Software Piracy Prevention through Diversity

Bertrand Anckaert
banckaer@elis.ugent.be

Bjorn De Sutter
brdsutte@elis.ugent.be

Koen De Bosschere
kdb@elis.ugent.be

Electronics and Information Systems Department
Ghent University
Sint-Pietersnieuwstraat 41
9000 Ghent, Belgium

ABSTRACT

Software piracy is a major concern for software providers, despite the many defense mechanisms that have been proposed to prevent it. This paper identifies the fundamental weaknesses of existing approaches, resulting from the static nature of defense and the impossibility to prevent the duplication of digital data. A new scheme is presented that enables a more dynamic nature of defense and makes it harder to create an additional, equally useful copy. Furthermore it enables a fine-grained control over the distributed software. Its strength is based on diversity: each installed copy is unique and updates are tailored to work for one installed copy only.

Categories and Subject Descriptors

K.5.1 [**Legal Aspects Of Computing**]: Hardware/Software Protection—*copyrights;licensing*; D.2.0 [**Software Engineering**]: General—*protection mechanisms*; D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement

General Terms

Economics, Legal Aspects

Keywords

Copyright Protection, Software Piracy Prevention, Identification, Authentication, Intellectual Property Protection, Diversity, Tailored Updates

1. INTRODUCTION

According to reports on software piracy [14, 17], no existing protective measures have been able to meet the major challenge posed by software piracy. Among the approaches that have been explored in recent history to address the problem of software piracy are legal, ethical and technical means.

DRM'04, October 25, 2004, Washington, DC, USA.

Legal means are based on the fear of consequences of violating piracy laws. But while in most software piracy cases the legal means are available, prosecution on a case by case basis is economically inviable. Furthermore, it is conceived as bad publicity and can take a long time.

Ethical measures relate to making software piracy morally unappealing. While the intentions are laudable, it takes even more effort and time to change the moral standards of a large group of people.

The existing technical means almost all have a static nature of defense, in which a protection mechanism is built into the distributed software. Once this protection is broken no further steps can be taken to protect the intellectual property. And since any static protection is eventually broken, the existing techniques are not satisfactory at all.

To tackle this problem, this paper presents an alternative technical protection scheme, whose strength is based on diversity. In the scheme we present, each installed copy of a program is unique. More precisely, each installed copy differs enough from all other installed copies to guarantee that successful attacks on its embedded copyright protection mechanism cannot be generalized successfully to other installed copies.

Furthermore, the proposed scheme includes software updates to migrate from a static nature of defense to a more dynamic one. In particular, software updates in our scheme are crafted to ensure that they work for one, and only one, installed copy. When updates are no longer provided for installed copies that are known to be illegitimate, a pirate needs to break through a new line of defense with every critical update.

An additional advantage of the proposed scheme is a fine-grained level of control over the distributed copies. This follows from the fact that a software provider in our scheme can enable the installation of a copy on an arbitrary number of machines, or even tolerate an arbitrary level of software piracy. We will refer to the latter as *piracy discrimination.*

The remainder of this paper is structured as follows: Section 2 gives an overview of related work and identifies the fundamental weaknesses of existing technical means. Section 3 introduces the software distribution model we will assume, and the new scheme is presented in Section 4. Claims for the benefits of the scheme are made in section 5 and the need for piracy discrimination is discussed in Section 6. Practical issues are considered in Section 7 and the additional costs of the proposed scheme are explored in Section 8. Finally, Section 9 concludes.

2. RELATED WORK

This section provides an overview of related work and identifies the fundamental weaknesses of the existing technical means for software piracy prevention.

2.1 Overview

Software diversity [5] as such has received little attention in the academic literature. Until now its application has been limited to protecting a trusted host against attacks from malicious code such as viruses. The fundamental idea is simple: in nature, genetic diversity provides protection against an entire species being wiped out by a single virus or disease. The same idea applies to software, with respect to resistance to the exploitation of software vulnerabilities and program-based attacks. This way an exploit crafted to succeed on one instance will not necessarily work against a second [18].

In contrast with software diversity, piracy prevention has drawn a lot of attention from both the software industry and the academic community. This large interest in piracy prevention is largely due to the huge financial losses attributed to software piracy, and has resulted in a plethora of technical means. These include hardware-based and software-based approaches.

All hardware-based approaches we are aware of use tokens. In these schemes, it is impossible to execute the program without the presence of a trusted hardware component, such as a specific CD, dongle, smart card, ... The link between the token and the software can be weak or strong. We call the link weak when the software merely checks for the presence of the token. When the software cannot run without the information or functionality of the token, we call the link strong.

The most common software-based approaches are also based on the use of a token such as a license key, a license file or an activation code. Likewise the token can be weakly or strongly linked to the software. An example of a strongly linked software token is an encrypted program where the token contains the decryption key.

