Darwin/x86

Mac OS X Binary Protection

I hesitated for a long time to write a page about Apple's binary protection scheme. As I saw it it was one thing for me to know how to get around the protection, quite another to disclose it. But at this point there are countless kernel extensions floating about. The three major ones would be dsmos, r3d3 and AppleDecrypt. Aside from the kernel extensions various patches to xnu have included the binary decryption inside the kernel.

So if you just want to get around Apple's binary protection you have plenty of choices. Go elsewhere because you're not going to find anything ready-made here. On the other hand if you want to learn a bit about how it works, stick around.

Back in October 2006 Amit Singh kicked things off with a basic analysis of how the kernel identifies protected binaries and what the kernel does upon encountering them. This article is entitled "Understanding Apple's Binary Protection in Mac OS X". You can of course see for yourself in the xnu source. The kernel does not do its own decryption and never has. Even the closed source versions from 10.4.4 through 10.4.7 did not do their own binary decryption. Instead the kernel passes it off to a hook function via code in osfmk/kern/page_decrypt.c.

Integrity Data

One curious thing about Amit Singh's page is that the program to print the magic poem does not work on Leopard. Typically it will print nothing. Firing up any program in gdb and examining memory at (-16 * 4096 + 0x1600) typically yields only NUL bytes. So what gives? The answer to this question is staring us right in the face in the open xnu source code. In all 10.4 versions of the source there is a dsmos_page_transform_hook function taking 2 parameters as follows

```
void
dsmos_page_transform_hook(dsmos_page_transform_hook_t hook,
                          void (*commpage_setup_dsmos_blob)(void**, int))
{
#ifdef i386
        /* finish initializing the commpage here */
        (*commpage_setup_dsmos_blob)(dsmos_blobs, dsmos_blob_count);
#endif

        /* set the hook now - new callers will run with it */
        dsmos_hook = hook;
}
```

However, in all 10.5 versions of the xnu source it looks like this:

```
void
dsmos_page_transform_hook(dsmos_page_transform_hook_t hook)
{
/*      printf("%s\n", __FUNCTION__); */
```

```
        /* set the hook now - new callers will run with it */
        dsmos_hook = hook;
}
```

As you can see, Apple removed the second parameter. Both parameters are function pointers. The first is stored in a global variable and the second is called immediately with some parameters. It is the second parameter that is involved in the commpage stuffing. The Tiger kernel code calls it with the parameters dsmos_blobs and dsmos_blob_count. Both globals are external to page_decrypt.c and are actually declared in osfmk/i386/commpage/commpage.c.

The dsmos_blobs variable is simply an array of void pointers and the dsmos_blob_count is simply an integer count of the number of used entries in dsmos_blobs. Also in commpage.c we can see how the kernel fills these in. At the end of commpage_populate_one the code stores an address into the next available slot and increments dsmos_blob_count. On a 32-bit system the eventual result is that dsmos_blob_count will be one and the first entry in dsmos_blobs will point to the kernel's address for the "system integrity data" for 32-bit processes. On a 64-bit system dsmos_blob_count will be two with the first entry pointing to the integrity data for 32-bit processes and the second one pointing to the integrity data for 64-bit processes.

It is clear simply by looking at Apple's open source code and examining the eventual result from a normal user-space program that the integrity data is eventually supposed to be present at these two locations. This is what the commpage_setup_dsmos_blob function pointer parameter does. The kernel immediately calls it and the kernel extension supplying the function pointer immediately stuffs the data into either 1 or 2 commpages.

Or at least that's what's supposed to happen. Funny thing about the hackintoshers. A number of them don't seem to be too bright. About a year ago now (summer of 2007) I decided to analyze the dsmos.kext that was floating around the internet. Sure enough it does not do this right. In fact, what it does is actually rather queer. The correct code would be to iterate over the blob pointers in the array and copy the blob to the memory pointed to by each one. It would look something like this:

```
static void commpage_setup(void **blobs, int count)
{
    for(int i=0; i<count; ++i)
    {
        memcpy(blobs[i], karma, INTEGRITY_BLOB_SIZE);
    }
}
```

Assume for that code that karma is an array of characters containing the magic poem and that INTEGRITY_BLOB_SIZE is 256 because the magic poem is in fact 256 bytes of data. The 256th and last byte (e.g. karma[255]) is a space (hex 0x20). Thus if you were to declare karma as a C string you would actually wind up with 257 bytes of string with the 257th byte (karma[256]) being a 0 byte. That byte is not to be copied to the commpage.

