol identification, Securitas does not need to assemble IP packets into application-level messages. Thus, it is suitable to both per-packet and per-flow classification.
- Securitas is very effective for both long-live flows (such as SMTP) and short-live flows (such as DNS) in real-world environments.

The remainder of the paper is organized as follows. Section II is dedicated to the related work. Then, we give an overview of the whole system in Section III. In Sections IV–VIII, we present the technical details of each module of Securitas. Next, in Section IX, we evaluate the whole system with the network traces of different application protocols. We compare the experimental results of Securitas to an existing algorithm in Section X. Limitations of our work are presented in Section XI. Finally, we conclude our work in Section XII.

II. Related Work

To identify application protocols from network traces, prior methods fall into three categories: port-based analysis, payload-based analysis [6]–[14], and behavior-based analysis [16]–[19]. Payload-based approaches can be classified

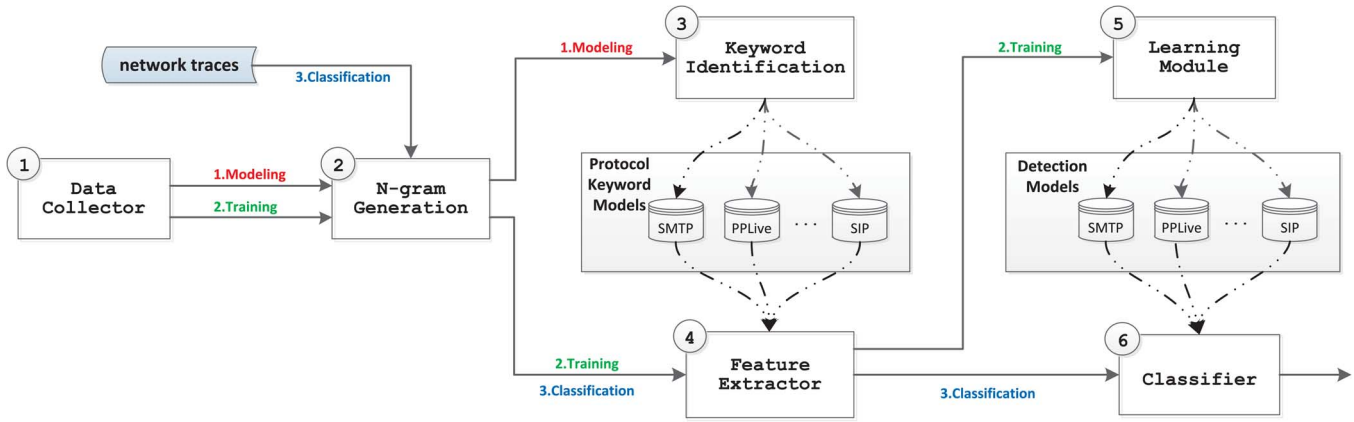


Fig. 2. Architecture of Securitas.

into two subcategories: 1) protocol parsing-based methods, and 2) protocol signature-based (i.e., application fingerprint) methods.

A. Protocol Parsing-Based Methods

Protocol parsing-based methods extract protocol features by analyzing the network traces and the binary code of protocols. Reverse engineering with manual efforts is a major instrument for protocol parsing, but it is very time-consuming and laborious, such as [20] and [21]. Several works paid their attentions on automating protocol parsing. Lim *et al.* proposed an analysis tool to extract output data formats on stripped executables [22]. Caballero *et al.* proposed an approach, shadowing [23], to automated protocol reverse engineering by tainting data received from the network. The methods proposed in [24] and [25] infer protocol message format by observing the dynamic execution of protocols. Cui *et al.* proposed a tool Tupni [26] that can reverse engineer file and network protocol formats with a richer set of information, such as record sequences, record types, etc. In comparison to such methods, we do not require the executable code of application protocols as a prior.

B. Protocol Signature-Based Methods

Haffner *et al.* proposed ACAS [6], which takes advantage of statistical machine learning techniques to identify protocol signatures based on application-level content. The proposed approach automated the construction of protocol signatures based on IP traffic payload content, which reveals the known instance of each protocol.

Ma *et al.* proposed a statistical and structural content model to automatically carry out protocol identification, sourced solely from flow content [8]. In [8], it is necessary to identify sessions from packet traces under analysis. In this case, a session is defined as flows of two directions, including application-level data sent by the initiator to the responder and data sent by the responder to the initiator.

Finamore *et al.* first introduced the concept of stochastic packet inspection (SPI) to conduct traffic classification based on packet payloads [11], [12]. They proposed a system called KISS, which is designed specifically for UDP traffic. The protocol signatures of SPI are derived by means of a Chi-square like hypothesis testing. Mantia *et al.* extended the concept of stochastic packet inspection to make it suitable for TCP traffic classification [13]. The SPI method is very effective for online classification, and it has high computational efficiency and classification accuracy in practice.

Recently, Tongaonkar *et al.* proposed SANTaClass [14], [27], a system for automated network traffic classification. SANTaClass is designed specifically for unknown traffic classification, which is a cutting-edge problem in networking and security. In SANTaClass, Tongaonkar *et al.* adopts an unsupervised approach to automatically extract statistical signatures from the packet payloads of network traces.

Our technique belongs to protocol signature-based methods, where statistical protocol signatures are extracted from the packet payloads of network traces automatically.

III. Architecture of Securitas

In this section, we first provide a bird's view over Securitas, a semantics-aware classification system, which takes network traces as input and automatically identifies the network traces of the target application protocol from mixed Internet traffic. As it is shown in Fig. 2, Securitas generally involves three major phases: (I) Modeling phase; (II) Training phase; and (III) Classification phase. We give a brief introduction to the three phases.

A. Modeling Phase

The Modeling phase is the first phase of Securitas, and it aims to build a statistical protocol keyword model from the network traces that correspond to a specific application protocol, where the protocol keyword model is referred to by us as a joint probability distribution of n-gram terms and protocol keywords. Note that we use the corresponding protocol keyword model to extract classification features in Feature Extractor. The Modeling phase has three major components: 1) Data Collector; 2) n-gram Generation; and 3) Keyword Identification.

B. Training Phase

After that, in the Training phase, we work on labeled training samples to build a protocol detection model for each target protocol. Our protocol detection model is based upon a supervised machine learning algorithm, and it is able to automatically distinguish the protocol traces from mixed Internet traffic. In Securitas, the Training phase contains four key components:

1) Data Collector; 2) n-gram Generation; 4) Feature Extractor; and 5) Learning Module.

Note that Securitas runs offline for a given protocol trace in the Modeling and Training phases.

C. Classification Phase

Finally, in the Classification phase, according to the protocol keyword model obtained in the Modeling phase and the detection model obtained in the Training phase, the Classifier Module makes decisions on each raw unseen network packet so that the unlabeled network packets are grouped into two classes, network packets that belong to the target protocol under analysis and those that are not. In this phase, there are three important functional components, including: 2) n-gram Generation; 4) Feature Extractor; and 6) Classifier.

IV. Data Collector

The objective of Data Collector is to collect two kinds of data. One is network traces that correspond to a specific application protocol (such as SMTP, DNS, BitTorrent, etc.), and we use them for modeling and training purposes. The other is network traces that actually do not stem from the target application protocol and are regarded by us as background traffic for training purpose.

In this part, to collect network traces of the target application protocols, we only resort to two simple strategies, transport-layer port numbers and protocol syntax check. However, our system is not limited to the above two strategies.

There are several other approaches to get raw packets of a specific protocol. For example, if the executable code of an application protocol is available, we can run it in a controlled environment to gather protocol data sets, such as the GT method [28] and the sandbox method [29]. In practice, Securitas can also use such technology to collect network traces for further analysis. We give a detailed description of our strategies in Section IX-A.

V. n-Gram Generation

The n-gram Generation module aims to provide the initial structure to the payloads of network packets. To this end, n-gram Generation translates each raw packet into a sequence of n-grams.

First of all, given many packets of the same application protocol, n-gram Generation first breaks up each packet into a set of n-grams. In the field of natural language processing, an n-gram is a subsequence of n elements contained in a given sequence of at least n elements. For instance, treating each character as an element, an example packet payload "`DATA\r\n`" can be represented with the following n-grams:

- 2-grams: DA, AT, TA, A\r, \r\n;
- 3-grams: DAT, ATA, TA\r, A\r\n;
- 4-grams: DATA, ATA\r, TA\r\n.

