#WWDC19

Advances in macOS Security

Garrett Jacobson, Security Engineering and Architecture
Kelly Yancey, Security Engineering and Architecture

Defense in depth

Gatekeeper

User privacy protection

Defense in Depth

Defense in Depth

No single technology or feature can deliver perfect security

Defense in Depth

No single technology or feature can deliver perfect security

macOS is designed with many layers of security

Defense in Depth

No single technology or feature can deliver perfect security

macOS is designed with many layers of security

Continuously improve the technologies and policies at each layer

Defense in Depth

Defense in Depth

One layer failing shouldn’t defeat all security

Defense in Depth

One layer failing shouldn’t defeat all security

Rely on multiple layers of protection, with different properties

- Delay the advance of an attacker
- Reduce the attack surface
- Create choke points that are easier to defend

Defense in Depth

Defense in Depth

Gatekeeper

Defense in Depth

Gatekeeper

User Privacy Protection

Gatekeeper

Protect users from
running malicious software

Gatekeeper

macOS Mojave

Gatekeeper

macOS Mojave

What does Gatekeeper check?

Gatekeeper

macOS Mojave

What does Gatekeeper check?

- Does it contain known malicious content?

Gatekeeper

macOS Mojave

What does Gatekeeper check?

- Does it contain known malicious content?
- Has it been tampered with?

Gatekeeper

macOS Mojave

What does Gatekeeper check?

- Does it contain known malicious content?
- Has it been tampered with?
- Does it meet the security policy?

Gatekeeper

macOS Mojave

What does Gatekeeper check?

- Does it contain known malicious content?
- Has it been tampered with?
- Does it meet the security policy?
- Does the user want to run it?

Gatekeeper

macOS Mojave

What does Gatekeeper check?

- Does it contain known malicious content?
- Has it been tampered with?
- Does it meet the security policy?
- Does the user want to run it?

When does Gatekeeper check?

Gatekeeper

macOS Mojave

What does Gatekeeper check?

- Does it contain known malicious content?
- Has it been tampered with?
- Does it meet the security policy?
- Does the user want to run it?

When does Gatekeeper check?

- **First launch** of **quarantined** apps launched via **LaunchServices**

Quarantine

Quarantine

Marks files that arrive on the system from a variety of external sources

Quarantine

Marks files that arrive on the system from a variety of external sources

Adds metadata about the source

Quarantine

Marks files that arrive on the system from a variety of external sources

Adds metadata about the source

Apps can opt-in to quarantining files

Quarantine

Marks files that arrive on the system from a variety of external sources

Adds metadata about the source

Apps can opt-in to quarantining files

Default for files written by App Sandboxed apps

Launch Services

Launch Services

Framework for finding and
launching applications

Launch Services

Framework for finding and
launching applications

Responsible for many common ways
to start apps

- Opening in Finder/Dock
- NSWorkspace
- Apps opened via document handlers or URLs

Launch Services

Framework for finding and launching applications

Responsible for many common ways to start apps

- Opening in Finder/Dock
- NSWorkspace
- Apps opened via document handlers or URLs

Apps via Launch Services

Launch Services

Launch Services

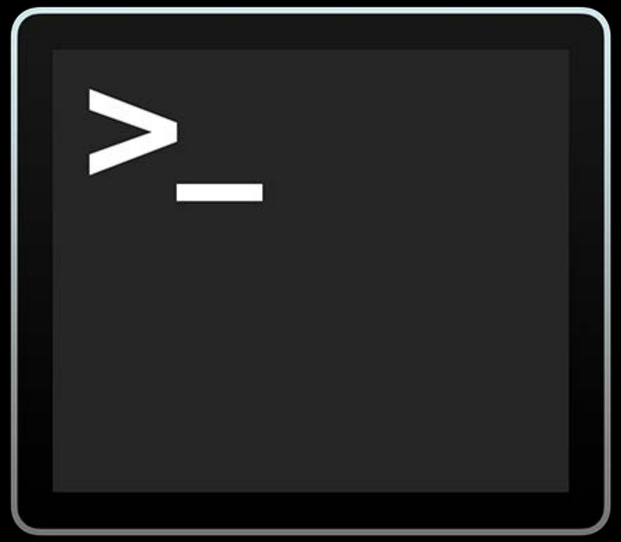
Not involved in other methods of loading code

- NSTask
- exec / posix_spawn
- NSBundle / dlopen

Launch Services

Not involved in other methods of loading code

- NSTask
- exec / posix_spawn
- NSBundle / dlopen



Gatekeeper

macOS Mojave

First use, quarantined

Malicious content scan	No known malicious content
Signature check	No tampering
Local policy check	Must be signed with Developer ID certificate
First launch prompt	User must approve

Gatekeeper

macOS Mojave 10.14.5

First use, quarantined

Malicious content scan	No known malicious content
Signature check	No tampering
Local policy check	Must be signed with Developer ID certificate. New Mac developers’ software requires notarization.
First launch prompt	User must approve

Gatekeeper

macOS Catalina

NEW

First use, quarantined

Malicious content scan	No known malicious content
Signature check	No tampering
Local policy check	All new software requires notarization
First launch prompt	User must approve

Gatekeeper
macOS Catalina

NEW

	 First use, quarantined	 First use, quarantined
Malicious content scan	No known malicious content	No known malicious content
Signature check	No tampering	No tampering
Local policy check	All new software requires notarization	All new software requires notarization
First launch prompt	User must approve	Users must approve software in bundles

Gatekeeper

macOS Catalina

NEW

	 First use, quarantined	 First use, quarantined	 Non-quarantined
Malicious content scan	No known malicious content	No known malicious content	No known malicious content
Signature check	No tampering	No tampering	–
Local policy check	All new software requires notarization	All new software requires notarization	–
First launch prompt	User must approve	Users must approve software in bundles	–

You can always choose to run any software on your system

Gatekeeper

The road ahead

Gatekeeper

The road ahead

Platform security is increasingly reliant on the constant validity of code signatures

Gatekeeper

The road ahead

Platform security is increasingly reliant on the constant validity of code signatures

If an app has no signature

Gatekeeper

The road ahead

Platform security is increasingly reliant on the constant validity of code signatures

If an app has no signature

- It’s impossible to detect tampering

Gatekeeper

The road ahead

Platform security is increasingly reliant on the constant validity of code signatures

If an app has no signature

- It’s impossible to detect tampering

If a bundle signature is broken

Gatekeeper

The road ahead

Platform security is increasingly reliant on the constant validity of code signatures

If an app has no signature

- It’s impossible to detect tampering

If a bundle signature is broken

- It’s very hard to differentiate malicious from mundane

Gatekeeper

The road ahead

Platform security is increasingly reliant on the constant validity of code signatures

If an app has no signature

- It’s impossible to detect tampering

If a bundle signature is broken

- It’s very hard to differentiate malicious from mundane

In a future version of macOS, unsigned code will not run by default

We Need Your Help

We Need Your Help

Sign and notarize all software you distribute

- Even if it doesn’t get quarantined

We Need Your Help

Sign and notarize all software you distribute

- Even if it doesn’t get quarantined

Don’t modify signed applications or bundles

We Need Your Help

Sign and notarize all software you distribute

- Even if it doesn’t get quarantined

Don’t modify signed applications or bundles

Loading code can fail

- Ensure your apps handle failures gracefully

User Privacy Protections

Kelly Yancey, Security Engineering and Architecture

User Privacy Protections

Recording capabilities

Files and folders

Automation

User Privacy Protections

Recording capabilities

Files and folders

Automation

User Privacy Protections

Recording capabilities

Files and folders

Automation

User Privacy Protections

Recording capabilities

Files and folders

Automation

Recording Protections

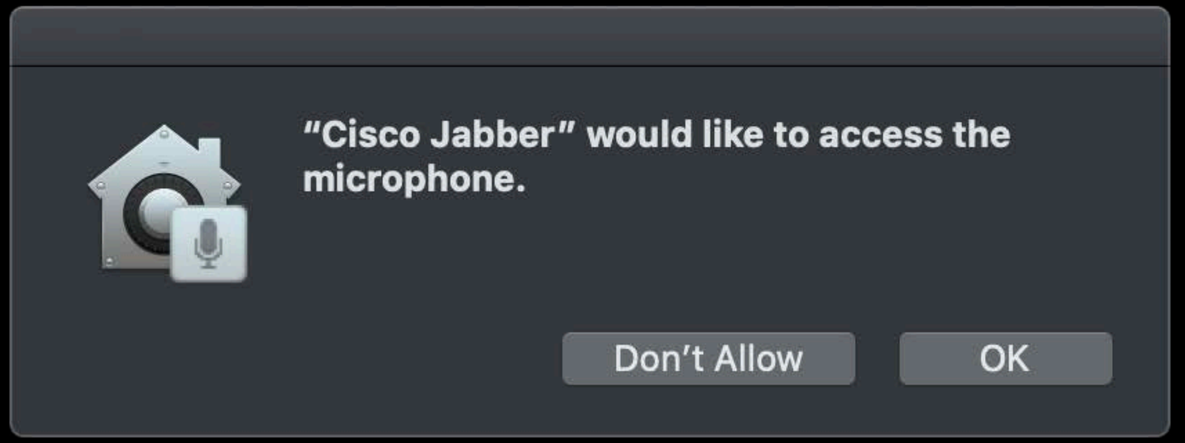
Camera

Microphone

Recording Protections

Camera

Microphone



Recording Protections in Catalina

NEW

Camera

Microphone

Recording Protections in Catalina

NEW

Camera

Microphone

Screen recording

Keyboard input monitoring

Recording Protections in Catalina

Screen recording and keyboard monitoring

NEW

Important to prevent apps from recording

- Contact information
- Private correspondence
- Account names or numbers
- Passwords
- And more

Recording Protections in Catalina

NEW

Camera

Microphone

Screen recording

Keyboard input monitoring

Recording Protections in Catalina

Screen recording

NEW

Recording the entire screen

```
guard let stream = CGDisplayStream(dispatchQueueDisplay: CGMainDisplayID(),
                                                 outputWidth: 1920,
                                                outputHeight: 1080,
                                                 pixelFormat: Int32(kCVPixelFormatType_32BGRA),
                                                  properties: nil,
                                                       queue: DispatchQueue.global(),
                                                     handler: frameHandler)
else {
    // Error occurred or user has not approved the app to record the screen.
    return
}
stream.start()
```

Recording Protections in Catalina

Screen recording

NEW

Recording the entire screen

```
guard let stream = CGDisplayStream(dispatchQueueDisplay: CGMainDisplayID(),
                                                 outputWidth: 1920,
                                                outputHeight: 1080,
                                                 pixelFormat: Int32(kCVPixelFormatType_32BGRA),
                                                  properties: nil,
                                                       queue: DispatchQueue.global(),
                                                     handler: frameHandler)
else {
    // Error occurred or user has not approved the app to record the screen.
    return
}
stream.start()
```

Recording Protections in Catalina

Screen recording

NEW

Recording the entire screen

```
guard let stream = CGDisplayStream(dispatchQueueDisplay: CGMainDisplayID(),
                                                outputWidth: 1920,
                                               outputHeight: 1080,
                                                pixelFormat: Int32(kCVPixelFormatType_32BGRA),
                                                 properties: nil,
                                                      queue: DispatchQueue.global(),
                                                    handler: frameHandler)
else {
    // Error occurred or user has not approved the app to record the screen.
    return
}
stream.start()
```

Recording Protections in Catalina

Screen recording

NEW

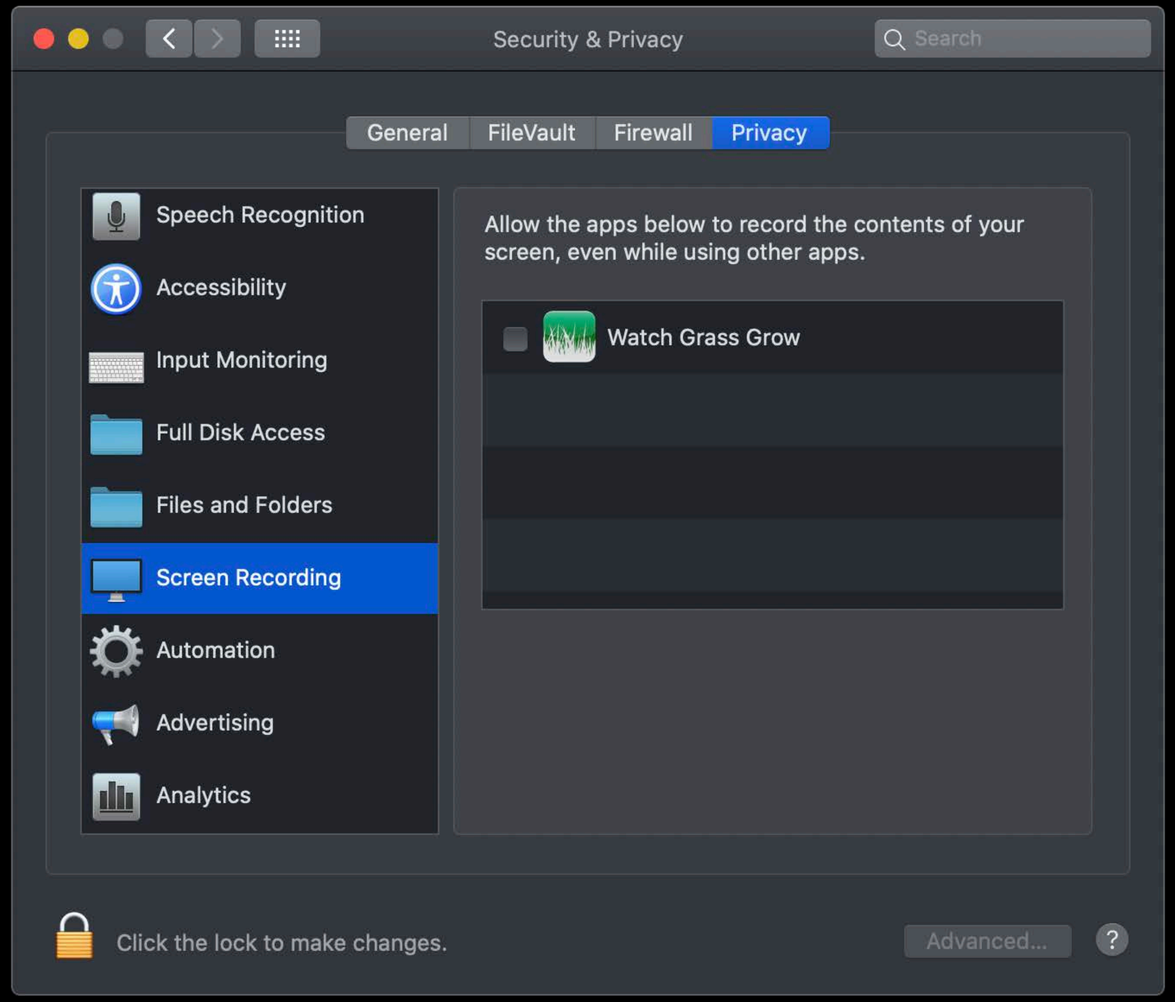
Recording the entire screen


```
guard let stream = CGD                                          ),


                                                                rmatType_32BGRA),


                                                                obal(),
                                                handler: frameHandler)
else {
    // Error occurred or user has not approved the app to record the screen.
    return
}
stream.start()
```

