Journal of Network and Computer Applications 36 (2013) 1070–1077

Contents lists available at SciVerse ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

Position-based automatic reverse engineering of network protocols

Jian-Zhen Luo*, Shun-Zheng Yu

School of Information Science and Technology, Sun Yat-Sen University, Guangzhou Higher Education Mega Center, Guangzhou 510006, PR China

ARTICLE INFO

Article history:
Received 15 July 2012
Received in revised form
18 December 2012
Accepted 13 January 2013
Available online 4 February 2013

Keywords:
Protocol reverse engineering
Protocol keyword
Variance analysis
Message format
State machine

ABSTRACT

Automatic protocol reverse engineering is a process of extracting protocol message formats and protocol state machine without access to the specification of target protocol. Protocol reverse engineering is useful for addressing many problems of network management and security, such as network management, honey-pot systems, intrusion detection, Botnet detection and prevention, and so on. Currently, protocol reverse engineering is mainly a manual and painstaking process which is time-consuming and error-prone. In this paper, we present a novel approach for automatic reverse engineering application-layer network protocols. We extract protocol keywords from network traces based on their support rates and variances of positions, reconstruct message formats, and infer protocol state machines. We implement our approach in a prototype system called AutoReEngine and evaluate it over four text-based protocols (HTTP, POP3, SMTP and FTP) and two binary protocols (DNS and NetBIOS). The results show that our AutoReEngine outperforms the existing algorithms.

1. Introduction

Network protocol specification which is referred to as both message format and protocol state machine has become increasingly important for addressing many management- and security-oriented network problems. For example, with assistance of protocol specifications, network management software can effectively identify and classify protocols or applications in monitored network traffic. Firewalls perform stateful tracking based on the specifications. Intrusion detection systems (IDS) (Paxson, 1999; Amiri et al., 2011) and vulnerability-specific filters (Wang et al., 2004) require the knowledge of protocol specifications to perform deep packet inspection. Based on the protocol specifications, penetration testing system generates network trace for the executable program of an application to reveal its potential vulnerabilities. As the dominant threats in today's Internet, Botnets (Dagon et al., 2007) make use of command-and-control (C&C) protocols to spread in the Internet by infecting vulnerable hosts, and enable the Botnet-attacker to control the infected hosts to execute abusive instructions, such as spamming and DoS attacks. The specifications of C&C protocols are significantly useful for security analysts to understand and take down Botnets. Finally, honey-pot (Provos, 2004) can deceive the attacker into attacking the honey-pot system to provide rich information about the activities and the mechanism of the attack. The honey-pot system needs to know about the specification of the protocol used by the attacker, so that it can generate scripts to interact with the attacker.

To date, protocol specifications are mainly derived from open source document manually. For some open protocols, such as HTTP, FTP and SMTP, their specifications are publicly available. However, there are tens of thousands of closed or private protocols, which are utilized by hackers or enterprises that incline to conceal the knowledge of their protocols. So there is no open document describing the detailed specifications of these protocols. Even for the open protocols, due to the evolution with new functionalities, it will take a long period before the updated specification is available. Thus, we have to perform protocol reverse engineering, a process of extracting the protocol specifications from network traffic generated by target protocols. Nevertheless, traditional techniques of protocol reverse engineering are manual processes, which are tedious, time-consuming, and error-prone. In practice, it takes a long time and great efforts to reverse-engineer closed protocols by means of manual analysis. For instance, it takes more than 10 years to reverse-engineer the CIFS/SMB protocol (Tridgell, 2003). To address these problems, automatic protocol reverse engineering is recently proposed to liberate network analysts from the cumbersome processes of reconstructing protocol specifications.

There are several challenges to reverse-engineer a protocol without prior information about it. Firstly, we have to achieve our task of reconstructing protocol specification under the premise that we have neither prior knowledge about protocol specification nor the type (text- or binary-based) of target protocol. Note that, a text-based protocol usually has message formats that consist of a

* Corresponding author. Tel.: +86 13751739597.
E-mail addresses: helu84@gmail.com (J.-Z. Luo), syu@mail.sysu.edu.cn (S.-Z. Yu).

1084-8045/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.jnca.2013.01.013

number of commands, status codes, and delimiters. A binary-based protocol usually has fixed fields with codes. We propose an unsupervised method for extracting message formats and protocol state machines. More importantly, the proposed method is applicable for both text- and binary-based protocols.

Secondly, there are some strings, such as some terms of hot topics in local news, that appear frequently in messages and are easy to be confused with some protocol keywords (e.g., "GET"). Thus, they are treated as noise in this paper and must be filtered out from the protocol keyword candidates. Different from noisy terms, protocol keywords' positions in messages are relatively fixed. A keyword is usually located in the beginning or at the end of a field or a message. Therefore, frequent strings with small variance of positions in the messages are extracted to be protocol keyword candidates.