While software watermarking and fingerprinting [6, 19] are not techniques that prevent the copying of software itself, they dissuade the pirate by increasing the likelihood of being caught. This is done by adding an identification to each released copy, making it possible to trace the original creator or legitimate owner of the copy.

One advantage of fingerprinting over most other copy protection techniques is that it is more difficult for an attacker to be sure that he has removed a fingerprint, than it is to be sure that a copy protection mechanism has been cracked. Whereas the latter can easily be tested, i.e. a copy works or does not work, the fact that no more fingerprint is present is not decidable, and hence a pirate can modify a fingerprinted program at will, but when he wants to use or redistribute it illegally, the fear that he could be identified remains. On the other hand, fingerprinting comes with the obvious disadvantage of its reliance on cumbersome legal measures.

Like fingerprinting, software aging [15], increases the likelihood of the pirate being caught. This technique relies heavily on program updates, which are crafted to ensure that more recent versions can read the output of older versions, but not vice versa. For this technique to work, it is assumed that illegitimate users interact only with the original pirate to obtain these updates. As a result pirated software becomes either decreasingly usable because it is not kept up to date, or the interaction with the original pirate increases and as a result his likelihood of being caught increases as well. With software aging, what a legitimate user pays for is the guarantee of continuing access to updates. This protection mechanism can therefore be seen as a dynamic form of protection.

Techniques such as tamper-proofing [3, 12] and obfuscation [7] do not prevent software piracy as such, but can be used to reinforce and protect other mechanisms. Tamper-proofing, which makes it hard to modify the program, makes it harder to remove embedded protection mechanisms. Obfuscation, which makes a program more difficult to analyze, can be used to hide the location of the protection mechanism.

A combined hardware-software approach is used by the Trusted Computing Platform Alliance (TCPA), the predecessor of the Trusted Computing Group [10]. Its intention is to define specifications for a hardware-assisted, OS-based, trusted system that would become an integral part of standard computing platforms. Microsoft also started a comparable initiative: Palladium. While it is hard to assert the security of these systems as they only exist on paper today, it seems unlikely that a universal adaptation of trusted-computing-enabled systems will happen in the near future.

2.2 Fundamental Flaws

None of the existing techniques for software piracy prevention provide adequate protection, since all of them have been broken.

Furthermore, any future software protection scheme will eventually be broken because it must depend on the operation of a finite state machine. Given enough time and effort, this finite state machine can be examined and ultimately modified at will by a malicious host running the software, because the owner of the software cannot impose restrictions on the hosts means to inspect the program. This contrasts with the model of a benign host and malicious code, such as in the case of worms and viruses. In these cases the host can prevent the code from modifying the host by restricting the actions that untrusted code is allowed to perform.

As a consequence, our problem in which the code is benign, and the host malicious, results in a more severe attack model, in which any static line of defense will sooner or later be broken. For example, only a few months were needed to create a key generator for the activation of Windows XP [11], which was one of the most complete systems, including on-line activation and links to the hardware.

The question thus becomes not whether or not a static protection will be broken, but when it will be broken, and what happens once it is broken. Unfortunately, once a copy is available that undoes the static copy protection or no longer carries the identification of the perpetrator, it can be distributed virtually unlimited and the software provider can no longer enforce its copyright.

In short, it is the static nature of existing defense mechanisms that makes them bound to fail.

Another reason why static software protection techniques are so susceptible to attacks is that, while the first copy is very expensive to produce, subsequent copies are inexpensive to reproduce and distribute. This is an important facilitating condition for software piracy. Hence its elimination will make software piracy less attractive.

Although it may at first sight seem counterproductive to increase the marginal cost of producing an additional, equally useful copy of a piece of software, we are convinced that there is no (other) silver bullet to prevent software piracy. This conviction is confirmed by the lack of protection provided by the current technologies. When marginal costs approach zero, they do so not only for the software producer, but also for the pirate. If software producers are to prevent piracy, the ease with which an equally useful copy can be created should be diminished.

As in the world of physical objects, where each object is unique and the cost to reproduce it is non-zero, we believe that the only way to achieve useful reproduction at non-zero costs is to make each legitimate copy unique. In fact, most of the technical mechanisms for piracy prevention already mimic this situation to some extent. This is most obvious for hardware-based mechanisms, as they combine the software with a unique, hard to duplicate, physical object. The software approaches also use a part that is unique for each installed copy, such as a license number, license file, activation code, decryption key or fingerprint. Software aging uses a key to identify legal owners of a copy and TCPA identifies the host computer and operating system.

A fundamental drawback of these schemes however is that these unique parts are not part of the original program. Instead they were added for the purpose of copyright protection. We believe that this is one of the reasons why they have proven to be relatively easily removed or circumvented.

