PROTECTING BRANDS OVERSEAS - ECONOMIC GLOBALIZATION AND TRADEMARKS

NEW PATENT LAWSUITS - RFCEXPRESS.COM

THE INTERPLAY BETWEEN TRADE SECRETS AND THE FIRST INVENTOR DEFENSE

Intellectual Property Today™

www.iptoday.com

I. P. Today, 381 W. Northwest Hwy., Palatine, IL 60067

A Publication of Omega Communications

August, 2007 $9.00 Volume 14, No. 8

An Introduction to Software Protection Concepts

By Kelce Wilson, PhD, MBA, JD

An Introduction to Software Protection Concepts

BY KELCE WILSON, PhD, MBA, JD

Kelce Wilson, PhD EE, MBA, is a registered patent attorney and provides software security services to the US Department of Defense. He can be reached at kwilson@softwareIPattorney.com.

The first step in creating an effective defense is learning about a threat, because a poor comprehension of a threat leads to inefficient and incomplete protection efforts. Therefore, an understanding of various types of software attacks, along with their accompanying goals, is critical to forming a solid software protection plan. There are different types of threats to intellectual property (IP) contained in software, which include license violations, reverse engineering and tampering.

Defensive protection methods that may work against license violations may be entirely useless to combat reverse engineering or tampering. Similarly, protections that may work against reverse engineering or tampering may be ineffective to enforce adherence to a license agreement. One reason for this is that these three sample classes of attacks may be carried out in furtherance of significantly different goals, and may involve different attack methods. "Attack" is the correct word, given the type of malicious activity that is often directed against software.

For an explanation of these different classes of attacks, consider the following hypothetical scenario: One of your clients has invested heavily in developing the IP in its flagship software product, and although sales are doing well, the development costs have not yet been fully paid off. The client has since discovered that the software contained a previously unrecognized inherent capability, and minimal changes in the program can drastically enhance the market value of the software. Your client's marketing department is ecstatic about selling an upgraded version, and its engineering department is relieved that few changes are required. Word has spread among the user community about the upcoming upgrade and enhanced capability. Good news? Perhaps not.

Unknown to the client, three copies of the software were stolen from overseas customers. One copy is about to be posted on a website in a country that is notorious for lax enforcement of IP rights. Due to national pride or concerns over international economic competitiveness, the government of that country makes only token efforts to enforce IP rights belonging to entities outside that country's borders. The website will allow unlimited, free, anonymous download of the software. Another copy of the stolen software was reverse engineered in a secretive overseas facility, and your client's IP is being leveraged to develop a product for a foreign competitor. Reverse engineering involves learning the secrets of how software operates. Because the competitor's engineers have studied your client's product, they have been able to develop a remarkably similar computer program, with some differences, but in a rather short period of time and with a relatively low cost. The competitor has thus obtained an economic advantage, although it will publicly proclaim efficient, independent development.

The third stolen copy fell into the possession of someone with skill in software tampering and who is also aware of the planned upgrade. Tampering is changing a copy of a computer file so that a program that depends on that file performs differently. Tampering often requires some degree of reverse engineering to find the data file or the part of the program that will be changed. A computer program patch, similar to those used for bug-fixes and software maintenance, has been developed from the third stolen copy, and is about to be posted on a website in another country that is notorious for lax enforcement of IP rights. The patch will soon be available for free, unlimited, anonymous download and will require minimal skill to use. With a few mouse clicks, anyone with a copy of your client's first software version will have most of the enhanced functionality of the planned upgrade version. Anyone can then visit the first website, download a copy of the first version, and then visit the second website to improve the pirated copy.

Approximately a week before the internet postings appear and the competitor releases its product, your client visits your office to discuss IP protection strategy for the upgraded version. Your client is also about to invest heavily in developing an unrelated software product. What plans do you recommend for protection of each product? How well will those plans protect your client's interests in view of the upcoming events?