Well, dsmos doesn't exactly do that. Instead what it does is this:

```
static void commpage_setup(void **blobs, int count)
{
    memcpy(blobs[count], karma, 255);
}
```

Case3:08-cv-03251-WHA Document93-1 Filed08/19/09

What!? Yeah, it doesn't make any sense to me either. The biggest mistake is that blobs[count] is clearly accessing the array beyond its bounds since in C its bounds are 0 through count-1. If you looked at the dsmos_blobs definition in commpage.c though you'd note that dsmos_blobs has three members. So what is the third member for? Good question. The kernel itself does not appear to use it. Furthermore, blobs[count] should always be zero because dsmos_blobs is global uninitialized data which would put it in a section that should be cleared to zero in early kernel startup.

The second mistake of course is to copy only 255 instead of 256 bytes. It's possibly just a misunderstanding that the integrity data is truly 256 bytes long and does not include a NUL terminator.

Pondering the magic poem is all well and good but ultimately it's a giant red herring. The magic poem appears to be checked by some Apple binaries. After all, I think that is the whole point of it. That way, even if somebody decrypts the binaries there is still a chance for the code to check for the magic poem in the commpage and start behaving oddly if the integrity data is missing or corrupt.

The conspiracy theorist in me thinks that Apple may have even intentionally released the broken dsmos code just to throw people off. I wonder though why they removed the integrity data from the Leopard releases. All I can figure is that it outlived its usefulness.

Decryption

If the magic poem doesn't have anything to do with the encryption, then what does? Looking back in page_decrypt.c we can see that the first parameter (only parameter on Leopard) to dsmos_page_transform_hook is stored in the dsmos_hook global. That global is initialized to &_dsmos_wait_for_callback which is a function located in the same file. What the function does is poll the value of dsmos_hook until it's something other than itself or NULL.

The reason it does this is that the kernel extension that calls dsmos_page_transform_hook is usually loaded by kextd which is run by launchd. With this scheme protected binaries will simply hang waiting for the decryption engine to be installed. As soon as the engine is installed the processes will resume.

The encryption itself is just basic AES-256. At the end of January 2008 Amit Singh finally posted a new article about the binary protection. This one is titled "TPM DRM" In Mac OS X: A Myth That Won't Die. In it, Singh posts some code which illustrates how one can retrieve the AES-256 keys from a genuine Mac and how one can then use those keys to encrypt his own binaries.

Of course, Singh didn't give up anything that wasn't already pretty well known. What is notable is that he's a respected software engineer who finally published what a lot of people already knew.

To be honest I think the TPM DRM myth persists because of the mystique of it. I mean, who's gonna believe that Apple's super-secret binary protection is really just two AES-256 keys stored in the clear on their hardware and used in a particular manner by regular old code supplied in a kernel extension? It's not like they even tried to hide it. Apple's decryption engine is hiding in plain sight, fully contained within Dont Steal Mac OS X.kext.

Poor-man's Decryption

One of the early methods used in various distributions of OS X was to pre-decrypt the protected binaries. This is surprisingly easy to accomplish. Recall that the decryption process occurrs as the executable is paged into memory by the kernel's execve function. That means that the running image is thus already decrypted. If you simply load a protected binary with GDB and break extremely early in process startup before dyld has a chance to modify the image you can just use GDB commands to dump regions of memory to a file.

You know which regions to dump because the Mach executable header tells you. Once you have dumped those regions you simply overwrite the encrypted portions of the binary with the decrypted portions and remove the encrypted flag from the header. You can do all of this using only gdb, dd, and otool.

Thus using only tools provided to you by Apple and a tiny bit of know-how you can circumvent Apple's binary decryption. Of course now that the specific decryption method is widely known it would be silly to waste time doing this. And it should also be noted that even if you do this you still don't have the integrity data in the commpage which may or may not be required depending on which OS release you are using.

Virtualization

If you are working on running OS X in a virtualizer you'd probably rather provide virtual hardware similar to Apple's real hardware and allow the real Apple decryption engine to do the work instead of supplying your own decryption engine. In this case you simply need to virtualize enough of the Apple SMC chip such that it can return the values for the 'OSK0' and 'OSK1' keys. On the open source front, Alex Graf has written a series of patches for QEMU which do this.

