Basic Considerations for Preventing Software Piracy

Andre Armstrong and Vic Luu

Preventing Software Piracy

This paper is for software developers and organizations that are considering protection and licensing of their software applications. It provides basic information that can be used to support and plan a decision to protect an application from piracy.

This document explores the impact piracy has on developers and organizations as well as identifies who pirates software and the reasons behind those actions. Additionally, it identifies what makes an application attractive to pirate and covers basic points that both developers and organizations need to consider when exploring the options for software protection and licensing.

The Impact of Software Piracy

Software is a necessary requirement in today's world of technological development. While continuously sought after, software is the intellectual property of the developer or the organization that produces it. While individual countries experience different levels of piracy, the Business Software Alliance (BSA) has reported that in 2001 about 40% of the software in use globally was pirated. The revenues lost to developers and organizations well exceed $10 billion for 2001. Even in the United States where the piracy rate was estimated to be 25% for 2001, the impact of software piracy is significant.

Developers and organizations need an option to exercise control over their intellectual property to prevent it from being altered, distributed or used by unlicensed, unregistered users. The lost revenue ultimately results in lower returns for stakeholders, developers, and employees. It culminates in a lack of additional funds that could be used for research and development, hiring of senior developers or for marketing of a new product.

With the current global average of software piracy rates it would seem possible that business and marketing plans could be seriously flawed when identifying projected sales or revenues for a new software application. Additionally, large-scale illegal software

distribution has taken place on Usenet and Internet bulletin boards and Internet users can easily locate pirated versions of software packages online.

Who Pirates Software?

There are a number of different types of application pirates, most of who fall into two categories – unintentional and intentional pirates.

Unintentional pirates are individuals who purchase applications and are unaware of licensing and registration issues. People who are either not exposed to software development practices or do not understand the ethical obligations of using software comprise the majority of these unintentional pirates. This group includes the loyal employee who installs a corporate copy of a word processing application on his or her home computer for personal use or a spreadsheet application to manage a home budget.

Some organizations purchase or obtain applications and intentionally violate the licensing and registration requirements. Organizations without an active security policy that stipulates that only software received from MIS can be installed on the company computer may be unknowingly accountable for pirated software. Employees who download or bring pirated applications from home into work are putting their employers at risk by running unlicensed software in the workplace.

Intentional pirates can be casual copiers, individuals or organizations that copy unprotected software with no intention of financially profiting from this action or of infringing on intellectual property rights. Counterfeiters are individuals or organizations that copy unprotected software with the intention of financially profiting from their actions or to inflict harm. Hackers break into applications for the fun of it and usually have no initial intent of financially profiting from their actions or causing harm, but these individuals may become crackers. Crackers copy or break into applications with the objective of financially profiting and have the intent of causing harm.

Developers and organizations should give due consideration to the target market for which the software was developed. Will the intended users of the software have enough technical resources or expertise to hack a specific application? An engineering application used by technical users is more likely to being hacked than a word processing application used in an office environment.

Additionally, consider the geographic location of the target market. If an application is to be sold into a software piracy hot spot, it may be worthwhile to incorporate maximum anti-piracy protection into the product.

Reasons for Pirating Software Applications

In addition to the fact that technology has made software piracy as simple as connecting to the Internet, downloading the application and burning the application to a CD, there are other reasons why software is pirated.

Crackers like to resell the application through an online auction at a substantially reduced price or take the time to increase their personal expertise on a specific application and provide knowledge transfer for financial gain. Crackers can pirate applications to steal features and include these features in their own products. Ex-employees can have issues with a specific coworker or their ex-employer and break an application, distributing the

software for free or perhaps gaining notoriety themselves by damaging the credibility of the manufacturer or the application.

Individuals and organizations may be unwilling to pay what they perceive as high prices for additional licenses or there may be no money left in the budget and they feel a need to copy or crack the application. The application may be copied for a short-term solution to an existing problem such as a project overrun of scope and budget. In this case, individuals or organizations can copy and deploy the software, which allows the hiring of additional resource(s) to complete the project within scope.

Hackers perceive breaking the application as a personal challenge and get personal satisfaction when they've hacked the application. Hackers have broken into applications to highlight the applications weakness in hopes that the software developer rectifies the identified weaknesses and produces a more secure product.

Convenience is often a factor for individuals and organizations that will copy an application for immediate use, making the copying process more convenient than obtaining a new license. While believing this one off event will not have any ethical or financial impact, the accumulation of these events leads to severe piracy issues worldwide.

What Makes an Application Attractive to Pirate?

What ultimately makes an application attractive to an individual or organization is that someone sees value and an opportunity to attain that value free of charge. Applications that are unprotected or insufficiently protected are the easiest to duplicate, thus providing an excellent target for a temporary solution, not to mention cost savings and convenience.

Hackers see the potential for personal satisfaction or recognition by successfully hacking a highly visible application or targeting applications with significant brand awareness. Crackers are attracted because they believe they can crack the application and sell reduced-cost copies for personal financial gain. Significant brand awareness is perceived as a product in demand, which equates to additional revenue opportunities for the cracker.