Next, we carry out a pilot study on the distribution of n-grams for two well-known protocols, SMTP and BitTorrent. In our analysis, we vary values of the parameter n for the two protocols. In Fig. 3, the x-axis denotes the rank of n-grams in terms of their frequency counts on the y-axis, and the x- and y-axes are both converted by us to logarithmic scale. Notice that the distribution of n-grams appears approximately linear on log-log

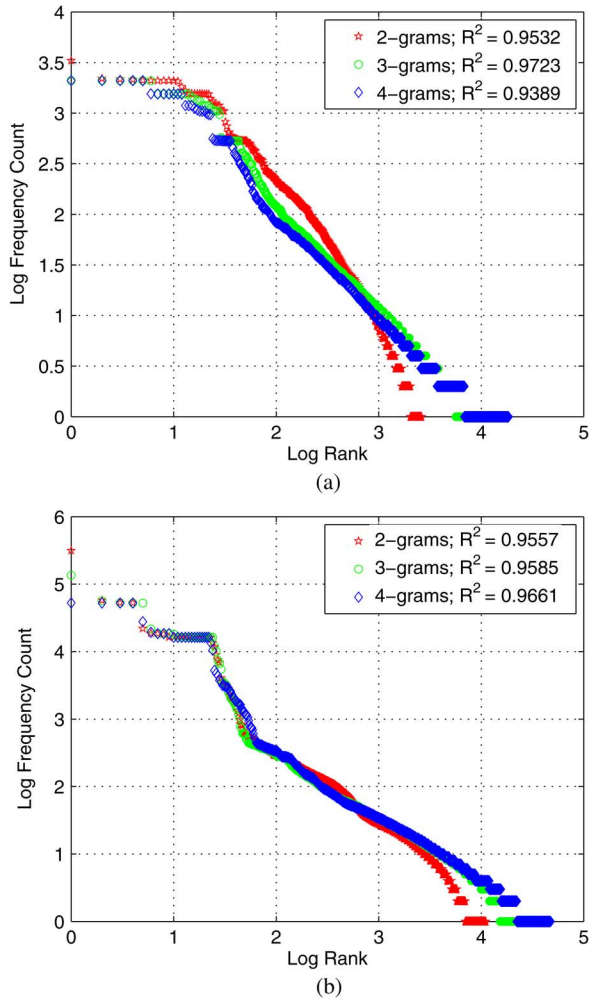


Fig. 3. n-gram distribution in SMTP and BitTorrent protocols for the first 16 B.

plot, which is a characteristic of Zipf distributions [30]. It is well known that the word occurrences in most human languages follow a Zipf distribution. In Fig. 3, we notice that the goodness-of-fit values R^2 improves for $n = 3$ compared to $n = 2$. Furthermore, we also observe that the distribution of n-grams becomes highly sparse for values of $n > 3$. Based on the above observations, we set the value of n in our study to be 3.

Note that the state space of n-grams contains more than 256^n unique elements. However, only a small part of the n-grams are format related. Thus, we need to reduce the huge size of n-gram states to avoid the curse of dimensionality. Toward this end, we select a subset of n-grams with high probability of appearance in our traces and use them to construct an n-gram vocabulary for the given application protocol. Furthermore, let W denote the parameter representing the size of the n-gram vocabulary.

According to the n-gram vocabulary obtained, n-gram Generation translates each raw packet payload into a sequence of n-grams. The n-gram sequences are the input to Keyword Identification and Feature Extractor. Obviously, the n-gram Generation module does not need to define/indicate field boundaries with separators, which are often leveraged in previous studies. Therefore, our solution is suitable for both textual and binary protocols. For the other parameter, we will discuss how to select an appropriate value of W in Section IX.

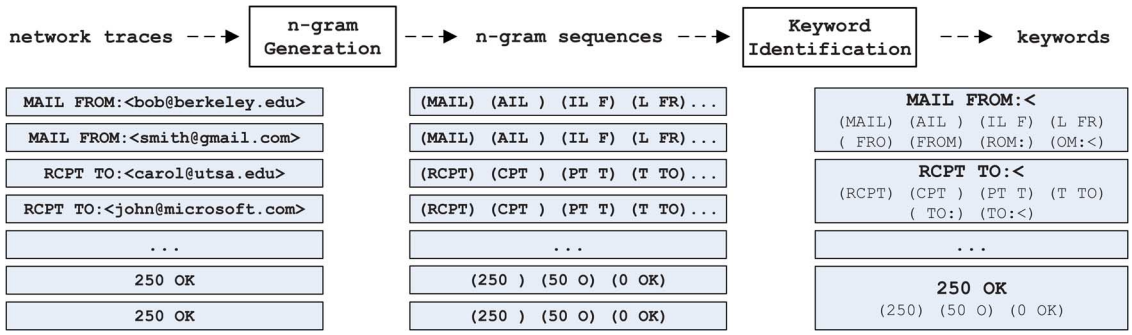


Fig. 4. Example of translating network packets into protocol keywords.

VI. Keyword Identification

In the Modeling phase, the second module is Keyword Identification. We perform offline analysis on the captured network traces that correspond to a specific application protocol. The input to Keyword Identification is a sequence of n-grams generated by n-gram Generation, and the output to Keyword Identification is a protocol keyword model, which is a joint probability distribution of n-gram terms and keywords. In Securitas, we use the corresponding protocol keyword model to extract classification features for each network packet in Feature Extractor.

A. Concept of Protocol Keyword

To begin with, we first introduce the concept of "protocol keyword" used in this paper. A protocol keyword is regarded by us as a set of n-grams, which mostly show up together in the packet payload and meanwhile denote the same semantic information. Note that the payload of each packet may have a complete or multiple protocol keywords, which are defined in the protocol specifications. For instance, the set of 3-grams {`RCP, CPT, PT_, T_T, _TO, TO:`} can be a keyword because "`RCPT TO:`" often shows up together in SMTP messages. In addition, a packet payload can also contain multiple keywords. For example, an example SMTP packet payload is as follows. There are three protocol keywords corresponding to `MAIL FROM`, `RCPT TO`, and `DATA`. We give an another example of a binary protocol in Appendix A.

```
MAIL FROM: <alice@gmail.com>
RCPT TO: <joe@BBN-VAX.ARPA>
RCPT TO: <bob@live.cn>
RCPT TO: <carol@microsoft.com>
DATA
```

B. LDA for Keyword Inference

To build the protocol keyword model for each target protocol, we first identify protocol keywords used in protocol communications. In this paper, we identify protocol keywords from raw packet payloads using a generative model called Latent Dirichlet Allocation (LDA) [31], [32]. The LDA model is widely used in statistics, and it is novel in the area of networking and security. By using the LDA model, our approach can group correlated n-grams together to form a protocol keyword. The LDA model treats each packet payload as a probabilistic mixture of keywords, where each keyword in turn represents a probabilistic distribution over n-grams. In Fig. 4, we give a simple example to illustrate the typical procedure of translating network packets into a set of protocol keywords. We describe the mechanism of LDA in detail.

Given a collection $D \equiv \{\{w_{m,i}\}_{i=1}^{N_m}\}_{m=1}^{M}$ of M packets over W n-gram terms, where $w_{m,i}$ represents the ith n-gram in packet m, and N_m is the total number of n-grams in packet m. Note that in the LDA model, each n-gram $w_{m,i}$ corresponds to a specific keyword indicator $z_{m,i}$. More specifically, each packet m is modeled as a probability distribution of protocol keywords (containing a total of K unique keywords), denoted $\vec{\theta}_m = p(\vec{z}|m)$, where each keyword $z = k \in \{1, \cdots, K\}$ is in turn a probability distribution over the n-gram terms $\vec{t} = \{u\}_{u=1}^{W}$, denoted $\vec{\varphi}_k = p(\vec{t}|k)$. The LDA model assumes the following generative process for each packet m in collection D, where $\mathrm{Dir}(\cdot)$ is the Dirichlet distribution, $\mathrm{Mult}(\cdot)$ is the multinomial distribution, and α and β are given hyper-parameters. .