Protection of the IP contained in software often requires the integration of two seemingly disparate bodies of knowledge: computer hacking and IP law. Expertise in IP law alone is not sufficient to plan for effective protection efforts on behalf of your client, because even with the broadest issued patent claims, airtight trade secret agreements, and thorough copyright registrations, your client may watch helplessly as off-shore hackers, pirates and websites destroy the economic value of a significant investment with impunity. Further, your client may even be forfeiting sales and licensing opportunities out of fear of IP theft, even though the risks can be managed to a reasonable level with the proper methods.

In contrast with the dire scenario given above, consider the following: Another client does not have a consumer-ready product, but has developed software modules that could be marketable to other software developers for inclusion in their products. However, the other developers insist on receiving source code in their projects and that client is uncertain about the other developer's compliance with the source code license agreements. Should your client forego the potential sales? Perhaps not. What if the software contains trade secrets? Can source code be sold to potentially untrustworthy parties while preserving trade secret status? The answer might be surprising.

The traditional modes of IP protection, patents, trademarks, copyrights, trade secrets and license agreements leave some threats unaddressed, and even a complete suite of legal protections does not enable full realization of marketing opportunities with reduced risks. For example, selling humanly-readable source code to untrustworthy parties will likely destroy trade secret status. Even the Digital Millennium Copyright Act (DMCA) allows exemptions for tampering and reverse engineering, so long as the purpose is for ensuring interoperability with independently-developed programs or compatibility with an operating system (OS) upgrade. *Digital Millennium Copyright Act*, 17 U.S.C. 1201(f). Note that this exemption is available even for competitors. However, just because a federal statute preserves a right for your client's competitors and customers to reverse engineer and tamper does not mean your client must cooperate by allowing such endeavors to be practical. The right protection tools, properly applied, can provide a series of revenue opportunities.

This article describes the roles of software attacker and defender, goals and tactics of attacks, various defensive options along with their applications and limitations, a basic protection paradigm, and some resources. The protection strategy proposed will be a combination of legal and technical protections. While legal and technical protections are in significantly different disciplines, IP law versus computer science, there is an area of overlap in which the two affect each other. A simultaneous understanding of both allows for a synergistic effect. For example, as will be seen later, technical protections may be used to enhance a software developer's ability to successfully identify piracy and cracking attempts, as well as identify the perpetrator, which can assist in developing evidence for copyright prosecutions and license violation disputes. In some situations, technical protections can even permit a software developer to ensure that a theft and reuse of IP leaves tell-tale signs that can assist with a patent infringement suit. However, a clear understanding of roles and attacks is needed first.

HACKERS, CRACKERS, PIRATES AND SCRIPT KIDDIES

Software attackers are often referred to as "hackers," although this label is overly broad and often used incorrectly. The particular label used to describe a software attacker should generally reflect a goal or the tactics used, in order to allow meaningful discussion. The term "pirate" may be used to describe someone who merely copies, or allows to be copied, a piece of software in violation of a license agreement. In this usage, the term "pirate" does not imply technical ability, although a successful act of piracy may require modification to an executable binary in order to ensure a copied program executes as desired. The term "cracker" may be used to describe someone with the ability to meaningfully examine or modify an executable binary or other critical computer file in order to accomplish a particular goal. For example, a cracker may modify a trial version of software so that it has full functionality even after the trial expiration date. Note that in this usage, cracking includes reverse engineering, even if no modifications to the binary are promulgated. The term "script kiddy," although diminutive, is a convenient term for describing those who use patches produced by crackers in order to enable acts of piracy or to enhance software functionality in violation of a license agreement.

The previously presented scenario of the three threats to a hypothetical client's software included all of the three roles just described. The first website is a piracy site, run and visited by software pirates. The competitor used software crackers. The computer program patch was developed by another cracker and is posted on a cracking site for download and use by script kiddies. Anyone visiting both websites to obtain the software and then to modify it with the patch is both a pirate and a script kiddy.