Recording Protections in Catalina

Approving for screen recording

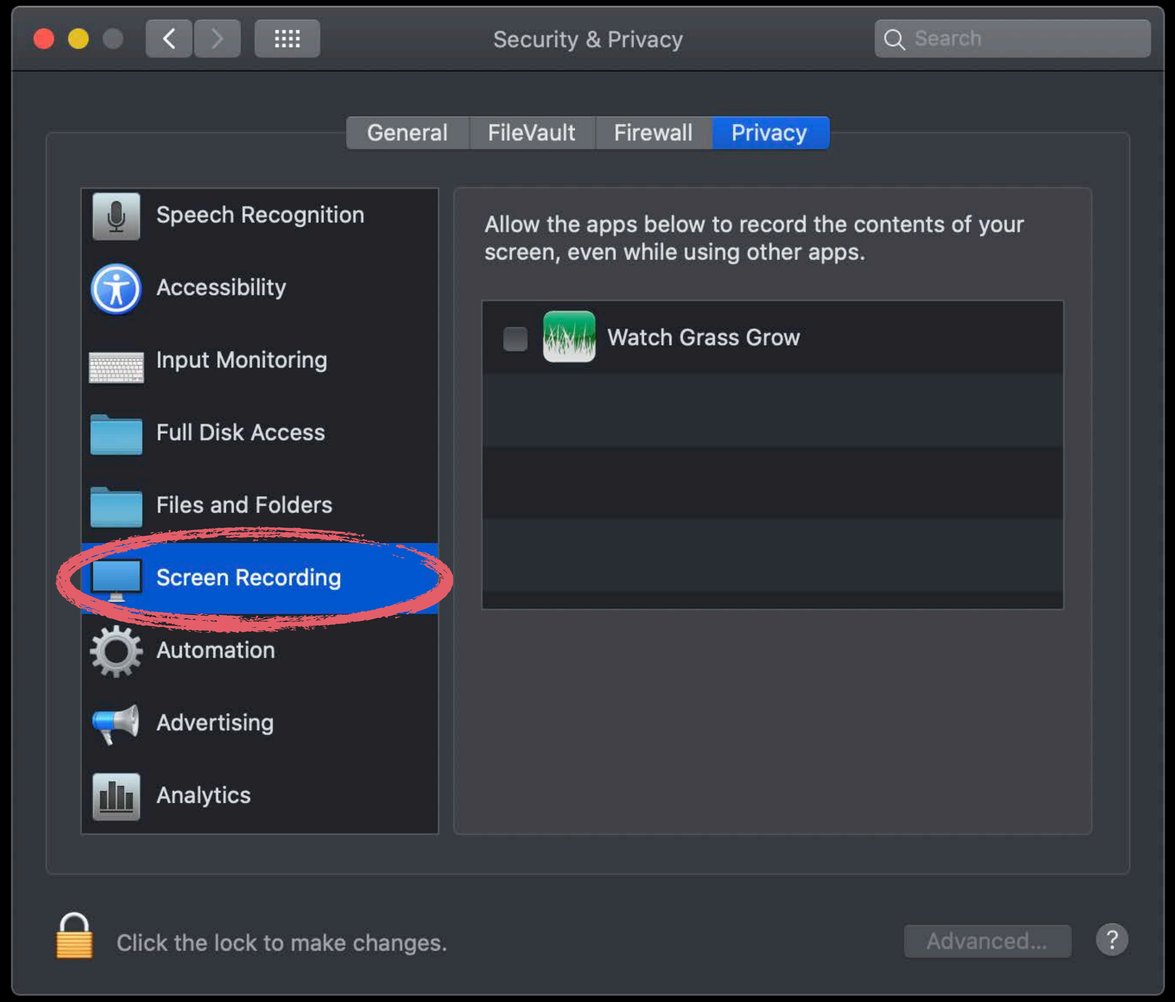
NEW



Recording Protections in Catalina

Approving for screen recording

NEW



Recording Protections in Catalina

Screen recording

NEW

Recording a window’s contents

```
func saveImage(forWindow windowId: CGWindowID, to url: URL) throws {
    let cgimage = CGWindowListCreateImage(.null, [.optionIncludingWindow], windowId,
                                          [.nominalResolution])!
    let imageRep = NSBitmapImageRep(cgImage: cgimage)
    let pngData = imageRep.representation(using: .png, properties: [:])
    try pngData!.write(to: url)
}
```

Recording Protections in Catalina

Screen recording

NEW

Recording a window’s contents

```
func saveImage(forWindow windowId: CGWindowID, to url: URL) throws {
    let cgimage = CGWindowListCreateImage(.null, [.optionIncludingWindow], windowId,
                                          [.nominalResolution])!
    let imageRep = NSBitmapImageRep(cgImage: cgimage)
    let pngData = imageRep.representation(using: .png, properties: [:])
    try pngData!.write(to: url)
}
```

Recording Protections in Catalina

Screen recording

NEW

Recording a window’s contents

```
func saveImage(forWindow windowId: CGWindowID, to url: URL) throws {
    let cgimage = CGWindowListCreateImage(.null, [.optionIncludingWindow], windowId,
                                          [.nominalResolution])!
    let imageRep = NSBitmapImageRep(cgImage: cgimage)
    let pngData = imageRep.representation(using: .png, properties: [:])
    try pngData!.write(to: url)
}
```

Recording Protections in Catalina

Screen recording

NEW

Recording a window’s contents

```
func saveImage(forWindow windowId: CGWindowID, to url: URL) throws {
    let cgimage = CGWindowListCreateImage(.null, [.optionIncludingWindow], windowId,
                                          [.nominalResolution])!
    let imageRep = NSBitmapImageRep(cgImage: cgimage)
    let pngData = imageRep.representation(using: .png, properties: [:])
    try pngData!.write(to: url)
}
```

Recording Protections in Catalina

Screen recording

NEW

Recording a window’s contents

```
func saveImage(forWindow windowId: CGWindowID, to url: URL) throws {
    let cgimage = CGWindowListCreateImage(.null, [.optionIncludingWindow], windowId,
                                          [.nominalResolution])!
    let imageRep = NSBitmapImageRep(cgImage: cgimage)
    let pngData = imageRep.representation(using: .png, properties: [:])
    try pngData!.write(to: url)
}
```

- ✓ App’s own windows
- ✓ Desktop or Menu Bar windows
- ✗ Other apps’ windows

Recording Protections in Catalina

Screen recording

NEW

Recording a window’s contents

```
func saveImage(forWinc
    let cgimage = CGWi                                    , windowId,

    let imageRep = NSB
    let pngData = imag
    try pngData!.write
}
```

- ✓ App’s own windows
- ✓ Desktop or Menu Bar windows
- ✗ Other apps’ windows

Recording Protections in Catalina

Screen recording

NEW

No approval necessary to query metadata about windows

```
let windows = CGWindowListCopyWindowInfo([.optionOnScreenOnly], kCGNullWindowID) as? [[String: AnyObject]]
```

Recording Protections in Catalina

Screen recording

NEW

No approval necessary to query metadata about windows

```
let windows = CGWindowListCopyWindowInfo([.optionOnScreenOnly], kCGNullWindowID) as? [[String: AnyObject]]
```

Recording Protections in Catalina

Screen recording

NEW

No approval necessary to query metadata about windows

```
let windows = CGWindowListCopyWindowInfo([.optionOnScreenOnly], kCGNullWindowID) as? [[String: AnyObject]]
```

Recording Protections in Catalina

Screen recording

NEW

No approval necessary to query metadata about windows

```
let windows = CGWindowListCopyWindowInfo([.optionOnScreenOnly], kCGNullWindowID) as? [[String: AnyObject]]
```

- `kCGWindowBounds`
- `kCGWindowNumber`
- `kCGWindowOwnerName`
- `kCGWindowOwnerPID`

Recording Protections in Catalina

Screen recording

NEW

No approval necessary to query metadata about windows

```
let windows = CGWindowListCopyWindowInfo([.optionOnScreenOnly], kCGNullWindowID) as? [[String: AnyObject]]
```

- kCGWindowBounds
- kCGWindowNumber
- kCGWindowOwnerName
- kCGWindowOwnerPID

Recording Protections in Catalina

Screen recording

NEW

No approval necessary to query metadata about windows

```
let windows = CGWindowListCopyWindowInfo([.optionOnScreenOnly], kCGNullWindowID) as? [[String: AnyObject]]
```

- ✓ `kCGWindowBounds`
- ✓ `kCGWindowNumber`
- ✓ `kCGWindowOwnerName`
- ✓ `kCGWindowOwnerPID`
- ✗ `kCGWindowName`
- ✗ `kCGWindowSharingState`

```swift
// Screen Recording Protections – Get Desktop Background Windows

func getDesktopWindowIds() -> [CGWindowID] {
    let windows = CGWindowListCopyWindowInfo([.optionOnScreenOnly], kCGNullWindowID)! as!
[[String: AnyObject]]
    let desktopWindowLevel = CGWindowLevelForKey(.desktopWindow) - 1
    let desktopWindows = windows.filter {
        let windowLevel = $0[kCGWindowLayer as String] as! CGWindowLevel
        return windowLevel == desktopWindowLevel
    }
    return desktopWindows.map {
        $0[kCGWindowNumber as String] as! CGWindowID
    }
}
```

```swift
// Screen Recording Protections – Get Desktop Background Windows

func getDesktopWindowIds() -> [CGWindowID] {
    let windows = CGWindowListCopyWindowInfo([.optionOnScreenOnly], kCGNullWindowID)! as!
[[String: AnyObject]]
    let desktopWindowLevel = CGWindowLevelForKey(.desktopWindow) - 1
    let desktopWindows = windows.filter {
        let windowLevel = $0[kCGWindowLayer as String] as! CGWindowLevel
        return windowLevel == desktopWindowLevel
    }
    return desktopWindows.map {
        $0[kCGWindowNumber as String] as! CGWindowID
    }
}
```

```swift
// Screen Recording Protections - Get Desktop Background Windows

func getDesktopWindowIds() -> [CGWindowID] {
    let windows = CGWindowListCopyWindowInfo([.optionOnScreenOnly], kCGNullWindowID)! as!
[[String: AnyObject]]
    let desktopWindowLevel = CGWindowLevelForKey(.desktopWindow) - 1
    let desktopWindows = windows.filter {
        let windowLevel = $0[kCGWindowLayer as String] as! CGWindowLevel
        return windowLevel == desktopWindowLevel
    }
    return desktopWindows.map {
        $0[kCGWindowNumber as String] as! CGWindowID
    }
}
```

```swift
// Screen Recording Protections — Get Desktop Background Windows

func getDesktopWindowIds() -> [CGWindowID] {
    let windows = CGWindowListCopyWindowInfo([.optionOnScreenOnly], kCGNullWindowID)! as!
[[String: AnyObject]]
    let desktopWindowLevel = CGWindowLevelForKey(.desktopWindow) - 1
    let desktopWindows = windows.filter {
        let windowLevel = $0[kCGWindowLayer as String] as! CGWindowLevel
        return windowLevel == desktopWindowLevel
    }
    return desktopWindows.map {
        $0[kCGWindowNumber as String] as! CGWindowID
    }
}
```

```swift
// Screen Recording Protections — Get Desktop Background Windows

func getDesktopWindowIds() -> [CGWindowID] {
    let windows = CGWindowListCopyWindowInfo([.optionOnScreenOnly], kCGNullWindowID)! as!
[[String: AnyObject]]
    let desktopWindowLevel = CGWindowLevelForKey(.desktopWindow) - 1
    let desktopWindows = windows.filter {
        let windowLevel = $0[kCGWindowLayer as String] as! CGWindowLevel
        return windowLevel == desktopWindowLevel
    }
    return desktopWindows.map {
        $0[kCGWindowNumber as String] as! CGWindowID
    }
}
```

```swift
// Screen Recording Protections - Get Desktop Background Windows

func getDesktopWindowIds() -> [CGWindowID] {
    let windows = CGWindowListCopyWindowInfo([.optionOnScreenOnly], kCGNullWindowID)! as!
[[String: AnyObject]]
    let desktopWindowLevel = CGWindowLevelForKey(.desktopWindow) - 1
    let desktopWindows = windows.filter {
        let windowLevel = $0[kCGWindowLayer as String] as! CGWindowLevel
        return windowLevel == desktopWindowLevel
    }
    return desktopWindows.map {
        $0[kCGWindowNumber as String] as! CGWindowID
    }
}
```

```swift
// Screen Recording Protections - Get Desktop Background Windows

func getDesktopWindowIds() -> [CGWindowID] {
    let windows = CGWindowListCopyWindowInfo([.optionOnScreenOnly], kCGNullWindowID)! as!
[[String: AnyObject]]
    let desktopWindowLevel = CGWindowLevelForKey(.desktopWindow) - 1
    let desktopWindows = windows.filter {
        let windowLevel = $0[kCGWindowLayer as String] as! CGWindowLevel
        return windowLevel == desktopWindowLevel
    }
    return desktopWindows.map {
        $0[kCGWindowNumber as String] as! CGWindowID
    }
}
```

```swift
// Screen Recording Protections - Get Desktop Background Windows

func getDesktopWindowIds() -> [CGWindowID] {
    let windows = CGWindowListCopyWindowInfo([.optionOnScreenOnly], kCGNullWindowID)! as!
[[String: AnyObject]]
    let desktopWindowLevel = CGWindowLevelForKey(.desktopWindow) - 1
    let desktopWindows = windows.filter {
        let windowLevel = $0[kCGWindowLayer as String] as! CGWindowLevel
        return windowLevel == desktopWindowLevel
    }
    return desktopWindows.map {
        $0[kCGWindowNumber as String] as! CGWindowID
    }
}
```

```swift
// Screen Recording Protections - Get Desktop Background Windows

func getDesktopWindowIds() -> [CGWindowID] {
    let windows = CGWindowListCopyWindowInfo([.optionOnScreenOnly], kCGNullWindowID)! as!
[[String: AnyObject]]
    let desktopWindowLevel = CGWindowLevelForKey(.desktopWindow) - 1
    let desktopWindows = windows.filter {
        let windowLevel = $0[kCGWindowLayer as String] as! CGWindowLevel
        return windowLevel == desktopWindowLevel
    }
    return desktopWindows.map {
        $0[kCGWindowNumber as String] as! CGWindowID
    }
}
```

