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
What does Gatekeeper check?
Gatekeeper
macOS Mojave

What does Gatekeeper check?
• Does it contain known malicious content?
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?
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
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

<table>
<thead>
<tr>
<th>Feature</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Malicious content scan</td>
<td>No known malicious content</td>
</tr>
<tr>
<td>Signature check</td>
<td>No tampering</td>
</tr>
<tr>
<td>Local policy check</td>
<td>Must be signed with Developer ID certificate</td>
</tr>
<tr>
<td>First launch prompt</td>
<td>User must approve</td>
</tr>
<tr>
<td>Feature</td>
<td>Status</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>Malicious content scan</td>
<td>No known malicious content</td>
</tr>
<tr>
<td>Signature check</td>
<td>No tampering</td>
</tr>
<tr>
<td>Local policy check</td>
<td>Must be signed with Developer ID certificate.</td>
</tr>
<tr>
<td>New Mac developers’ software requires notarization.</td>
<td></td>
</tr>
<tr>
<td>First launch prompt</td>
<td>User must approve</td>
</tr>
</tbody>
</table>
## Gatekeeper macOS Catalina

### First use, quarantined

<table>
<thead>
<tr>
<th>Feature</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Malicious content scan</td>
<td>No known malicious content</td>
</tr>
<tr>
<td>Signature check</td>
<td>No tampering</td>
</tr>
<tr>
<td>Local policy check</td>
<td>All new software requires notarization</td>
</tr>
<tr>
<td>First launch prompt</td>
<td>User must approve</td>
</tr>
<tr>
<td>Feature</td>
<td>macOS Catalina</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>Malicious content scan</td>
<td>No known malicious content</td>
</tr>
<tr>
<td>Signature check</td>
<td>No tampering</td>
</tr>
<tr>
<td>Local policy check</td>
<td>All new software requires notarization</td>
</tr>
<tr>
<td>First launch prompt</td>
<td>User must approve</td>
</tr>
</tbody>
</table>
### Gatekeeper

**macOS Catalina**

<table>
<thead>
<tr>
<th>Feature</th>
<th>First use, quarantined</th>
<th>First use, quarantined</th>
<th>Non-quarantined</th>
</tr>
</thead>
<tbody>
<tr>
<td>Malicious content scan</td>
<td>No known malicious content</td>
<td>No known malicious content</td>
<td>No known malicious content</td>
</tr>
<tr>
<td>Signature check</td>
<td>No tampering</td>
<td>No tampering</td>
<td>–</td>
</tr>
<tr>
<td>Local policy check</td>
<td>All new software requires notarization</td>
<td>All new software requires notarization</td>
<td>–</td>
</tr>
<tr>
<td>First launch prompt</td>
<td>User must approve</td>
<td>Users must approve</td>
<td>–</td>
</tr>
</tbody>
</table>

NEW
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

“Cisco Jabber” would like to access the microphone.

[Buttons: Don’t Allow, OK]
Recording Protections in Catalina

Camera

Microphone
Recording Protections in Catalina

Camera

Microphone

Screen recording

Keyboard input monitoring
Recording Protections in Catalina
Screen recording and keyboard monitoring

Important to prevent apps from recording
• Contact information
• Private correspondence
• Account names or numbers
• Passwords
• And more
Recording Protections in Catalina

Camera
Microphone
Screen recording
Keyboard input monitoring
Recording Protections in Catalina
Screen recording

Recording the entire screen

```swift
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

Recording the entire screen

```swift
guard let stream = CGDisplayStream(displayQueueDisplay: 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

Recording the entire screen

guard let stream = CGDisplayStream(dispatchQueueDisplay: CGMainDisplayID(),
    outputWidth: 1920,
    outputHeight: 1080,
    pixelFormat: Int32(kCVPixelFormatType_32BGRA),
    queue: DispatchQueue.global(),
    handler: frameHandler)

else {
    // Error occurred or user has not approved the app to record the screen.
    return
}

stream.start()
Recording the entire screen

```swift
guard let stream = CGDisplayStream(displayID: 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
Recording Protections in Catalina

Approving for screen recording
Recording Protections in Catalina

Approving for screen recording
Recording Protections in Catalina

Screen recording

Recording a window’s contents

```swift
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

Recording a window’s contents

```swift
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

Recording a window’s contents

```swift
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

Recording a window’s contents

```swift
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

Recording a window’s contents

```swift
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

Recording a window’s contents

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

```swift
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)
}
```

“Watch Grass Grow” would like to record this computer’s screen.