Software defenders are typically the developers, who are also often the owners. However, this is not always the case; and further, there is an industry developing in which technical defensive measures that are intended to frustrate and impede crackers are inserted by experienced service providers under contract. As an editorial note here, I will emphasize that homegrown technical security measures are rarely effective, despite glowing promises made by your client's engineering department. I have seen multiple situations in which the most self-confident software developers turn red-faced and start offering lame excuses when their "hacker-proof" safeguards are bypassed by "ethical hackers," i.e. crackers under contract to perform testing, as if the protections were trivial or non-existent. In one incident, I was challenged with a piece of software that had been protected by programming experts with years of experience. I was able to crack it in less than seven minutes after installing it on my computer. The developer's representative called me, so I took the CD out of the mailing envelope and, with him on the phone, completely bypassed all the protections before he finished telling me about how solidly the program was protected.

As part of a software security education campaign, I have given demonstrations in which I modify trial software from a widely-known developer to behave like a paid-up version, in less time than it would take to visit the developer's website and pay the registration fee. The demonstration is so quick, that I have been asked to repeat it because someone's attention was momentarily diverted and I was finished before they thought I was going to start. Cracking some software is not only cheaper, but can even be quicker than taking the honest route.

NETWORK ATTACKS VS. SOFTWARE ATTACKS

Before examining the attacks in more detail, it is useful to draw a distinction between computer network attacks and software attacks. While there is a fundamental difference, the required attacker skill sets overlap significantly. In a computer network attack, the attacker seeks to obtain access to a computer itself, often but not always, located remotely from the attacker. Success by the attacker is achieved when the desired level of access is obtained, allowing files to be added, deleted, moved, copied, executed, or modified. Once the attacker has gained access, subsequent actions are generally fairly simple. Defensive measures may be taken after an attack, such as removing a targeted computer from a network. Preventative measures may also be updated to keep pace with evolving network attack methods. Even as difficult and frustrating as it may seem for information technology (IT) managers to deal with daily penetration attempts and virus updates, the defensive game against computer network attacks is considerably easier to win than the defensive game against software attacks.

This is because, in software attacks, the attacker often controls the environment in which the software resides. That is, the software resides on the attacker's computer and an experienced attacker can modify and restore files and data stored on the hard drive and even in firmware. Thus, the attacker can often try multiple methods, develop tools, consult with others attackers, and reload the software at the attacker's leisure, until succeeding in the goal. Even though attack methods may evolve over time, the software usually cannot receive further updates from the defender to compensate.

LICENSE VIOLATIONS, REVERSE ENGINEERING, AND TAMPERING

Of the three types of threats mentioned earlier, license violations, reverse engineering and tampering, the most easily understood is license violations. Often this includes piracy, in which unauthorized copies of the software are disseminated and used. Other license violations include using software past an agreed-upon date or for purposes prohibited by terms stated in an agreement. License violations are typically motivated by a desire to gain an economic benefit.

Tampering has three primary motivations: removal of anti-piracy technical protections to facilitate a license violation, sabotage, or to enhance operation. Examples of enhanced operation include:

disabling "nag screens" that pester a user into paying for a program

enhancing a player's abilities in multiplayer online games

removing limitations placed on a trial copy of software, such as the inability to save a file in a common format.

Intercepting and operating on data between two software modules may also be defined as tampering, even if the software modules themselves remain intact.

Reverse engineering has two primary motivations: facilitating tampering and also industrial espionage in order to reduce the development costs of a competing product. Examples of license violation cases abound, but some interesting cases that describe tampering and reverse engineering include *Krause v. Titleserv Inc.*, 402 F.3d 119, (2d Cir. 2005), *Storage Technology Corp. v. Custom Hardware Engineering & Consulting Inc.*, 421 F.3d 1307, (Fed.Cir. 2005), and *Leapfrog Enterprises, Inc. v. Fisher-Price Inc., Mattel Inc.*, No.Civ.A. 03-027-GMS, 2006 WL 891001 (D.Del. Mar. 30, 2006). All three cases involve reverse engineering, but *Kraus* and *Storage Technology* further include tampering. Other issues are involved in these cases, but it is interesting to note that, although the evidence clearly showed cracking attacks, the plaintiff did not prevail in any of these three cases.