Recording Protections in Catalina

NEW

Camera

Microphone

Screen recording

Keyboard input monitoring

Recording Protections in Catalina

NEW

Camera

Microphone

Screen recording

Keyboard input monitoring

Recording Protections

Keyboard input monitoring

NEW

No approval necessary to monitor events for own app

```
NSEvent.addLocalMonitorForEvents(matching: .any, handler: { event in
    // Do something with the event
    return event
})
```

Recording Protections

Keyboard input monitoring

NEW

No approval necessary to monitor events for own app

```
NSEvent.addLocalMonitorForEvents(matching: .any, handler: { event in
    // Do something with the event
    return event
})
```

NEW

```
// Keyboard Event Recording Protections

func callback(proxy: CGEventTapProxy, type: CGEventType, event: CGEvent,
              userInfo: UnsafeMutableRawPointer?) -> Unmanaged<CGEvent>? {
    // Do something with the event.
    return Unmanaged.passUnretained(event)
}

let eventMask = (1 << CGEventType.keyDown.rawValue) | (1 << CGEventType.keyUp.rawValue)
let eventTap = CGEvent.tapCreate(tap: .cghidEventTap,
                                 place: .tailAppendEventTap,
                                 options: .listenOnly,
                                 eventsOfInterest: CGEventMask(eventMask),
                                 callback: callback,
                                 userInfo: nil)
```

NEW

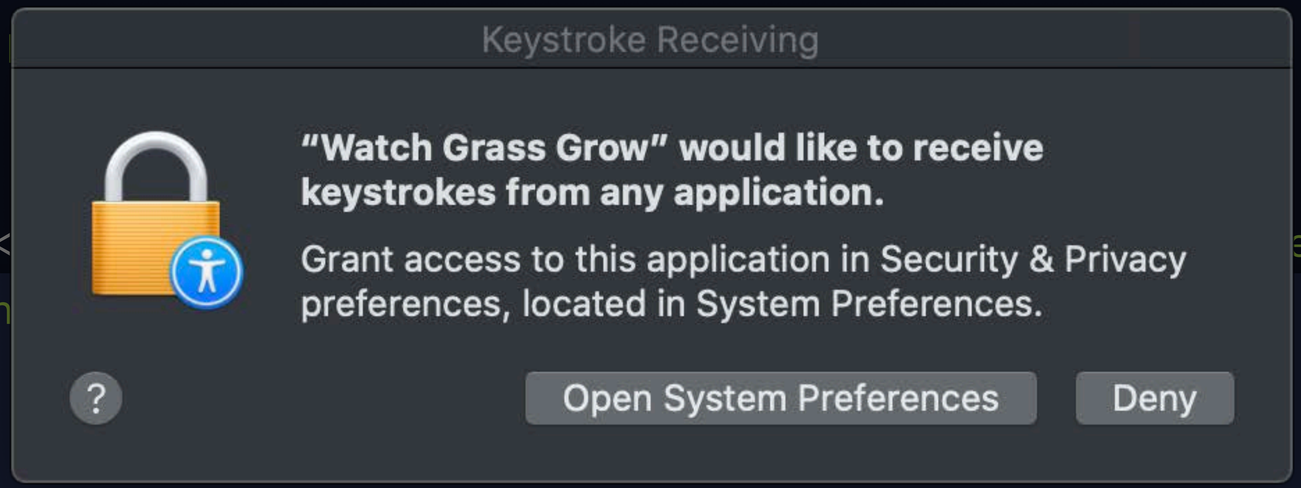
```
// Keyboard Event Recording Protections

func callback(proxy: CGEventTapProxy, type: CGEventType, event: CGEvent,
              userInfo: UnsafeMutableRawPointer?) -> Unmanaged<CGEvent>? {
    // Do something with the event.
    return Unmanaged.passUnretained(event)
}

let eventMask = (1 << CGEventType.keyDown.rawValue) | (1 << CGEventType.keyUp.rawValue)
let eventTap = CGEvent.tapCreate(tap: .cghidEventTap,
                                 place: .tailAppendEventTap,
                                 options: .listenOnly,
                                 eventsOfInterest: CGEventMask(eventMask),
                                 callback: callback,
                                 userInfo: nil)
```

NEW

```
// Keyboard Event Recording Protections

func callback(proxy: CGEventTapProxy, type: CGEventType, event: CGEvent,
              userInfo: UnsafeMutableRawPointer?) -> Unmanaged<CGEvent>? {
    // Do something with the event.
    return Unmanaged.passUnretained(event)
}

let eventMask = (1 << CGEventType.keyDown.rawValue) | (1 << CGEventType.keyUp.rawValue)
let eventTap = CGEvent.tapCreate(tap: .cghidEventTap,
                                 place: .tailAppendEventTap,
                                 options: .listenOnly,
                                 eventsOfInterest: CGEventMask(eventMask),
                                 callback: callback,
                                 userInfo: nil)
```

NEW

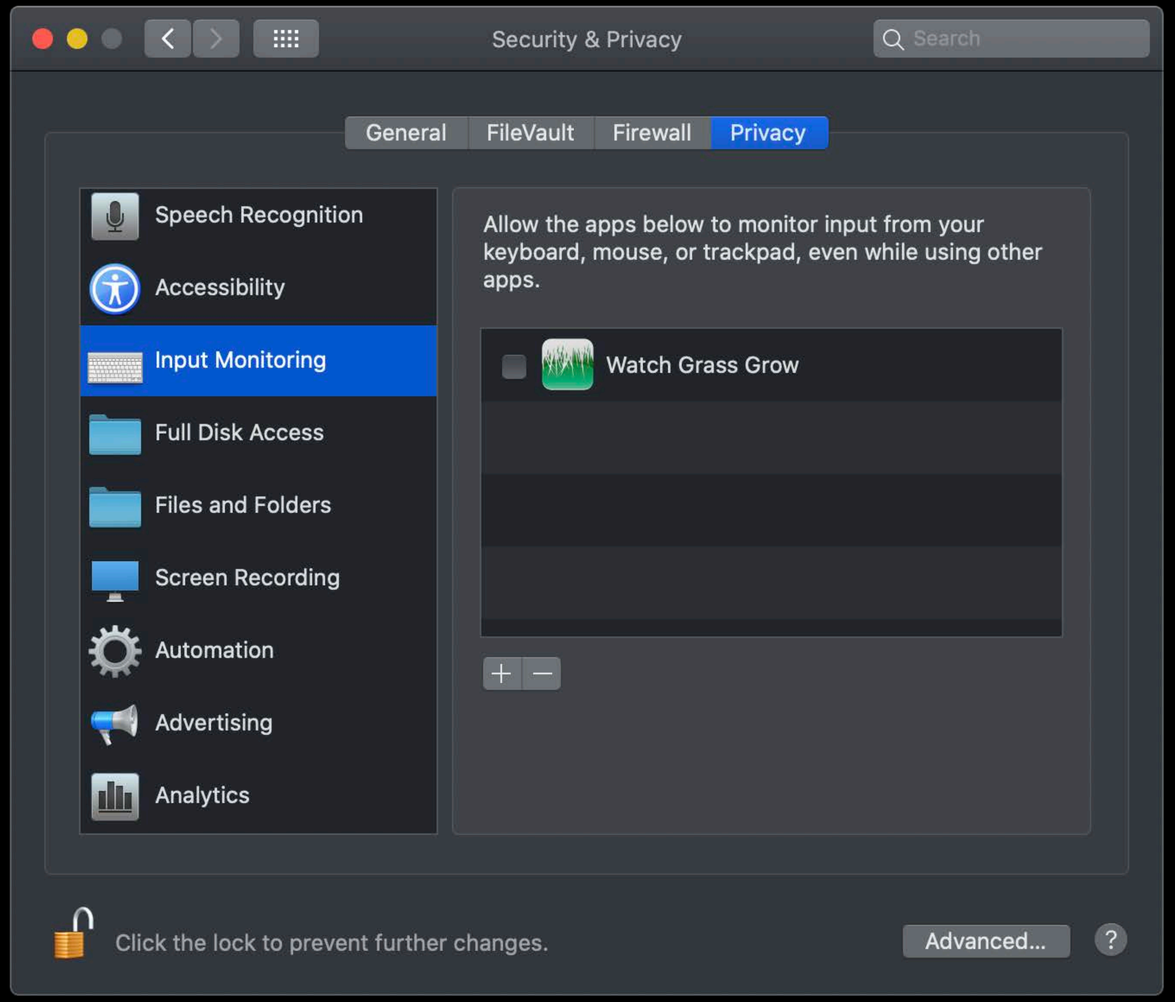
```
// Keyboard Event Recording Protections

func callback(proxy: CGEventTapProxy, type: CGEventType, event: CGEvent,
              userInfo: UnsafeMutableRawPointer?) -> Unmanaged<CGEvent>? {
    // Do something with the event.
    return Unmanaged.
}

let eventMask = (1 <<                                          eyUp.rawValue)
let eventTap = CGEven
                                  eventsOfInterest: CGEventMask(eventMask),
                                          callback: callback,
                                          userInfo: nil)
```



Recording Protections

Approving for keyboard input monitoring

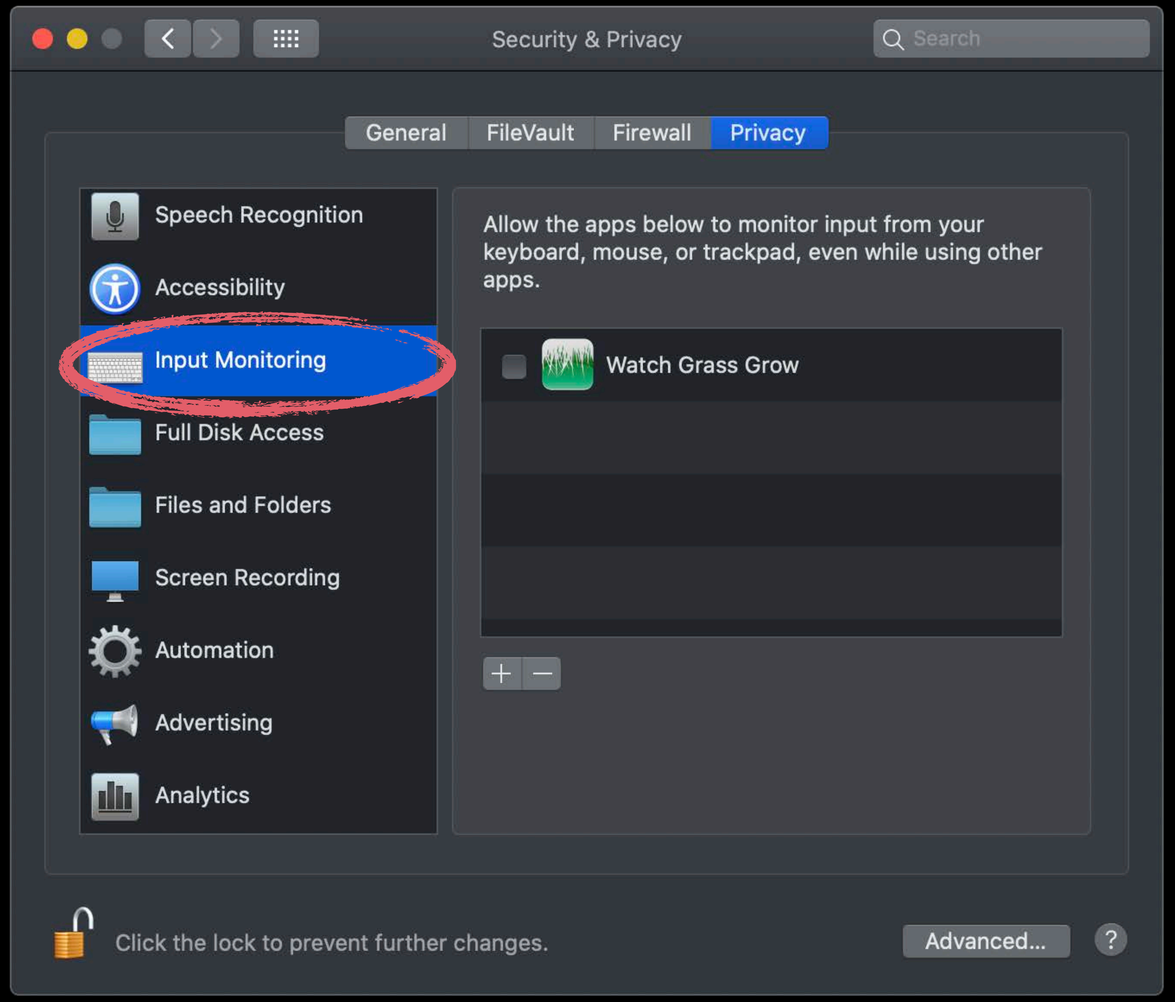
NEW



Recording Protections

Approving for keyboard input monitoring

NEW



Recording Protections

Checking keyboard input monitoring approval

NEW

```
let accessType = IOHIDCheckAccess(kIOHIDRequestTypeListenEvent)
switch accessType {
    case kIOHIDAccessTypeGranted:
        // User has approved the app to listen to all keystrokes
        …
    case kIOHIDAccessTypeDenied:
        // Denied; approval dialog has been displayed.
        …
    case kIOHIDAccessTypeUnknown:
        // Denied; approval dialog has not yet been displayed.
        …
    default:
        // Unknown status; assume denied.
        …
}
```

Recording Protections

Checking keyboard input monitoring approval

NEW

```
let accessType = IOHIDCheckAccess(kIOHIDRequestTypeListenEvent)
switch accessType {
    case kIOHIDAccessTypeGranted:
        // User has approved the app to listen to all keystrokes
        …
    case kIOHIDAccessTypeDenied:
        // Denied; approval dialog has been displayed.
        …
    case kIOHIDAccessTypeUnknown:
        // Denied; approval dialog has not yet been displayed.
        …
    default:
        // Unknown status; assume denied.
        …
}
```