The advantage of this method is that you do not have to supply the key material nor the decryption engine. On the host side simply retrieve OSK0 and OSK1 from the genuine Mac host and on the guest side simply virtualize the SMC to at least return the values for those two keys.

Cat and Mouse

Occasionally I wonder if Apple could do anything to break OS X on non-Mac machines without breaking it on Mac machines. Some people speculate that Apple may begin using the TPM even though they do not now. That is unlikely.

Ultimately any binary protection relies on encrypting the binaries and hoping that it takes people a while to figure out how to decrypt them. Apple could change the encryption method. Perhaps they would use the same keys but something other than AES-256. Perhaps they would rotate the keys by a certain number of bits before passing them into the AES-256 decryption routine. Perhaps they would swap the two keys so the first is used for the second half of the page and the second for the first half of the page.

In the end it's a lot of work for a lot of nothing. And so far the trend is to remove what is known to be broken. For example, the Leopard releases no longer require integrity data in the commpage. It may be the case that by the time Snow Leopard is released the encryption will be gone altogether.

Or it may be the case that Apple leaves the encryption in. The simple fact that you have to do something to break the encryption is enough to convince a large number of people that they'd be better off just buying a Mac from Apple.

Final Thoughts

I think it's safe to say that Apple's "hard work" wasn't "guarded" particularly well. Regardless I suggest you give Apple the courtesy of adhering to the seond part of their message: "Please don't steal!"

My favorites | Sign in

xnu-dev

A fork of the Apple® Darwin XNU kernel for generic x86 PCs

Search projects

Project Home Downloads Wiki Issues Source

New issue | Search Open issues for Search | Advanced search | Search tips

Issue 72: Kernel Panic ACPI releated?

‹ Prev 141 of 144 Next ›
Back to list

2 people starred this issue and may be notified of changes.

Status: New
Owner: ----
Type-Defect
Priority-Medium

Sign in to add a comment

Reported by teeeeeec...@googlemail.com, Nov 27, 2008

```
We are not accepting any more bug reports for sleep/resume problems.
Otherwise continue answering the questions:

Which CPU do you have?
AMD Athlon(tm) 64 Processor 3200+


Which version of Voodoo are you using? (e.g alpha 13 or beta 1)
RC1

What is the problem you are having?
Kernel Panic:
Unable to find driver for this plattform.
ACPI/Users/user/voodoo/xnu-1228.7.5/iokit/kernel/IOPlatformExport.cpp:1411


Did you follow the beta test plan and debug procedure? If yes, specify
which test failed, and a log of your debug procedure:
No

If it works on another kernel, please tell us which :
9.4

The debug procedure should have asked you to attach several files to this
report. Please attach them.

Please provide any additional information below.
```

TC@tcs.local.tar.bz2
91.7 KB Download

Comment 1 by mercurysquad, Nov 27, 2008

```
Try booting with  -f
```

Comment 2 by teeeeeec...@googlemail.com, Nov 27, 2008

```
I always remove extensions.mkext, but tried it anyway.
No success.
This time screenshot attached.
System boots with system.kext 9.4 (but then I cant' start most of the applications)
With system.kext 9.5 -> panic, Picture attached (sorry for the flashlight).
```

CIMG0001.JPG
421 KB Download

Comment 8 by teeeeeec...@googlemail.com, Nov 29, 2008

```
i got it to work using system_kext from here:
http://www.infinitemac.com/showthread.php?t=1361
the one shipped in the update package didn't work for me.
so everything fine! thx for your help!
```

Comment 9 by teeeeeec...@googlemail.com, Nov 30, 2008

```
Q: i am using the kernel without any of those kexts:
"This kernel does not have any in-built decryption, so you will need dsmos.kext or
AppleDecrypt.kext."
So why is it working anyways and what problem might i get in future if i don't have them?
```

Comment 10 by mercurysquad, Nov 30, 2008

```
It could be because you had a pre-patched AMD setup, so your binaries are already decrypted and you don't
need dsmos or AppleDecrypt (yet!).
```

```
I suggest you install one of those kexts to be on the safe side, in case you run an update and no longer have
decrypted versions.
```