CODE LIFTING

Another form of attack, which is a hybrid of tampering and reverse engineering, is code lifting. In a code lifting attack, a software cracker learns enough about the software to identify the portion containing important IP, but rather than learning the IP well enough to independently reproduce it, the identified section of the software is copied and used in a purportedly different program. The competitor's program may actually be little more than window dressing around the copied software so that it appears to be different. With code lifting, one software developer can leverage another developer's IP, even without understanding it. *Lexmark Intern. Inc. v. Static Control Components Inc.*, 387 F.3d 522, (6th Cir. 2004) provides an example of code lifting, and the plaintiffs allegations in *Davidson & Associates v. Jung, 422 F.3d 630 (8th Cir. 2005)* imply code lifting.

The remaining portion of this article will describe protection measures that could have been used by the plaintiffs in *Krause*, *Storage Technology*, *Leapfrog*, *Lexmark* and *Davidson* that likely would have achieved their goals without the need for litigation.

LINES OF DEFENSE

Software protection has two primary lines of defense: the attacker's willingness and the attacker's ability. To successfully crack software, an attacker must have both. Thus, protection mechanisms may be divided into two categories: deterrence-based and technology-based. A technology-based protection, i.e. a technical protection, is one that introduces costs to an attacker in terms of time, money and/or skill. Examples include encrypting portions of a program and moving critical data into a USB dongle, although more complex and generally more effective methods will be described later.

Deterrent protections include the legal mechanisms of patents, copyrights, trademarks, trade secrets and contracts.

However, marketing pressures and social pressures may also be used as deterrents. For example, the movie industry has an educational campaign included in many movie previews that is aimed at convincing consumers that copying movies and songs is equivalent to shoplifting.

In general, a deterrence protection requires a reaction, whereas a technical protection often does not. However, a technical protection can additionally be used to assist a deterrent, such as by using a "phone home" feature, in which an attempt to crack or pirate a piece of software results in the attacker's computer sending an email to a monitoring organization. Further, digital media files, such as mp3s, mpegs, and even executable programs may be watermarked or fingerprinted to assist in collecting evidence for a potential lawsuit.

Watermarking is the insertion of information into a digital media file that allows the file to be uniquely associated with the author. For example, placing secret information such as a digital signature in the file will enable someone to assert authorship at a later time, because it is unlikely that someone else would have placed that same information in the file. Fingerprinting is the watermarking of files with different information, based on where the file is sent. For example, a software developer has three orders for a program, encrypts each customer's name with public key cryptography using the developer's secret encryption key, and then embeds the encrypted name of a customer in the copy provided to that customer. Thus, each customer gets a unique version of the software. Later, if a pirated copy of the program is discovered, the developer can likely determine which customer had been sharing the program. Further, in an authorship dispute, the developer can assert that because the customer's name was encrypted with the developer's secret key, the developer is the author of the original program and it was not developed independently.

Effective watermarking and fingerprinting can assist litigation efforts, such as by providing evidence of piracy or code-lifting. However, both techniques involve complicated technical issues, which are beyond the scope of this article. For more information, see Christian Collberg and Clark Thoborson, *Software Watermarking: Models and Dynamic Embeddings*, Symposium on Principles of Programming Languages (1999).

Since an attacker needs to have both the willingness and the ability, there are often opportunities to use both deterrent and technical protections simultaneously. Only one needs to work, but often one type alone may be insufficient. So redundancy, when practical, is an effective defensive tactic. In many circumstances, deterrent protections are likely the quickest to fail.

FAILURE OF DETERRENCE

Unfortunately for deterrent protections, there are at least four components that could fail individually, reducing the effectiveness of the protection. These are (1) there must be a threat that offsets the attacker's perceived value of a successful attack; (2) the entity making the threat must be able to identify that a software attack has occurred; (3) the identity of the attacker must be reliably discernable; and (4) the threat must be credible. A myriad of problems could arise to derail a deterrent protection, such as a patent being invalidated or trade secret status being lost; the attacker may be able to remain anonymous or be able to implicate an innocent person; the attacker may be judgment-proof or outside of a cooperative jurisdiction; and it may become socially unacceptable or too expensive to carry out the threat. For example, suing teenage girls for swapping mp3 files may be perceived negatively by some aspects of society. Additionally, many software developers may not be able to afford litigation, even after discovering IP theft.