Recording Protections

Requesting keyboard input monitoring approval

NEW

```
if IOHIDRequestAccess(kIOHIDRequestTypeListenEvent) {
    // The user has approved the app to listen to all keystrokes.
    …
} else {
    // App may not listen to all keystrokes.
    // Approval dialog displayed if it has not previously been displayed.
    …
}
```

Recording Protections

Requesting keyboard input monitoring approval

NEW

```
if IOHIDRequestAccess(kIOHIDRequestTypeListenEvent) {
    // The user has approved the app to listen to all keystrokes.
    …
} else {
    // App may not listen to all keystrokes.
    // Approval dialog displayed if it has not previously been displayed.
    …
}
```

User Privacy Protections

Recording capabilities

Files and folders

Automation

User Privacy Protections

Recording capabilities

Files and folders

Automation

User Privacy Protections

Recording capabilities

Files and folders

Automation

User Privacy Protections

Recording capabilities

Files and folders

- Data that requires user consent to access

Automation

User Privacy Protections

Recording capabilities

Files and folders

- Data that requires user consent to access
- Private data managed by the system

Automation

User Privacy Protections

Recording capabilities

Files and folders

- Data that requires user consent to access
- Private data managed by the system

Automation

User Privacy Protections

Recording capabilities

Files and folders

- Data that requires user consent to access
- Private data managed by the system

Automation

User Data Protections

Data that requires user consent to access

User Data Protections

Data that requires user consent to access

Contacts

Calendars

Reminders

Photos

User Data Protections

Data that requires user consent to access

Contacts

Calendars

Reminders

Photos

User Data Protections

Data that requires user consent to access

Contacts

Calendars

Reminders

Photos

User Data Protections

Data that requires user consent to access

NEW

Contacts

Calendars

Reminders

Photos

User Data Protections

Data that requires user consent to access

NEW

Contacts

Calendars

Reminders

Photos

Desktop

Documents

Downloads

iCloud Drive

Third-party cloud storage

Removable volumes

Network volumes

User Data Protections

Data that requires user consent to access

NEW

Contacts

Calendars

Reminders

Photos

Desktop

Documents

Downloads

iCloud Drive

Third-party cloud storage

Removable volumes

Network volumes

User Data Protections

Data that requires user consent to access

NEW

- Contacts
- Calendars
- Reminders
- Photos
- Desktop
- Documents
- Downloads
- iCloud Drive
- Third-party cloud storage
- Removable volumes
- Network volumes

User Data Protections

Data that requires user consent to access

NEW

- Contacts
- Calendars
- Reminders
- Photos
- Desktop
- Documents
- Downloads
- iCloud Drive
- Third-party cloud storage
- Removable volumes
- Network volumes

User Data Protections

Data that requires user consent to access

NEW

Contacts

Calendars

Reminders

Photos

Desktop

Documents

Downloads

iCloud Drive

Third-party cloud storage

Removable volumes

Network volumes

User Data Protections

Data that requires user consent to access

NEW

Contacts

Calendars

Reminders

Photos

Desktop

Documents

Downloads

iCloud Drive

Third-party cloud storage

Removable volumes

Network volumes

User Data Protections

Data that requires user consent to access

NEW

Contacts

Calendars

Reminders

Photos

Desktop

Documents

Downloads

iCloud Drive

Third-party cloud storage

Removable volumes

Network volumes

User Data Protections

Data that requires user consent to access

NEW

Contacts

Calendars

Reminders

Photos

Desktop

Documents

Downloads

iCloud Drive

Third-party cloud storage

Removable volumes

Network volumes

User Data Protections

Data that requires user consent to access

NEW

Contacts

Calendars

Reminders

Photos

Desktop

Documents

Downloads

iCloud Drive

Third-party cloud storage

Removable volumes

Network volumes

User Data Protections

Data that requires user consent to access

NEW

Contacts

Calendars

Reminders

Photos

Desktop

Documents

Downloads

iCloud Drive

Third-party cloud storage

Removable volumes

Network volumes

User Data Protections

Minimizing consent prompts by inferring user intent

NEW

User Data Protections

Minimizing consent prompts by inferring user intent

NEW

Double-clicking on files in Finder

Dragging and dropping

Selecting files via a NSOpenPanel or NSSavePanel

User Data Protections

Minimizing consent prompts by inferring user intent

NEW

Double-clicking on files in Finder

Dragging and dropping

Selecting files via a NSOpenPanel or NSSavePanel

User Data Protections

Minimizing consent prompts by inferring user intent

NEW

Double-clicking on files in Finder

Dragging and dropping

Selecting files via a NSOpenPanel or NSSavePanel

User Data Protections

Minimizing consent prompts by inferring user intent

NEW

User Consent

User Intent

User Data Protections

Minimizing consent prompts by inferring user intent

NEW

User Consent	User Intent
Reactive	Proactive

User Data Protections

Minimizing consent prompts by inferring user intent

NEW

User Consent	User Intent
Reactive	Proactive

User Data Protections

Minimizing consent prompts by inferring user intent

NEW

User Consent	User Intent
Reactive	Proactive
Prompt interrupts workflow	Inferred transparently from workflow

User Data Protections

Minimizing consent prompts by inferring user intent

NEW

User Consent	User Intent
Reactive	Proactive
Prompt interrupts workflow	Inferred transparently from workflow

User Data Protections

Minimizing consent prompts by inferring user intent

NEW

User Consent	User Intent
Reactive	Proactive
Prompt interrupts workflow	Inferred transparently from workflow
Applies to whole class of data	Applies to just the selected files

User Data Protections

Minimizing consent prompts by inferring user intent

NEW

User Consent	User Intent
Reactive	Proactive
Prompt interrupts workflow	Inferred transparently from workflow
Applies to whole class of data	Applies to just the selected files

User Data Protections

Minimizing consent prompts by inferring user intent

NEW

User Consent	User Intent
Reactive	Proactive
Prompt interrupts workflow	Inferred transparently from workflow
Applies to whole class of data	Applies to just the selected files

User Data Protections

Minimizing consent prompts by inferring user intent

NEW

User Consent	User Intent
Reactive	Proactive
Prompt interrupts workflow	Inferred transparently from workflow
Applies to whole class of data	Applies to just the selected files

User Data Protections

Minimizing consent prompts by inferring user intent

NEW

User Consent	User Intent
Reactive	Proactive
Prompt interrupts workflow	Inferred transparently from workflow
Applies to whole class of data	Applies to just the selected files

User Data Protections

Accessing sidecar files

```
<key>CFBundleDocumentTypes</key>
<array>
    <dict>
        <key>CFBundleTypeRole</key>
        <string>None</string>
        <key>CFBundleTypeExtensions</key>
        <array>
            <string>srt</string>
        </array>
        <key>CFBundleTypeName</key>
        <string>Subtitle File</string>
        <key>NSIsRelatedItemType</key>
        <true/>
    </dict>
</array>
```

User Data Protections

Accessing sidecar files

```
<key>CFBundleDocumentTypes</key>
<array>
    <dict>
        <key>CFBundleTypeRole</key>
        <string>None</string>
        <key>CFBundleTypeExtensions</key>
        <array>
            <string>srt</string>
        </array>
        <key>CFBundleTypeName</key>
        <string>Subtitle File</string>
        <key>NSIsRelatedItemType</key>
        <true/>
    </dict>
</array>
```

User Data Protections

Accessing sidecar files

```
<key>CFBundleDocumentTypes</key>
<array>
    <dict>
        <key>CFBundleTypeRole</key>
        <string>None</string>
        <key>CFBundleTypeExtensions</key>
        <array>
            <string>srt</string>
        </array>
        <key>CFBundleTypeName</key>
        <string>Subtitle File</string>
        <key>NSIsRelatedItemType</key>
        <true/>
    </dict>
</array>
```

```
class SubtitleSidecar: NSObject, NSFilePresenter {
    lazy var presentedItemOperationQueue = OperationQueue.main
    var primaryPresentedItemURL: URL?
    var presentedItemURL: URL?
    init(withMovieURL movieURL: URL) {
        primaryPresentedItemURL = movieURL
        presentedItemURL = movieURL.deletingPathExtension().appendingPathExtension("srt")
    }
    func readData() -> Data? {
        var data: Data?
        var error: NSError?
        let coordinator = NSFileCoordinator.init(filePresenter: self)
        coordinator.coordinate(readingItemAt: presentedItemURL!, options: [], error: &error) {
url in
            data = try! Data.init(contentsOf: url)
        }
        return data
    }
}
```

```
class SubtitleSidecar: NSObject, NSFilePresenter {
    lazy var presentedItemOperationQueue = OperationQueue.main
    var primaryPresentedItemURL: URL?
    var presentedItemURL: URL?
    init(withMovieURL movieURL: URL) {
        primaryPresentedItemURL = movieURL
        presentedItemURL = movieURL.deletingPathExtension().appendingPathExtension("srt")
    }
    func readData() -> Data? {
        var data: Data?
        var error: NSError?
        let coordinator = NSFileCoordinator.init(filePresenter: self)
        coordinator.coordinate(readingItemAt: presentedItemURL!, options: [], error: &error) {
url in
            data = try! Data.init(contentsOf: url)
        }
        return data
    }
}
```

```
class SubtitleSidecar: NSObject, NSFilePresenter {
    lazy var presentedItemOperationQueue = OperationQueue.main
    var primaryPresentedItemURL: URL?
    var presentedItemURL: URL?
    init(withMovieURL movieURL: URL) {
        primaryPresentedItemURL = movieURL
        presentedItemURL = movieURL.deletingPathExtension().appendingPathExtension("srt")
    }
    func readData() -> Data? {
        var data: Data?
        var error: NSError?
        let coordinator = NSFileCoordinator.init(filePresenter: self)
        coordinator.coordinate(readingItemAt: presentedItemURL!, options: [], error: &error) {
url in
            data = try! Data.init(contentsOf: url)
        }
        return data
    }
}
```

```
class SubtitleSidecar: NSObject, NSFilePresenter {
    lazy var presentedItemOperationQueue = OperationQueue.main
    var primaryPresentedItemURL: URL?
    var presentedItemURL: URL?
    init(withMovieURL movieURL: URL) {
        primaryPresentedItemURL = movieURL
        presentedItemURL = movieURL.deletingPathExtension().appendingPathExtension("srt")
    }
    func readData() -> Data? {
        var data: Data?
        var error: NSError?
        let coordinator = NSFileCoordinator.init(filePresenter: self)
        coordinator.coordinate(readingItemAt: presentedItemURL!, options: [], error: &error) {
url in
            data = try! Data.init(contentsOf: url)
        }
        return data
    }
}
```

```
class SubtitleSidecar: NSObject, NSFilePresenter {
    lazy var presentedItemOperationQueue = OperationQueue.main
    var primaryPresentedItemURL: URL?
    var presentedItemURL: URL?
    init(withMovieURL movieURL: URL) {
        primaryPresentedItemURL = movieURL
        presentedItemURL = movieURL.deletingPathExtension().appendingPathExtension("srt")
    }
    func readData() -> Data? {
        var data: Data?
        var error: NSError?
        let coordinator = NSFileCoordinator.init(filePresenter: self)
        coordinator.coordinate(readingItemAt: presentedItemURL!, options: [], error: &error) {
url in
            data = try! Data.init(contentsOf: url)
        }
        return data
    }
}
```

```
class SubtitleSidecar: NSObject, NSFilePresenter {
    lazy var presentedItemOperationQueue = OperationQueue.main
    var primaryPresentedItemURL: URL?
    var presentedItemURL: URL?
    init(withMovieURL movieURL: URL) {
        primaryPresentedItemURL = movieURL
        presentedItemURL = movieURL.deletingPathExtension().appendingPathExtension("srt")
    }
    func readData() -> Data? {
        var data: Data?
        var error: NSError?
        let coordinator = NSFileCoordinator.init(filePresenter: self)
        coordinator.coordinate(readingItemAt: presentedItemURL!, options: [], error: &error) {
url in
            data = try! Data.init(contentsOf: url)
        }
        return data
    }
}
```

```
class SubtitleSidecar: NSObject, NSFilePresenter {
    lazy var presentedItemOperationQueue = OperationQueue.main
    var primaryPresentedItemURL: URL?
    var presentedItemURL: URL?
    init(withMovieURL movieURL: URL) {
        primaryPresentedItemURL = movieURL
        presentedItemURL = movieURL.deletingPathExtension().appendingPathExtension("srt")
    }
    func readData() -> Data? {
        var data: Data?
        var error: NSError?
        let coordinator = NSFileCoordinator.init(filePresenter: self)
        coordinator.coordinate(readingItemAt: presentedItemURL!, options: [], error: &error) {
url in
            data = try! Data.init(contentsOf: url)
        }
        return data
    }
}
```

```
class SubtitleSidecar: NSObject, NSFilePresenter {
    lazy var presentedItemOperationQueue = OperationQueue.main
    var primaryPresentedItemURL: URL?
    var presentedItemURL: URL?
    init(withMovieURL movieURL: URL) {
        primaryPresentedItemURL = movieURL
        presentedItemURL = movieURL.deletingPathExtension().appendingPathExtension("srt")
    }
    func readData() -> Data? {
        var data: Data?
        var error: NSError?
        let coordinator = NSFileCoordinator.init(filePresenter: self)
        coordinator.coordinate(readingItemAt: presentedItemURL!, options: [], error: &error) {
url in
            data = try! Data.init(contentsOf: url)
        }
        return data
    }
}
```

```
class SubtitleSidecar: NSObject, NSFilePresenter {
    lazy var presentedItemOperationQueue = OperationQueue.main
    var primaryPresentedItemURL: URL?
    var presentedItemURL: URL?
    init(withMovieURL movieURL: URL) {
        primaryPresentedItemURL = movieURL
        presentedItemURL = movieURL.deletingPathExtension().appendingPathExtension("srt")
    }
    func readData() -> Data? {
        var data: Data?
        var error: NSError?
        let coordinator = NSFileCoordinator.init(filePresenter: self)
        coordinator.coordinate(readingItemAt: presentedItemURL!, options: [], error: &error) {
url in
            data = try! Data.init(contentsOf: url)
        }
        return data
    }
}
```

```
class SubtitleSidecar: NSObject, NSFilePresenter {
    lazy var presentedItemOperationQueue = OperationQueue.main
    var primaryPresentedItemURL: URL?
    var presentedItemURL: URL?
    init(withMovieURL movieURL: URL) {
        primaryPresentedItemURL = movieURL
        presentedItemURL = movieURL.deletingPathExtension().appendingPathExtension("srt")
    }
    func readData() -> Data? {
        var data: Data?
        var error: NSError?
        let coordinator = NSFileCoordinator.init(filePresenter: self)
        coordinator.coordinate(readingItemAt: presentedItemURL!, options: [], error: &error) {
url in
            data = try! Data.init(contentsOf: url)
        }
        return data
    }
}
```

```
class SubtitleSidecar: NSObject, NSFilePresenter {
    lazy var presentedItemOperationQueue = OperationQueue.main
    var primaryPresentedItemURL: URL?
    var presentedItemURL: URL?
    init(withMovieURL movieURL: URL) {
        primaryPresentedItemURL = movieURL
        presentedItemURL = movieURL.deletingPathExtension().appendingPathExtension("srt")
    }
    func readData() -> Data? {
        var data: Data?
        var error: NSError?
        let coordinator = NSFileCoordinator.init(filePresenter: self)
        coordinator.coordinate(readingItemAt: presentedItemURL!, options: [], error: &error) {
url in
            data = try! Data.init(contentsOf: url)
        }
        return data
    }
}
```

