Resolving Code Signing Crashes on Launch

This thread has been locked by a moderator.

440

This post is part of a cluster of posts related to the trusted execution system. If you found your way here directly, I recommend that you start at the top.

Share and Enjoy
—
Quinn “The Eskimo!” @ Developer Technical Support @ Apple
`let myEmail = "eskimo" + "1" + "@" + "apple.com"`

Resolving Code Signing Crashes on Launch

A code signing crash has the following characteristic information:

```
Exception Type:  EXC_CRASH (SIGKILL (Code Signature Invalid))
```

IMPORTANT Most developers never see a code signing crash because they use Xcode to build and sign their product. Xcode’s code signing infrastructure detects problems that could cause a code signing crash, and its automatic code signing fixes them for you! If you’re having problems with code signing crashes and you can use Xcode but aren’t, consider making the switch Xcode.

The most common code signing crash is a crash on launch. To confirm that, look at the thread backtraces:

```
Backtrace not available
```

If you see valid thread backtraces this is not a crash on launch. Go back to Resolving Trusted Execution Problems and read through the *Code Signing Crashes After Launch* section.

If you see no thread backtraces, your code didn’t run at all. The trusted execution system has blocked it. In most cases there is some evidence of the problem in the system log. For example:

```
type: error
time: 2022-05-19 06:29:17.640331 -0700
process: taskgated-helper
subsystem: com.apple.ManagedClient
category: ProvisioningProfiles
message: com.example.apple-samplecode.OverClaim: Unsatisfied entitlements: com.apple.overclaim
```

This indicates that the OverClaim app, with bundle ID `com.example.apple-samplecode.OverClaim`, claimed a restricted entitlement, `com.apple.overclaim`, that wasn’t authorised by a provisioning profile.

For more information about provisioning profiles, see TN3125 Inside Code Signing: Provisioning Profiles. Specifically, the Entitlements on macOS section discusses the concept of restricted entitlements. For general information about the system log, see Your Friend the System Log.

Normalise the Entitlements Property List

Entitlement property list files look like text and so it’s tempting to edit them with a text editor. This can lead to all sorts of problems. If you have code whose entitlements property list contains comments, non-Unix line endings, or other weird formatting, the trusted execution system may block it. To avoid such problems, normalise your entitlements property list before passing it to `codesign`. For example:

```
% plutil -convert xml1 MyApp.plist
% codesign -s III --entitlements MyApp.plist MyApp.app
```

Problems like this typically show up on older systems. Modern systems use DER-encoded entitlements, as discussed in The future is DER section of TN3125.

A related gotcha is line breaks. Consider this entitlements property list file:

```
% cat MyApp.plist
…
<plist version="1.0">
<dict>
    <key>
com.apple.security.cs.disable-library-validation</key>
    <true/>
</dict>
</plist>
```

This is a valid property list but it doesn’t do what you think it does. It looks like it claims the `com.apple.security.cs.disable-library-validation` entitlement but in reality it claims `\ncom.apple.security.cs.disable-library-validation`. The system treats the latter as a restricted entitlement and thus requires it to be authorised by a profile. Of course no such profile will authorise that entitlement, and so the app is blocked by the trusted execution system.

Similarly, consider this:

```
% cat MyApp.plist
…
<plist version="1.0">
<dict>
    <key> com.apple.security.cs.disable-library-validation</key>
    <true/>
</dict>
</plist>
```

This claims `com.apple.security.cs.disable-library-validation `, note the leading space, and that’s also blocked by the trusted execution system.

Check for Unauthorised Entitlements

Sometimes the system log may not make it obvious what’s gone wrong. It may be easier to work this out by looking at the built program. The most common cause of problems like this is the app claiming a restricted entitlement that’s not authorised by a provisioning profile.

To start your investigation, dump the entitlements to check for restricted entitlements:

```
% codesign -d --entitlements - "OverClaim.app"
…/OverClaim.app/Contents/MacOS/OverClaim
[Dict]
    [Key] com.apple.application-identifier
    [Value]
        [String] 5A9ME9E2Y8.com.example.apple-samplecode.OverClaim
    [Key] com.apple.developer.team-identifier
    [Value]
        [String] 5A9ME9E2Y8
    [Key] com.apple.overclaim
    [Value]
        [Bool] true
    [Key] com.apple.security.get-task-allow
    [Value]
        [Bool] true
```

In this case all the entitlements except `com.apple.security.get-task-allow` are restricted.

Note If there are no restricted entitlements, something else has gone wrong. Go back to Resolving Trusted Execution Problems and look for other potential causes.

Now check that the provisioning profile was embedded correctly and extract its payload:

```
% ls -l "OverClaim.app/Contents/embedded.provisionprofile"
…  OverClaim.app/Contents/embedded.provisionprofile
% security cms -D -i "OverClaim.app/Contents/embedded.provisionprofile" -o "OverClaim-payload.plist"
```

Check that the profile applies to this app by dumping the `com.apple.application-identifier` entitlement authorised by the profile:

```
% /usr/libexec/PlistBuddy -c "print :Entitlements:com.apple.application-identifier" OverClaim-payload.plist
5A9ME9E2Y8.com.example.apple-samplecode.*
```

This should match the `com.apple.application-identifier` entitlement claimed by the app.

Repeat this for all the remaining restricted entitlements:

```
% /usr/libexec/PlistBuddy -c "print :Entitlements:com.apple.developer.team-identifier" OverClaim-payload.plist
5A9ME9E2Y8
% /usr/libexec/PlistBuddy -c "print :Entitlements:com.apple.overclaim" OverClaim-payload.plist
Print: Entry, ":Entitlements:com.apple.overclaim", Does Not Exist
```

In this example the problem is the `com.apple.overclaim` entitlement, which is claimed by the app but not authorised by the profile. If that’s the case for your program, you have two choices:

- If you program doesn’t need this entitlement, update your code signing to not claim it.
- If you program relies on this entitlement, update your profile to authorise it.