1) Choose a sample keyword distribution $\vec{\theta}_m$ from $\mathrm{Dir}(\vec{\alpha})$.
2) For each n-gram $w_{m,i}$ in packet m:
   a) Choose a keyword indicator $z_{m,i} \in \{1, \cdots, K\}$ from $\mathrm{Mult}(\vec{\theta})$ for n-gram $w_{m,i}$.
   b) For each keyword indictor $z_{m,i}$, choose the corresponding n-gram $w_{m,i}$ from $\mathrm{Mult}(\vec{\varphi}_{z_{m,i}})$.

Note that determining keyword indicators $\vec{z}$ of the LDA model is the core problem for Keyword Identification. By using $\vec{z}$, we can easily calculate the aforementioned two types of distributions: 1) the n-gram distribution for each keyword k, denoted $\vec{\varphi}_k$; and 2) the keyword distribution for each packet m, denoted $\vec{\vartheta}_m$. In the rest of this paper, we use parameter sets $\phi = \{\vec{\varphi}_k\}_{k=1}^{K}$ and $\theta = \{\vec{\vartheta}_m\}_{m=1}^{M}$ to denote the above two types of distributions, respectively.

Next, we want to discuss how to estimate the parameter $\vec{z}$ in the LDA model. Given collection $D \equiv \{\{w_{m,i}\}_{i=1}^{N_m}\}_{m=1}^{M}$, our target posterior distribution is $p(\vec{z}|\vec{w})$, and it can be formulated as follows:

$$p(\vec{z}|\vec{w}) = \frac{p(\vec{z}, \vec{w})}{p(\vec{w})}. \tag{1}$$

Note that exact inference of the target distribution $p(\vec{z}|\vec{w})$ in the LDA model is extremely difficult. First, because denominator $p(\vec{w}) = \prod_{u=1}^{W} \sum_{k=1}^{K} p(w_u, z_u = k)$ involves a summation over K^W items, and thus it generally does not factorize in practice [31], [33]. Second, because exact probabilistic inference is generally computationally intractable in complex graphical models, which have many correlations between the variables (i.e., nodes). Accordingly, we can hardly achieve an

exact-form solution for the Bayesian inference problem defined in (1).

C. Approximate Inference

In practice, we usually resort to three candidate strategies to obtain an approximate inference result for a Bayesian inference problem, including: 1) Variational Inference; 2) Markov Chain Monte Carlo (MCMC); and 3) Expectation Propagation (EP). Among the above strategies, we choose the MCMC algorithm as it is tolerant to local optima, requires little memory, and is competitive in speed [34]. Specifically, we use an MCMC algorithm called Gibbs sampling [35]. Gibbs sampling is an iterative algorithm, in which each iteration obtains the value of each variable by updating a value drawn from the target distribution of that variable (as conditioned by the rest of variables).

For the present problem, $p(\vec{z}|\vec{w})$ shown in (1) denotes our target function. Note that $\vec{w} = \{w_{(m,i)} = t, \vec{w}_{\neg(m,i)}\}$ and $\vec{z} = \{z_{(m,i)} = k, \vec{z}_{\neg(m,i)}\}$, where $\vec{w}_{\neg(m,i)}$ and $\vec{z}_{\neg(m,i)}$ respectively represent $w_{(m,j)}$ and $z_{(m,j)}$ for any m and any $j \neq i$. The basic procedure of Gibbs sampling for keyword inference is as follows.

1) To begin with, we select an arbitrary initial guess for all the keyword indictors $\vec{z}$.
2) Next, we draw a new value for sample $z_{(m,i)}$ from the posterior distribution $p(z_{(m,i)}|z_{\neg(m,i)}, \vec{w})$ conditioned on the rest of variables $z_{\neg(m,i)}$.
3) We make use of the most recent values and update the variable with its new value as soon as it has been sampled.
4) We proceed the above operations for all tokens (m, i) repeatedly.

From the joint distribution, we can derive the full conditional posterior distribution for a keyword index $z_{(m,i)}$ (see [36] for detailed derivation), and the proportional relationship is as follows:

$$p\left(z_{(m,i)} = k|\vec{z}_{\neg(m,i)}, \vec{w}\right) \propto \frac{n_k^{(t)} - 1 + \beta}{\sum_{i=1}^{W} n_k^{(t)} - 1 + W\beta} \cdot \frac{n_m^{(k)} - 1 + \alpha}{\sum_{k=1}^{K} n_m^{(k)} - 1 + K\alpha} \quad (2)$$

where $n_k^{(t)}$ is the number of times that vocabulary term t is assigned to keyword k, and $n_m^{(k)}$ is the number of times keyword k has been observed with an n-gram of packet m.

It can be proved that after a sufficient number of iterations, the above distribution will converge to the true distribution. In natural language processing, we often use perplexity [37] as a measure to make sure the parameters we estimated are generalizable. In Securitas, perplexity is employed by us for two purposes: 1) determine the optimal number of protocol keywords K in collection D; and 2) determine if the Gibbs sampling algorithm has truly converged after L iterations

$$\text{perplexity}(D) = \exp\left\{-\frac{\sum_{m=1}^{M} \log p(\vec{w}_m)}{\sum_{m=1}^{M} N_m}\right\}. \quad (3)$$

D. Protocol Keyword Model

In this paper, the protocol keyword model is referred to by us as a joint probability distribution of n-gram terms $\vec{t}$ and keywords $\vec{z}$. Once the keyword assignments $\vec{z}$ are obtained, the protocol keyword model $\phi = \{\vec{\varphi}_k\}_{k=1}^{K}$ can be estimated according to the following expression:

$$\varphi_{k,t} = \frac{n_k^{(t)} + \beta}{\sum_{t=1}^{W} n_k^{(t)} + W\beta}. \quad (4)$$

In Securitas, we use the corresponding protocol keyword model to extract classification features for each network packet in Feature Extractor.

VII. Feature Extractor

According to the protocol keyword model identified by Keyword Identification, Feature Extractor infers a keyword distribution for unseen packet $\widetilde{m}$ as its classification features. The input to Feature Extractor is network packets, where each packet payload is represented as a sequence of n-grams. The output to Feature Extractor is the keyword distribution of packet $\widetilde{m}$.

Next, we give a detailed description of how to extract our classification features. Given n-grams of unseen packet $\widetilde{m}$, denoted by $\widetilde{\vec{w}}$ and the protocol keyword model ϕ, we can estimate the posterior distribution of keywords $\widetilde{\vec{z}}$ for packet $\widetilde{m}$ using the following Gibbs sampling update rule:

$$p(\widetilde{z}_i = k|\widetilde{w}_i = t, \widetilde{\vec{z}}_{-i}, \widetilde{\vec{w}}_{-i}; \mathcal{M}) \propto \frac{n_k^{(t)} + \widetilde{n}_k^{(t)} - 1 + \beta}{\sum_{i=1}^{W}\left(n_k^{(t)} + \widetilde{n}_k^{(t)} - 1\right) + W\beta} \cdot \frac{n_{\widetilde{m}}^{(k)} - 1 + \alpha}{\sum_{k=1}^{K} n_{\widetilde{m}}^{(k)} - 1 + K\alpha} \quad (5)$$

where variable $\widetilde{n}_k^{(t)}$ represents the number of times that vocabulary term t is assigned to keyword k in packet $\widetilde{m}$. Note that after a sufficient number of iterations, we acquire the keyword assignment $\widetilde{\vec{z}}$ for $\widetilde{m}$, which is then used to calculate a vector $[\vartheta_{\widetilde{m},1}; \vartheta_{\widetilde{m},2}; \ldots; \vartheta_{\widetilde{m},k}; \ldots; \vartheta_{\widetilde{m},K}]$. Note that $\vartheta_{\widetilde{m},k}$ for any k is defined as follows:

$$\vartheta_{\widetilde{m},k} = \frac{n_{\widetilde{m}}^{(k)} + \alpha}{\sum_{k=1}^{K} n_{\widetilde{m}}^{(k)} + K\alpha}. \quad (6)$$

We regard the above vector as our classification feature for packet $\widetilde{m}$.