User Data Protections

NSOpenPanel and NSSavePanel changes

NEW

Open and save panels are now hosted out-of-process

User Data Protections

NSOpenPanel and NSSavePanel changes

NEW

Open and save panels are now hosted out-of-process

User Data Protections

NSOpenPanel and NSSavePanel changes

NEW

Open and save panels are now hosted out-of-process

Class inheritance and view hierarchies have changed

User Data Protections

NSOpenPanel and NSSavePanel changes

NEW

Open and save panels are now hosted out-of-process

Class inheritance and view hierarchies have changed

User Data Protections

NSOpenPanel and NSSavePanel changes

NEW

Open and save panels are now hosted out-of-process

Class inheritance and view hierarchies have changed

Cannot invoke the OK button using the `ok` method

User Data Protections

NSOpenSavePanelDelegate changes

NEW

NSOpenSavePanelDelegate

```
func panel(_ sender: Any, userEnteredFilename filename: String, confirmed okFlag: Bool) ->
String?
```

Cannot rewrite the user’s selection

User Data Protections

NSOpenSavePanelDelegate changes

NEW

NSOpenSavePanelDelegate

```
func panel(_ sender: Any, validate url: URL) throws
func panel(_ sender: Any, didChangeToDirectoryURL url: URL?)
```

App has not yet been granted access to the file

Access may trigger a consent prompt

User Data Protections

APIs for testing filesystem authorization

FileManager

```
func isReadableFile(atPath path: String) -> Bool
func isWritableFile(atPath path: String) -> Bool
```

BSD

```
int access(const char *path, int mode)
```

User Data Protections

APIs for testing filesystem authorization

FileManager

```
func isReadableFile(atPath path: String) -> Bool
func isWritableFile(atPath path: String) -> Bool
```

BSD

```
int access(const char *path, int mode)
```

User Data Protections

Purpose string keys

NEW

Desktop folder	`NSDesktopFolderUsageDescription`
Documents folder	`NSDocumentsFolderUsageDescription`
Downloads folder	`NSDownloadsFolderUsageDescription`
iCloud Drive or Third-party Cloud Provider	`NSFileProviderDomainUsageDescription`
Removable volumes	`NSRemovableVolumesUsageDescription`
Network volumes	`NSNetworkVolumesUsageDescription`

User Data Protections

Purpose string keys

NEW

Desktop folder	`NSDesktopFolderUsageDescription`
Documents folder	`NSDocumentsFolderUsageDescription`
Downloads folder	`NSDownloadsFolderUsageDescription`
iCloud Drive or Third-party Cloud Provider	`NSFileProviderDomainUsageDescription`
Removable volumes	`NSRemovableVolumesUsageDescription`
Network volumes	`NSNetworkVolumesUsageDescription`

User Privacy Protections

Recording capabilities

Files and folders

- Data that requires user consent to access
- Private data managed by the system

Automation

User Privacy Protections

Recording capabilities

Files and folders

- Data that requires user consent to access
- Private data managed by the system

Automation

User Data Protections

Private data managed by the system

User Data Protections

Private data managed by the system

Mail

Messages

Safari browsing history

HTTP cookies

Call history

iTunes backups

Time machine backups

User Data Protections

APIs for testing filesystem authorization

FileManager

```
func isReadableFile(atPath path: String) -> Bool
func isWritableFile(atPath path: String) -> Bool
```

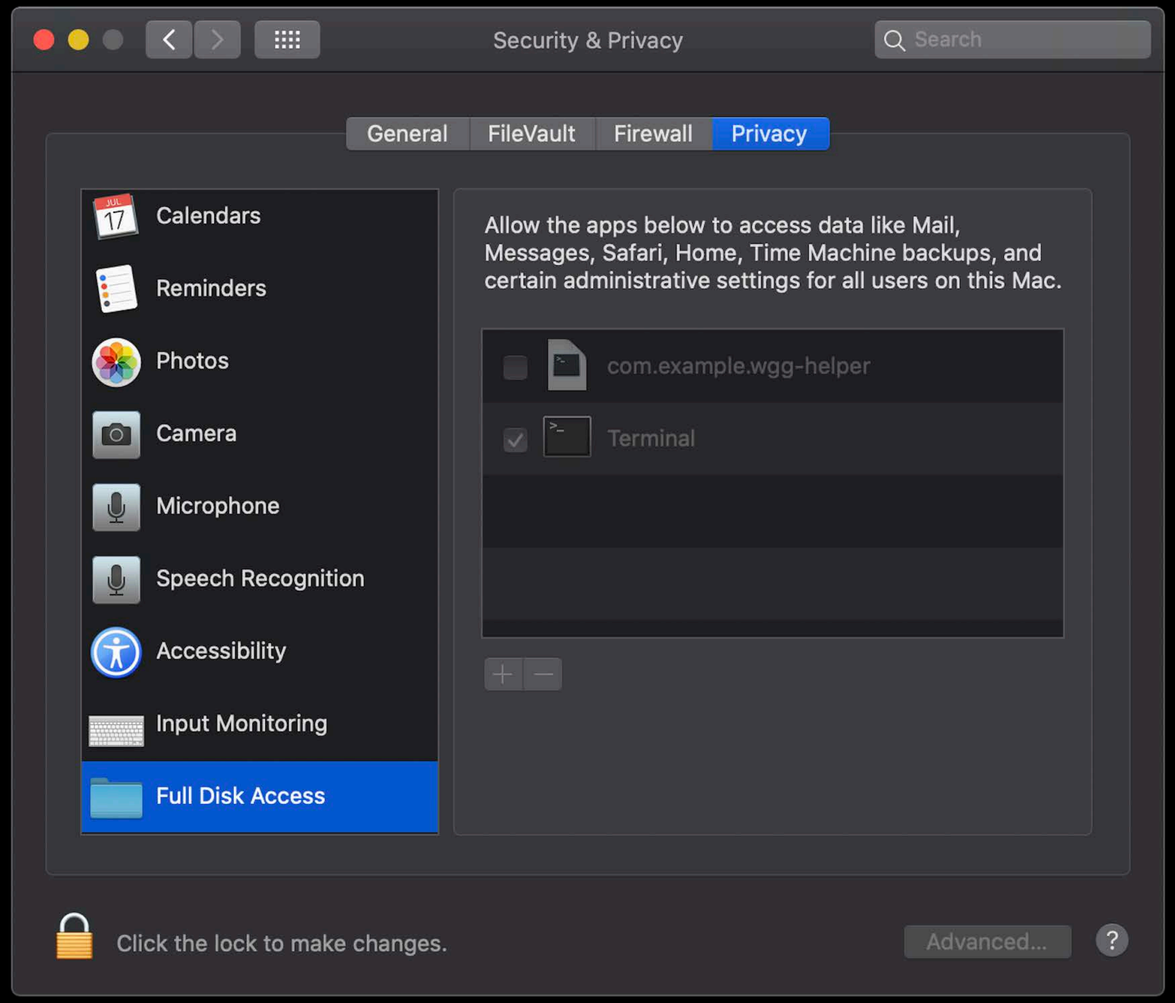
BSD

```
int access(const char *path, int mode)
```