3. SOFTWARE DISTRIBUTION MODEL

In the remainder of the paper, we will consider a simple software distribution model that consists of the following participants:

- **software providers**, who want to maximize their profits now and in the future;
- **legitimate users**, who are willing to pay for the software and want to use it without being impaired by the piracy prevention mechanism;
- **pirates**, who have the technical skills and the desire to circumvent the piracy prevention mechanism and want to minimize the risk of being caught;
- **illegitimate users**, who have no technical skills and want to enjoy the same privileges as legitimate users without proper compensation.

We will further assume that the number of pirates is limited.

In this paper, we do not consider the form of piracy where parts of an application are reused by a competing software provider. Instead we focus on piracy where it relates to the copying of an entire, possibly altered, application. As we will show the scheme presented in this paper impedes most forms of software piracy under realistic assumptions.

4. THE PIRACY PREVENTION SCHEME

The core of our protection scheme consists of two levels of diversification. At the first level each distributed copy is different. At the second level every installation of a specific copy is different. We will refer to a specific copy installed on a specific machine as an *instance*, and to an instance-specific update as a *tailored update*.

An instance must be activated through interaction with the software provider and contains links to the hardware to insure that an instance cannot simply be copied to another machine.

The software provider maintains a database that keeps track of the legitimate instances and their characteristics. The instances are crafted in such a way that they differ significantly, allowing the creation of updates fit for one instance and one instance only in such a way that a tailored update cannot easily be generalized for other instances.

When a user requests an update, he needs to identify the instance he wants to obtain the update for. The software provider then checks if the request is legitimate, looks up the characteristics of the instance and generates a tailored update.

5. BENEFITS OF THE SCHEME

This section discusses how the proposed scheme overcomes the limitations of existing piracy prevention techniques and how it impedes most forms of software piracy.

5.1 Improvements over Previous Approaches

The protection scheme presented in this paper overcomes the fundamental flaws common to almost all existing technical means for software piracy prevention. As a result of using tailored updates, the protection mechanism migrates from a static nature of defense to a dynamic nature. Since every instance is unique and since tailored updates are only provided for legitimate updates, the pirate needs to break through a new line of defense with every critical update. An update is said to be critical if it is necessary to ensure a continued secure functioning of the software, including data exchange with other (updated) copies. If such an update is not applied, illegitimate software cannot be kept sound and up to date.

The marginal cost of producing an additional, equally useful copy is increased as a result of the diversity and the links to the hardware. While we cannot prevent the duplication of digital data this prevents the copy from having practical value. The part of the program giving it its uniqueness, as common to many protection schemes, is in this scheme extended to the program as a whole, resulting in a very strong link, which has the advantage that it the cannot be separated as easily.

The scheme can furthermore be seen as a form of fingerprinting in which the program itself is the fingerprint that identifies a particular instance. As such it inherits the advantages of distributing unique versions: each copy can be identified which makes it possible to track unauthorized copies to their source. It provides a means for generating unique registration numbers which can be verified by the programs themselves at various points and by the manufacturer to insure that no corruptions have taken place in the distribution. The proposed scheme doesn't rely on legal measures however.

Of the existing techniques, software aging has the merit of being the only scheme with a dynamic nature of defense. We believe however that the used software distribution model and the made assumptions are unrealistic. Furthermore, the number of forms of software piracy against which it provides protection is too limited.

In the software distribution model used for software aging [15], the original pirate is held responsible for the subsequent updates. The authors assume that legitimate and illegitimate users do not cooperate and that illegitimate users interact only with the original pirate, both for the purpose of obtaining the original software and for obtaining updates. We believe that it is more realistic to see the pirate as an ubiquitous entity, meaning that the pirate providing the initial software and the pirate providing subsequent updates are not necessarily the same person or organization. Furthermore we believe that legitimate and illegitimate users do interact to exchange updates. The model we use, and in which our scheme protects against piracy, is hence much more realistic. This is confirmed by the Global Report on Software Piracy [17]: *The unauthorized copying of personal computer software for use in the office or at home or "sharing" of software among friends and co-workers is the most pervasive form of piracy encountered and is estimated to be responsible for more than half the total revenues lost by the industry.*

In our protection scheme, the relaxation of the constraints imposed by the model used for software aging is possible through the diversity of instances. As a result, the exchange of updates between legitimate users and illegitimate users poses no threat, as these are not fit for the illegitimate instances.

By contrast, the fact that an update works for all instances in the scheme of Jakobsson *et al.* introduces an additional complication for software aging: when two requests for an update for the same instance occur, they cannot distinguish between a repeated request as result of an interrupted transmission and a request by a clone. To overcome this problem they introduce conflict resolution. In our scheme, an update only works for one instance and other instances have no use for this update. Hence we can simply transmit a legitimate update every time it is requested.

