Blocking Code Injection on iOS and OS X

Yesterday I posted (twitter) a set of linker flags that can be set that will block types of code injection on iOS and OS X that came from a little known check inside the dynamic linker. This is an explanation as to how and why those flags work and what they do.

###Background The dynamic linker (dyld) is the process that loads and runs binaries on OS X and iOS. This process also has some very special environment variables that can modify the normal behavior of it (You can check out the whole list here: man-page or web). One commonly used environment variable is `DYLD_INSERT_LIBRARIES`:

```
DYLD_INSERT_LIBRARIES
    This is a colon separated list of dynamic libraries to load before
    program. This lets you test new modules of existing dynamic shared
    flat-namespace images by loading a temporary dynamic shared library
    Note that this has no effect on images built a two-level namespace
    shared library unless DYLD_FORCE_FLAT_NAMESPACE is also used.
```

This is commonly used to inject dylibs into applications that modify behavior or patch specific functionality. This is how the vast majority of modifications on existing applications are run on jailbroken devices. However it also has some more mundane uses, such as for injecting code while performing analysis and debugging when in Xcode.

When an application is launched the binary is run through dyld and that processes the binary file. This finds what libraries it needs to load and link against to generate a complete symbol table. Doing this requires parsing through the binary header, while it does this it can trigger flags in dyld based on what segments are present in the binary. There is a special flag that will be set for binaries that are marked as "restricted". This special flag means that the dynamic linker should ignore any set environment variables.

###Stopping dyld from Loading Code There are three ways to flag a binary as "restricted" to the dynamic linker.

1. **Set restricted status by entitlements**

This option is only available to applications on OS X with special entitlements.

2. **setuid and setgid**

   Any application that makes these two calls are going to be marked as restricted by the linker as a security measure.

3. **Restricted Segment of Header**

   The final way to mark a binary as restricted is by telling the linker to add new section to the binary header that is named "__RESTRICT" and has a section named "__restrict" when you compile it. This can be done in Xcode by adding the following flags into your "Other Linker Flags"

   ```
   -Wl,-sectcreate,__RESTRICT,__restrict,/dev/null
   ```

   This segment type is not mentioned anywhere on Apple's documentation for the Mach-O ABI. Google results for how it works are also very sparse. The only place that this can be found documented is actually in the source code for dyld.

###Notes * If Apple ever removes the checks for this type of segment in the binary header you aren't going to be causing problems to your app.

- **This should only be added to build configurations that you plan to distribute the resulting binary**. Marking debug builds as restricted can cause problems when you go to debug using Instruments, guard malloc, and many third party debugging tools that use library injection.
- The flags listed above generate an empty section (size zero) in the binary, if you wish to validate your own binaries then you can specify a file name instead of "/dev/null" and it will store that file in the binary's header. Adding your own file there can be useful if you plan on validating that your binary is correctly signed and not modified.