Comment 11 by teeeeeec...@googlemail.com, Nov 30, 2008

```
Hi mercurysquad, that makes sense, am just gonna try to understand the whole
thing...have already done so and added dsmos!
thanks for the answer,
tc
```

Sign in to add a comment

©2009 Google - Code Home - Terms of Service - Privacy Policy - Site Directory - Project Hosting Help

Hosted by Google code

pastebin - collaborative debugging tool View Help

Recent Posts

- nask0
  1 min ago
- nask0
  1 min ago
- Lambd
  1 min ago
- Anonymous
  1 min ago
- zef
  2 mins ago
- Anonymous
  3 mins ago
- migi
  4 mins ago
- Anonymous
  5 mins ago
- Anonymous
  6 mins ago
- Anonymous
  6 mins ago
- Make new post

News

For news and feedback see my blog.

Free subdomains

Want your own xyz.pastebin.com sub-domain for your community? Just type the address into your browser address bar. See help for details

About

Pastebin is a tool for collaborative debugging or editing, See help for details.

Credits

Software developed by Paul Dixon

Posted by prasys on Sun 16 Aug 09:22

report abuse | download | new post

```
1. My hardware is pretty much compatible with Mac OS without much
   fuss.. I built this machine specifically for doing Mac OS stuff so
   as to make this easier and more vanilla, but with some hacking you
   should be able to apply this method to any Leopard/Snow Leopard
   installation.. it's the concepts that are important to understand,
   not the exact steps.
2.
3. What you need
4.
5. A computer that can run Mac OS:
6.
7. Something with an Intel chipset from the past year or two is
   ideal.
8. Intel Core 2 Duo processor of some sort
9. BIOS set to AHCI for your SATA
10. SATA or USB hard drive and SATA cd/dvd drive
11. USB keyboard/mouse (as this is what Macs have)
12. Fairly recent NVidia or ATI video card
13.
14. Here is what I used:
15. Gigabyte EP45T-UD3P motherboard
```

Intel E8600 Processor
NVidia 7950 GT PCI Express graphics card
Microsoft USB wired keyboard
Microsoft USB wired optical mouse
8GB OCZ DDR3 Memory
1.5TB SATA hard disk (this MUST be partitioned into smaller chunks; the bootloader will not work with a single 1.5TB partition)

About DSDT

The first challenge is booting. In order to successfully boot you need a cool bootloader like PC EFI v9 or Chameleon 2.0 RC1.
My computer worked fine in Leopard (10.5) but I couldn't get things to boot right in 10.6 until I created a DSDT patch.

The DSDT is part of the ACPI stuff which is part of the BIOS.. basically it's like a program that tells the operating system what hardware the computer has and how to access it.
It is not the only way the operating system discovers hardware but it is one of the ways and Mac OS expects certain devices to look a certain way. Computers are all different and made by different companies and people so there's a lot of variation in the general PC market, but Apple's obviously only got 4-5 computers they manufacture so they aren't writing their code to account for every variation.

Anyway, you will likely need to fix your DSDT. The nice thing about these great bootloaders we have available for OSX86 is that they not only emulate EFI for us, but they allow manipulating the DSDT by creating a patch file and putting it in the right place for the bootloader to find. The bootloader will replace the in-memory DSDT table with the one we provide before executing the operating system, so we have a chance to tweak things and make them look the way Mac OS wants.

The first tool you need for this is DSDT Patcher. Read the thread here or Google for it.
This includes iasl which is what you need to compile/decompile the DSDT, and includes a tool to make some fixes for you automatically. This is good to get started with, but you should get familiar with what's going on, how to decompile by hand and how to compile by hand as you will probably want to tweak it further.

The other tool I found useful is ACPI Patcher. Read the thread here to learn more about it.

You need some way to boot into Leopard to use some of this stuff, but if you're creative enough you can get your DSDT other ways (Linux, Windows)
Leopard is also a good way to test your DSDT prior to trying it with Snow Leopard. Most things are fixable through the DSDT if your computer is compatible with Mac OS already.

I got the sound on my motherboard (alc889a) to work without a modified kext by editing the DSDT.