5.2 Countered Forms of Piracy

Many forms of software piracy [13, 17] exist and it is important that a protection scheme impedes as many of them as possible. We therefore continue with a survey of these forms and show how the scheme makes them inviable.

5.2.1 Cracks and serials

Cracks and serials are forms of software piracy that consist of legally obtaining an evaluation version and subsequently entering a copied license code or applying a generic patch that undoes the copy protection.

This a widespread form of piracy. It is so popular because of the small amount of information that needs to be exchanged illegally. It is clear that it is easier to illegally distribute and obtain a license code or a patch than a complete program. The problem for defenders is to find a way to increase the difficulty of enabling an additional copy by reducing the uniformity of the distributed software. In our scheme, since all distributed evaluation versions are different, a patch for one instance will not necessarily work on another instance. Moreover the distributed evaluation versions can be designed to be incompatible with the license codes of other copies.

5.2.2 Softlifting and hard disk loading

The term softlifting refers to the act of piracy where one copy is legally obtained and installed on more computers than allowed.

Hard disk loading is the unauthorized installation of copies of software onto the hard disks of personal computers, often as an incentive for the end user to buy the hardware.

Both forms consist of installing software on more computers than allowed by the license. In these cases, we can expect exchange of updates between the legitimate user of a copy and the illegitimate users of the same copy. We assume that a software provider cannot easily become aware of these forms of piracy because of the limited size of the communities sharing a copy.

The scheme proposed in Section 4 involves interaction between the user and the software provider to activate an instance as a primary, static line of defense against this form of piracy. Even though this protection can eventually be broken, the diversity insures that only one copy of the software will be broken at a time.

When the first line of defense is broken, a second, dynamic line of defense is provided by the diversification at installation. As a result the characteristics of an illegitimate instance will not be considered legitimate and updates will not be provided. This way these instances cannot be kept sound and up to date. However, this diversification can be turned off as well in order to insure that the illegal instances are identical to the legal instance. To enable the illegal instances the links between the hardware and the software need to be broken. We note that it can be harmful to change the installed code to remove the links to the hardware as it might cause subsequent updates to fail. Changing the program might change the characteristics which determine the update. An alternative attack would be to emulate the hardware.

Both lines of defense need to be broken before illegitimate users can fully enjoy the software. As a result of the diversification an attack would ideally work on only one distributed copy. As each copy needs to be cracked separately and as we assume a limited number of pirates, we can conclude that, given the limited size of the sharing communities, this form of piracy will no longer have a significant impact on the revenues of the software provider.

5.2.3 Internet piracy and software counterfeiting

Internet piracy is the act of making unauthorized copies of copyrighted software available to others electronically. Software counterfeiting is the illegal duplication and distribution of copyrighted software in a form designed to make it appear legitimate.

Both forms consist of installing a piece of software on more computers than allowed. The scale is typically larger than of the forms discussed in the previous paragraph. We assume that there cannot be a continuing interaction between the pirate and the illegitimate users, as the exposure of the pirate and thus the risk of legal action against him would be too high.

We only consider the case in which both lines of defense are already broken. We limit ourselves to this case because of the following reasons: if the first line of defense is not broken and the user is charged per activation of an instance, this will bring no harm to the software provider. It will instead be a free distribution and advertising channel. If

the second line of defense is not broken illegitimate users will not be able to keep their software sound and up to date, since we insure that the update for a specific instance cannot easily be derived from an update targeted toward another instance.

In practice, the majority of versions pirated this way has the same origin. For example, most of the pirated versions of windows XP were tied to a few volume license product keys [16]. Given the large scale, the software provider probably can become aware of the piracy, for example, by searching the Internet for illegally distributed copies. Alternatively, as these illegitimate versions need to be kept sound and up to date and there cannot be a continuing interaction with the original pirate, many requests for updates will be made from many different locations for the same instance. This would also rise the suspicion of the software provider. If an instance is considered to be corrupted, the software provider will no longer provide updates for these instances, thereby impairing the illegitimate users.

5.2.4 Mischanneling

Mischanneling refers to the form of piracy where, e.g., an academic license is used for commercial purposes. To our knowledge this is the only form of piracy of an entire program against which our scheme does not provide protection. In fact, we are not aware of any technical protection against this form of piracy.

6. IMPACT OF INCREASED PROTECTION ON THE USER BASE AND PIRACY DISCRIMINATION

This section provides an economic model of the impact of protection on the user base. It is loosely based on the work by Conner and Rumelt [8] and Altinkemer and Guan [1]. We also discuss how the proposed scheme allows the software provider to act appropriately.