VIII. Learning Module and Classifier Module

A. Learning Module

The Learning Module is the fifth component of Securitas, and its objective is to develop a classifier that can automatically identify the network traces that correspond to a specific application protocol from mixed Internet traffic. To this end, the Learning Module works on labeled training samples using supervised machine learning techniques. The input to the Learning Module is the classification features obtained by Feature Extractor, and the output to the Learning Module is the corresponding protocol detection model for each target protocol.

In the Learning Module, we leverage the aforementioned vector $[\vartheta_{\widetilde{m},1}; \vartheta_{\widetilde{m},2}; \ldots; \vartheta_{\widetilde{m},k}; \ldots; \vartheta_{\widetilde{m},K}]$ as its K-dimension classification feature for packet $\widetilde{m}$. Next, the detection model

TABLE I
SUMMARY OF THE TRACES

	Size(B)	Packets	Flows	Time period
RealTrace-I	613.17G	1.38G	156.7M	21st, May 2012
RealTrace-II	287.79G	0.47G	74.9M	22nd, July 2012

for each target protocol is trained offline using labeled training samples, denoted by $\{\vartheta_{\widetilde{m}}\}_{m=1}^{Q}$, where Q represents the total number of the training samples. To select the most appropriate classifier for protocol identification, in this process we apply and test three well-known supervised machine learning algorithms, including Support Vector Machine (SVM), C4.5 Decision Tree, and Bayes Network.

B. Classifier Module

As it is shown in Fig. 2, the Classifier Module is the last module of Securitas, and it makes decisions on each raw packet according to the detection model produced by the Learning Module. The unlabeled network packets are grouped into two classes, network packets that belong to the target application protocol under analysis and those that are not.

IX. Experimental Evaluation

To test the effectiveness of Securitas in protocol identification, our research has involved the implementation and extensive evaluation of Securitas with seven typical protocols (i.e., BitTorrent, PPLive, SMTP, DNS, FTP, SIP, and CIFS/SMB). In Section IX-A, we first describe the data set used for evaluating Securitas, next define the evaluation metrics, then discuss the parameters used in Securitas, and finally present our experimental results.

A. Data Set

In our experiments, we work on seven typical and stateful protocols to quantitatively evaluate the effectiveness of our system, including BitTorrent, PPLive, SMTP, DNS, FTP, SIP, and CIFS/SMB. Note that our target protocols involve both connection-oriented protocols (such as TCP) and connection-less protocols (such as UDP). Simultaneously, the target protocols also contain both textual and binary protocols. Next, we give a brief description of the data sets.

The network traces of each application protocol under analysis were obtained from a backbone router of a major ISP in China, which offers diverse services in reality. We present a summary of the network traces in Table I, and both two traces contain many popular applications generating TCP and UDP traffic. In particular, we select four connection-oriented protocols, including BitTorrent, CIFS/SMB, FTP, and SMTP, and three connection-less protocols, including PPLive, SIP, and DNS. More specifically, CIFS/SMB, DNS, FTP, SIP, and SMTP are well-known protocols, which are mainly responsible for file sharing access, domain name services, files transferring, multimedia communication sessions controlling, and e-mail communications, respectively. In addition, BitTorrent and PPLive are peer-to-peer (P2P) applications for file-sharing and streaming video, and they are representatives of new and proprietary protocols. All of the above application protocols are expected to offer a great amount of Internet traffic.

There are several approaches to get raw packets of a specific protocol, that is the ground truth. For example, if the executable code of an application protocol is available, we can run it in a controlled environment to gather protocol data sets, such as the GT method [28] and the sandbox method [29]. In this part, to collect network traces of the target application protocols, we only resort to two simple strategies, transport-layer port numbers and protocol syntax check. However, our system is not limited to the above two strategies.

- Port filter: To collect network traces of CIFS/SMB, DNS, FTP, and SMTP, we simply used the TCP and UDP port numbers to filter traffic—TCP port 445 for CIFS/SMB, UDP port 53 for DNS, TCP port 21 for FTP, and TCP port 25 for SMTP.
- Protocol syntax check: BitTorrent, SIP, and PPLive can use arbitrary ports for communication, and thus port-based classification may become unreliable. In this paper, we develop a DPI classifier, which is explicitly designed for the above protocols, to carry out protocol syntax check.

In the experimental evaluation, we use 10-fold cross validation over the packet traces for measuring the precision and recall of Securitas.

B. Evaluation Metrics for Effectiveness

For a specific application protocol under analysis, we define the following three data sets for further analysis:

- *True Positives* (TP): {the set of packets where each packet is parsed by Securitas as the packet of the application protocol and indeed generated by the application protocol};
- *False Positives* (FP): {the set of packets where each packet is parsed by Securitas as the packet of the application protocol but was not actually generated by the application protocol};
- *False Negatives* (FN): {the set of packets where each packet is parsed by Securitas as not belonging to the application protocol but actually is generated by the application protocol}.

Based on these data sets, we are able to define three metrics (i.e., precision, recall, and F-Measure) to evaluate the quality of our proposed system

$$\mathtt{recall} = \frac{\mathtt{TP}}{\mathtt{TP} + \mathtt{FN}} \tag{7}$$

$$\mathtt{precision} = \frac{\mathtt{TP}}{\mathtt{TP} + \mathtt{FP}} \tag{8}$$

$$\mathtt{F-Measure} = 2 * \frac{\mathtt{precision} * \mathtt{recall}}{\mathtt{precision} + \mathtt{recall}}. \tag{9}$$

Since precision and recall measures respectively reflect two fronts of the whole system, F-Measure is a compromise between precision and recall.

C. Parameter Selection

Our approach involves the following key parameters in the whole process: 1) parameter W: n-gram vocabulary size; 2) parameter L: maximum iteration count for Gibbs sampling algorithm; and 3) parameters α and β: hyperparameters for

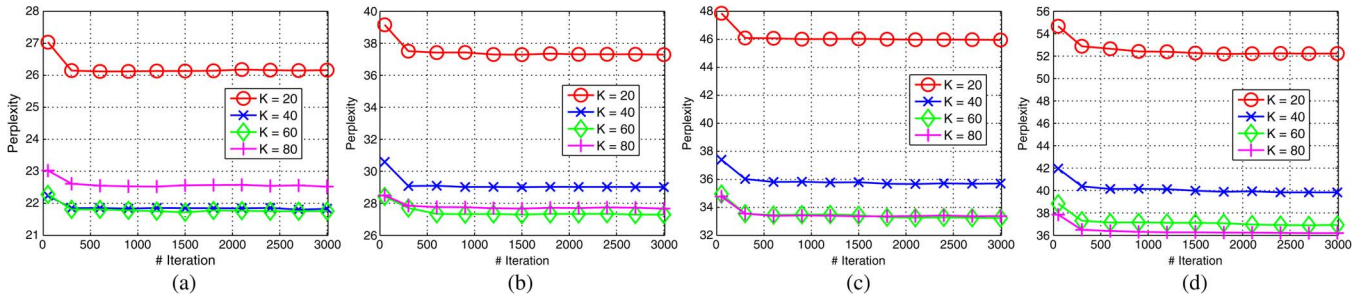


Fig. 5. Selection of L for SMTP protocol. (a) $W = 500$. (b) $W = 1000$. (c) $W = 1500$. (d) $W = 2000$.

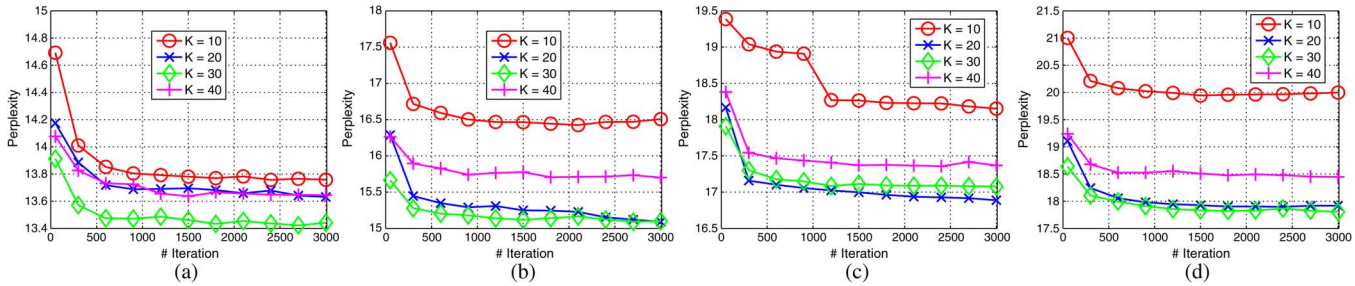


Fig. 6. Selection of L for BitTorrent protocol. (a) $W = 500$. (b) $W = 1000$. (c) $W = 1500$. (d) $W = 2000$.