Here is my DSDT compiled: EP45T-UD3P-DSDT.aml
And here it is decompiled: EP45T-UD3P-DSDT.dsl

Hint: to decompile an .aml file, use:

```
iasl -d /Path/DSDT.aml
```

to compile the .aml file from a .dsl file, use:

```
iasl -ta /Path/DSDT.dsl
```

Notable things to look at are the HDEF (just search the dsl file) - this makes my AppleHDA work with no modification, and the HPET which makes the AppleIntelCPUPowerManagement.kext load without kernel panic (but it is still preferred to block it, more about this later)

BIOS setup

This is for my motherboard, but the same ideas should apply for any others.

NOTE: right now each time I boot snow leopard I lose my BIOS settings. It's annoying but one way to make it a little less annoying is to save your settings in a profile inside the BIOS (F11) and after you reboot go back into the BIOS and load it with F12

UPDATE: CMOS reset fixed thanks to Stellarola. Make your RTC device look like this in your DSDT. It might be called RTC0, doesn't matter..

```
                Device (RTC)
                {
                    Name (_HID, EisaId ("PNP0B00"))
                    Name (_CRS, ResourceTemplate()
                    {
                        IO (Decode16, 0x0070, 0x0070, 0x00, 0x02)
                    })
                }
```

I have 6 yellow SATA ports that are from the Intel ICH10 and a few purple SATA ports + IDE port which are added by Gigabyte (I think it's JMicron). You need to turn off the 'Onboard' controller in the BIOS. It will disable the purple ports + IDE port.

Make sure you set the SATA mode to 'AHCI' not raid or IDE.

Enable the HPET and put in 64 bit mode (in Power management settings)

Disable the UART serial ports and the parallel port.

Set onboard azalia audio to Auto so you get sound.

Enable the gigabit LAN ports.

Enable firewire/USB 2.0

Files for installation

You are going to need Snow Leopard. The one that was given out at WWDC was build 10A380. As of this writing the most current is 10A394 but it is only distributed through Software Update. So basically start with an actual install disc (or dmg) of whatever the latest Snow Leopard is.

At the minimum you need a decrypt kext. The popular ones floating around are dsmos.kext and AppleDecrypt.kext. Note that you can use old ones from Leopard but one of the big features of Snow Leopard is the 64 bit kernel, which means you'll need new kext which include both a 32 bit and a 64 bit version in the binary, if you want to be able to boot in 64 bit mode.

I have collected the following kexts which in my opinion make for a perfect install:

dsmos.kext_for_snow_64.tar
NullCPUPowerManagement.kext_for_snow_64.tar
OpenHaltRestart.kext_for_snow_64.tar
PlatformUUID.kext_for_snow_64.tar

dsmos - required to decrypt encrypted binaries
NullCPUPowerManagement - attaches in place of the real AppleIntelCPUPowerManagement which doesn't work right on PCs (kernel panic, cpu running hot)
OpenHaltRestart - very simple kext that makes reboot and shutdown work on PCs
PlatformUUID - sets the platform UUID so that you have a uniform UUID from the very start of the boot.. this UUID is used in your preferences files (among other things) and I think it is important for a clean install to have this right from the first boot.

Bootloader files:

These are for the first boot:
PCefi_v9_Installer_Final 2.dmg
boot - snowboot file (goes in /boot)

This is what we want to end up using in the end:
Chameleon-2.0-r431.pkg
smbios.plist - modify this to your tastes

You also need an EFI string for your video card. Search the

forums, this is pretty easy to come by.

Installation

The easiest way to do this is by using an already working system. If you have a working Intel Mac with Snow Leopard running already, that's ideal. Basically the idea is to boot the working system, connect the target drive to that computer, and use the OSInstall.mpkg to install. There are lots of guides about how to do this on Leopard already but it boils down to this:

1. Connect the target drive.

2. Use Disk Utility to partition in using GPT - GUID Partition Table
Important: Right click on the new volume on your desktop, Get Info, Click the little lock icon in the bottom right corner to authenticate and UNCHECK 'Ignore ownership on this volume' or else the permissions will be all screwed up after you install.

3. Make sure that the new volume's root directory is owned by the root user.
sudo chown 0:0 /Volumes/Snow # substitute whatever you called your volume in place of Snow

4. Mount the Mac OS X Snow Leopard DVD (or image)

5. In terminal:
open /Volumes/Mac\ OS\ X\ Install\ DVD/System/Installation/Packages/OSInstall.mpkg

If you don't have a working Snow Leopard system to do it from, you can use Leopard maybe, but I had problems with the install failing at the end under Leopard.. It should certainly be doable but might require a little more effort.