In the following discussion, we index each potential user by i. Let p be the price of the software, v_i the value of the software to the user and c_i the cost of obtaining and maintaining an illegitimate copy.

From an economical point of view, the sets of legitimate users (L), illegitimate users (I) and users who do without (D) would then be given by:

$$\begin{array}{lcll} L & = & \{i : p & \leq \quad min(v_i, c_i)\} \\ I & = & \{i : c_i & < \quad min(v_i, p)\} \\ D & = & \{i : v_i & < \quad min(c_i, p)\} \end{array}$$

We assume that an increase in the level of protection l does not influence legitimate users, meaning that the price p and value of the software v_i remains unchanged with changing levels of protection. The cost of obtaining and maintaining an illegitimate copy increases however with increasing levels of protection as the cost of breaking the protection mechanism increases. As a result, the number of potential pirates decreases, making it harder to obtain illegitimate software, cracks or updates. With set x_l denoting set x under protection level l, the following property holds: $n > m \Rightarrow L_m \subseteq L_n \wedge D_m \subseteq D_n$. Only users out of I_m might have moved to L_n or D_n when the protection level changed from m to n. Under these conditions increasing the level of protection can only increase the number of legitimate users, not decrease it.

This simple approach does not take network externalities into account however. The market of software is very susceptible to network externalities. A larger group of users that uses the same piece of software leads to an increase of the exchangeability of data and an increase in the production of complementary goods. This corresponds to Metcalfe's law which states that the usefulness of a network equals the square of the number of users.

Let $U = L \cup I$ be the total number of users, v_i^0 the value of the software to user i in the absence of other users and $f_i(U)$ the increase in value when there are U software users. Each f_i is positive and increasing in U. The sets of users now become:

$$\begin{array}{lcll} L & = & \{i : p & \leq \quad min(v_i^0 + f_i(U), c_i)\} \\ I & = & \{i : c_i & < \quad min(v_i^0 + f_i(U), p)\} \\ D & = & \{i : v_i^0 + f_i(U) & < \quad min(c_i, p)\} \end{array}$$

Under these conditions, the set of legitimate users L can decrease with increasing levels of protection l. Some of the otherwise illegitimate users might, e.g., choose to do without, resulting in a decrease of the user base U and a decrease of the value added because of network externalities $f_i(U)$, as a result of which p might become higher than $min(v_i^0 + f_i(U), c_i)$.

The impact of the higher protection level on L is determined by the strength of network externalities and the protection elasticity of the user base U. Strong protection mechanisms are thus useful only when network externalities are weak or when the protection elasticity of the user base is low. This is, e.g., the case for niche products with a high consumer surplus and when the product is not easily substitutable. In other cases it is desirable to tolerate some level of piracy, thereby minimizing the impact on the user base.

Another incentive to tolerate some level of piracy are switching costs. Piracy can help to lock-in consumers in an earlier phase and lead to higher profits in a later phase. In our model, software providers want to maximize their profits now, but also in the future. Users that have illegitimately used a program for a while and want to turn to a legal version can be expected to stick to the software they are used to because switching implicates additional costs. These include learning costs, transaction costs (because the installation and implementation of a new software system is not a trivial task) and artificial costs: an update can for example be cheaper than the full version. In some cases the pirate does not have the financial capabilities of obtaining the software legally, but he can be expected to do so in the future.

The proposed scheme enables a fine-grained control over the distributed copies and allows a software provider to tolerate an arbitrary level of piracy. This is made possible through the activation and the tailored updates. A software provider can choose which instances are activated and for which instances updates are provided. Furthermore he knows the origin of the instance for which an activation or update is requested because of the diversity.

A software provider could this way tolerate, e.g., piracy of a Chinese version. This way the user base can grow and the consumers grow accustomed to the software. When they will have to choose which software to buy in the future, they will be more likely to buy the software they are familiar with. The distribution of a Chinese-language version or, for example, regional codes as in DVDs can be used as a

simple form of diversification to assure that the software is not suited for other markets.

Other cases would require a more fine-grained level of diversification and would better exploit the full strength of the proposed scheme. As an example, two installations of a copy for private use could be allowed to enable a user to use the same copy on a desktop and on a laptop. On the other hand a copy for commercial use can be limited to one installation or exactly as many as agreed upon in the license.

The relevance of piracy discrimination in practice is best illustrated by the following citation: *Although about three million computers get sold every year in China, people don't pay for the software. Someday they will, though. And as long as they're going to steal it, we want them to steal ours. They'll get sort of addicted, and then we'll somehow figure how to collect sometime in the next decade.* (Public dialog between Bill Gates, founder and CEO of Microsoft Corporation, and Warren Buffet, chairman of Berkshire Hathaway Inc., 1998).

