# From Hack to Counterattack:

## A Guide to Protecting Software Products Against the Top 8 Piracy Threats

**WHITE PAPER**

Nowadays, an ever increasing number of Independent Software Vendors (ISV) chose to implement some sort of protection in order to ensure that they are being fully paid upon end-users utilizing their applications.

### Table of Contents

According to the BSA/IDC 2010 Piracy Report - for every $100 worth of legitimate software sold, an additional $75 worth of unlicensed software also made its way into the market generating absolutely no revenue to the original ISV.

## Types of Software Attacks and How to Defend Against Them

Nowadays, Independent Software Vendors (ISV) must implement some type of protection in order to ensure that they are being fully paid upon end-users utilizing their applications.

The basic idea behind any software protection solution is to restrict the use of unlicensed software by some form of licensing scheme. This can be achieved via hardware tokens (also known as dongles), software-only licenses, or homegrown solutions. In general, hardware-based protection keys provide the most robust security, while software-based keys provide support for Trialware and Electronic Software Distribution.

External hardware-based solutions provide the highest level of security currently available. Unfortunately, hackers are a persistent nuisance that cost Software Vendors billions of dollars in lost revenue worldwide. Therefore, it's important that the selected solution and the actual implementation seal off all potential hacker entry points.

**This white paper examines a variety of the most common hacking techniques and the best counterattacks available for protecting applications from piracy.**

## HACK #1: Patching Executable Files or Dynamically Linked Libraries

In this scenario, the software cracker attempts to disassemble and/or debug executable files or dynamically linked libraries in order to retrieve the protected code. The executable file(s) is then patched in order to modify run-time flow, or to remove calls in the code. Commonly, the software cracker sends a small standalone patch executable (called a "crack") that the end-user runs in order to patch the licensed software.

### Counterattack

The goal is to make the process of patching executable files more complex and costly than purchasing the software. There is a direct proportion between the number of protected file(s) and the amount of time required by the software cracker in order to remove the protection. The most convenient way to protect a large number of executable and dynamically linked libraries is by using automatic protection tools, also known as "packers". Injecting the protection code into the compiled binary files reduces the need to laboriously write protection code for every individual file. The most effective tools also offer anti-piracy specific measures such as anti-debugging mechanisms, executable file encryption and data file encryption mechanisms allowing the protected application to encrypt and decrypt data files on-the-fly.

## HACK #2: Brute Force Attack

A brute force attack scenario, whether generic or specific, consists of hackers exhaustively attempting to compromise every security measure of the secret being protected. A generic hack means that the dongle itself is compromised - no amount of implementation improvement can counter such a hack, and all applications protected by the hacked dongle are at threat.

Specific hacks break only one specific dongle implementation for one particular software application – this in result does not pose a risk for other software vendors using the same dongle.

### Counterattack

Brute force attack can be overcome if the secured data is large enough, i.e. cracking all combinations becomes virtually impossible. In order to protect the algorithms that are central to the security of a dongle, the dongle should have a secure kernel and communication channel that allows storing the cryptographic key. The security kernel of a dongle consists of a 128-bit AES encryption key allowing for the protection of the algorithms central to the dongle's operation.

Forces driving piracy down in many economies included vendor legalization programs, government education, enforcement actions and major technological shifts – such as employment of Digital Rights Management (DRM).

## HACK #3: Emulating Protection Keys

In an emulation (record/playback) attack scenario, all information exchanged between the application and the dongle is recorded – in an attempt to replace the device driver.

The use of middleware, usually a software application, provides a platform which attempts to mimic the requests and responses provided by the host. The dongle is replaying previously recorded calls as if the actual key is returning the calls.

Limited functionality emulators only record and replay previously recorded calls, whereas fully-functional emulators also emulate the key, including its encryption.

In order to create a fully-functional emulator, the software cracker requires access to the encryption key. Successful emulation of the two-way communication means that the application has been compromised. Emulating the protection keys is not a generic hack as it only affects the application copies on which it may exist, and can be easily amended.

### Counterattack

Protection keys rely on a secure channel between the application and the hardware key; data that passes between the protected application and the key is encrypted. On-chip 128-bit randomized AES encryption ensures a robust link between the application and the hardware key fully securing the communication channel. The channel employs a different encryption key for every session. This means that the cracker attempting to record data passing through the secure channel would not be able to replay the data, since the encryption key used to encrypt the data will differ from that used to decrypt the data. In addition, the protected application can detect limited-functionality emulators by utilizing the encryption engine of the hardware key to decrypt hard-coded data. The emulator itself has no way of correctly decrypting such data, as it has no access to the encryption key.

