The pirate life for me

Jake Carroll | Mar 2, 2010 10:53 AM
X-ray: Jake Carroll puts on an eye-patch and looks into the history of software piracy and protection.

Way back when, in the 1970s, a young San Franciscan man was unwrapping some fine shiny software for his Apple II.

A few days later, he started wondering about the real value of the 0s and 1s on that floppy disk. After eating a month old Twinkie and riding a sugar high, he decided to try something brash. He was going to copy that software, somehow, and give it to a friend.

That day was the beginning of software piracy.

This month we're investigating software piracy. Piracy has a significant impact on both the hardware and software industry, to the point where we've had to change the landscape and economics of the gadgets and tools we buy, in order to accommodate the phenomenon. Don your pirate hat, talk to some guy called Uncle Torrence and put a 'z' on the end of every word. We're sailing into uncharted waters.

The ethos
Over the years, longitudinal studies have been carried out using psychological profiling in an attempt to understand why people pirate software. Studies such as the Triandis Theory of Interpersonal Behaviour in understanding software piracy (Robinson et al, 2009) and Factors Contributing to the Understanding of Software Piracy among College Students (Liang et al, 2005) have attempted to broaden our knowledge. The findings of the studies revolve around:

- The economics of stealing (you're getting something for nothing).
- Soft distribution mechanism (because its software, it can be shuffled from place to place without thought or regard for the actual source).
- Convenience (digital content protection mechanisms and physical media are so convoluted, the argument goes, that its more simple to just jump on bit torrent and do a little torrent window shopping).
- The white hat conquest (the crackers of our world enjoy breaking software protection mechanisms as an intellectual challenge, and nothing more).
- The black hat conquest (the profiteers of mass duplication, sale and counterfeit of software).

Regardless of piracy being morally wrong, and criminal, there are different motivators for different subsets of users

different subsets of users.

From the dawn of time

To begin with, piracy was almost an innocent concept; people really didn't think too much about the implications. In the days of the cassette tape and 5.25in floppy media, copying things was a slightly painful process. Neither ProDOS nor DOS 3.3 had any built in copy applications, or utilities, so third party tools such as Disk Muncher came about. Still nobody thought much of it in terms of crime or illegal behaviour. At some point in time, somebody realised it was hurting profits. Of all the somebodies... it was a games company.

The humble checksum

To begin with, copy protection methodologies were primitive. Software was installed with a sub routine that checked to see if the original hexadecimal offset was present on the media. If not, software could make the inference that the media was a 'copy' and not the original. In a similar situation, checksums and 'intentional corruption' were used as a means for applications and software engineers to prevent piracy.

The use of checksum techniques on shared media meant that when a user intentionally copied something from a disk onto another media type, it would slightly change something about that data, even if it were the headers of a file or table of contents. A small routine would run around and 'sum' looking for a known result, and if the result was wrong, the inference was the user was attempting to run from an illegitimate source. This got old pretty fast. People got sick of being tied to one form of media for all reads, loads and application dependence.

Dongle

In 1980, a software firm produced a piece of software known as WordCraft. The company became the first to ship a hardware protection mechanism to control software usage and piracy. Enter, the 'Dongle'. The premise behind the dongle was simple. You couldn't start a piece of software without the dongle being inserted into the computer, be it through a serial port, or a PET cassette port (as it was on the Commodore 64).

Over time, the concept of the dongle became more and more refined, with everything from simple UART (Universal Asynchronous Receiver Transmitter) systems being used to 'test' the validity of a dongle, to small microprocessors even being embedded in the devices. The more complex these systems became, ironically, the simpler they were to crack. The reason for this was the exposure through bus tracing of the protocol being used to negotiate the transaction between the software a user would run, and the hardware dongle. In trivialised dongle systems, the designer might simply put a return statement into execution to the effect of:

IF dongleUID[] is PRESENT

THEN return 1;

ELSE return 0;

Such Boolean return statements are easily cracked by inserting spurious code on the localhost, or target machine.

Dongles have become more intelligent however, with cryptographic sums, crypto processors, hardware drivers and real time clocks (RTC) being employed. Interestingly, the dongle was the precursor to a whole line of new protection technologies to follow that, without the dongle, probably would never have existed.

The hardware checksum

By the mid to late 1980s dongles were being beaten, and the software checksum was a bit of a joke. Somebody got the idea that a global hardware level checksum might be a useful mechanism in slowing down the rapid growth of piracy. The idea was simple. Take a big bit of hardware, that has some known components and give each component some kind of 'signature' or assigned value. That value would then be added to the rest of the components in the host, to form the global checksum for the device.

When hardware piracy and the modifications being used to defeat software checks in hardware became apparent, the use of the global checksum came into its own. The global sum, if interrupted by a newly introduced component or unauthorised part, would fail, potentially rejecting the user from running a certain type of software or proprietary code. For those following closely, the Xbox 360 and PlayStation 3 both use a hardware level checksum across their components. Logically, the introduction of mod chips into the consoles has been difficult, as a result. Mess with the hardware, and you're messing with the software.

