iphone - Binary Stripping in a Nutshell - Stack Overflow

There are actually two kinds of stripping.

Stripping a binary removes the symbols which are not needed from your binary's symbol table. The symbol table contains the name of every object in your program as well as debugging information. This is not necessary for your program to run, since programs do not ordinarily access their own functions through the symbol table.

The symbol table provides useful debugging information when your program crashes, so you can get a backtrace showing which functions were being called when the program crashed. If you strip the symbols, the backtrace will only contain memory addresses but no function names. You should never strip an application that you are debugging.

Stripping a binary also makes it slightly harder to reverse engineer an executable, but I am hoping you are not that paranoid.

Stripping a binary does not make your program load faster. The symbol table will not be loaded into memory unless it is needed for a backtrace if your program crashes.

You can strip a binary from the command line using the `strip` command. I am not quite sure how to trigger this from Xcode.

```
strip MyExecutable
```

Dead stripping is a different process which removes unused functions and data from your code. This happens at the link stage, when the binary is created. This may reduce the size of your code. The difference will depend

on how much unused data is in your program; it may be a lot and it may be nothing at all. Dead stripping can make your application very slightly faster since it increases the locality of the hot code. (If it makes a big difference, there is something seriously wrong with your application.)

There are no general drawbacks to dead stripping, so I would always turn it on. You can enable dead stripping through Xcode, it corresponds to the linker flag `-dead_strip`.

```
gcc -o MyExecutable -Wl,-dead_strip ....
```

Note: Dead stripping works by putting each function in a separate subsection. If you write assembly files, you can use the `.subsections_via_symbols` directive to put each symbol in a separate subsection, allowing individual symbols to be dead stripped by the linker. GCC always produces that directive in its assembly output, as far as I can tell.

References:

https://developer.apple.com/library/mac/#documentation/General/Conceptual/MOSXAppProgrammingGuide/Performance/Performance.html