7. PRACTICAL CONSIDERATIONS

This section discusses a number of practical problems and possible solutions related to the proposed scheme. First, the dependency on updates is discussed, then we explore the possibilities to diversify executables and to insure that an update only works for one instance.

7.1 Reliance on Updates

The dynamic nature of the scheme is only possible through updates. In the presence of the Internet, they can be distributed easily and the updating process can be done automatically by the program. Nowadays software updates are used for the following purposes:

- to fix bugs;
- to add security patches;
- to support new hardware and new file formats;
- to keep a program compatible with other programs;
- to add new functionality.

The first four categories are considered to be critical. Updates increase the cost c_i for illegitimate users. If they want to enjoy the same privileges as legitimate users, they need to find an update suited for their instance with every update.

We believe that, for most types of commercial software, a frequency of one update every couple of months will inflict enough damage on illegitimate users to persuade them to become legitimate.

However, if the frequency of updates normally required is too low, we can artificially increase the necessity for updates. Obviously, introducing bugs or security flaws to make updates necessary is not an option and introducing new hardware is too expensive.

For document-producing programs we can apply software aging [15]. This way instances that are not kept up to date are unable to read the output of more recently updated instances. As such we can consider the output to be of a different file format. This decreases the value of a pirated instance as it cannot be kept compatible with legitimate instances.

Legitimate users could be favored by providing them with access to a collection of add-ons or extra features. As a result, the value of the program for legitimate users will be higher than for illegitimate users as these add-ons or features will not work for illegitimate instances.

Another approach is to move from the model in which a user pays for the continued use of the software to a model where a user rents the software for a limited amount of time. The software could disable itself, unless it is updated to enable the software for an additional time-period. This is however a static form of defense, making it possible for a pirate to remove the disabling code. While the diversity insures that each copy needs to be cracked separately, a full, cracked version could still be distributed, defeating this defense mechanism. Therefore this approach must be combined with other types of updates, which should be tailored not to work with instances where the time limitation is removed.

7.2 Diverse Instances and Tailored Updates

The core of the protection scheme requires the instances to differ in such a way that updates can be tailored to work for one instance and one instance only. In this section, we will discuss a possible approach to achieve this.

A typical program consists of a large number of files, containing code or data. A distributed update contains the necessary information to convert the program to a newer version. Obviously, no information needs to be included regarding unmodified files as this would only make the update larger. Also, new files need to be included entirely.

There are two possibilities for changed files: in a full-file update the entire file is included, whereas with an incremental update only the changes over the installed version are specified. The former has the advantage that when different users have different installed versions, e.g., because some users have updated their software more regularly than others, the same update can still be provided to all the users. An incremental update has the advantage that it is smaller, but different updates are necessary for different installed versions. This problem is sometimes solved by providing updates incremental to the original version of the files, which needs to be provided by the user by inserting the installation disk. We will now discuss how our scheme can be put into practice in both of these cases.

7.2.1 Full-file updates

When using full-file updates it is useless to apply the diversification within a single file as the full file will be included in the update. We thus need to diversify the interfaces between the different files to insure that a file in a tailored update cannot function correctly with an illegitimate instance.

For data files, the content can be encrypted and decrypted using an instance-specific key. The same technique can be applied for the interface between code files, in which arguments passed to functions and the values returned can be encrypted. The keys could be hidden by techniques for white-box cryptography [4] to make it more difficult to circumvent this protection.

Alternatively, all data, arguments or return values can be transformed to an instance-specific domain, provided that the computations are also transformed to this domain.

instance	machine code	assembly	
1	29 c2	sub	%eax,%edx
	83 c2 ff	add	$-1,%edx
	19 c9	sbb	%ecx,%ecx
	83 c1 01	add	$1,%ecx
2	29 c9	sub	%ecx,%ecx
	29 c2	sub	%eax,%edx
	83 fa 01	cmp	$1,%edx
	83 d1 00	adc	$0,%ecx

Figure 1: Two corresponding code fragments for two instances of Intel machines/software.

instance	machine code	assembly		update
1	29 c2	sub	%eax,%edx	00 00
	83 c2 ff	add	$-1,%edx	00 00 00
	19 c9	sbb	%ecx,%ecx	00 00
	c1 e9 1f	shr	$31,%ecx	42 28 1e
2	29 c9	sub	%ecx,%ecx	00 00
	29 c2	sub	%eax,%edx	00 00
	83 c2 ff	add	$-1,%edx	00 38 f1
	83 d1 00	adc	$0,%ecx	00 00 00

Figure 2: The two corresponding code fragments after their update, and the bitwise difference (update) with the original code.

7.2.2 Incremental updates

The main incentive for full-file updates is that it facilitates the distribution as the same update can be used by each user. Clearly this is no longer a valid argument in our protection scheme since it requires the updates to be instance-specific.