Applications that currently have large revenues are the crackers' biggest targets. Successful copying and selling of the application can produce large financial rewards. Consider this: an application that sells one million units per year at $300 per unit grosses revenues of $300 million. A cracker could sell 10,000 units at half the price and could make $1.5 million from a single application.

Applications that have a high production cost per unit attract crackers – if they can crack the application, they can and sell copies and make personal financial gain. Selling only a few units, crackers can make a substantial financial return. A $10,000 application that's cracked can be sold as a special promotion for $5,000 - $8,000 and selling only 10-20 copies can make considerable revenue for the cracker.

Applications that are touted as highly secure are immediate targets for hackers. Security is seen as a challenge by the hacker and he or she gets great personal satisfaction or recognition from cracking a "secure" application.

Applications that sell in large volumes at a low price attract crackers who see these applications as a gold mine. Applications that sell at an inexpensive purchase price are targeted and the financial return from selling numerous copies is substantial for the cracker. Ten thousand copies of a pirated application selling for $100 can be moved at a special promotional price of $80 and return $80,000 for the pirate.

Does this happen? Microsoft identified that 90 percent of its software sold at online auctions is counterfeit (Microsoft's Worldwide Anti-Piracy Group 2000).

Considerations for Application Protection

Developers and organizations need to consider whether their application is likely to be targeted and, if so, how much effort the pirates will put into breaking the added protection. Additionally, the target market needs to be addressed. Is this a niche market where if pirated, you risk losing the target market? Where do you intend to sell the application in the world and what are the piracy rates for those world regions? Consider how cost effective it is to add the protection to the application. Ideally, there is a good balance between the protection plan and the financial benefits for that plan. Alternatively, an organization may choose, after analyzing the market, to incur some costs associated with piracy initially, in exchange for potential exposure from people sharing the application.

Timing is a significant factor to be considered by developers and organizations. If a product is scheduled for immediate release, then the amount of time available to add protection to the application is limited. The same may hold true if the development schedule is tight in an effort to meet a deadline that cannot be adjusted and allow for the additional development time necessary to add the required protection.

Resource availability comes into play when neither the funding nor the developers are available to add the desired protection. Organizations need to know what resources are available to them. When determining the protection plan for an application, organizations need to consider what the cost will be in terms of resources and time to implement the desired protection scheme.

Software protection can be implemented in a single phase or several phases over a period. A phased approach could use the application "shelling" option, which wraps a quick fix protective layer of code around the application's executable. At a later stage, API calls can be added to further enhance the application security. Alternatively, consider protecting one or more modules using API calls with a future intention of protecting the complete application using multiple API calls enhanced with various implementation methods.

You can also consider adding licensing options to your application at the same time that you apply your security plan. Licensing options are not restricted to unlimited usage for every application purchased. Consideration should be given to large modular applications for adding the option of renting or leasing by module. Additional licensing options such as counters can be added to source code to allow for demonstrations of the application. Datawords can be useful for a date storage function used for purchasing an application for a fixed period for a fixed price. Other licensing options can include time-sensitive or limited-version features that can be used after registering the software, or an application that can restrict functionality based on the user registration. Options may also include the ability to rent software applications or trial periods for related software modules with options to "upgrade" to fully licensed versions of the software.

When considering application protection, there are two choices to consider. These are protection through shelling and protection through API calls. If API calls are used, many algorithms can be implemented to increase the protection. API-level implementation has implications for the resources required and the time to implement the solution but offers the most flexibility and options for protection and licensing. Based on the cost benefit analysis performed developers and organizations can choose the best balance of security implementation for the identified application.

Acceptable Level of Protection Using Shelling and Hardware Keys

The acceptable level of protection is obtained by the use of the automated Sentinel Shell feature. The Sentinel shell is a protective, encrypted layer that wraps the executable file or DLL. The Sentinel Shell automatically handles functions such as checking and verification by using an internal proprietary algorithm with a table of query/responses to perform periodic token checks. It doesn't require programming skills and can be applied in a matter of minutes. An application can only be run if the user has the correct hardware key attached.

The protection works with calls being made via the Sentinel system driver to cells on a pre-programmed hardware key. The calls query the key to verify the algorithm; the key, if present, responds with an algorithm response and the application determines if this is the correct response. No response or an incorrect response will cause the application to stop running. The Sentinel Shell solution may also be the appropriate option if the source code is not available.

Higher Protection Using API Calls and Hardware Keys

Additional protection can be achieved using automated Sentinel Shell and Sentinel API calls. Sentinel APIs must be embedded directly into the source code. Because additional programming is required for this level of protection, developers typically allow additional time to implement this solution. A rule of thumb is to allow a development day per feature you are protecting, where the "feature" may be the entire application.

After the security plan is outlined and the security level and any license management options are outlined, the developer integrates the protection with the hardware key. The developer adds Sentinel API function calls to their source code and pre-programs the cells in the hardware keys. The developer determines the number, types and when the calls are made and what action to take if no response is made or if the response is incorrect.