The naughty 90s

Permutations of existing techniques were coming thick and fast. Ever more vendors shipped ever increasingly convoluted solutions using dongles, keys, checksums and validation methods. With the advent of the Internet and popularisation of network-connected computers, challenge and response codes became popular. A user would receive their software in a box with an activation key on it. They would install the software, and then the software would ask for a validation code. The end user would put their activation key in and a validation key would come back from the software vendor server. This would then be input into the software - and provided it 'matched up' on both sides, the product would be activated.

This gave rise to the ever-popular keygen. Key generators became popular when the power of decompilation tools (and thus, reverse engineering) became accessible to those outside of large-scale software development teams. The premise of the keygen was simple. Pull apart the proprietary code by disassembling it into near-machine code, then figure out what sequences of keys or algorithms were used to create said keys. Once this information is obtained, put it into software that can run externally to the application, and generate based upon the algorithm.

Key generators today represent the largest 'market' for open piracy, often being created for everything from games to applications and operating systems. For many of the groups and individuals that make key generators, it's all about the challenge of the reverse engineering method and technique, rather than simply stealing software.

Of deformities and wobble sectors

Along the time line, it's now early turn of the century. Software companies are under threat, people are losing jobs, and the industry is under a slow but definite landslide. Here is where the 'fun' stopped and things became serious.

The RedBook CD standard was well in place for audio CDs, as were the respective standards for data DVDs. It became painfully apparent that it wasn't a problem for people with the right equipment to 1:1 copy media, no questions asked. The vendors had an ace up their sleeves, though. Along came SafeDisc.

Command and Conquer: Red Alert 2 started the trend. The disc would copy fine. Everything seemed sensible. It would then be installed from, and everything still seemed sane. Then the problems would arise. People couldn't play from their backup burnt media. Was the burner missing something? It sure was!

Weak sector and Eight to Fourteen Modulation (EFM) encoding were the key to the protection mechanism in Macrovision's SafeDisc.

First, the concept of the 'exclusive or' (XOR) needs to be understood. XOR is a bitwise operation. XORing a number with 1 flips a bit, but XORing with 0 does not. If your input bit reading from a disk was 1, and it was XORed with 1, the result would be 0. If your input bit was 1, and it was XORed with 0, the result would be 1.

The next part of the equation comes in understanding of how a CD/DVD drive actually works. A laser in the most modern of drives still finds it hard to detect frequent changes between physical pits and lands. As a result, changing the number of pits and lands frequently is a recipe for failure. In this respect, when reading/writing a CD or DVD, the number of these pit/land changes must be minimised. This is where EFM was invented. It turned eight bits into 14 bits, with an XOR lookup table alongside, as a hardware encoder/decoder.

The next step in understanding SafeDisc is the Digital Sum Value (DSV).

The DSV is an integer that changes at each point along the media. For every pit on the disc, the DSV is incremented by one, and for every land it is decremented by one. For instance, take the following series of pits and lands:

Pit-Pit-Pit-Pit-Land-Land-Land-Pit-Pit-Land-Land-Land-Land.

Here, the sequence of pits to lands would be +6, -7.

Here is where SafeDisc and several other copy protection techniques kick in. SafeDisc's weak sectors are already XORed with the verified outputs of what a sector XOR engine will be. Once this hits the EFM encoder on the way 'in', because it's already XOR'ed, it'll be twice the pattern that it was previously, and thus have to traverse twice the length on read 'in' or write 'out'. The

that it was previously, and thus have to traverse twice the length on read 'in' or write 'out'. The algorithm used for calculating these merged bit patterns is far too intensive and slow for a burner to deal with. When confronted with these long chains of weak sectors a drive doesn't know what to do, so it throws the read away as garbage data or a read error. When this happens, the SafeDisc module gets the hint, understands that the media is forged, and refuses to run.

Hyperstrong

So, with SafeDisc emulated, broken and forged by tools such as Daemon Tools, the companies have had to try harder on numerous fronts. Proprietary media, barcode data (BD+ watermarks on Blu-Ray media) and low-level hand shaking are all part of the game.

At the stream level, High Bandwidth Digital Content Protection (HDCP) is stopping direct packet ripping in its tracks on the consumer set-top front by creating a handshake tunnel using a 56-bit key and a small stream cipher consisting of up to 20 permutations for each device. This can be thought of as analogous to an SSH tunnel between two UNIX hosts, but made purely for pixels and sound.

The most modern copy protection techniques use a combination of:

- Weak sector/media signature identification (SafeDisc et al).
- Global hardware checksums.
- One-time keys and activation techniques.
- Software enclosed environments.

In an abridged sense, we arrive at this point, where the hypervisor is born.

The hypervisor, used by many digitally sensitive modern environments such as the PlayStation 3, BD-ROM drives and large scale server platforms (VMware VSphere, Linux KVM et al), is a method for encasing a running set of known routines, in which software will work - guiding direct hardware level access. Without the instructions the hypervisor can provide, and effectively 'useless' hardware underneath, the end user has no choice but to run through the provided mechanism. When the user runs through this mechanism, they play by the hypervisor's rules. The PlayStation 3's content protection remains unbroken, to this day, proving that total control over hardware, software and middleware is an ideal means to plug the piracy hole.

Software engineering, hardware engineering and protection development are all lucrative pockets in which to dwell. In a world where e-protection needs must continue to grow, digital content protection is only going to become more important. Wouldn't you rather play for the good guys, than the bad?