A further problem with deterrent protections is that the software may have such significant value that the attack is worth attempting. A rational software attacker will likely use an analysis similar to the one outlined here: The value of the attack is the sum of the economic utility of a successful attack, such as the purchase price of the software or cost of independent development, and notoriety or other ego factors. The cost of the deterrent is the cost of the threat discounted by the probability of being caught and the probability of the threat being carried out. If the discounted cost of the threat is below the perceived value of the attack, a rational attacker will proceed. An irrational attacker, unfortunately, cannot be deterred, and even more unfortunately, many irrational attackers may be quite skilled.

Thus, attacks should almost always be expected. Deterrence can only be expected to work on a percentage of people, meaning that a technical protection mechanism should be considered if the expected percentage is not high enough. Because software security is a specialized field, adding technical protections may delay release dates by affecting testing and maintenance, as well as add the costs of consulting and/or purchasing a protection insertion tool. Economic models suitable for setting a defender's technical protection budget are rather complicated. There are multiple proprietary budgeting tools, the results of which can vary significantly, but there are no thorough models currently available for public discussion.

Detracting issues with deterrent protections include the incompatibility of various methods, such as the incompatibility between the enabling disclosure required for a patent and the secrecy required for maintaining a trade secret. However, technical protections can work with all types of deterrent protections identified above, including some that can work with patents equally well as with trade secrets. Further, some technical protections can bolster a software developer's claim to trade secret status, and even extend the circumstances in which trade secret status can be claimed.

For example, in *QSR Soft Inc. v. Restaurant Technology Inc.*, No. 06C2734, 2006 WL 3196928 (N.D.Ill. Nov. 02, 2006), the plaintiff placed information on a website and yet retained trade secret status. The information was protected by a password, which the court apparently found to be a reasonable safeguard for protecting against disclosure to the public. Although it is only speculation at this point, in the future it may be possible for a software developer to openly sell source code in unencrypted text files, protected by obfuscation, which is described later, while preserving trade secret status of the IP contained in the software. If such a result ever does become settled law, it opens the door to selling source code modules for inclusion in other developers' products, but without yielding some important IP rights that would have previously been lost.

LICENSE ENFORCEMENT

The technical protections are best understood when described in relationship to the classes of attacks. Node locking, dongles, media checks, and remote validation are license enforcement techniques. Node locking adapts the software or a data file needed by the software to a specific computer, so that if it is copied, minor differences between the two computers will prevent the software from running. Dongles are physical devices, often connected to a USB port, that contain critical data or a portion of the program, so that the program should not run without the dongle connected to the computer. The idea behind a dongle is that, although copying software may be easy, the hardware device cannot be duplicated easily. However, dongle emulators and patches that remove many programs' dependence on dongles often appear on the internet. Media checks are intended to only run the software when the original CD provided in the shrinkwrap package is in the computer. However, there are many tutorials on the internet for tampering with software to bypass these checks. Remote validation requires a computer to connect to the internet and visit a website each time a user wants to run the software. This isn't always transparent to the users, and can often lead to unhappy customers when a network goes down.

PREVENTING TAMPERING

Technical protections against tampering include integrity checking, anti-debug measures, and to some extent, obfuscation. Obfuscation will be discussed in more detail in relation to anti-reverse engineering, since that is the primary benefit. Integrity checking is when the software examines itself for changes. For example, a piece of software may look at its own binary code to determine whether a media check is intact, or whether a cracker has modified the program to omit the media check. If the integrity check fails, the software has multiple options. It may simply stop, it may repair itself with a self-healing routine, or it may "phone home" by sending an email to a monitoring organization to alert someone to the crack. Or perhaps, if the program outputs data, such as digital images, the integrity check may put a hidden code such as a watermark or fingerprint in the output to indicate that the data was produced with a cracked version of the program.

Anti-debug measures warrant some background explanation. To crack a program, an attacker typically needs to disassemble it. This turns the binary file into a series of assembly language commands using a "debugger" so that a human can understand what the program is designed to do. An overly simplistic description of a debugger is that it is a program which allows a human to see what commands the target software will send to the CPU before the CPU gets the commands, and then monitor the results of the commands when the CPU receives them. With a debugger, a cracker can test changes to instructions and data to determine what modifications will make the software behave the way that the cracker, not the author, wishes it to behave. Once the changes are determined, the cracker may write a patch to automatically modify the program, and post the patch on a website for script kiddies to use on their copies of the software.