Recent research by the author [2] shows that the number of equivalent code sequences is exponential in the number of instructions and is huge for any code sequence of considerable length. Therefore it is possible to generate a large number of binary differing code files that are equivalent, i.e. they perform the same operation. As a result every instance can have code files that differ significantly on a binary level. When an update only specifies which bits in the existing file need to be flipped to migrate to the new version, it is clear that this will not work for other instances.

An example might help to clarify things. Suppose we have two users, user 1 and user 2, and that the original program executed the following operation: `%ecx = (%eax == %edx))` (`%eax`,`%ecx` and `%edx` are registers). Then the code for each user could be as depicted in Figure 1.

If this code, for some reason, needs to be updated to (`%ecx = (%eax != %edx))`, then we could migrate the code to the code sequences depicted in Figure 2.

The rightmost column indicates which bits of the original version need to be flipped for each user to migrate to the new version. Clearly, these updates would not work for the wrong instance.

However, it would be relatively easy to generalize these incremental updates to instance-independent updates: a pirate could simply monitor which files have changed by applying the update and include these files entirely in a full-file update.

Fortunately, this form of diversification can be combined with the ones discussed for full-file updates. If we, e.g., encode function arguments, they will be wrongfully decoded if the file containing that function is crafted for the wrong instance, thereby impairing this form of generalization. In our running example, one of the instances could take the negate of one of the arguments before calling the function and the function could take the negate of the passed argument before executing the operation. This way, the correct functioning is assured for that instance, but if the file was separated from that instance and moved to another instance without this conversion, it would clearly no longer work correctly.

As incremental updates allow for an additional technique to make diverse code files and as this technique can be combined with the techniques discussed for full-file updates, it is the preferred updating method for our scheme.

7.3 Identification of Legitimate Instances

When an update is requested, the instance for which it has to be tailored needs to be identified. This can be done through a simple serial number of some sort which is assigned to an instance at activation. The database would then keep track of the serial numbers that identify legal instances. The database also contains the necessary information to reconstruct the characteristics of that instance. If the instance is considered to be legitimate, the software provider tailors an update that migrates that instance to the new version of the software.

The software provider does not need to worry about illegitimate users that request an update for a legitimate instance as it will not work for their instances. However, he will keep a log of the different requests to track illegitimate copies as described in Section 5.2.3. When many requests for the same instance have different origins, this will rise the suspicion of the software provider and he will classify this instance as illegitimate.

To assure that the original legitimate owner is not damaged as a result (he might not have been aware of the piracy), he should contact him and provide him with an update that migrates his instance to a new instance, with a new serial number.

Clearly, this update should not be provided to the illegitimate users. However, if illegitimate users would choose to become legitimate through correct compensation in order to obtain full access to the updates in the future, a similar process can be used to migrate their illegitimate instance to a new legitimate instance.

We also note that a legitimate user could accidentally request an update for the wrong instance. As a result his software might no longer function correctly. Again, we do not want to damage the legitimate user that has, e.g., made a typographical error when submitting the serial number. Therefore, each update should check whether it is applied to the expected instance, e.g., by comparing a checksum over the files that will be changed to the expected checksum.

The integrity of the database needs to be assured for the correct operation of the scheme. This can be achieved by the usual means to ensure data integrity for data centers, including an off-site mirror data center.

8. COSTS OF THE PROPOSED SCHEME

The cost of the development and maintenance of the infrastructure required by our scheme is considerable and

should be justified by the additional revenues that can be expected from the number of illegitimate users that will turn legitimate in the presence of our software piracy prevention scheme. Let d denote a protection level with diversity, w a protection level without diversity and C_l the total cost of the software at protection level l. Clearly, the application of the scheme is only worthwhile if $p|L_d \setminus L_w| \geq C_d - C_w$. As already noted in Section 6, $|L_d \setminus L_w|$ is determined by the strength of network externalities and the protection elasticity of the user base and as such depends on the market conditions under which the software distributor operates. These conditions will also determine the optimal level of piracy discrimination.

The key element of the proposed scheme, diversity, signifies an additional cost $C_d - C_w$ for the software provider. While distributing identical copies provides an enormous economic leverage, this is not the case for non-identical copies.

The creation of non-identical copies and tailored updates is considerably more complicated than having a uniform installed base. The fixed costs include the development of the software for the diversification of programs and the creation of tailored updates, while the marginal costs include the computing and distribution costs per instance.