Grant access to this application in Security & Privacy preferences, located in System Preferences.
Recording Protections in Catalina
Screen recording

No approval necessary to query metadata about windows

```swift
let windows = CGWindowListCopyWindowInfo([.optionOnScreenOnly], kCGNullWindowID) as? [[String: AnyObject]]
```
Recording Protections in Catalina
Screen recording

No approval necessary to query metadata about windows

```swift
let windows = CGWindowListCopyWindowInfo([.optionOnScreenOnly], kCGNullWindowID) as? [[String: AnyObject]]
```
Recording Protections in Catalina
Screen recording

No approval necessary to query metadata about windows

```swift
let windows = CGWindowListCopyWindowInfo([.optionOnScreenOnly], kCGNullWindowID) as? [[String: AnyObject]]
```
Recording Protections in Catalina
Screen recording

No approval necessary to query metadata about windows

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

- kCGWindowBounds
- kCGWindowNumber
- kCGWindowOwnerName
- kCGWindowOwnerPID
Recording Protections in Catalina

Screen recording

No approval necessary to query metadata about windows

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

- `kCGWindowBounds`
- `kCGWindowNumber`
- `kCGWindowOwnerName`
- `kCGWindowOwnerPID`
Recording Protections in Catalina
Screen recording

No approval necessary to query metadata about windows

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

- kCGWindowBounds
- kCGWindowNumber
- kCGWindowOwnerName
- kCGWindowOwnerPID
- kCGWindowName
- kCGWindowSharingState
// 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
    }
}
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
    }
}
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
    }
}
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
    }
}
// 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
    }
}
// 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
    }
}
/ Screen Recording Protections - Get Desktop Background Windows

```swift
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
    }
}
```
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
    }
}
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

- Camera
- Microphone
- Screen recording
- Keyboard input monitoring
Recording Protections in Catalina

Camera
Microphone
Screen recording
Keyboard input monitoring
Recording Protections

Keyboard input monitoring

No approval necessary to monitor events for own app

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

Keyboard input monitoring

No approval necessary to monitor events for own app

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

let eventTap = CGEventTap

Recording Protections

Approving for keyboard input monitoring
Recording Protections

Approving for keyboard input monitoring

[Image of a screenshot showing the Input Monitoring option in the Security & Privacy settings on a Mac]
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.
        ...
}
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

```c
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

```c
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

“Watch Grass Grow” would like to access your photos.

Your photo library is used to view & save photos of important grass-growing milestones.

Don’t Allow  OK
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

- Desktop
- Documents
- Downloads
- iCloud Drive
- Third-party cloud storage
- Removable volumes
- Network volumes
User Data Protections
Data that requires user consent to access

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

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

- 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

- 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

- 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

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

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

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

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
User Data Protections
Minimizing consent prompts by inferring user intent

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

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

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

<table>
<thead>
<tr>
<th>User Consent</th>
<th>User Intent</th>
</tr>
</thead>
</table>

NEW
# User Data Protections

Minimizing consent prompts by inferring user intent

<table>
<thead>
<tr>
<th>User Consent</th>
<th>User Intent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reactive</td>
<td>Proactive</td>
</tr>
</tbody>
</table>
## User Data Protections
Minimizing consent prompts by inferring user intent

<table>
<thead>
<tr>
<th>User Consent</th>
<th>User Intent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reactive</td>
<td>Proactive</td>
</tr>
</tbody>
</table>
# User Data Protections

Minimizing consent prompts by inferring user intent

<table>
<thead>
<tr>
<th>User Consent</th>
<th>User Intent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reactive</td>
<td>Proactive</td>
</tr>
<tr>
<td>Prompt interrupts workflow</td>
<td>Inferred transparently from workflow</td>
</tr>
</tbody>
</table>
### User Data Protections
Minimizing consent prompts by inferring user intent

<table>
<thead>
<tr>
<th>User Consent</th>
<th>User Intent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reactive</td>
<td>Proactive</td>
</tr>
<tr>
<td>Prompt interrupts workflow</td>
<td>Inferred transparently from workflow</td>
</tr>
</tbody>
</table>
# User Data Protections

Minimizing consent prompts by inferring user intent

<table>
<thead>
<tr>
<th>User Consent</th>
<th>User Intent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reactive</td>
<td>Proactive</td>
</tr>
<tr>
<td>Prompt interrupts workflow</td>
<td>Inferred transparently from workflow</td>
</tr>
<tr>
<td>Applies to whole class of data</td>
<td>Applies to just the selected files</td>
</tr>
</tbody>
</table>
## User Data Protections
Minimizing consent prompts by inferring user intent