LDA. Next, we would like to discuss how to select suitable values for the above parameters.

1) n-Gram Vocabulary Size (W): Recall that in Section V, we set the value of n to be 3 for n-gram Generation. Note that the state space of n-grams contains more than 256^3 unique elements, however only a small part of which are formats-related. Thus, the huge size of n-gram state space needs to be reduced to remove redundancy and to avoid the curse of dimensionality for further analysis. Toward this end, we select a subset of n-grams (denoted by W) with high probability of appearance in our traces to construct the n-gram vocabulary of the application protocol. In the following evaluations of Securitas, we vary the ranges of $W \in \{500, 1000, 1500, 2000\}$.

2) Maximum Iteration Count for Gibbs Sampling (L): Recall that in Keyword Identification, we make use of Gibbs sampling algorithm to find protocol keywords for a given application protocol. We used perplexity as the metrics to ensure that the LDA model estimated using Gibbs sampling is generalizable. As Gibbs sampling is an iterative algorithm, we need to first select an appropriate maximum iteration count, denoted by L, for the algorithm to converge. In the following, we choose two example protocols (i.e., SMTP and BitTorrent) to show the procedure of how to select a suitable value of L.

For SMTP, we carry out our experiments for $K \in \{20, 40, 60, 80\}$ and $W \in \{500, 1000, 1500, 2000\}$. Fig. 5 shows the perplexity values for the above values of K and W for SMTP. Note that we observe the perplexity values generally converge by 500 iterations for SMTP. For BitTorrent, we carry out the experiments for $K \in \{10, 20, 30, 40\}$ and $W \in \{500, 1000, 1500, 2000\}$. Fig. 6 shows the perplexity values for the above values of K and W for BitTorrent. Note that we observe the perplexity values typically converge by 1000 iterations for BitTorrent. For the final evaluation of Securitas, we select a conservative value of $L = 3000$ to ensure convergence for off-line analysis, and $L = 500$ for online analysis.

3) Hyperparameters for LDA (α, β): The LDA model presented in Section VI is associated with three basic parameters: α, β, and K. Given the values for parameters α and β in the range of [0, 1], the optimal value of K can be calculated according to the perplexity metrics. In this part, we empirically set $\alpha = 0.1$ and $\beta = 0.01$, which consistently result in good model quality in practice.

D. Effectiveness Results

In this paper, we use real-world network traces to evaluate the quality of our system in terms of precision and recall. Next, we present the precision and recall results of Securitas to show how different values of W and supervised machine learning algorithms affect the performance of Securitas.

1) Precision and Recall: In the evaluations, we conduct the experiments under three supervised machine learning algorithms (i.e., Support Vector Machine, C4.5 Decision Tree, and Bayes Network) for $W \in \{500, 1000, 1500, 2000\}$.

Table II(a) shows the experimental results for BitTorrent protocol. Note that the precision values of BitTorrent vary in the range of 92.45%–96.35% for different parameter settings. In addition, the recall values of BitTorrent vary in the range of 90.26%–99.47% for different parameter settings. For BitTorrent, we observe that Securitas achieves the best experimental results under C4.5 Decision Tree algorithm for $W = 500$.

Table II(b) shows the experimental results for CIFS/SMB protocol. Note that the precision values of CIFS/SMB vary in the range of 98.91%–99.73% for different parameter settings. Furthermore, the recall values of CIFS/SMB vary in the range of 97.4%–99.79% for different parameter settings. For CIFS/SMB, we observe that Securitas achieves the best experimental results under Bayes Network algorithm for $W = 1000$.

TABLE II
EXPERIMENTAL RESULTS ON SEVEN PROTOCOLS. (a) BITTORRENT. (b) CIFS/SMB. (c) DNS. (d) PPLIVE. (e) SIP. (f) SMTP. (g) FTP

(a)

	W	SVM			C4.5 Decision Tree			Bayes Network		
		Rec.%	Prec.%	F-Mea.%	Rec.%	Prec.%	F-Mea.%	Rec.%	Prec.%	F-Mea.%
BitTorrent	500	99.30	93.00	96.05	**98.88**	**95.14**	**96.97**	90.26	96.35	93.20
	1000	98.92	93.00	95.87	99.06	94.81	96.89	90.77	95.90	93.27
	1500	99.30	92.53	95.80	98.84	94.77	96.76	90.82	95.71	93.20
	2000	99.47	92.45	95.61	98.85	94.73	96.60	90.82	95.62	93.16

(b)

	W	SVM			C4.5			Bayes Network		
		Rec.%	Prec.%	F-Mea.%	Rec.%	Prec.%	F-Mea.%	Rec.%	Prec.%	F-Mea.%
CIFS/SMB	500	97.61	98.91	98.26	99.62	99.63	99.63	99.73	99.64	99.69
	1000	97.40	98.97	98.18	98.04	99.54	98.78	**99.79**	**99.64**	**99.72**
	1500	97.84	99.14	98.49	99.68	99.68	99.68	99.79	99.64	99.72
	2000	98.64	99.17	98.91	99.61	99.73	99.67	99.79	99.64	99.72

(c)

	W	SVM			C4.5			Bayes Network		
		Rec.%	Prec.%	F-Mea.%	Rec.%	Prec.%	F-Mea.%	Rec.%	Prec.%	F-Mea.%
DNS	500	98.82	96.65	97.72	99.78	99.30	99.54	98.96	99.67	99.31
	1000	98.89	97.70	98.29	**99.85**	**99.46**	**99.66**	98.21	99.73	98.97
	1500	98.79	96.45	97.60	99.70	99.38	99.54	98.92	99.67	99.29
	2000	98.82	94.16	96.43	99.79	99.37	99.58	99.26	99.66	99.45

(d)

	W	SVM			C4.5			Bayes Network		
		Rec.%	Prec.%	F-Mea.%	Rec.%	Prec.%	F-Mea.%	Rec.%	Prec.%	F-Mea.%
PPLive	500	93.13	94.83	94.88	94.11	95.65	94.88	78.81	88.15	83.22
	1000	**95.42**	**95.55**	**95.49**	93.57	97.21	95.36	76.32	94.63	84.50
	1500	91.80	94.17	92.97	93.99	94.58	94.29	89.43	87.18	88.29
	2000	92.05	94.63	93.32	91.91	95.40	93.62	81.05	90.10	85.34

(e)

	W	SVM			C4.5			Bayes Network		
		Rec.%	Prec.%	F-Mea.%	Rec.%	Prec.%	F-Mea.%	Rec.%	Prec.%	F-Mea.%
SIP	500	99.56	98.23	98.89	99.75	100.00	99.88	99.987	99.998	99.993
	1000	99.90	98.34	99.12	99.98	100.00	99.988	**99.991**	**99.999**	**99.995**
	1500	99.88	98.61	99.24	99.98	100.00	99.99	99.991	99.996	99.993
	2000	99.92	98.75	99.33	99.98	100.00	99.99	99.991	99.996	99.993

(f)

	W	SVM			C4.5 Decision Tree			Bayes Network		
		Rec.%	Prec.%	F-Mea.%	Rec.%	Prec.%	F-Mea.%	Rec.%	Prec.%	F-Mea.%
SMTP	500	93.37	98.77	95.99	**94.61**	**98.63**	**96.58**	88.50	98.59	93.28
	1000	94.26	98.29	96.23	94.78	98.17	96.43	89.71	98.39	93.84
	1500	93.62	95.74	94.67	94.74	95.55	95.15	78.66	98.26	87.37
	2000	94.25	95.48	94.86	94.42	95.51	94.96	85.75	99.79	92.24

(g)

	W	SVM			C4.5 Decision Tree			Bayes Network		
		Rec.%	Prec.%	F-Mea.%	Rec.%	Prec.%	F-Mea.%	Rec.%	Prec.%	F-Mea.%
FTP	500	94.44	99.46	96.89	95.88	99.55	97.68	95.79	99.53	97.62
	1000	94.73	99.52	97.07	96.52	99.56	98.01	90.59	98.51	94.39
	1500	96.86	97.69	97.27	**97.28**	**98.99**	**98.13**	87.16	98.69	92.57
	2000	95.02	97.21	96.10	94.41	97.20	95.79	87.80	98.53	92.86

Table II(c) shows the experimental results for DNS protocol. Note that the precision values of DNS vary in the range of 94.16%–99.73% for different parameter settings. In addition, the recall values of DNS vary in the range of 98.21%–99.85% for different parameter settings. For DNS, We observe that Securitas achieves the best experimental results under C4.5 Decision Tree algorithm for $W = 1000$.