The computing costs consist of the additional cost per instance for the purchase and maintenance of hardware needed for the creation of non-identical copies and tailored updates, as well as the costs associated to the database that keeps track of legitimate instances and contains the necessary information to create tailored updates. To reduce the computing costs, the process of diversification and the creation of tailored updates should be fully automated. This could be achieved by adding an additional pass in the tool chain during compilation or at link time [9]. This way it will not interfere with the source code and it could be guaranteed to be semantic-preserving (except for the case where we want to prevent that a license code is valid for multiple instances).

The pressing of CDs is no longer economically viable when all distributed copies are unique. In this case it is more advantageous to burn them, but the cost per disk will be higher. While it might be acceptable to distribute the initial software this way, physical distribution of non-identical updates on a regular basis would significantly increase the cost per instance. Fortunately, in many cases, the updates could be digitally distributed, given the widespread use of the Internet. As a result, the distribution costs for the updates can be minimized.

Furthermore the debugging process will be complicated as error reports will be instance-dependent. This could be circumvented by storing a mapping from each specific instance to the original software, and by applying the reverse mapping to bug-reports.

While the effort will likely be worthwhile for larger software providers as the costs can be distributed over a larger number of repentant illegitimate users, the effort might be too big for smaller software providers. In the latter case a specialized company could take care of the generation and distribution of copies and updates. As the process can take place at link time, this requires no disclosure of source code.

9. CONCLUSION

This paper presented a new scheme for the protection of software against piracy. Its strength is based on diversity: each installed copy is unique. Updates are tailored to fit one instance and one instance only. This way the ease with which a useful additional copy can be created is diminished. Furthermore, as illegitimate instances cannot be kept sound and up to date unless a new line of defense is broken with every critical update, this results in a dynamic nature of defense.

We pointed out the improvements over previous approaches and argued that it makes most forms of software piracy more difficult in a realistic model under realistic assumptions.

Acknowledgments

The authors would like to thank the Flemish Institute for the Promotion of Scientific-Technological Research in the Industry (IWT), the Fund for Scientific Research - Belgium - Flanders (FWO) and Ghent University for their financial support.

10. REFERENCES

[1] K. Altinkemer and J. Guan. Analyzing protection strategies for online software distribution. *Journal of Electronic Commerce Research*, 4(1):34–48, 2003.

[2] B. Anckaert, B. De Sutter, and K. De Bosschere. Steganography for executables. Technical Report R104.003, ELIS, Ghent University, 2004.

[3] H. Chang and M. Atallah. Protecting software code by guards. *Security and Privacy in Digital Rights Management, LNCS*, 2320:160–175, 2002.

[4] S. Chow, P. Eisen, H. Johnson, and P. Van Oorschot. White-box cryptography and an AES implementation. *Selected Areas in Cryptography, LNCS*, 2595:250–270, 2003.

[5] F. Cohen. Operating system evolution through program evolution. *Computers and Security*, 12(6):565–584, 1993.

[6] C. Collberg and C. Thomborson. Software watermarking: Models and dynamic embeddings. In *Principles of Programming Languages*, pages 311–324, 1999.

[7] C. Collberg, C. Thomborson, and D. Low. A taxonomy of obfuscating transformations. Technical Report 148, Dept. of Computer Science, Univ. of Auckland, 1997.

[8] K. Conner and R. Rumelt. Software piracy: an analysis of protection strategies. *Management Science*, 37(2):125–139, 1991.

[9] B. De Bus, D. Kästner, D. Chanet, L. Van Put, and B. De Sutter. Post-pass compaction techniques. *Communications of the ACM*, 46:41–46, 2003.

[10] E. Felten. Understanding trusted computing: will its benefits outweigh its drawbacks. *IEEE Security and Privacy*, 1(03):60–62, 2003.

[11] Heise Online. *Crack und Keymaker aktivieren Windows XP*, February 2002. http://heise.de/newsticker/meldung/24775.

[12] B. Horne, L. Matheson, C. Sheehan, and R. Tarjan. Dynamic self-checking techniques for improved tamper resistance. *Security and Privacy in Digital Rights Management, LNCS*, 2320:141–159, 2002.

[13] International Planning and Research Corporation. *Software Management Guide*, 2003.

[14] International Planning and Research Corporation. *First Annual BSA and IDC Global Software Piracy Study*, July 2004.

[15] M. Jakobsson and M. Reiter. Discouraging software piracy using software aging. *Security and Privacy in Digital Rights Management, LNCS*, 2320:1–12, 2002.

[16] Microsoft. *Microsoft Knowledge Base Article - 326904*.

[17] Software and Information Industry Association. *Report on global software piracy*, 2000.

[18] P. van Oorschot. Revisiting software protection. *Information Security, LNCS*, 2851:1–13, 2003.

[19] R. Venkatesan, V. Vazirani, and S. Sinha. A graph theoretic approach to software watermarking. *Information Hiding, LNCS*, 2137:157–168, 2001.