The Sentinel SuperPro hardware key holds up to 28 algorithms, allowing the use of multiple algorithms to protect one application. For higher levels of security, the developer may randomly query multiple algorithms each time the hardware key is checked. Developers are able to camouflage real software locks with varying numbers of random queries increasing the difficulty of cracking the application source code. Developers can perform actions on responses from random queries, which are ultimately ignored, increasing the number of queries to record, thereby thwarting pirates who attempt to hack an application through a playback method.

Examples of random query pseudo-code:

```
X = Rand() AND 3              //Example 1
for i = 0 to X
        R = query( rand() )  //Provides for up to 3 random queries
EndFor
```

```
while NO userInput                          //Example 2
        userInput = getCmd()
        R = query( rand() )  //Provides random queries while waiting for
                                    //customer input
endWhile

if (rand() MOD 2 = 0) then    //Example 3
        R = Query (Q)         //Performs a real query if the random number is even
        if (R <> Expected R) then <error response>
        endif
endif
```

An additional security option can be implemented by using a random index to a table of known call/response pairs. Divide the table into smaller tables spacing their placement within the code apart from one another. Store queries in one table and responses in another and use different indexes to the tables that are related through a calculation.

Example of a random index to a table of known call/response pairs:

```
tSize = SizeOfQueryTable    //Constant for the number of known queries
X = Rand() MOD tSize        //Generate a random index into the known table
Q = A[X].Q                          //Perform a real query using the random index
R = Query (Q)
if R <> A[X].R then <error response>
endif
```

This implementation prevents identifying if all the real queries have been sent. If a software crack works once, it is no assurance that another attempt will work.

Do not declare a security violation for tests that fail. Show fail with a non-descript message like “Memory Corruption” rather than “Incorrect Hardware Response”. Placing random queries after a failed test may help confuse a hacker as to which failure was real. Be sure to vary error responses to hide both when and where an error was detected.

Verify that an inappropriate action caused a specific error by unplugging the key and prove communication with it failed. Perform illegal activities such as decrementing a memory cell or writing to locked memory to verify errors.

Highest Protection Using API Calls and Hardware Keys

Implement additional protection such as the examples above and include the following options as well.

Issue many calls throughout the source code. A safe guideline is to issue one evaluation query for every $20 of the application cost. Protect DLLs as well as executables to increase the size of a code patch. Placing some calls in the code that are rarely entered so a seemingly “cracked” application still fails occasionally.

Separating the Call, Test and React sequences by time and code will increase the amount of code to trace and give hackers a bigger challenge and disguise real software locks as a random query by increasing the lock’s scope.

Removing the need for code to contain the correct response value will allow the use of a response to perform an action. Require additional time to trace through the process of using a response or use the response as an array index, constant, calculation element or checksum. Use the response as a pointer or a jump-table index to control the program's execution and consider encrypting or decrypting sections of the code with a response.

Do not declare a testing function with a query and call it from the code, this causes a jump in execution into a single body of code which can be easily bypassed; one body of code requires only one patch.

Macros can be used for code modularization. A copy of macrocode can replace each call when it's time to compile. Code patches for each software lock macro should be required.

Disable interrupts by locking out the keyboard as well as some debuggers. Be conscious of time spent with interrupts off and perform some critical calculations in Ring Zero (OS kernel mode) on Intel machines.

Executing different software locks forces a hacker to simulate multiple environments as well as attempt to break the security. Other options for increasing the level of protection include the following: vary the security code executed depending on system clock time, date, available disk space, date of installation directory, and system information like processor and OS version.

Conclusion

Software piracy is a serious issue that impacts the bottom line for software developers. By implementing a security plan for software protection, software developers gain the benefits of protection from piracy as well as obtain the ability to implement additional license models. A security implementation plan that balances the time and resources with the desired outcome is possible given the wide range of security options. Developers can additionally choose a phased approach to the security implementation if time or resources are constrained in the short term.

Corporate Overview:

Founded in 1984, Rainbow Technologies (NASDQ: RNBO) is a leading provider of digital content and transaction security solutions for the Internet and eCommerce. Rainbow applies its core technology to a variety of Internet applications from securing software, custom high assurance security hardware, complete eBusiness services and content/transaction security solutions for corporate networks, Instant Private Web appliances, Virtual Private Networks (VPNs), and the Internet. Rainbow's products include: secure Web server acceleration solutions; anti-piracy, and Internet software licensing and distribution solutions; Public Key Infrastructure (PKI)-based security solutions; voice, data and satellite information security and authentication solutions and USB-based Web authentication keys.

Rainbow Technologies is comprised of Rainbow eSecurity Group for digital content and transaction security solutions for the wired and wireless worlds and Rainbow Mykotronx for custom security and high assurance security solutions. Both Rainbow eSecurity and Rainbow Mykotronx are **ISO-9001** certified.

www.rainbow.com