Table II(d) shows the experimental results for PPLive protocol. Note that the precision values of PPLive vary in the range of 87.18%–97.21% for different parameter settings. In addition, the recall values of PPLive vary in the range of 76.32%–95.42% for different parameter settings. For PPLive, we observe that Securitas achieves the best experimental results under SVM algorithm for $W = 1000$.

Table II(e) shows the experimental results for SIP protocol. Note that the precision values of SIP vary in the range of 99.12%–99.91% for different parameter settings. Moreover, the recall values of SIP vary in the range of 81.95%–99.87% for different parameter settings. For SIP, we observe that Securitas achieves the best experimental results under C4.5 Decision Tree algorithm for $W = 500$.

Table II(f) shows the experimental results for SMTP protocol. Note that the precision values of SMTP vary in the range of 95.48%–99.79% for different parameter settings. Furthermore,

the recall values of SMTP vary in the range of 78.66%–94.78% for different parameter settings. For SMTP, we observe that Securitas achieves the best experimental results under C4.5 Decision Tree algorithm for $W = 500$.

Table II(g) shows the experimental results for FTP protocol. Note that the precision values of FTP vary in the range of 97.20%–99.56% for different parameter settings. Furthermore, the recall values of FTP vary in the range of 87.16%–97.28% for different parameter settings. For FTP, we notice that Securitas achieves the best experimental results under C4.5 Decision Tree algorithm for $W = 1500$.

2) Discussion: In the evaluations, SVM algorithm with Radial Basis Function (RBF) kernel is used under the parameters for $\gamma = 0.5$ and $C = 1000$. For C4.5 Decision Tree, there are two important parameters, the confidence factor used for pruning and the minimum number of instances per leaf. The two parameters are set to 0.25 and 2, respectively.

Note that in Table II, we observe the trend that, among the three algorithms we choose, C4.5 Decision Tree algorithm in most cases outperforms that of SVM and Bayes Network for different protocols. In addition, we also observe the recall of Securitas increases for higher values of W, and the precision values of Securitas generally decrease for higher values of W. F-Measure is a compromise between precision and recall.

In our experimental settings, the optimal parameters for Securitas are $W = 1000$ under C4.5 Decision Tree algorithms, and the corresponding precision and recall values on sampled network traces are with average approximately 97.4% and 98.4%, respectively. The experimental results on seven protocols clearly show that Securitas has the ability to conduct the accurate and automated network protocol identification.

X. Comparison to Existing Algorithm

In the experimental evaluation part, we also compare our results of protocol identification to SPI for both UDP and TCP traffic, as both Securitas and SPI are tools for protocol identification by analyzing the payloads of network traces. The method of SPI is first proposed by Finamore *et al.* to carry out Internet classification explicitly targeting UDP traffic [12]. Note that SPI is capable of classifying UDP traffic with good results in different scenarios, such as client–server protocols, VoIP, and P2P Internet applications. Next, Mantia *et al.* extend the concept of stochastic packet inspection to make it suitable for TCP traffic classification [13].

To carry out application protocol identification, SPI first needs to reassemble IP packets into UDP or TCP flows, respectively. Then, for UDP traffic, SPI keeps at least 80 packets of each flow or each endpoint prior to that a classification decision can be taken. Furthermore, to address TCP traffic, SPI adopts a different strategy. Note that SPI needs to observe 16 different flows of each endpoint before an classification decision can be made, where each flow involves at least 5 packets. In order to compare our results to SPI, we also leverage precision and recall introduced in Section IX-B as our metrics to measure the performance of Securitas and SPI.

A. Precision and Recall

In our comparison experiments for the two methods, we choose two connection-oriented protocols, SMTP and CIFS/SMB, and two connection-less protocols, PPLive and DNS.

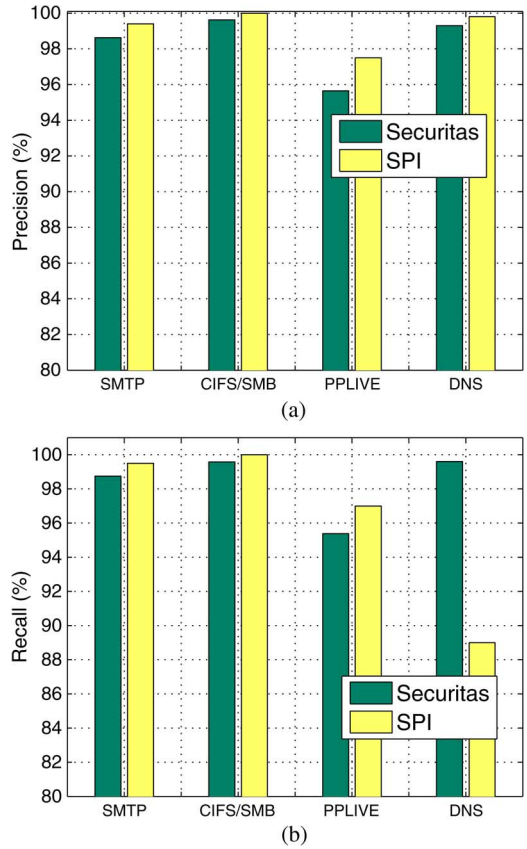


Fig. 7. Experimental results of SPI and Securitas. (a) Precision. (b) Recall.

As it is shown in Fig. 7, for SMTP and CIFS/SMB protocols, the precision of both methods is above 99%, and the recall is above 98%. Note that the experimental results of SPI are slightly better than Securitas. We observe that there are 1.6 million SMTP flows involved in our testing data set, which hold about 3.54 GB, and SPI is only capable of parsing 2580 flows, which account for 67% of bytes carried by SMTP flows. Similarly, we have 5.6 million CIFS/SMB flows in our testing data set, which refer to 702 MB. Since only 18 CIFS/SMB endpoints have at least 80 packets, SPI can account for 0.2% of bytes carried by CIFS/SMB flows.

Fig. 7 also shows the results by running Securitas and SPI for PPLive and DNS protocols. We observe that Securitas outperforms that of SPI on recall of DNS protocol. In other cases, the results of the two methods are very close. However, there are about 1.4 million PPLive flows involved in our testing data set, which hold about 10.38 GB. SPI is able to parse 18 359 flows, which account for 90.5% of bytes carried by PPLive flows. Similarly, we observe about 38 million DNS flows in our testing data set, which refer to 11.26 GB. Due to only 6702 DNS flows having at least 80 packets, SPI is capable of parsing about 5% of bytes carried by DNS flows.

B. Computational Complexity

Note that Securitas runs offline for a given protocol trace in the Modeling and Training phases and runs online in the

TABLE III
SUMMARY OF TIME COMPLEXITY OF SECURITAS'S MODULES. (A) MODELING PHASE. (B) TRAINING PHASE. (C) CLASSIFICATION PHASE

(a)

n-gram Generation	Keyword Identification
O(M*l)	O(M*K*L)

(b)

n-gram Generation	Feature Extractor	Learning Module
O(N*l)	O(N*K*L)	O(N*K^2)

(c)

n-gram Generation	Feature Extractor	Classifier Module
O(l)	O(K*L)	O(h)

Classification phase. Next, we present a theoretic analysis about computational complexity for each phase.

