Computer Networks 57 (2013) 451–474

Contents lists available at SciVerse ScienceDirect

Computer Networks

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

Automatic protocol reverse-engineering: Message format extraction and field semantics inference

Juan Caballero [a,*], Dawn Song [b]

[a] IMDEA Software Institute, Madrid, Spain
[b] University of California, Berkeley, CA, USA

ARTICLE INFO

Article history:
Received 20 December 2011
Received in revised form 25 May 2012
Accepted 10 August 2012
Available online 11 September 2012

Keywords:
Protocol reverse-engineering
Active botnet infiltration
Message format extraction
Field semantics inference
Command-and-control protocol

ABSTRACT

Understanding the command-and-control (C&C) protocol used by a botnet is crucial for anticipating its repertoire of nefarious activity. However, the C&C protocols of botnets, similar to many other application layer protocols, are undocumented. Automatic protocol reverse-engineering techniques enable understanding undocumented protocols and are important for many security applications, including the analysis and defense against botnets. For example, they enable active botnet infiltration, where a security analyst rewrites messages sent and received by a bot in order to contain malicious activity and to provide the botmaster with an illusion of successful and unhampered operation.

In this work, we propose a novel approach to automatic protocol reverse engineering based on dynamic program binary analysis. Compared to previous work that examines the network traffic, we leverage the availability of a program that implements the protocol. Our approach extracts more accurate and complete protocol information and enables the analysis of encrypted protocols. Our automatic protocol reverse-engineering techniques extract the message format and field semantics of protocol messages *sent* and *received* by an application that implements an unknown protocol specification. We implement our techniques into a tool called Dispatcher and use it to analyze the previously undocumented C&C protocol of MegaD, a spam botnet that at its peak produced one third of the spam on the Internet.

1. Introduction

Protocol reverse-engineering techniques extract the specification of unknown or undocumented network protocols and file formats. Protocol reverse-engineering techniques are needed because many protocols and file formats, especially at the application layer, are closed (i.e., have no publicly available specification). For example, malware often uses undocumented network protocols such as the command-and-control (C&C) protocols used by botnets to synchronize their actions and report back on the nefarious activities. Commercial off-the-shelf applications also use a myriad of undocumented protocols and file formats. Closed network protocols include Skype's protocol [1]; protocols used by instant messaging clients such as AOL's ICQ [2], Yahoo!'s Messenger [3], and Microsoft's MSN Messenger [4]; and update protocols used by antivirus tools and browsers. Closed file formats include the DWG format used by Autodesk's AutoCAD software [5] and the PSD format used by Adobe's Photoshop software [6].

A detailed protocol specification can enable or enhance many security applications. For example, in this work we enable active botnet infiltration by extracting the specification of the C&C protocol used by the MegaD spam botnet and use it for deep packet inspection and rewriting of the C&C communication. Protocol specifications are also the input for generic protocol parsers used in network

* Corresponding author. Tel.: +34 913363742.
E-mail addresses: juan.caballero@imdea.org (J. Caballero), dawnsong@cs.berkeley.edu (D. Song).

1389-1286/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.comnet.2012.08.003

Table 1
Field attributes used in this work. Each attribute captures a property of the field.

Attribute	Value
Field range	Start and end offsets in message
Field boundary	Fixed-length (*l*), variable-length (*Length*), variable-length (*Delimiter*)
Field dependencies	Length (x_i), delimiter (x_i), checksum ($x_i, \ldots, x_j$)
Field semantics	The type of data the field carries. A value from Table 3

monitoring [7,8] and can be used to build protocol-aware fuzzers that explore deeper execution paths than random fuzzers can [9], as well as to generate accurate fingerprints required by fingerprinting tools that remotely distinguish among implementations of the same specification [10].

Currently, protocol reverse-engineering is mostly a time-consuming and error-prone manual task. Protocol reverse-engineering projects such as the ones targeting the MSN Messenger and SMB protocols from Microsoft [11,12],[1] the Yahoo! Messenger protocol [14], or the OSCAR and ICQ protocols from AOL [15,16], have all been long term efforts lasting years. In addition, protocol reverse-engineering is not a once-and-done effort, since existing protocols are often extended to support new functionality. Thus, to successfully reverse engineer a protocol in a timely manner and keep up the effort through time, automatic protocol reverse-engineering techniques are needed.

Previous work on automatic protocol reverse-engineering proposes techniques that take as input network data [17–19]. Those techniques face the issue of limited protocol information available in network traces and cannot address encrypted protocols. To address those limitations, we present a new approach for automatic protocol reverse-engineering, which leverages the availability of a program that implements the protocol. Our approach uses dynamic program binary analysis techniques and is based on the intuition that monitoring how the program parses and constructs protocol messages reveals a wealth of information about the message structure and its semantics.

Compared to network traces, program binaries contain richer protocol information because they represent the implementation of the protocol, which is the most detailed description of the protocol in absence of the specification. Understanding the protocol implementation can be beneficial even for protocols with a publicly available specification, because implementations often deviate from the specification. In addition, for encrypted protocols, the program binary knows the cryptographic information required to decrypt and encrypt protocol data. Thus, we can wait until the program decrypts the received network data to start our analysis and stop it before the program encrypts the network data to be sent in response, thus revealing the structure and semantics of the underlying protocol.

1.1. Our work in context

This work comprises research published in two conference articles. The first article appeared in the proceedings of the 14th ACM Conference on Computer and Communications Security (CCS 2007). It presented a system called *Polyglot* [20], which implemented the first approach for automatic protocol reverse-engineering using dynamic binary analysis. Polyglot uses the intuition that monitoring the execution of a program that implements the protocol reveals a wealth of information about the protocol. Polyglot extracts only the message format of a received message. The second article appeared in the proceedings of the 16th ACM Conference on Computer and Communications Security (CCS 2009). It presented a system called *Dispatcher* [21], which in addition to the techniques introduced in Polyglot, implemented techniques to extract the message format for a sent message. It also implemented semantics inference techniques for both sent and received messages, which we had previously introduced in a Technical Report in 2007 [22].

After the publication of Polyglot, other research groups published automatic protocol reverse-engineering techniques that used dynamic binary analysis for extracting the protocol grammar [23–25] and the protocol state-machine [26]. The works that focus on protocol grammar extraction use the approach we introduced in Polyglot of monitoring the execution of a program that implements the protocol. Their techniques target two issues: (1) they consider the message format to be hierarchical [23–25], rather than flat as considered in Polyglot and (2) they extend the problem scope from extracting the message format as done in Polyglot, to extracting the protocol grammar by combining information from multiple messages [23,25]. In Dispatcher we still focus only on message format extraction because it is a pre-requisite for both protocol grammar and state-machine extraction, but we consider the hierarchical structure of the protocol messages. In this work, we present a unified view of the techniques introduced in Polyglot and Dispatcher that considers the hierarchical structure of protocol messages. We also unify the protocol nomenclature used across the different protocol reverse-engineering works.

2. Overview and problem definition

In this section we introduce automatic protocol reverse-engineering and its goals, describe the scope of the problem we address, introduce common protocol elements and terminology, formally define the problem, and provide an overview of our approach.

2.1. Automatic protocol reverse-engineering

The goal of automatic protocol reverse-engineering is given an undocumented protocol or file format to extract

[1] Microsoft has since publicly released the specification of both protocols as part of their Open Specification initiative [13].