First boot

You should have ended up with a stock, never booted system on your target drive.
Now to make it bootable..
The reason we're doing this 2-phase boot is because Chameleon 2.0 RC1 cannot yet make the .mkext archive for Snow Leopard on its own, so we need to boot with something else the first time.

NOTE: Here is my Extensions.mkext that works for me.. If this works for you, then you can skip this first boot step entirely and go right to chameleon, but if you need other kexts besides the 4 I'm using here then you'll have to make it yourself.
Extensions.mkext

NOTE: You can actually make the mkext in Leopard but just make sure you're specifying the proper directories like /Volumes/Snow Leopard/System/Library/Extensions not /System/Library/Extensions

1. Install PC EFI v9 on the target disk.
2. Replace the boot file in the root of the target volume with the snow boot file I linked to above.
3. Copy the 4 kexts I linked above into /System/Library/Extensions on the target volume (make sure you're not doing this to your working system.. /Volumes/Whatever/System/Library/Extensions)
4. Copy the DSDT.aml you created to the root of the new volume
5. Edit the UUID in the PlatformUUID.kext's Info.plist. Make it the same as the UUID in the smbios.plist file (which we aren't using yet), but will when we get chameleon installed. You can just generate a new UUID or use the one I generated in the smbios.plist. Google around if you don't know how to generate one.
6. Edit your /Library/Preferences/SystemConfiguration/com.apple.Boot.plist and add your EFI string. For me it looked like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
        <key>Kernel</key>
        <string>mach_kernel</string>
        <key>Kernel Flags</key>
        <string></string>
```

```
169.         <key>device-properties</key>
170.
     <string>b2020000010000000100000a602000010000000002010c00d041030a0100000001010600000101010600000007fff04000e0000004e00560050004d
171. </dict>
172. </plist>
173.
174.
175.
176.
177. Unmount the volume and boot it up on your real system. Use -v in
     the bootloader so you can see the messages as it boots. Add -x64
     to boot the 64 bit kernel instead of the 32 bit. This only applies
     to the first bootloader, not to Chameleon, which we will install
     after. At this point you should be seeing the Welcome video if
     your graphics string is right. You might even hear sound if your
     HDEF device is working. The only thing wrong with your system at
     this point is that your smbios will be messed up. When you go
     through the registration process your system will have a weird
     name, whatever it gets from smbios, instead of 'MacBook' or 'Mac
     Pro' or whatever. We will fix this later.
178.
179. Go through the registration to create your user account, etc.
180.
181. UPDATE: As an alternative to registering with the PC EFI v9 boot
     (with no smbios stuff) you can just boot single user mode and
     instead of keeping your Extensions.mkext in /Extra you can just go
     ahead and put it where it really belongs. This simplifies things a
     great deal and this is what I recommend now. If you do this you
     can just move on to installing Chameleon after you do the
     kextcache thing in single user mode, as it will then be able to
     boot with Chameleon. As an alternative, skip the EFI v9 thing and
     use EFI v10. It's the concepts that are important here not the
     exact steps, remember that!
182.
183. Boot in single user mode by using the boot parameter -s
184.
185. Use the following commands:
186. mount -uw /
187. kextcache -v 1 -t -l -m
     /System/Library/Caches/com.apple.kext.caches/Startup/Extensions.mkext
     /System/Library/Extensions /Extra/AdditionalExtensions # remove
     /Extra/AdditionalExtensions if you don't keep any under there
188.
189.
190. Permanent boot
191.
192. Now, we want to install Chameleon. The only catch is that
     Chameleon 2.0 RC1 needs you to create an Extensions.mkext for it
     (the -f thing doesn't work yet with it for snow leopard). So,
     install the Chameleon package. If you choose the Chameleon EFI it
     will read the com.apple.Boot.plist from your hidden EFI partition
     instead of from the normal partition. If you don't know how to do
     this then just use the regular one. Don't install the extra kexts,
     but you can install the themes.
193.
194. 1. Now, remove anything it put in /Extra/Extensions. Just remove
     the whole directory.
195. 2. Create /Extra/AdditionalExtensions (or whatever you want to
     call it, just not Extensions, because chameleon can't yet process
     them into a mkext on its own)
196. sudo mkdir /Extra/AdditionalExtensions
197.
198. 3. Move the 4 kexts you installed in /System/Library/Extensions to
     your /Extra/AdditionalExtesions directory
199. sudo mv /System/Library/Extensions/dsmos.kext
     /Extra/AdditionalExtensions/
200. sudo mv /System/Library/Extensions/NullCPUPowerManagement.kext
     /Extra/AdditionalExtensions/
201. sudo mv /System/Library/Extensions/PlatformUUID.kext
     /Extra/AdditionalExtensions/
202. sudo mv /System/Library/Extensions/OpenHaltRestart.kext
     /Extra/AdditionalExtensions/
203.
204. 4. Make sure your filesystem root is owned by the root user
205. sudo chown 0:0 /
206.
207. 5. Remove the caches if they exist
208. sudo rm -fr /System/Library/Caches
209.
210. 6. Create the Extensions.mkext for Chameleon
211. sudo kextcache -v 1 -t -l -m /Extra/Extensions.mkext
     /Extra/AdditionalExtensions /System/Library/Extensions
212.
```