1) Modeling Phase: In the Modeling phase, Securitas contains two important functional modules, including: 2) n-gram Generation, and 3) Keyword Identification. The computational complexity of the two modules is presented in Table III(a), where l is the first constant number of bytes of a packet, M denotes the size of samples for modeling, K is the number of attributes (i.e., keywords), and L is the iteration count in Gibbs sampling algorithm. As $K * L \gg l$, the overall computational complexity of the Modeling phase is $\mathrm{O}(M * K * L)$.

2) Training Phase: In the Training phase, Securitas has: 2) n-gram Generation; 4) Feature Extractor; and 5) Learning Module three key functional modules. In Securitas, we chose C4.5 decision tree as our basic classification algorithm. The time complexity of each module is presented in Table III(b), where l is the first constant number of bytes of a packet, N is the size of samples for training, K is the number of attributes (i.e., keywords), and L is the iteration count in Gibbs sampling algorithm. Note that, in practice, we have the following relations, $K * L \gg l$ and $L > K$. Thus, the overall computational complexity of the Training phase is $O(N * K * L)$.

3) Classification Phase: In the Classification phase, Securitas uses three key functional modules, including: 2) n-gram Generation; 4) Feature Extractor; and 6) Classifier Module, to process a network packet. The computational complexity of each module is presented in Table III(c), where l is the first constant number of bytes of a packet, K is the number of attributes (i.e., keywords), L is the iteration count in Gibbs sampling algorithm, and h represents the maximum depth of the tree. We notice that in practice $K * L \gg l, h$. Thus, to classify a signature, the overall time complexity of Securitas is $\mathrm{O}(K * L)$. The computational complexity of SPI is $\mathrm{O}(G * 2^b)$, where G is the size of groups, and b is the count of bits per group. In practice, K is nearly equal to G, and L is usually greater than 2^b. Therefore, compared to SPI, Securitas needs more time to process a packet for online classification.

C. Discussion

Up until now, SPI is one of the best solutions in the field of protocol identification. We note that both SPI and Securitas have a high performance in classification accuracy. In addition, SPI also performs better than Securitas in computational efficiency for online classification.

However, Securitas is a more robust system, and it distinguishes itself from SPI on the following important aspects.

1) To make a classification decision, Securitas observes one packet, and in SPI, the number of packets is 80 (16 flows $\times$ 5 packets).
2) Securitas does not require to assemble IP packets into TCP or UDP flows to implement protocol identification, thus it can handle more flows with limited memory resources.
3) Securitas works very effectively for both long-live flows (such as SMTP) and short-live flows (such as DNS).

XI. LIMITATIONS

In this section, we briefly summarize three limitations of our proposed system.

Trace Dependency: The quality of the keywords produced by Securitas is limited by the diversity of the network traces used for modeling and training. More specifically, for a specific protocol, if some protocol states rarely appear in our collected protocol traces, it is impossible for Securitas to infer the protocol keywords associated with these states. Thus, as a network trace-based approach, the completeness of our inferred protocol signatures is limited by the variety of behaviors observed in the given trace.

Statistics Dependency: Basically, Securitas has to perform statistical analysis on the payloads of observed network traces. Therefore, Securitas is more inclined to infer protocol keywords, which are statistically significant in the trace. Specifically, some binary protocols may have keywords whose sizes are smaller than n, and in Securitas, such keywords are combined with adjacent bytes to form n-grams. Note that not all these n-grams are statistically significant, and thus some keywords may be ignored in our analysis. In addition, similarly to previous deep packet inspection (DPI)-based approaches, once the payloads of network traces are encrypted, Securitas becomes ineffective.

Computational Efficiency: Our feature extraction process relies on an iterative algorithm (i.e., Gibbs sampling), which is very time-consuming. Compared to existing state-of-the-art solutions, such as SPI, Securitas needs more time to process a network packet. Thus, the computational efficiency of Securitas represents one of our main limitations.

XII. CONCLUSION

In this paper, we propose a semantics-aware classification system, Securitas, which takes network packet traces as input and automatically infers the application of the network traces. Our approach is purely based on network packet traces, and it requires neither protocol executable code nor any prior knowledge on protocol message format. Securitas works with both asynchronous and synchronous application protocols. We have implemented and extensively evaluated Securitas with both textual and binary protocols (BitTorrent, CIFS/SMB, DNS, FTP, PPLive, SIP, and SMTP). Our experimental results show that Securitas can accurately identify each application protocol from mixed network traffic with high precision and recall rates.

TABLE IV
SUMMARY OF ELAPSED TIME OF SECURITAS'S MODULES FOR ONLINE CLASSIFICATION. "ms" DENOTES MILLISECOND (10^{-3} s), AND "μs" DENOTES MICROSECOND (10^{-6} s)

Protocol	n-gram Generation	Feature Extractor	Classifier Module
BitTorrent	62.4 µs	6.3 ms	19.8 µs
CIFS/SMB	77.8 µs	11.1 ms	23.0 µs
DNS	76.2 µs	6.9 ms	17.4 µs
PPLive	71.5 µs	5.1 ms	25.2 µs
SIP	74.7 µs	2.1 ms	4.7 µs
SMTP	71.0 µs	12.5 ms	33.5 µs
FTP	59.7 µs	4.5 ms	19.5 µs

APPENDIX A

This part is to give an example of a binary protocol, and the first 16 B of an example DNS packet are shown as follows:

```
Byte 1−8 : 0xac \0x5d \0x81 \0x80
           \0x00 \0x01 \0x00 \0x02
Byte 9−16 : 0x00 \0x00 \0x00 \0x01
            \0x63 \0x05 \0x62 \0x61.
```

The following n-grams are used to form a protocol keyword of DNS:

{0x81\0x80\0x00, \0x80\0x00\0x01, \0x00\0x01\0x00, \0x01\0x00\0x02, \0x00\0x02\0x00, \0x02\0x00\0x00, \0x00\0x00\0x00, \0x00\0x00\0x01}.

APPENDIX B

Our experiments were all executed on a cluster machine where each node had four quad-core Xeon processors running at 2.13 GHz with 16 GB RAM. The results of elapsed time of Securitas's modules for online classification are in Table IV.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their insightful comments and suggestions on this paper.