## HACK #4: Modifying Key Memory

Licensing data is normally stored in specific area in the memory of a software protection key. In this scenario, the software cracker will attempt to access the dongle's memory in order to modify its licensing terms. For example, the cracker will attempt to change a depleted execution-based license into a perpetual license, or enable a feature that was not initially paid for. This can dramatically decrease any current or future applications' usage revenues.

### Counterattack

To prevent key memory modification, the licensing implementation has to include usage of the built-in Read-Only Memory (ROM). ROM is a segment of memory that can contain data to which only the protection application can access, but not overwrite, unlike regular Read/Write memory segments. This memory segment can only be updated using remote updates (C2V and V2C file types), and the creation of such updates is only possible for the specific vendor.

## HACK #5: Device Sharing

Device sharing refers to multiple PC's sharing a single license without authorization, i.e. one dongle is used by several machines on the same network operating on multiple concurrent terminals. This scenario does not jeopardize the security of the application itself; rather it has the potential of dramatically decreasing current or potential application revenues as multiple users share a common copy of a single application. The use of a static key for encryption results has greater susceptibility to this attack.

A software publisher's own protection implementation must be thorough, complete, and to a certain extent creative. Only when both the security infrastructure and its implementation are maximized is the software truly protected. This may sound like an enormous task to accomplish, and while it is by no means trivial, it is definitely achievable. Implementing effective protection can not only increase your product's longevity, but it can also significantly increase your company's revenue, making it well worth the effort.

There are two potential scenarios under which device sharing can be done:

- Multiple VMware Sessions: In this scenario, multiple VMware sessions are concurrently running on a single hardware platform. For example, each VMware session utilizes its own instance of the driver to communicate with the dongle allowing for multiple virtual operating systems access to a single application.
- USB Sharing Hub: In this scenario, multiple computers sharing a USB hub are concurrently using a single license dongle as if individually communicating with it. This can allow multiple PC's to run an application even though only a single hardware license exists.

**Counterattack**

Software vendors need to ensure that the quantity of computers or device drivers accessing the dongle does not exceed the purchased number of customers. Secure tunneling of a dongle can help safeguard against multiple devices or machines trying to simultaneously access a single dongle.

Every use of the application will open an instance of the driver; two or more instances of the driver will not be able to communicate with the dongle through the secure channel. The dongle driver will protect against these scenarios and enforce only the allowed number of licenses to be run.

## HACK #6: Cloning Hardware Keys

In this unlikely scenario, the cracker attempts to reverse-engineer a hardware protection key and manufacture exact duplicates. This is an extremely difficult and costly attack both in terms of reverse-engineering tools and expertise, not to mention the ongoing production costs of such dongles.

**Counterattack**

The countermeasure of such attacks can be either hardware or software based, preferably both. One method of reducing the likelihood of this attack is by implementing customized hardware components that are not available for purchase off-the-shelf, and that contain specific anti-tampering measures. In terms of software, the firmware used is to be protected against extraction/cloning/copying.

## HACK #7: Time Tampering

This relatively simple scenario occurs when the end-user attempts to tamper with the machine system clock on which the protected software is running to an earlier date, thus re-enabling a time-limited license that has already expired.

**Counterattack**

One method of securing trial licenses is to simply offer limited or crippled software versions. However, if you would like to create time-based licenses in order to offer software subscriptions or complete time-limited trials, time-tampering is a risk that must be assessed and battled.

An additional method for securing time-based licenses is using a battery powered RTC (Real Time Clock) built into the hardware key. Equipping the dongle with the ability to check and store the current system time, allows the developers to use this information in an attempt to prevent time tampering by detecting any significant changes in system time. Thus, the application can be preprogrammed to disable or run in a restricted mode.

## General Strategies for Protection

- Use Both the API-Based Implementation and Automatic Protection
- Insert Multiple Calls in Your Code
- Encrypt/Decrypt Data Using the Protection Key's On-Chip Encryption Engine

## HACK #8: Tampered License Requests and Updates

This scenario refers to the actual implementation of license requests and updates. If such requests are verified at the software level, i.e. by the device drivers of the hardware key, crackers may attempt to attack this code and force all licenses to be automatically granted or applied. This can even apply to licenses which have already expired or license updates that have already been applied, potentially granting full access to revoked/expired dongles.

