A Study of Software Protection Techniques

Jan M. Memon Asma Khan Amber Baig Asadullah Shah
Department of Computer Science
Isra University
Hyderabad, Pakistan
{janmohd, asmakhan, amberbaig, asadullah}@isra.edu.pk

Abstract- **Software piracy and tampering is a well known threat the world is faced with. There have been a lot of attempts to protect software from reverse engineering and tampering. It appears as if there is an ongoing war between software developers and crackers, both parties want to get an upper hand over each other as the time passes. Some of the ample techniques of software protection are reviewed, including multi-block hashing scheme, hardware based solutions, checksums, obfuscation, guards, software aging, cryptographic techniques and watermarking. All of these techniques play their parts imparted on them to protect the software from malicious attacks.**

Keywords: *reverse engineering, software protection, piracy, obfuscation, software aging, watermarking, checksums*

I. INTRODUCTION

Piracy of software has always been one of the burning threats to the software industry [33, 34, 35], especially after the advent of the Internet and the availability of many software analysis tools. Computer software is an important asset to any organization developing it as massive investment of time; money and intellectual capital are involved in its production. However, once produced, software is at risk to theft and misuse. It is estimated that tens of billion dollars of revenue is lost by the software industry due to software piracy alone. Piracy is no longer the only issue, but software tampering with the malicious intent of planting a Trojan horse in the end user's system is a worrisome possibility.

Today's complex software is of much value to its creator, whether that be a company with many products, or the only product of a small company. Software piracy is a major trade and industry problem. Great losses to software producers are due to unauthorized use and distribution of their products. Of much concern is the protection of the software, such that it will always retain the functionality which its creators intended, always protect the intellectual belongings embedded in the program, and spoil attempts to make illegitimate copies of the program. Much has been done to thwart network originated attacks, but little has been done to thwart hardware and software based attacks on the intellectual property embedded within a program. These attacks include modifications to a program to omit critical checks, such as license file checks, or reverse engineering of a key piece of a program's functionality.

Reverse engineering can also leads to code-lifting that consists of reusing a crucial part of a code, without necessarily understanding the internals of how it works, in some other software. These attacks are potentially very costly to the original software developer as they allow a competitor to abolish the developer's competitive edge by rapidly filling a technology gap through insights gleaned from examining the software.

In this paper, some of the approaches are discussed which are employed by software developers to provide ample software protection. In the next section, Multi-block hashing scheme is discussed. In section 3 some of the widely used hardware based approaches are discussed followed by checksums in section 4. One of the most promising software protection techniques, obfuscation is discussed in section 5, followed by Guards and Software Aging in sections 6 and 7, respectively. In section 8, some of the cryptographic techniques are discussed for protecting Java programs from reverse engineering along with some watermarking techniques in section 9. Comparison of these techniques is made in section 10 and finally, the conclusions are drawn in section 11.

II. MULTI-BLOCK HASHING SCHEME

In this scheme, a binary program is divided into several different sized independent blocks; each block does not contain any branching instruction to the next block. The blocks are encrypted with the hash value of the previous block. The last block is encrypted with the hash value of the second last block; the second last block is encrypted with the hash value of the third last block and so on. The first block is not encrypted, it is considered as the basic block. A *program controller* is built to contain the decrypting routine and is placed at the end of the program. Each block contains *pointer* to the *program controller* to decrypt the next block with its *hash value*. When the program executes, the basic block calculates its hash value and passes it to the program controller to decrypt the next block and continues its execution. The next block also does the same thing and this mechanism continues till the end. The hash values are calculated dynamically during program

T. Sobh (ed.), Innovations and Advanced Techniques in Computer and Information Sciences and Engineering, 249–253.

execution, so the adversary cannot alters the program statically as the binary codes are in encrypted form after protection [1].

III. HARDWARE BASED SOLUTIONS

The most common hardware based protection approach uses a trusted processor, a tamper-resistant hardware, which checks and verifies every piece of software that is requested to be run on the computer [4, 6]. The hardware, for example, stores all of the keys necessary to verify digital signatures, decrypt license files, decrypt software before running it, and encrypt messages during any message transfer. The same software or media could be encrypted in a different way for each trusted processor that would execute it, because each processor would have a distinctive decryption key. This would put quite a dent in the piracy problem, as disseminating your software or media files to others would require the same hardware, which the software pirates cannot produce.

The other approach is to associate the software with the hardware of a particular machine. So that when the software runs, it secretly sends the serial number associated with the hardware to the software company [2]. In yet another approach, a movable piece of hardware, such as a *smart card* or a *dongle* or a *secure digital memory card*, is used to protect the software. The software is prevented from running unless some specific information from the trusted hardware is retrieved [3, 5, 8].

In addition, all the hardware based protection mechanisms suffer from the general drawback of lacking user friendliness, which reduce the importance of using such mechanisms for software protection.

IV. CHECKSUMS

A straightforward technique is to calculate the checksum of a program using any message digest algorithm and integrate it with the program. When the program is run, a checksum of the original program is calculated and compared with the already computed checksum. If they both match, the program is considered unmodified [8]. However, once the checksum is detected in a program, an attacker can remove them or patch around them.

The mechanism of integrating checksum within a program is also used in software tamper resistance checks of applications through dynamic monitoring. Two different processes are created, *Monitor* or *M – process*, and *Program* or *P – process*. The *M – process* is designed explicitly to monitor the control flow of the main program process. The *P-process* sends information on its instantiated control flow at a fixed period to the *M –process*. If there is a violation of the control flow conditions captured within the *M – process*, the *M–process* takes an anti-tamper action such as termination of the *P-process*. The length and checksum of the *P–process* is integrated in the *M–process*. When the program is run, the *M–process* computes the length and checksum of the *P–process* to ensure that an adversary has not added some extra instructions to the *P–process* [7].

V. OBFUSCATION

Obfuscation transforms a program into another program that has an equivalent behavior but which is harder to understand [9, 10, 11]. Obfuscation has been proposed as the solution to problems such as protection of transient secrets in programs, protection of algorithms, license management for software, and protection of digital watermarks in programs, software-based tamper resistance, and protection of mobile agents [12].

Obfuscation transformation quality is a combination of four measures; *potency*, which measures how much obfuscation is applied to the program; *resilience*, which measures how well the transformation holds up under attack from reverse engineering tools; *stealth*, which measures how well the obfuscated code blends in with the rest of the program; and *cost*, which measures performance penalty which a transformation incurs on an obfuscated application [19].

A research was conducted on whether automated obfuscation tools can produce obfuscated code that is resistant enough to analysis so that deobfuscation always requires significant manual analysis (or manual guidance of deobfuscation tools). If those techniques are sufficient to force a manual component to deobfuscation, they have provided a positive cost/benefit tradeoff between obfuscation and deobfuscation since the obfuscation techniques were automatically applied without human involvement, and are therefore relatively inexpensive [12].

An experiment was conducted on varying degrees of obfuscated programs and it was found that reverse engineering tools performed better on un-obfuscated codes as compared to the obfuscated ones [32]. An approach of using class *De-compilers* to provide security of Java applications, used in software *REVEAL* [14], is to first construct source code through De-compiler and then apply some obfuscation transformations to the generated source code. A theoretical basis for software obfuscation was provided on the basis of complexity problem, where proposed techniques were implemented using a prototype tool [16]. It was proved that a general obfuscator cannot be created that can transform a vast number of applications and perform a variety of tasks; instead, specific obfuscators should be created for performing specific tasks [15].