<table>
<thead>
<tr>
<th>User Consent</th>
<th>User Intent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reactive</td>
<td>Proactive</td>
</tr>
<tr>
<td>Prompt interrupts workflow</td>
<td>Inferred transparently from workflow</td>
</tr>
<tr>
<td>Applies to whole class of data</td>
<td>Applies to just the selected files</td>
</tr>
</tbody>
</table>
# User Data Protections

Minimizing consent prompts by inferring user intent

<table>
<thead>
<tr>
<th>User Consent</th>
<th>User Intent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reactive</td>
<td>Proactive</td>
</tr>
<tr>
<td>Prompt interrupts workflow</td>
<td>Inferred transparently from workflow</td>
</tr>
<tr>
<td>Applies to whole class of data</td>
<td>Applies to just the selected files</td>
</tr>
</tbody>
</table>
## User Data Protections
Minimizing consent prompts by inferring user intent

<table>
<thead>
<tr>
<th>User Consent</th>
<th>User Intent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prompt interrupts workflow</td>
<td>Inferred transparently from workflow</td>
</tr>
<tr>
<td>Applies to whole class of data</td>
<td>Applies to just the selected files</td>
</tr>
</tbody>
</table>
## User Data Protections
Minimizing consent prompts by inferring user intent

<table>
<thead>
<tr>
<th>User Consent</th>
<th>User Intent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reactive</td>
<td>Proactive</td>
</tr>
<tr>
<td>Prompt interrupts workflow</td>
<td>Inferred transparently from workflow</td>
</tr>
<tr>
<td>Applies to whole class of data</td>
<td>Applies to just the selected files</td>
</tr>
</tbody>
</table>
User Data Protections
Accessing sidecar files

```xml
<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

Open and save panels are now hosted out-of-process
User Data Protections
NSOpenPanel and NSSavePanel changes

Open and save panels are now hosted out-of-process
User Data Protections
NSOpenPanel and NSSavePanel changes

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

Class inheritance and view hierarchies have changed
User Data Protections
NSOpenPanel and NSSavePanel changes

Open and save panels are now hosted out-of-process
Class inheritance and view hierarchies have changed
User Data Protections
NSOpenPanel and NSSavePanel changes

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

NSOpenSavePanelDelegate

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

Cannot rewrite the user’s selection
User Data Protections
NSOpenSavePanelDelegate changes

NSOpenSavePanelDelegate

```swift
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

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

BSD

```c
int access(const char *path, int mode)
```
User Data Protections
APIs for testing filesystem authorization

FileManager

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

BSD

```c
int access(const char *path, int mode)
```
# User Data Protections

## Purpose string keys

<table>
<thead>
<tr>
<th>Folder</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desktop folder</td>
<td>NSDesktopFolderUsageDescription</td>
</tr>
<tr>
<td>Documents folder</td>
<td>NSDocumentsFolderUsageDescription</td>
</tr>
<tr>
<td>Downloads folder</td>
<td>NSDownloadsFolderUsageDescription</td>
</tr>
<tr>
<td>iCloud Drive or Third-party Cloud Provider</td>
<td>NSFileProviderDomainUsageDescription</td>
</tr>
<tr>
<td>Removable volumes</td>
<td>NSRemovableVolumesUsageDescription</td>
</tr>
<tr>
<td>Network volumes</td>
<td>NSNetworkVolumesUsageDescription</td>
</tr>
</tbody>
</table>
# User Data Protections

**Purpose string keys**

<table>
<thead>
<tr>
<th>Folder Description</th>
<th>Usage Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desktop folder</td>
<td>NSDesktopFolderUsageDescription</td>
</tr>
<tr>
<td>Documents folder</td>
<td>NSDocumentsFolderUsageDescription</td>
</tr>
<tr>
<td>Downloads folder</td>
<td>NSDownloadsFolderUsageDescription</td>
</tr>
<tr>
<td>iCloud Drive or Third-party Cloud Provider</td>
<td>NSFFileProviderDomainUsageDescription</td>
</tr>
<tr>
<td>Removable volumes</td>
<td>NSRemovableVolumesUsageDescription</td>
</tr>
<tr>
<td>Network volumes</td>
<td>NSNetworkVolumesUsageDescription</td>
</tr>
</tbody>
</table>
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

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

BSD

```c
int access(const char *path, int mode)
```
User Data Protections
Authorizing for data access via the filesystem
User Data Protections
Authorizing for data access via the filesystem
User Data Protections

Authorizing for data access via the filesystem
User Data Protections
Authorizing for data access via the filesystem

![Security & Privacy window with Full Disk Access permissions](image)
User Data Protections
Authorizing for data access via the filesystem

Click the lock to prevent further changes.
User Data Protections
Authorizing for data access via the filesystem

Security & Privacy

- Calendars
- Reminders
- Photos
- Camera
- Microphone
- Speech Recognition
- Accessibility
- Input Monitoring
- Full Disk Access

Allow the apps below to access data like Mail, Messages, Safari, Home, Time Machine backups, and certain administrative settings for all users on this Mac.