Preventing the use of a debugger greatly impedes a cracker's progress. Measures to do this include debugger detection and programming the software to perform a series of tricks that violate some of the assumptions made by most debugger programs. For example, debuggers often make assumptions as to which bits in the target file represent data and which represent instructions. While cracking programs (under DMCA exemptions!) I have seen many different tricks, some of which are exceptionally clever. But after seeing them once, they're usually easy to defeat in the next piece of software that uses them.

PREVENTING REVERSE ENGINEERING

Anti-reverse engineering protections are arguably the first line of defense against cracking attempts, since reverse engineering is a necessary step in tampering, and tampering is a necessary step in cracking any technical protections used for license enforcement. For example, a full line of technical defense could include a license enforcement method (such as a dongle or node locking) to prevent piracy, anti-tamper defense (such as self-healing to protect the license enforcement method) and a "phone home" capability to report the cracking attempt. Additionally, one of the anti-reverse engineering protections described below will hopefully discourage the majority of crackers before they make any meaningful progress.

The anti-reverse engineering protection methods include encryption, anti-decompilation, and obfuscation. Obfuscation, when done correctly, can be a formidable anti-reverse engineering protection. The goal of software obfuscation is to change the program so that a human cannot understand it, but yet the computer returns the same result. There are two primary types: source code obfuscation and binary obfuscation. Source code is generally what a computer programmer writes and often bears some resemblance to the English language. Source code is put through a process known as compilation to turn it into a binary file comprising a series of numbers on which the CPU operates. Good source code obfuscation will turn a nearly trivial program, comprised of only a handful of lines, into multiple pages of incomprehensible instructions, even while resulting in the same functionality. The earlier speculation about preserving trade secret status in source code modules that were openly sold was referring to a potential use of source code obfuscation. Binary obfuscation rearranges the binary file, but it does not affect the source code.

Encryption keeps the version of the file stored on disk in a manner that it cannot be analyzed either by a human or a debugger. The program is only decrypted to an executable state while it is running, and when it stops, the decrypted version is erased from memory. Memory grabbing is a common technique used by crackers to defeat encryption protection. Decompilation is a step beyond disassembly, because the goal is to produce a high-level language, such as C, which is easier to understand than assembly code. Decompilation is a difficult process and is easily derailed, so at least for now, software defenders appear to be winning this particular battle.

Technical protections should be used as a second line of defense whenever a developer is uncertain about the enforceability or cost of enforcing legal protections. Examples include an overseas market in countries with poor IP enforcement, use scenarios in which piracy is unlikely to be detected, judgment-proof users, the prevalence of irrational attackers who cannot be deterred, and high value software that passes a rational attacker's cost-benefit analysis. Other issues that drive a need for technical protections include developer liability and potentially expanding export opportunities. For example, guarding against disgruntled programmers or viruses may help reduce a developer's liability to customers. Some integrity checking tools may allow a program to monitor itself for the presence of a backdoor inserted by a programmer who plans to blackmail the developer after quitting employment, and when the developer has already sold the program to important customers. Plaintiff attorneys may eventually see a link between easily cracked software and product liability tort claims. Obfuscation and anti-tamper techniques have applicability in foreign military sales. Eventually, many DoD contractors may see a requirement for technical protections inserted into their contracts, even for purely domestic projects.

PROTECTION ISSUES

In many situations, a good technical protection defense does not need to be perfect or entirely impenetrable. Often, a developer only needs to raise an attacker's cost up to the point where an attacker could have better spent his time, skill and money simply developing a competing product from scratch. Unfortunately, since cracking tools and techniques are evolving, technical protection measures have a shelf life that is likely far below patent and copyright terms. Good security will avoid a crack-once, cracked-forever scenario, in which even if the initial cost is high for a crack, the subsequent marginal costs for extending the crack to other copies of the software will be relatively low. There are ways to protect against a single patch working for multiple copies of the software, but such methods are beyond the scope of this introductory-level article.