User Data Protections

Authorizing for data access via the filesystem

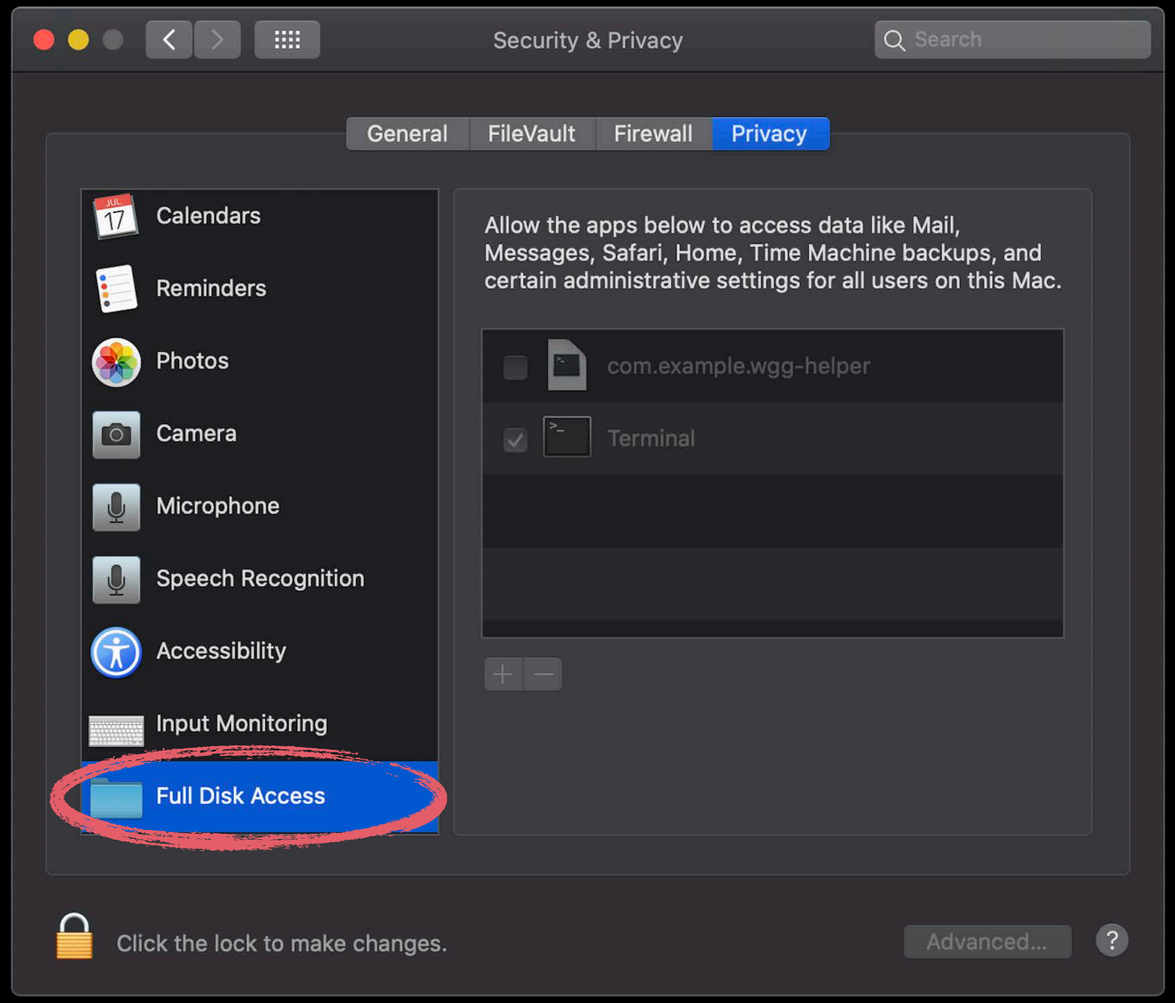
NEW



User Data Protections

Authorizing for data access via the filesystem

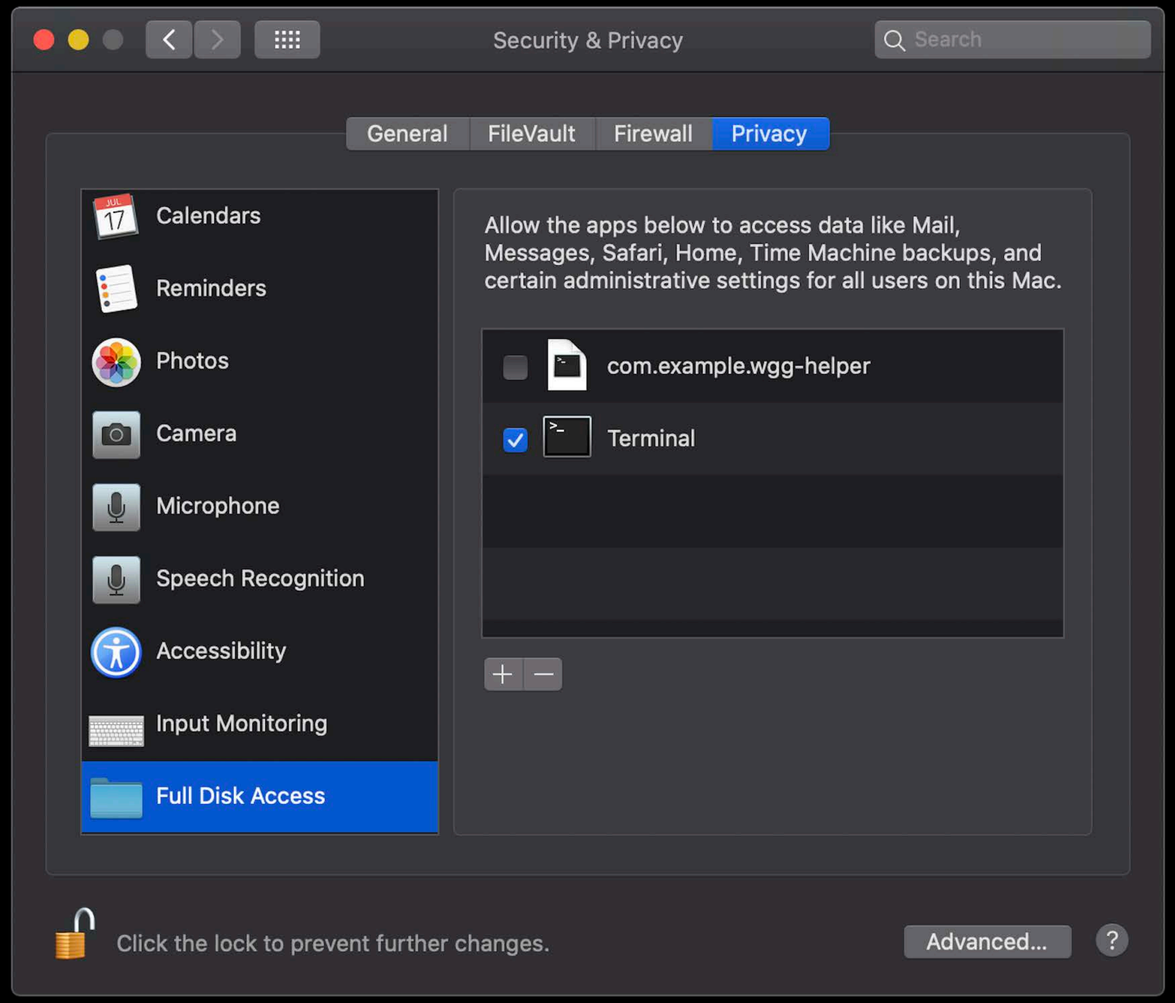
NEW



User Data Protections

Authorizing for data access via the filesystem

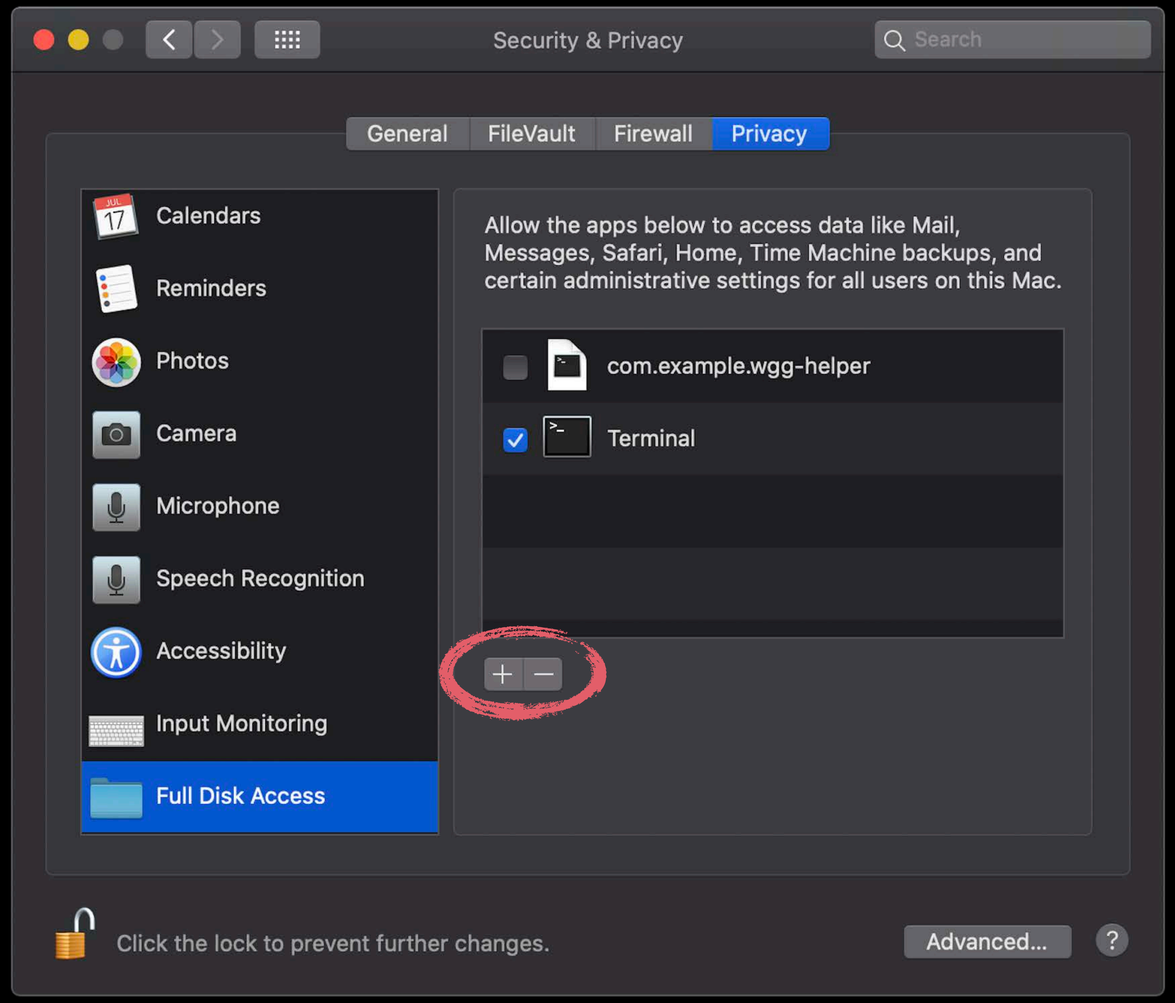
NEW



User Data Protections

Authorizing for data access via the filesystem

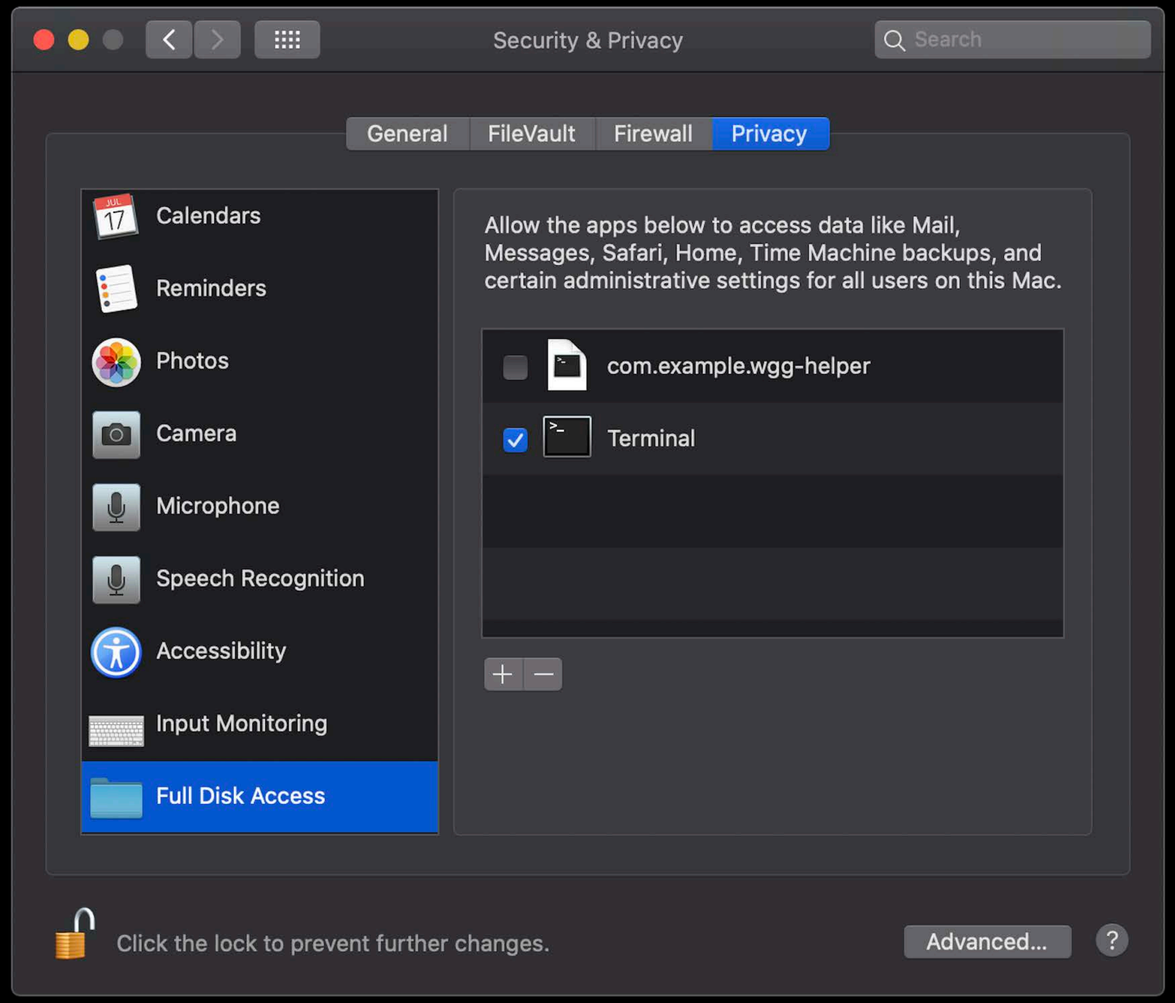
NEW



User Data Protections

Authorizing for data access via the filesystem

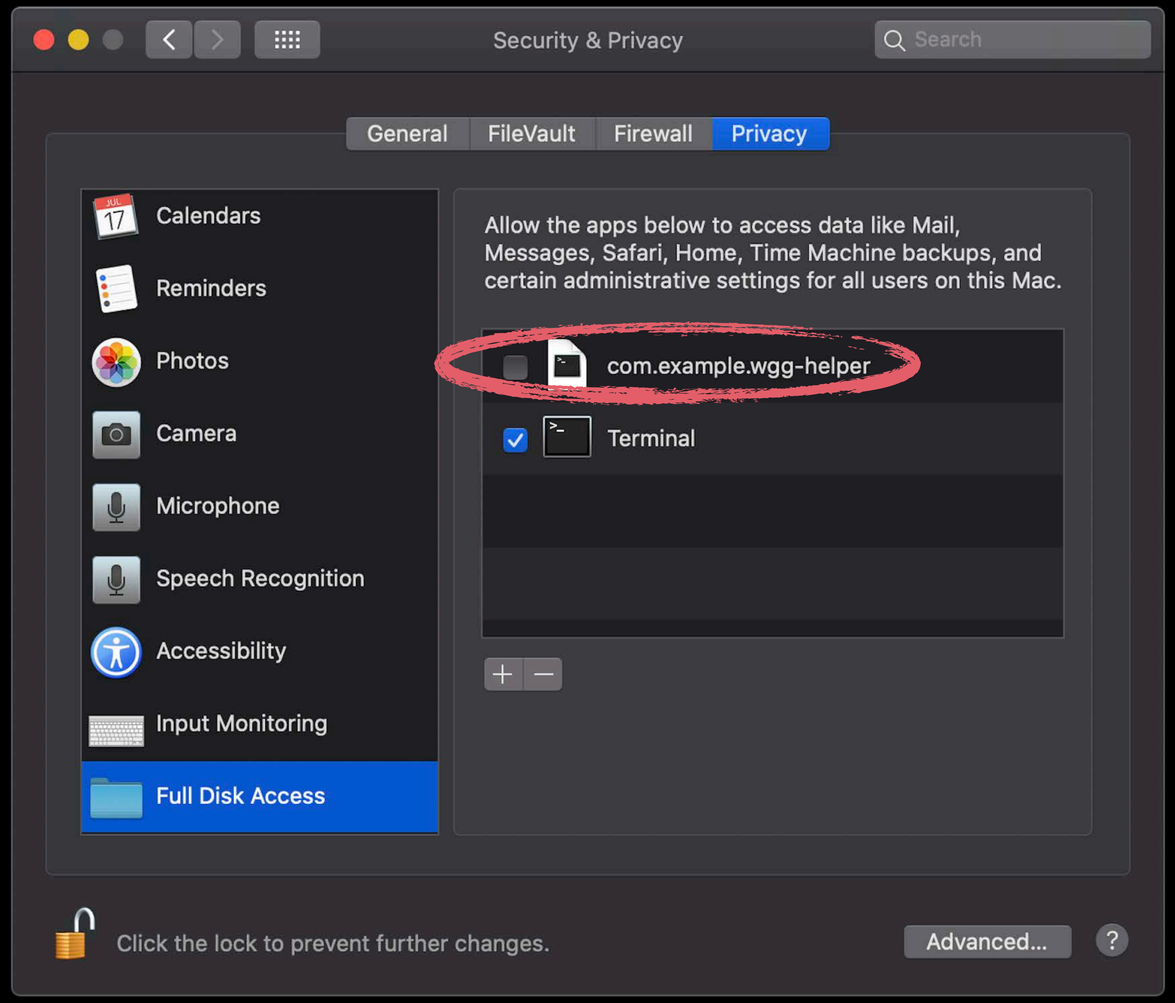
NEW



User Data Protections

Authorizing for data access via the filesystem

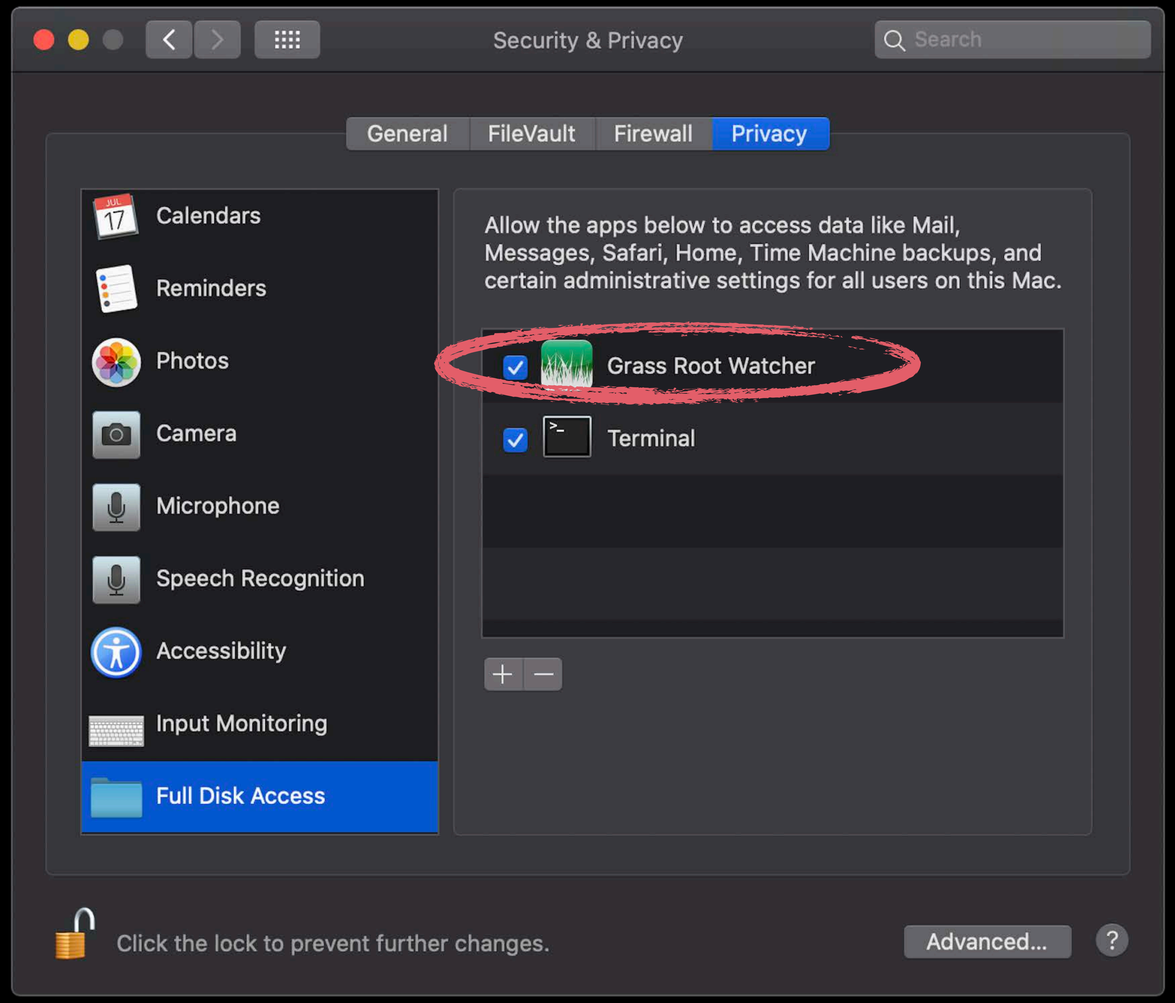
NEW



User Data Protections

Authorizing for data access via the filesystem

NEW



User Data Protections

Private data managed by the system

User Data Protections

Private data managed by the system

Data available via pre-approval for “Full Disk Access”

