Computers & Security (2006) **25**, 221—228

Computers & Security

www.elsevier.com/locate/cose

An empirical examination of the reverse engineering process for binary files

Iain Sutherland [a,*], George E. Kalb [b], Andrew Blyth [a], Gaius Mulley [a]

[a] *School of Computing, University of Glamoran, Treforest, Wales, UK*
[b] *The Johns Hopkins University, Information Security Institute Baltimore, Maryland, USA*

Received 18 November 2004; accepted 4 November 2005

KEYWORDS
Reverse engineering;
Software protection;
Process metrics;
Binary code;
Complexity metrics

Abstract Reverse engineering of binary code file has become increasingly easier to perform. The binary reverse engineering and subsequent software exploitation activities represent a significant threat to the intellectual property content of commercially supplied software products. Protection technologies integrated within the software products offer a viable solution towards deterring the software exploitation threat. However, the absence of metrics, measures, and models to characterize the software exploitation process prevents execution of quantitative assessments to define the extent of protection technology suitable for application to a particular software product. This paper examines a framework for collecting reverse engineering measurements, the execution of a reverse engineering experiment, and the analysis of the findings to determine the primary factors that affect the software exploitation process. The results of this research form a foundation for the specification of metrics, gathering of additional measurements, and development of predictive models to characterize the software exploitation process.
© 2005 Elsevier Ltd. All rights reserved.

Introduction

Deployed software products are known to be susceptible to software exploitation through reverse engineering of the binary code (executable) files. Numerous accounts of commercial companies reverse engineering their competitor's product, for purposes of gaining competitive advantages, have been published (Bull et al., 1995; Chen, 1995; Tabernero, 2002). Global movement towards the use of industrial standards, commercially supplied hardware computing environments, and common operating environments achieves software engineering goals of interoperability, portability, and reusability. This same global movement results in a reduced cost of entry for clandestine software exploiters to successfully reverse engineer a binary code file. A software exploiter, with rudimentary skills, possesses a threat to recently deployed commercial software product because (1) machine-code instruction set and executable file

* Corresponding author.
E-mail address: isutherl@glam.ac.uk (I. Sutherland).

formats (Tilley, 2000) are routinely published, (2) hex editors, dissemblers, software in-circuit emulators tools are readily available via Internet sources, and (3) similar attack scenarios involving reverse engineering of binary code files are readily accessible through numerous hacking websites. There are also legitimate reasons for reverse engineering code in such cases as legacy systems (Muller et al., 2000; Cifuentes and Fitzgerald, 2000) and so there is a body of published academic material (Weide et al., 1995; Interrante and Basrawala, 1988; Demeyer et al., 1999; Wills and Cross, 1996; Gannod et al., 1988) to which a software exploiter could refer although the main focus of this effort is at source code level (Muller et al., 2000).

The commercial software product developer is forced to employ various protection technologies to protect both the intellectual property content and the software development investment represented by the software asset to be released into the marketplace. The commercial software product developer must determine the appropriate protection technologies that are both affordable and supply adequate protection against the reverse engineering threat for a desired period of performance.

The absence of predictive models that characterize the binary reverse engineering software exploitation process precludes an objective and quantitative assessment of the time since first release of the software asset to when software exploitation is expected to successfully extract useful information content. Similar to parametric software development estimation models (e.g., COCOMO), the size and complexity of the binary code file to be reverse engineered are considered to be a prime contributing factor to the time and effort required to execute the reverse engineering activity. Additionally, the skill level of the software exploiter is also considered to be a primary contributing factor. This paper describes the execution of an experiment to derive empirical data that will validate a set of proposed attributes that are believed to be the primary factors affecting the binary reverse engineering process.

Background

An insider is assumed to have access to developmental information resources pertaining to the commercial software product including the product source code. An outsider does not have access to this information and must resort to analysis of available software product resources. Such available software product resources may be little more than the binary code file as released from the original developer. The outsider is forced to execute a binary reverse engineering activity beginning with the binary code file and concluding when some desired end goal has been achieved.

The entry criterion is defined as the time when the outsider first obtains a copy of the binary code file so as to commence the reverse engineering process. The commercial software product vendor must assume that this entry criterion coincides with the first market release of the product.

The exit criterion is determined by the time when the outsider has satisfied a particular end goal for the software exploitation process. Unlike software development activities where the singular end goal is to deliver a reasonably well-tested software product to an end user given the available funding and schedule resources, binary reverse engineering activities may have multiple software exploitation end goals (Kalb). The first software exploitation end goal is defined as obtaining sufficient information regarding the software product's operational function, performance, capabilities, and limitation. Satisfying this first software exploitation end goal enables the software exploiter to transfer the information gathered to other software products that are either in development or are already deployed. The second software exploitation end goal builds upon the first and is defined as enabling minor modifications to alter/enhance the deployed software product. Satisfying this second software exploitation end goal enables (1) circumvention of existing performance limiters and protection technologies to enhance the operational performance of the deployed software product, and/or (2) insertion of malicious code artefacts to corrupt the execution of the deployed software product. The third software exploitation end goal builds upon the previous two and is defined as enabling major modifications to enhance the operational performance of the deployed software product. Satisfying this third software exploitation end goal enables a significant alteration of the deployed software product's functional and operational performance characteristics.

Regardless of the particular software exploitation end goal to be obtained, the software exploitation process must be defined to base a series of experiments that will enable the capturing of measurement data. This software exploitation process commences when the exploiter acquires the binary code file that represents the subject for the reverse engineering activity. For network-centric computing, this acquisition step is rather expediently performed and may be no more effort than locating the particular executable or load file that will be the subject of subsequent reverse engineering activities. For commercial software