The entitlement allowlist in the profile is built by the Apple Developer website based on the capabilities enabled on your App ID. To change this allowlist, modify your App ID capabilities and rebuild your profile. Some capabilities are only available on some platforms and, within that platform, for some distribution channels. For these details for macOS, see Developer Account Help > Reference > Supported capabilities (macOS). Some capabilities require review and approval by Apple. For more on this, see Developer Account Help > Reference > Provisioning with managed capabilities.

Check the Signing Certificate

If your program’s entitlements look good, the next most likely problem is that your program was signed by a signing identity whose certificate is not authorised by the profile. To debug this, first extract the certificate chain from your program:

```
% codesign -d --extract-certificates=signed-with- "OverClaim.app"
…
% for i in signed-with-* ; do mv "${i}" "${i}.cer" ; done
```

The first certificate is the one that matters:

```
% certtool d "signed-with-0.cer"
Serial Number      : 53 D8 68 CC B5 32 83 DE 72 D9 6A C9 BF 84 78 25
…
Subject Name       :
    Other name     : UT37G6R4Z9
    Common Name    : Apple Development: Quinn Quinn (7XFU7D5254)
    OrgUnit        : 5A9ME9E2Y8
    Org            : Quinn Quinn
    Country        : US
…
```

Now check this against each of the certificates authorised by the profile. Start by extracting the first one:

```
% plutil -extract DeveloperCertificates.0 raw -o - OverClaim-payload.plist | base64 -D > "authorised0.cer"
% certtool d "authorised0.cer"
Serial Number      : 46 AB EF 2C 52 54 DE FD D1 76 9D 3A 41 7C 9E 43
…
Subject Name       :
    Other name     : UT37G6R4Z9
    Common Name    : Mac Developer: Quinn Quinn (7XFU7D5254)
    OrgUnit        : 5A9ME9E2Y8
    Org            : Quinn Quinn
    Country        : US
…
```

That’s not a match. So try the next one:

```
% plutil -extract DeveloperCertificates.1 raw -o - OverClaim-payload.plist | base64 -D > authorised1.cer
% certtool d "authorised1.cer"
Serial Number      : 53 D8 68 CC B5 32 83 DE 72 D9 6A C9 BF 84 78 25
…
Subject Name       :
    Other name     : UT37G6R4Z9
    Common Name    : Apple Development: Quinn Quinn (7XFU7D5254)
    OrgUnit        : 5A9ME9E2Y8
    Org            : Quinn Quinn
    Country        : US
…
```

This matches, which means the profile applies to this code.

IMPORTANT When checking for a match, look at the `Serial Number` field. Don’t just rely on the `Common Name` field. A common mistake is to have two signing identities whose certificates have identical common names but the profile only lists one of them.

If you get to the end of the list of certificate list in the profile and don’t find the certificate that the program was signed with, you know what the problem is: Your program is signed with a signing identity whose certificate is not listed in its profile. To fix this, either:

- Reconfigure your code signing to use a signing identity whose certificate is listed.
- Or update the profile to include the certificate of the signing identity you’re using.

Check for Expiration

If your certificates aren’t the problem, check that nothing has expired. Start with the certificate from the app’s signature:

```
% certtool d "signed-with-0.cer"
Serial Number      : 53 D8 68 CC B5 32 83 DE 72 D9 6A C9 BF 84 78 25
…
Not Before         : 10:52:56 Apr 21, 2022
Not After          : 10:52:55 Apr 21, 2023
…
```

Also check the expiry date on the profile:

```
% plutil -extract ExpirationDate raw -o - OverClaim-payload.plist
2023-04-21T11:02:58Z
```

If either has expired, update it and re-sign your product.

IMPORTANT Developer ID-signed code and installers include a secure timestamp. When the system checks the expiry date on a Developer ID certificate, it only checks that the certificate was valid at the time that the code was signed, base on that secure timestamp. Thus, an old Developer ID-signed app will continue to run after it’s certificate has expired.

Check the Supported Devices

If everything else checks out, the last thing to check is that the profile authorises the code to run on this machine. There are two cases here:

- Developer ID profiles authorise the code on all machines.
- Other profiles authorise the code on a specific list of machines.

If you think you have a Developer ID profile, confirm that by looking for the `ProvisionsAllDevices` property:

```
% plutil -extract "ProvisionsAllDevices" xml1 -o - "OverClaim-payload.plist"
…No value at that key path or invalid key path: ProvisionsAllDevices
```

If that’s not the case, get the `ProvisionedDevices` property and verify that the current machine’s provisioning UDID is listed there:

```
% plutil -extract "ProvisionedDevices" xml1 -o - "OverClaim-payload.plist"
…
<array>
    …
    <string>A545CA26-B8D7-5B38-A98C-53DA798BE342</string>
    …
</array>
</plist>
% system_profiler SPHardwareDataType
…
      Provisioning UDID: A545CA26-B8D7-5B38-A98C-53DA798BE342
…
```

If you get to the end any everything looks OK, your provisioning profile is not the cause of this crash. Return to Resolving Trusted Execution Problems for more suggestions.

Revision History

- 2022-06-08 Added the *Normalise the Entitlements Property List* section.
- 2022-05-20 First posted.

Gatekeeper Code Signing Notarization

Reply

Posted 4 months ago by eskimo

Add a Comment