- com.example.wgg-helper
  - Terminal

Click the lock to prevent further changes.
User Data Protections
Authorizing for data access via the filesystem

![Security & Privacy settings window showing Full Disk Access permissions for Grass Root Watcher and Terminal apps.](image-url)
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

Mail
Messages
Safari browsing history
HTTP cookies
Call history
iTunes backups
Time machine backups
User Data Protections
Private data managed by the system

Mail  Trash
Messages
Safari browsing history
HTTP cookies
Call history
iTunes backups
Time machine backups
User Data Protections
Private data managed by the system

Mail
Messages
Safari browsing history
HTTP cookies
Call history
iTunes backups
Time machine backups
Trash
User Data Protections
Protecting your trash

**FileManager**

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

**NSWorkspace**

```swift
func recycle(_ URLs: [URL], completionHandler handler: (([URL : URL], Error?) -> Void)? = nil)
```
User Data Protections
Protecting your trash

FileManager

```swift
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

```swift
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

```swift
func trashItem(at: URL,
    resultingItemURL: 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

Govern ability to synthesize mouse clicks or key presses

Important to prevent malware from clicking through security dialogs
Important to prevent malware from clicking through security dialogs

```swift
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)
```
Important to prevent malware from clicking through security dialogs

```swift
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)
```
Important to prevent malware from clicking through security dialogs

```swift
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)
```
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)
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

```swift
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)
// 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)
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)
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)
let accessType = IOHIDCheckAccess(kIOHIDRequestTypePostEvent)

switch accessType {
    case kIOHIDAcessTypeGranted:
        // User has approved the app to generate keystrokes or move the mouse pointer.
        ...
    case kIOHIDAcessTypeDenied:
        // Denied; approval dialog has been displayed.
        ...
    case kIOHIDAcessTypeUnknown:
        // Denied; approval dialog has not yet been displayed.
        ...
    default:
        // Unknown status; assume denied.
        ...
}

Automation Authorization
Synthetic input events
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

tell application "Keynote"

Watch Grass Grow → Keynote
Automation Authorization

Apple Events

User authorization required to automate other apps via Apple Events

"Watch Grass Grow" wants access to control "Keynote". Allowing control will provide access to documents and data in "Keynote", and to perform actions within that app.

Permission is required to automate creation of Keynote presentations showcasing grass growth.

Don’t Allow  OK
Automation Authorization
Apple Events

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

Examples:

```swift
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

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

Example: Is caller approved to automate Keynote?

```swift
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?

```swift
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?

```swift
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?

```swift
let target = NSAppleEventDescriptor(bundleIdentifier: "com.apple.iWork.Keynote")
let permission = AEDeterminePermissionToAutomateTarget(target.aeDesc,
                                                     typeWildCard, typeWildCard, true)
```
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: NSOSSErrorDomain, code: Int(permission), userInfo: nil)
        // Handle error
}
Automation Authorization
Apple Events

```swift
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

```swift
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

```swift
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
}
```
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
}
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
}
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
}
```swift
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
}
```
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
    }
}
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

• Screen recording
• Keyboard input monitoring
• Common document locations
New User Privacy Protections in Catalina

MDM support

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ScreenCapture</td>
<td>Screen Recording. Access may not be granted, only denied</td>
</tr>
<tr>
<td>ListenEvent</td>
<td>Keyboard Input Monitoring. Access may not be granted, only denied</td>
</tr>
<tr>
<td>SystemPolicyDesktopFolder</td>
<td>Desktop Folder</td>
</tr>
<tr>
<td>SystemPolicyDocumentsFolder</td>
<td>Documents Folder</td>
</tr>
<tr>
<td>SystemPolicyDownloadsFolder</td>
<td>Downloads Folder</td>
</tr>
<tr>
<td>SystemPolicyRemovableVolumes</td>
<td>Removable Volumes</td>
</tr>
<tr>
<td>SystemPolicyNetworkVolumes</td>
<td>Network Volumes</td>
</tr>
</tbody>
</table>
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
System Extensions and DriverKit  
Tuesday, 10:00

Designing for Privacy  
Wednesday, 2:00

Cryptography and Your Apps  
Wednesday, 3:00