Finally, some site-specific strings, such as URL, appear frequently in relatively fixed positions in messages. However, different from the protocol keywords, the site-specific strings appear only in a set of sessions associated with accesses to specific web site. Hence, the site-specific strings can be filtered out by considering the support rate among site-specific session sets (see Section 3).

In this paper, we explore these challenges and present an approach for automatic protocol reverse engineering based on support rate and variance of strings' positions. We implement our approach on a practical system called AutoReEngine to extract protocol message formats and infer protocol state machines from network traces.

The rest of this paper is organized as follows. We firstly study the related work in Section 2, and present the background, models and problem definition in Section 3. Then, we describe techniques of extracting frequent strings in Section 4, and demonstrate the approaches of extracting message formats and inferring protocol state machines in Section 5 and Section 6, respectively. Finally, we evaluate the proposed approach in Section 7 and make a conclusion in Section 8.

2. Related work

The concept of protocol reverse engineering has been proposed for a long time. However, traditional approach is time-consuming and error-prone. Usually, it takes a long period to perform manual reverse-engineering, just as the example of re-engineering CIFS/SMB protocol (Tridgell, 2003) mentioned in Section 1. To address the problem, automatic protocol reverse engineering has been proposed and attracted a great deal of attention in research in recent years. Automatic protocol reverse engineering aims to extract protocol message format and state machine without neither manual participant nor prior knowledge about the target protocol specification. It greatly reduces the workload of network analysts, and improves the efficiency of protocol analysis.

An early attempt was made to discover protocol format using automatic method derived from the Protocol Information Project lead by Beddoe (2004). Beddoe proposes to apply sequence alignment algorithm (Needleman and Wunsch, 1970; Smith and Waterman, 1981) found in the bio-informatics field to extract protocol structure and infer message fields from network traces. Cui et al. (2007) propose a tool called Discoveror to extract message formats from network traffic leveraging recursive clustering and type-based sequence alignment. Quite different from Protocol Information Project (Beddoe, 2004) and Discoveror (Cui et al., 2007), our approach is based on the Apriori algorithm (Agrawal and Srikant, 1994) which is widely used in the field of data mining, and considers position-based features of protocol keywords.

Automatic protocol reverse engineering is also related to the field of application replay and protocol identification. ScriptGen (Leita et al., 2005) and RolePlayer (Cui et al., 2006) apply byte-wise sequence alignment techniques to analyze network trace and generate correct replies to continue the interaction with a malicious program automatically. They focus on identifying fields that are variable among messages during the communication, while we aim to reconstruct the whole structure of message and infer protocol state machine.

Wang et al. (2012) utilize multiple sequence alignment (MSA) technique to generate regular expression signatures with a certain subset of standard syntax rules for identifying application protocols. They aim to extract unique features of target applications for innocuous application identification.

In the past few years, dynamic analysis (Newsome, 2005) techniques have been applied to the field of automatic protocol reverse engineering. The systems based on dynamic analysis techniques take both executable binary programs and messages of target protocol as input to infer message format by observing the way how the programs processed messages they have received or sent. Polyglot (Caballero et al., 2007), Tupni (Cui et al., 2008), AutoFormat (Lin et al., 2008), Prospex (Comparetti et al., 2009) and Dispatcher (Caballero et al., 2009) are some typical systems based on dynamic analysis techniques. Compared to approaches by only taking network trace as input, such as Protocol Information Project and Discoveror, approaches based on dynamic analysis can extract more semantic information, and have higher accuracy. However, the binary programs implementing target protocols are not always available or easy to be obtained. Moreover, more and more malware programs make use of obfuscation interference techniques to protect themselves from being detected and reverse-engineered. On the other hand, the approaches using only network traces need not obtain binary programs of target protocols and it is much easier to collect network traces than to obtain executable binaries. So it is much convenient and easy to dispose a system based on these approaches. Hence, although limited semantic information is inferred, the approaches using only network traces are still very practicable.

3. Problem definition

The basic unit of communication between processes on Internet hosts is session (Ma et al., 2006). Each session is identified by its five-tuple key consisting of initiator address, initiator port number, responder address, responder port number, and the type of transport layer protocol. A session is a pair of opposite flows, and each flow consists of a sequence of messages sent by the initiator to the responder or by the responder to the initiator.

There are many types of messages in a session. The structure of message is determined by the message format. The message formats consist of many fields. Taking HTTP as example, some fields (e.g., "GET *URL*") contain the keywords (e.g., "GET") and their corresponding data part (e.g., "*URL*"), and other fields contain status code, such as "200 OK". On the other hand, the order in which different types of messages should be sent during a session is determined by the protocol state machine.

In this paper, we denote the alphabet as

$$D = \{\backslash 0x00, \backslash 0x01, \ldots, \backslash 0xFF\}. \quad (1)$$

Each value in the alphabet D represents a single-byte character. A string set over D, denoted by I, can be expressed as

$$I = \{i_k \mid i_k = \overline{d_0 d_1 \ldots d_l \ldots d_{n_k}} \wedge d_l \in D,\ l = 0,1,2,\ldots,n_k\}. \quad (2)$$