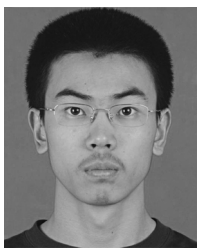
REFERENCES

[1] C. Meiners, E. Norige, A. X. Liu, and E. Torng, "FlowSifter: A counting automata approach to layer 7 field extraction for deep flow inspection," in *Proc. IEEE INFOCOM*, 2012, pp. 1746–1754.
[2] Z. Li *et al.*, "NetShield: Massive semantics-based vulnerability signature matching for high-speed networks," in *Proc. ACM SIGCOMM*, 2010, pp. 279–290.
[3] N. Borisov, D. J. Brumley, and H. J. Wang, "A generic application-level protocol analyzer and its language," in *Proc. NDSS*, 2007.
[4] R. Pang, V. Paxson, R. Sommer, and L. Peterson, "Binpac: A yacc for writing application protocol parsers," in *Proc. 6th ACM SIGCOMM Conf. Internet Meas.*, 2006, pp. 289–300.
[5] J. St Sauver, "A look at the unidentified half of Netflow (with an additional tutorial on how to use the Internet2 Netflow data archives)," 2008 [Online]. Available: http://www.internet2.edu/presentations/jt2008jan/20080122-stsauver.pdf
[6] P. Haffner, S. Sen, O. Spatscheck, and D. Wang, "ACAS: Automated construction of application signatures," in *Proc. ACM SIGCOMM MineNet*, 2005, pp. 197–202.
[7] J. Kannan, J. Jung, V. Paxson, and C. E. Koksal, "Semi-automated discovery of application session structure," in *Proc. ACM SIGCOMM IMC*, 2006, pp. 119–132.
[8] J. Ma, K. Levchenko, C. Kreibich, S. Savage, and G. M. Voelker, "Unexpected means of protocol inference," in *Proc. ACM SIGCOMM IMC*, 2006, pp. 313–326.
[9] W. Cui, J. Kannan, and H. J. Wang, "Discoverer: Automatic protocol reverse engineering from network traces," in *Proc. 16th USENIX SS*, 2007, Art. no . 14.
[10] A. Este, F. Gringoli, and L. Salgarelli, "Support vector machines for tcp traffic classification," *Comput. Netw.*, vol. 53, no. 14, pp. 2476–2490, 2009.
[11] A. Finamore, M. Mellia, M. Meo, and D. Rossi, "Kiss: Stochastic packet inspection," in *Proc. Traffic Monitoring Anal.*, 2009, pp. 117–125.
[12] A. Finamore, M. Mellia, M. Meo, and D. Rossi, "KISS: Stochastic packet inspection classifier for UDP traffic," *IEEE/ACM Trans. Netw.*, vol. 18, no. 5, pp. 1505–1515, Oct. 2010.
[13] G. La Mantia, D. Rossi, A. Finamore, M. Mellia, and M. Meo, "Stochastic packet inspection for TCP traffic," in *Proc. IEEE ICC*, 2010, pp. 1–6.
[14] A. Tongaonkar, R. Keralapura, and A. Nucci, "Santaclass: A self adaptive network traffic classification system," in *Proc. IFIP Netw. Conf.*, 2013, pp. 1–9.
[15] Y. Wang *et al.*, "A semantics aware approach to automated reverse engineering unknown protocols," in *Proc. 20th IEEE ICNP*, 2012, pp. 1–10.
[16] T. Karagiannis, K. Papagiannaki, and M. Faloutsos, "Blinc: Multilevel traffic classification in the dark," *Comput. Commun. Rev.*, vol. 35, no. 4, pp. 229–240, 2005.
[17] L. Bernaille, R. Teixeira, and K. Salamatian, "Early application identification," in *Proc. ACM CoNEXT*, 2006, Art. no. 6.
[18] M. Iliofotou, M. Faloutsos, and M. Mitzenmacher, "Exploiting dynamicity in graph-based traffic analysis: Techniques and applications," in *Proc. 5th CoNEXT*, 2009, pp. 241–252.
[19] J. Zhang, C. Chen, Y. Xiang, W. Zhou, and A. Vasilakos, "An effective network traffic classification method with unknown flow detection," *IEEE Trans. Netw. Service Manage.*, vol. 10, no. 2, pp. 133–147, Jun. 2013.
[20] "Gaim instant messaging client," [Online]. Available: http://gaim.sourceforge.net
[21] A. Tridgell, "How Samba was written," August 2003 [Online]. Available: http://www.samba.org/ftp/tridge/misc/french_cafe.txt
[22] J. Lim, T. Reps, and B. Liblit, "Extracting output formats from executables," in *Proc. 13th WCRE*, 2006, pp. 167–178.
[23] J. Caballero, H. Yin, Z. Liang, and D. Song, "Polyglot: Automatic extraction of protocol message format using dynamic binary analysis," in *Proc. 14th ACM CCS*, 2007, pp. 317–329.
[24] P. M. Comparetti, G. Wondracek, C. Kruegel, and E. Kirda, "Prospex: Protocol specification extraction," in *Proc. 30th IEEE SP*, 2009, pp. 110–125.
[25] C. Y. Cho, D. Babić, R. Shin, and D. Song, "Inference and analysis of formal models of botnet command and control protocols," in *Proc. 17th ACM Conf. Comput. Commun. Security*, 2010, pp. 426–439.
[26] W. Cui, M. Peinado, K. Chen, H. J. Wang, and L. Irun-Briz, "Tupni: Automatic reverse engineering of input formats," in *Proc. 14th ACM CCS*, 2008, pp. 391–402.
[27] A. Tongaonkar, R. Torres, M. Iliofotou, R. Keralapura, and A. Nucci, "Towards self adaptive network traffic classification," *Comput. Commun.*, 2014, to be published.
[28] F. Gringoli *et al.*, "GT: Picking up the truth from the ground for internet traffic," *Comput. Commun. Rev.*, vol. 39, no. 5, pp. 12–18, 2009.
[29] D. Oktavianto and I. Muhardianto, *Cuckoo Malware Analysis*. Birmingham, U.K.: Packt, 2013.
[30] C. D. Manning and H. Schötze, *Foundations of Statistical Natural Language Processing*. Cambridge, MA, USA: MIT Press, 1999.
[31] D. Blei, A. Ng, and M. Jordan, "Latent Dirichlet allocation," *J. Mach. Learning Res.*, vol. 3, pp. 993–1022, Mar. 2003.
[32] B. Croft, "Language models for information retrieval," in *Proc. 19th Int. Conf. Data Eng.*, 2003, pp. 3–7, invited talk.
[33] J. M. Dickey, "Multiple hypergeometric functions: Probabilistic interpretations and statistical uses," *J. Amer. Statist. Assoc.*, vol. 78, no. 383, pp. 628–637, 1983.
[34] D. Gamerman and H. F. Lopes, "Markov chain Monte Carlo: Stochastic simulation for Bayesian inference," 2006.
[35] T. L. Griffiths and M. Steyvers, "Finding scientific topics," *Proc. Nat. Acad. Sci. USA*, vol. 101, pp. 5228–5235, 2004.
[36] G. Heinrich, "Parameter estimation for text analysis," University of Leipzig, Leipzig, Germany, Tech. Rep., 2008.
[37] L. Azzopardi, M. Girolami, and K. van Risjbergen, "Investigating the relationship between language model perplexity and ir precision-recall measures," in *Proc. 26th Annu. ACM SIGIR*, 2003, pp. 369–370.

Xiaochun Yun (M'13) received the B.S. and Ph.D. degrees in computer science from the Harbin Institute of Technology, Harbin, China, in 1993 and 1998, respectively.

He is a Professor with the Institute of Information Engineering, Chinese Academy of Sciences (CAS), Beijing, China. His research interests include network and information security.

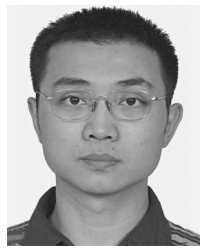
Prof. Yun is a member of the Advisory Committee for Chinese State Informatization and a member of the subject matter expert group of information security technology for the National High Technology Research and Development Program of China. He was honored with the first prize of the Chinese National Award for Science and Technology Progress in 2002 and 2011. He received the Best Paper Award from ICNP 2012.



Yipeng Wang (M'14) received the Ph.D. degree in computer science from the Chinese Academy of Sciences (CAS), Beijing, China, in 2014.

He is an Assistant Professor with the Institute of Information Engineering, CAS. He has published more than 15 research papers in refereed international journals and conferences, such as the IEEE/ACM TRANSACTIONS ON NETWORKING, the IEEE International Conference on Network Protocols (ICNP), and the International Conference on Applied Cryptography and Network Security (ACNS). His research interests are in networking, security, and machine learning, in particular network protocol inference.

Dr. Wang won the Best Paper Award at ICNP 2012.



Yongzheng Zhang (M'13) received the B.S. and Ph.D. degrees in computer science from the Harbin Institute of Technology, Harbin, China, in 2001 and 2006, respectively.

He is a Professor and Ph.D. Supervisor with the Institute of Information Engineering, Chinese Academy of Sciences (CAS), Beijing, China. His research interests include network security, particularly cyberspace security situational awareness.

Prof. Zhang was honored with the first prize of the Chinese National Award for Science and Technology Progress in 2011.

Yu Zhou (M'13) received the Ph.D. degree in computer science from the Harbin Institute of Technology, Harbin, China, in 2009.

He is an Assistant Professor with the Institute of Information Engineering, Chinese Academy of Sciences (CAS), Beijing, China. Before joining CAS, he was a Postdoctoral Research Fellow with Shanghai Jiao Tong University, Shanghai, China, from 2010 to 2012. His current research interests include security problems in multimedia.

Dr. Zhou served as a TPC member of the Second International Workshop on Emerging Multimedia Systems and Applications, and as a reviewer of the *Journal of Systems Science & Complexity*, the Second International Workshop on Emerging Multimedia Systems and Applications, and the 19th Asia-Pacific Conference on Communications.