### Counterattack

The granting of licenses and the application of license updates should be performed by the hardware key itself, outside the reach of debuggers. Failing to obtain a valid license results in locked functionality. For example, the encryption engine should remain disabled until a license is successfully granted. Similarly, license updates should only be applied after the hardware key itself verifies the signature.

## General Protection Strategies

These following general software protection strategies are recommended for publishers employing hardware-based protection keys:

### Use Both the API-Based Implementation and Automatic Protection

Maximize security by using the API to implement calls to the software protection keys, and protect the program with an automatic protection tool that offers anti-debugging and anti-reverse engineering measures. Using one protection method does not preclude the use of the other.

### Insert Multiple Calls in Your Code

Inserting many calls throughout the code to the protection key and binding data from the key with the software functionality frustrates those attempting to crack your software. Multiple calls increase the difficulty in tracing a protection scheme.

### Encrypt/Decrypt Data Using the Protection Key's On-Chip Encryption Engine

The encryption process, which takes place before distributing the software, and the decryption processes, which take place during run-time of the protected application, should be performed inside a hardware protection key, beyond the reach of any debugging utility. The protection key should employ a highly secure public encryption algorithm such as AES. Encrypting data with protection keys using such algorithms considerably enhances software security. Cracking the software necessitates the software cracker to decrypt the data. The encryption key length represents a trade-off between security and performance. A short encryption key provides better performance while reducing the security level. 128-bit AES provides a well balanced choice, as it is faster than 256-bit AES, yet still impractical to attack.

It is important to remember that protection is only as strong as the weakest link. An effective implementation must build upon a software protection solution that offers all the building blocks for a secure implementation:

- An encryption engine that employs a proven public encryption algorithm such as AES
- Automatic protection with anti-debugging detection and other anti-reverse engineering measures
- A secure communication channel
- ROM – Read Only Memory

LicensingLive!™ (lahy'sun sing lahyv'), adj. n. [SAFENET, INTERACTIVE] 1. Immediate access to the best practices and emerging challenges associated with software packaging, pricing, fulfillment, delivery and management. 2. A forum bringing together software vendors, industry analysts, licensing consultants and technology vendors.

## Conclusion

A software publisher's own protection implementation must be thorough, complete, and to a certain extent creative. Only when both the security infrastructure and its implementation are maximized is the software truly protected. This may sound like an enormous task to accomplish, and while it is by no means trivial, it is definitely achievable. Implementing effective protection can not only increase your product's longevity, but it can also significantly increase your company's revenue, making it well worth the effort.

## SafeNet Sentinel Software Monetization Solutions

SafeNet has more than 25 years of experience in delivering innovative and reliable software licensing and entitlement management solutions to software and technology vendors worldwide. Easy to integrate and use, innovative, and feature-focused, the company's family of Sentinel® Software Monetization Solutions are designed to meet the unique license enablement, enforcement, and management requirements of any organization, regardless of size, technical requirements or organizational structure. Only with SafeNet are clients able to address all of their anti-piracy, IP protection, license enablement, and license management challenges while increasing overall profitability, improving internal operations, maintaining competitive positioning, and enhancing relationships with their customers and end users. With a proven history in adapting to new requirements and introducing new technologies to address evolving market conditions, SafeNet's more than 25,000 customers around the globe know that by choosing Sentinel, they choose the freedom to evolve how they do business today, tomorrow, and beyond.

For more information on SafeNet's complete portfolio of Software Monetization Solutions for installed, embedded, and cloud applications or to download a free evaluation of our award winning products please visit:
http://www.safenet-inc.com/sentinel/

To download a FREE SENTINEL HASP Developer Kit, visit:
http://www3.safenet-inc.com/Special/hasp/safenet-hasp-srm-order/default.asp

### Join the Conversation

Sentinel Online
www.Safenet-inc.com/sentinel

LicensingLive! POWERED BY SAFENET
www.LicensingLive.com

Twitter
http://twitter.com/#!/LicensingLive

LinkedIn
http://bit.ly/LinkedInLicensingLive

YouTube
http://www.youtube.com/user/LicensingLive

BrightTALK BrightTalk
http://www.brighttalk.com/channel/5572

**Contact Us:** For all office locations and contact information, please visit **www.safenet-inc.com**
**Follow Us:** www.safenet-inc.com/connected