Test for authorization using fileManager API

Guide user to security and privacy preference pane if necessary

User Data Protections

Private data managed by the system

Mail

Messages

Safari browsing history

HTTP cookies

Call history

iTunes backups

Time machine backups

User Data Protections

Private data managed by the system

NEW

Mail

Messages

Safari browsing history

HTTP cookies

Call history

iTunes backups

Time machine backups

User Data Protections

Private data managed by the system

NEW

Mail

Messages

Safari browsing history

HTTP cookies

Call history

iTunes backups

Time machine backups

Trash

User Data Protections

Private data managed by the system

NEW

Mail

Messages

Safari browsing history

HTTP cookies

Call history

iTunes backups

Time machine backups

Trash

User Data Protections

Protecting your trash

FileManager

```
func trashItem(at url: URL,
               resultingItemURL outResultingURL: AutoreleasingUnsafeMutablePointer<NSURL?>?)
```

NSWorkspace

```
func recycle(_ URLs: [URL], completionHandler handler: (([URL : URL], Error?) -> Void)? = nil)
```

User Data Protections

Protecting your trash

FileManager

```
func trashItem(at url: URL,
               resultingItemURL outResultingURL: AutoreleasingUnsafeMutablePointer<NSURL?>?)
```

Do not need Full Disk Access to move a file to the trash

Just need authorization to the file being moved

User Data Protections

Protecting your trash

FileManager

```
func trashItem(at url: URL,
               resultingItemURL outResultingURL: AutoreleasingUnsafeMutablePointer<NSURL?>?)
```

Do not need Full Disk Access to move a file to the trash

Just need authorization to the file being moved

User Data Protections

Protecting your trash

FileManager

```
func trashItem(at url: URL,
               resultingItemURL outResultingURL: AutoreleasingUnsafeMutablePointer<NSURL?>?)
```

Do not need Full Disk Access to move a file to the trash

Just need authorization to the file being moved

Caller retains access to the file, even once it is in the trash

User Data Protections

Protecting your trash

FileManager

```
func trashItem(at url: URL,
               resultingItemURL outResultingURL: AutoreleasingUnsafeMutablePointer<NSURL?>?)
```

Do not need Full Disk Access to move a file to the trash

Just need authorization to the file being moved

Caller retains access to the file, even once it is in the trash

User Data Protections

Recording capabilities

Files and folders

- Data that requires user consent to access
- Private data managed by the system

Automation

User Privacy Protections

Recording capabilities

Files and folders

- Data that requires user consent to access
- Private data managed by the system

Automation

Automation Authorization

Automation Authorization

Synthetic input events

Apple Events

Automation Authorization

Synthetic input events

Automation Authorization

Synthetic input events

Governs ability to synthesize mouse clicks or key presses

Important to prevent malware from clicking through security dialogs

Automation Authorization

Synthetic input events

Important to prevent malware from clicking through security dialogs

```
let spaceKey = CGKeyCode(kVK_Space)
let keyDown = CGEvent(keyboardEventSource: nil, virtualKey: spaceKey, keyDown: true)!
let keyUp = CGEvent(keyboardEventSource: nil, virtualKey: spaceKey, keyDown: true)!

keyDown.post(tap: .cghidEventTap)
keyUp.post(tap: .cghidEventTap)
```

Automation Authorization

Synthetic input events

Important to prevent malware from clicking through security dialogs

```
let spaceKey = CGKeyCode(kVK_Space)
let keyDown = CGEvent(keyboardEventSource: nil, virtualKey: spaceKey, keyDown: true)!
let keyUp = CGEvent(keyboardEventSource: nil, virtualKey: spaceKey, keyDown: true)!

keyDown.post(tap: .cghidEventTap)
keyUp.post(tap: .cghidEventTap)
```

Automation Authorization

Synthetic input events

Important to prevent malware from clicking through security dialogs

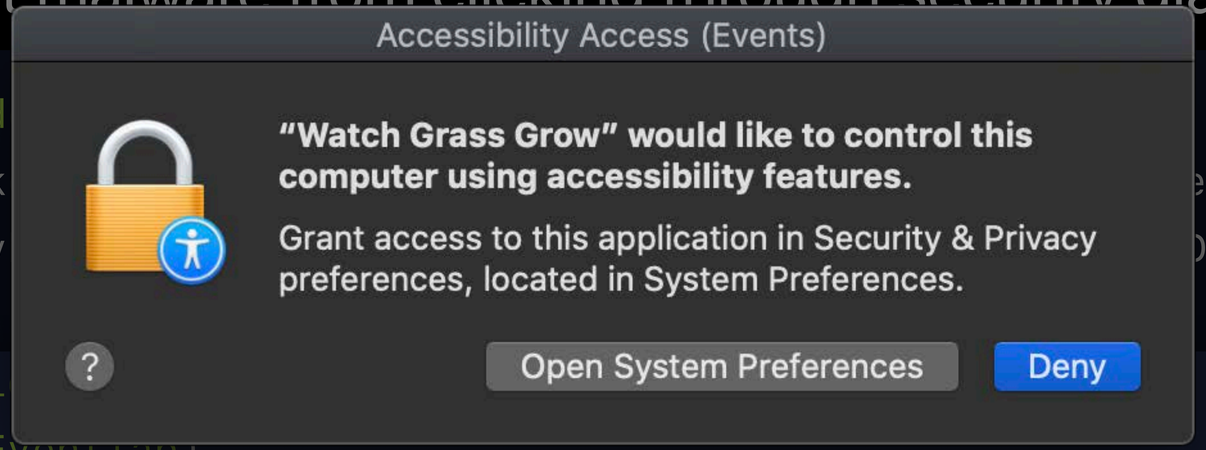
```
let spaceKey = CGKeyCode(kVK_Space)
let keyDown = CGEvent(keyboardEventSource: nil, virtualKey: spaceKey, keyDown: true)!
let keyUp = CGEvent(keyboardEventSource: nil, virtualKey: spaceKey, keyDown: true)!

keyDown.post(tap: .cghidEventTap)
keyUp.post(tap: .cghidEventTap)
```

Automation Authorization

Synthetic input events

Important to prevent malware from clicking through security dialogs

```
let spaceKey = CGKeyCod
let keyDown = CGEvent(k                                    eyDown: true)!
let keyUp = CGEvent(key                                    Down: true)!

keyDown.post(tap: .cghi
keyUp.post(tap: .cghidEventTap)
```



```
// User Data Protections – Listening to Keyboard Events in the Background

func callback(proxy: CGEventTapProxy, type: CGEventType, event: CGEvent,
              userInfo: UnsafeMutableRawPointer?) -> Unmanaged<CGEvent>? {
    // Do something with the event.
    return Unmanaged.passUnretained(event)
}

let eventMask = (1 << CGEventType.keyDown.rawValue) | (1 << CGEventType.keyUp.rawValue)
let eventTap = CGEvent.tapCreate(tap: .cghidEventTap,
                                 place: .tailAppendEventTap,
                                 options: .listenOnly,
                                 eventsOfInterest: CGEventMask(eventMask),
                                 callback: callback,
                                 userInfo: nil)
```

```
// User Data Protections – Listening to Keyboard Events in the Background

func callback(proxy: CGEventTapProxy, type: CGEventType, event: CGEvent,
              userInfo: UnsafeMutableRawPointer?) -> Unmanaged<CGEvent>? {
    // Do something with the event.
    return Unmanaged.passUnretained(event)
}

let eventMask = (1 << CGEventType.keyDown.rawValue) | (1 << CGEventType.keyUp.rawValue)
let eventTap = CGEvent.tapCreate(tap: .cghidEventTap,
                                 place: .tailAppendEventTap,
                                 options: .listenOnly,
                                 eventsOfInterest: CGEventMask(eventMask),
                                 callback: callback,
                                 userInfo: nil)
```

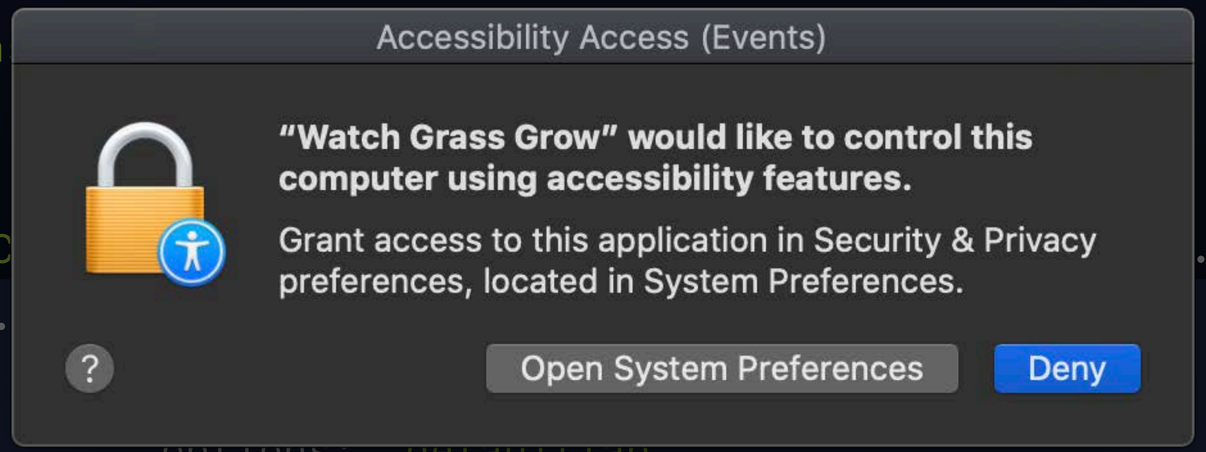
```swift
// User Data Protections – Listening to Keyboard Events in the Background

func callback(proxy: CGEventTapProxy, type: CGEventType, event: CGEvent,
              userInfo: UnsafeMutableRawPointer?) -> Unmanaged<CGEvent>? {
    // Do something with the event.
    return Unmanaged.passUnretained(event)
}

let eventMask = (1 << CGEventType.keyDown.rawValue) | (1 << CGEventType.keyUp.rawValue)
let eventTap = CGEvent.tapCreate(tap: .cghidEventTap,
                                 place: .tailAppendEventTap,
                                 options: .listenOnly,
                                 eventsOfInterest: CGEventMask(eventMask),
                                 callback: callback,
                                 userInfo: nil)
```

```swift
// User Data Protections — Synthesizing Keyboard Events

func callback(proxy: CGEventTapProxy, type: CGEventType, event: CGEvent,
              userInfo: UnsafeMutableRawPointer?) -> Unmanaged<CGEvent>? {
    // Do something with the event.
    return Unmanaged.passUnretained(event)
}

let eventMask = (1 << CGEventType.keyDown.rawValue) | (1 << CGEventType.keyUp.rawValue)
let eventTap = CGEvent.tapCreate(tap: .cghidEventTap,
                                 place: .tailAppendEventTap,
                                 options: .defaultTap,
                                 eventsOfInterest: CGEventMask(eventMask),
                                 callback: callback,
                                 userInfo: nil)
```

```
// User Data Protections — Synthesizing Keyboard Events

func callback(proxy: CGEventTapProxy, type: CGEventType, event: CGEvent,
              userInfo: UnsafeMutableRawPointer?) -> Unmanaged<CGEvent>? {
    // Do something with the event.
    return Unmanaged.passUnretained(event)
}

let eventMask = (1 << CGEventType.keyDown.rawValue) | (1 << CGEventType.keyUp.rawValue)
let eventTap = CGEvent.tapCreate(tap: .cghidEventTap,
                                 place: .tailAppendEventTap,
                                 options: .defaultTap,
                                 eventsOfInterest: CGEventMask(eventMask),
                                 callback: callback,
                                 userInfo: nil)
```

```
// User Data Protections — Synthesizing Keyboard Events

func callback(proxy: CGEventTapProxy, type: CGEventType, event: CGEvent,
              userInfo: UnsafeMutableRawPointer?) -> Unmanaged<CGEvent>? {
    // Do something with the event.
    return Unmanaged.passUnretained(event)
}

let eventMask = (1 << CGEventType.keyDown.rawValue) | (1 << CGEventType.keyUp.rawValue)
let eventTap = CGEvent.tapCreate(tap: .cghidEventTap,
                                 place: .tailAppendEventTap,
                                 options: .defaultTap,
                                 eventsOfInterest: CGEventMask(eventMask),
                                 callback: callback,
                                 userInfo: nil)
```

```
// User Data Protections – Synthesizing Keyboard Events

func callback(proxy: CGEventTapProxy, type: CGEventType, event: CGEvent,
              userInfo: UnsafeMutableRawPointer?) -> Unmanaged<CGEvent>? {
    // Do something with the event.
    return Unmanaged.pa
}

let eventMask = (1 << C                                    .keyUp.rawValue)
let eventTap = CGEvent.
                                 options: .defaultTap,
                    eventsOfInterest: CGEventMask(eventMask),
                             callback: callback,
                             userInfo: nil)
```