```
213. Here is a script you can keep in /Extra to help you with this
     later.. just rename it to .sh (take off the .txt) and chmod 755
     it:
214. make-mkext.sh
215. 7. Put your smbios.plist into /Extra
216. 8. You can also move the DSDT.aml into /Extra (Chameleon will read
     it from either place)
217. 9. Copy your
     /Library/Preferences/SystemConfiguration/com.apple.Boot.plist to
     /Extra (edit if needed too, you can add Timeout, etc)
218. sudo cp
     /Library/Preferences/SystemConfiguration/com.apple.Boot.plist
     /Extra/
219.
220.
221. Hopefully it'll work! At this point you should be able to have a
     perfect system.
222.
223. When booting with chameleon you can select which kernel to boot by
     using the parameter arch=i386 for 32 bit or arch=x86_64 for 64
     bit. It will boot the 64 bit kernel by default if your system is
     capable. Don't use -f.
224.
225. To fix your machine's name, go to System Preferences -> Sharing
     and change it to Soandso's Mac Pro or whatever
226.
227. Turn off the sleep in Energy saver settings as it causes problems
     for a lot of people (it might not wake up).
228.
229. -solar
230.
231. solar@heliacal.net
```

Submit a correction or amendment below (click here to make a fresh posting)
After submitting an amendment, you'll be able to view the differences between the old and new posts easily.

Syntax highlighting: None

To highlight particular lines, prefix each line with @@

My hardware is pretty much compatible with Mac OS without much fuss.. I built this machine specifically for doing Mac OS stuff so as to make this easier and more vanilla, but with some hacking you should be able to apply this method to any Leopard/Snow Leopard installation.. it's the concepts that are important to understand, not the exact steps.

What you need

A computer that can run Mac OS:

Something with an Intel chipset from the past year or two is ideal.
Intel Core 2 Duo processor of some sort
BIOS set to AHCI for your SATA
SATA or USB hard drive and SATA cd/dvd drive
USB keyboard/mouse (as this is what Macs have)
Fairly recent NVidia or ATI video card

Here is what I used:
Gigabyte EP45T-UD3P motherboard
Intel E8600 Processor
NVidia 7950 GT PCI Express graphics card
Microsoft USB wired keyboard
Microsoft USB wired optical mouse
8GB OCZ DDR3 Memory
1.5TB SATA hard disk (this MUST be partitioned into smaller chunks; the bootloader will not work with a single 1.5TB partition)

About DSDT

The first challenge is booting. In order to successfully boot you need a cool bootloader like PC EFI v9 or Chameleon 2.0 RC1.
My computer worked fine in Leopard (10.5) but I couldn't get things to boot right in 10.6 until I created a DSDT patch.

The DSDT is part of the ACPI stuff which is part of the BIOS.. basically it's like a program that tells the operating system what hardware the computer has and how to access it.
It is not the only way the operating system discovers hardware but it is one of the ways and Mac OS expects certain devices to look a certain way. Computers are all different and made by different companies and people so there's a lot of variation in the general PC market, but Apple's obviously only got 4-5 computers they manufacture so they aren't writing their code to account for every variation.

Your Name

Send

Remember me so that I can delete my post
How long should your post be retained?
a day a month forever
Good for email conversations / temporary data