Software has a bad reputation for schedule slips and cost over-runs, so introducing new security requirements is likely to run into opposition. And when the requirements are introduced, you can expect them to catch the blame for all sorts of problems. There is some reason for apprehension about software security. Software features that ease maintenance, such as orderly, sensible logic flow, often make cracking easier. As a result, many changes that make cracking harder also complicate maintenance and bug-fixes by preventing the use of patches. Further, many technical protections are not interoperable. To effectively manage the challenges, security should be planned from the first day of development.

There are parallels between legal protection strategies and technical protection strategies, such as budgeting based on IP value, and being careful about overexposing protection mechanisms. Just as filing too many patent applications may expose too much IP, overusing a technical protection exposes it to multiple crack attempts. One concern is that a protection used on one piece of software may enable an attacker to learn how to defeat another piece of software that is similarly protected. Attackers learn. Don't provide them with educational materials.

Cost and budget offers another way to compare legal and technical protections. Patents have a moderate initial cost, but can have a prohibitive cost to enforce. Trade secret protection may have a relatively low initial cost, merely preparing and signing NDA's, but there are real costs in terms of workforce inefficiency and missed collaboration opportunities, due to restrictions on information flow. Copyrights may be inexpensive to obtain, but enforcement may create bad public relations. Technical protections, in contrast, have only moderate up-front costs, and generally no costs for enforcement, because they operate automatically. Typically, costs for technical protections will decrease over time as the developer becomes more proficient and experienced. Further, unlike patents, technical protections have no examination requirement, so a developer can protect anything the developer believes comprises IP, even if a patent examiner would disagree about novelty.

BASIC STRATEGY

A basic software protection strategy is to deter, detect, and drive up costs. Legal protection provides deterrence. Technical protections provide means for detection and driving up attacker costs. To address the scenario presented in the earlier part of this article, I would recommend the following: The client should fingerprint all copies of the software it sells, if practical, but if not, at least watermark it. If the overseas competitor did any code lifting, a fingerprint could have enabled the developer to identify the customer from which the software was stolen. Alternatively, watermarking could have enabled the developer to identify whether the competitor used code lifting to steal IP and shorten its development cycle. A license enforcement protection, itself protected by self-healing, would prevent web site postings of the program from cutting into sales. A "phone home" protection that detected tamper attempts could assist in discovering the identity of the skilled cracker, while obfuscation, either source code or binary, could slow or prevent the cracker's ability to develop a patch for the new version of the program. Self-healing can also be used on critical portions of the program to prevent upgrade via patch, but there may be practical reasons to limit self-healing to small portions of the program, such as the license management sections. There are a myriad of other technical protections to recommend, but the ones listed here form a decent starter set that address many of the issues in the presented scenario.

RESOURCES

Following are some resources you may wish to investigate for possible referrals for your clients – either the specific listed organizations or else competitors offering similar products. I am not affiliated with these organizations, nor do I have any financial relationship with any of them, and I am not endorsing any of the products. The list is necessarily abbreviated for space, and admittedly omits the majority of software security providers. The descriptions of even the included products omit many features in the interest of brevity.

EnforcIT by Arxan is predominantly an anti-tamper tool, but offers protection against code-lifting and reverse engineering. EnforcIT inserts "guards" into the program which perform integrity checks, including self-healing, and also performs binary file obfuscation.

TransCoder by Cloakware is predominantly an anti-reverse engineering tool that performs source code obfuscation. Anti-tamper and anti-code lifting are inherent capabilities, due to the strong likelihood that any changes to the binary executable will introduce undesirable consequences.

Dotfuscator by PreEmptive Solutions is a binary obfuscation tool to prevent reverse engineering, and also provides anti-tamper and anti-code lifting protection out of inherency. Additionally, PreEmptive offers a "phone home" tool and service for monitoring of cracking activity.

Armadillo by Silicon Realms offers license enforcement and anti-cracking protection. Sentinel by Rainbow and Hasp by Aladdin are commonly-used dongle solutions for license enforcement. IPT