Automation Authorization

Synthetic input events

NEW

```
let accessType = IOHIDCheckAccess(kIOHIDRequestTypePostEvent)
switch accessType {
    case kIOHIDAccessTypeGranted:
        // User has approved the app to generate keystrokes or move the mouse pointer.
        …
    case kIOHIDAccessTypeDenied:
        // Denied; approval dialog has been displayed.
        …
    case kIOHIDAccessTypeUnknown:
        // Denied; approval dialog has not yet been displayed.
        …
    default:
        // Unknown status; assume denied.
        …
}
```

Automation Authorization

Synthetic input events

NEW

```
let accessType = IOHIDCheckAccess(kIOHIDRequestTypePostEvent)
switch accessType {
    case kIOHIDAccessTypeGranted:
        // User has approved the app to generate keystrokes or move the mouse pointer.
        …
    case kIOHIDAccessTypeDenied:
        // Denied; approval dialog has been displayed.
        …
    case kIOHIDAccessTypeUnknown:
        // Denied; approval dialog has not yet been displayed.
        …
    default:
        // Unknown status; assume denied.
        …
}
```

Automation Authorization

Synthetic input events

Apple Events

Automation Authorization

Synthetic input events

Apple Events

Automation Authorization

Apple Events

User authorization required to automate apps via Apple Events

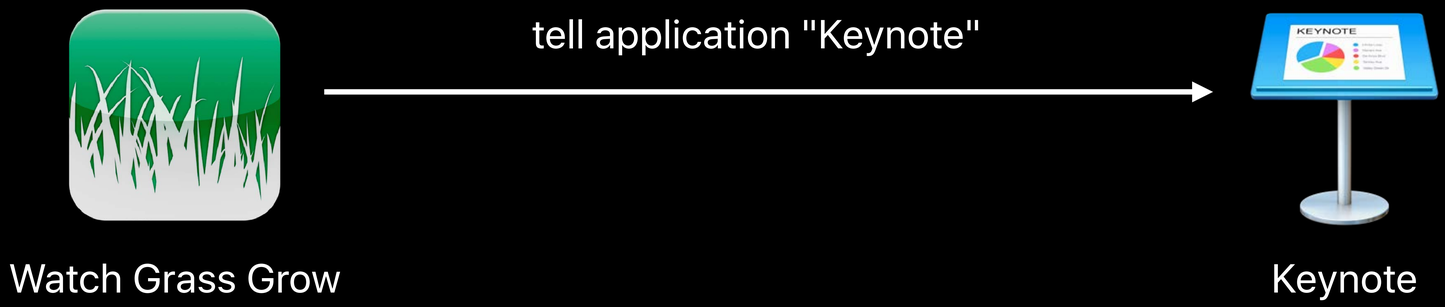
Watch Grass Grow

Keynote

Automation Authorization

Apple Events

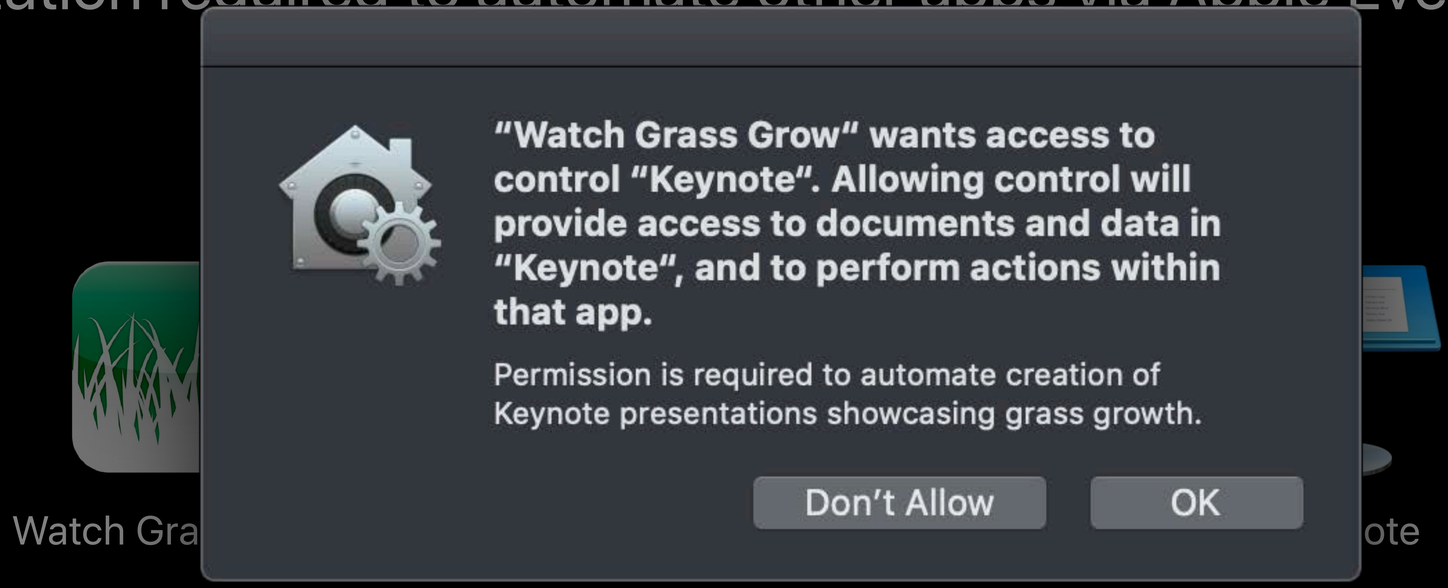
User authorization required to automate apps via Apple Events



Automation Authorization

Apple Events

User authorization required to automate other apps via Apple Events



Automation Authorization

Apple Events

Exceptions for events that don’t expose privacy-sensitive data to sender.

Examples:

```
NSWorkspace.shared.hideOtherApplications()
NSWorkspace.shared.activateFileViewerSelecting([URL(string: "/etc/hosts")!])
NSWorkspace.shared.launchApplication("TextEdit")
NSWorkspace.shared.openFile("/var/log/system.log", withApplication: "TextEdit")
NSWorkspace.shared.open(URL(string: "https://developer.apple.com/wwdc/")!)
NSWorkspace.shared.selectFile("/etc/hosts", inFileViewerRootedAtPath: "/etc")
```

Automation Authorization

Apple Events

API for querying approval status

```
func AEDeterminePermissionToAutomateTarget(_ target: UnsafePointer<AEAddressDesc>!,
                                           _ theAEEventClass: AEEventClass,
                                           _ theAEEventID: AEEventID,
                                           _ askUserIfNeeded: Bool) -> OSStatus
```

Automation Authorization

Apple Events

Example: Is caller approved to automate Keynote?

```
let target = NSAppleEventDescriptor(bundleIdentifier: "com.apple.iWork.Keynote")
let permission = AEDeterminePermissionToAutomateTarget(target.aeDesc,
                                                       typeWildCard, typeWildCard, false)
```

Automation Authorization

Apple Events

Example: Is caller approved to automate Keynote?

```
let target = NSAppleEventDescriptor(bundleIdentifier: "com.apple.iWork.Keynote")
let permission = AEDeterminePermissionToAutomateTarget(target.aeDesc,
                                                       typeWildCard, typeWildCard, true)
```

Automation Authorization

Apple Events

Example: Is caller approved to automate Keynote?

```
let target = NSAppleEventDescriptor(bundleIdentifier: "com.apple.iWork.Keynote")
let permission = AEDeterminePermissionToAutomateTarget(target.aeDesc,
                                                       typeWildCard, typeWildCard, true)
```

Automation Authorization

Apple Events

Example: Is caller approved to automate Keynote?

```
let target = NSAppleEventDescriptor(bundleIdentifier: "com.apple.iWork.Keynote")
let permission = AEDeterminePermissionToAutomateTarget(target.aeDesc,
                                                       typeWildCard, typeWildCard, true)
```

Automation Authorization

Apple Events

```
switch permission {
    case noErr:
        // User has previously approved automation.
    case OSStatus(errAEEventNotPermitted):
        // No purpose string or user has previously denied automation.
    case OSStatus(errAEEventWouldRequireUserConsent):
        // Status unknown: would require authorization prompt.
    case OSStatus(procNotFound):
        // Status unknown: target app not running.
    default:
        let error = NSError(domain: NSOSStatusErrorDomain, code: Int(permission),
                            userInfo: nil)
        // Handle error
}
```

Automation Authorization

Apple Events

```
switch permission {
    case noErr:
        // User has previously approved automation.
    case OSStatus(errAEEventNotPermitted):
        // No purpose string or user has previously denied automation.
    case OSStatus(errAEEventWouldRequireUserConsent):
        // Status unknown: would require authorization prompt.
    case OSStatus(procNotFound):
        // Status unknown: target app not running.
    default:
        let error = NSError(domain: NSOSStatusErrorDomain, code: Int(permission),
                            userInfo: nil)
        // Handle error
}
```

Automation Authorization

Apple Events

```
switch permission {
    case noErr:
        // User has previously approved automation.
    case OSStatus(errAEEventNotPermitted):
        // No purpose string or user has previously denied automation.
    case OSStatus(errAEEventWouldRequireUserConsent):
        // Status unknown: would require authorization prompt.
    case OSStatus(procNotFound):
        // Status unknown: target app not running.
    default:
        let error = NSError(domain: NSOSStatusErrorDomain, code: Int(permission),
                            userInfo: nil)
        // Handle error
}
```

Automation Authorization

Apple Events

```
switch permission {
    case noErr:
        // User has previously approved automation.
    case OSStatus(errAEEventNotPermitted):
        // No purpose string or user has previously denied automation.
    case OSStatus(errAEEventWouldRequireUserConsent):
        // Status unknown: would require authorization prompt.
    case OSStatus(procNotFound):
        // Status unknown: target app not running.
    default:
        let error = NSError(domain: NSOSStatusErrorDomain, code: Int(permission),
                            userInfo: nil)
        // Handle error
}
```

Automation Authorization

Apple Events

```
switch permission {
    case noErr:
        // User has previously approved automation.
    case OSStatus(errAEEventNotPermitted):
        // No purpose string or user has previously denied automation.
    case OSStatus(errAEEventWouldRequireUserConsent):
        // Status unknown: would require authorization prompt.
    case OSStatus(procNotFound):
        // Status unknown: target app not running.
    default:
        let error = NSError(domain: NSOSStatusErrorDomain, code: Int(permission),
                            userInfo: nil)
        // Handle error
}
```

Automation Authorization

Apple Events

```
switch permission {
    case noErr:
        // User has previously approved automation.
    case OSStatus(errAEEventNotPermitted):
        // No purpose string or user has previously denied automation.
    case OSStatus(errAEEventWouldRequireUserConsent):
        // Status unknown: would require authorization prompt.
    case OSStatus(procNotFound):
        // Status unknown: target app not running.
    default:
        let error = NSError(domain: NSOSStatusErrorDomain, code: Int(permission),
                            userInfo: nil)
        // Handle error
}
```

Automation Authorization

Apple Events

```
switch permission {
    case noErr:
        // User has previously approved automation.
    case OSStatus(errAEEventNotPermitted):
        // No purpose string or user has previously denied automation.
    case OSStatus(errAEEventWouldRequireUserConsent):
        // Status unknown: would require authorization prompt.
    case OSStatus(procNotFound):
        // Status unknown: target app not running.
    default:
        let error = NSError(domain: NSOSStatusErrorDomain, code: Int(permission),
                            userInfo: nil)
        // Handle error
}
```

Automation Authorization

Apple Events

```
switch permission {
    case noErr:
        // User has previously approved automation.
    case OSStatus(errAEEventNotPermitted):
        // No purpose string or user has previously denied automation.
    case OSStatus(errAEEventWouldRequireUserConsent):
        // Status unknown: would require authorization prompt.
    case OSStatus(procNotFound):
        // Status unknown: target app not running.
    default:
        let error = NSError(domain: NSOSStatusErrorDomain, code: Int(permission),
                            userInfo: nil)
        // Handle error
}
```

Automation Authorization

Apple Events

```
switch permission {
    case noErr:
        // User has previously approved automation.
    case OSStatus(errAEEventNotPermitted):
        // No purpose string or user has previously denied automation.
    case OSStatus(errAEEventWouldRequireUserConsent):
        // Status unknown: would require authorization prompt.
    case OSStatus(procNotFound):
        // Status unknown: target app not running.
    default:
        let error = NSError(domain: NSOSStatusErrorDomain, code: Int(permission),
                            userInfo: nil)
        // Handle error
}
```

Automation Authorization

Apple Events

```
switch permission {
    case noErr:
        // User has previously approved automation.
    case OSStatus(errAEEventNotPermitted):
        // No purpose string or user has previously denied automation.
    case OSStatus(errAEEventWouldRequireUserConsent):
        // Status unknown: would require authorization prompt.
    case OSStatus(procNotFound):
        // Status unknown: target app not running.
    default:
        let error = NSError(domain: NSOSStatusErrorDomain, code: Int(permission),
                            userInfo: nil)
        // Handle error
}
```

Automation Authorization

Apple Events

```
switch permission {
    case noErr:
        // User has previously approved automation.
    case OSStatus(errAEEventNotPermitted):
        // No purpose string or user has previously denied automation.
    case OSStatus(errAEEventWouldRequireUserConsent):
        // Status unknown: would require authorization prompt.
    case OSStatus(procNotFound):
        // Status unknown: target app not running.
    default:
        let error = NSError(domain: NSOSStatusErrorDomain, code: Int(permission),
                            userInfo: nil)
        // Handle error
}
```

User Privacy Protections

Recording capabilities

Files and folders

- Data that requires user consent to access
- Private data managed by the system

Automation

New User Privacy Protections in Catalina

Summary

NEW

Screen recording

Keyboard input monitoring

Common document locations

New User Privacy Protections in Catalina

MDM support

NEW

`ScreenCapture`	Screen Recording. Access may not be granted, only denied.
`ListenEvent`	Keyboard Input Monitoring. Access may not be granted, only denied.
`SystemPolicyDesktopFolder`	Desktop Folder
`SystemPolicyDocumentsFolder`	Documents Folder
`SystemPolicyDownloadsFolder`	Downloads Folder
`SystemPolicyRemovableVolumes`	Removable Volumes
`SystemPolicyNetworkVolumes`	Network Volumes

Summary

Sign and notarize all the software you distribute

Do not modify signed bundles

Be aware of user consent requirements

Handle user’s data with care

Summary

Sign and notarize all the software you distribute

Do not modify signed bundles

Be aware of user consent requirements

Handle user’s data with care

More Information

developer.apple.com/wwdc19/701

System Extensions and DriverKit	Tuesday, 10:00
Designing for Privacy	Wednesday, 2:00
Cryptography and Your Apps	Wednesday, 3:00


WWDC19