Job(s) Bless Us!
Privileged Operations on macOS
@aronskaya

MacPaw
Software Engineer,
Anti-malware team,
Triage team

WWC Kyiv macOS Chapter Lead
Agenda

Intro to privileged operations API on macOS

First CleanMyMac's security issue, reported by TALOS

CleanMyMac on hackerone

Comparison of privileged operations implementation on and SETAPP

Summary & Takeaways
Intro to privileged operations API on macOS
High-level APIs

- SMJobBless()
- AuthorizationExecuteWithPrivileges()
High-level APIs

SMJobBless()

⚠️ AuthorizationExecuteWithPrivileges()

This function poses a security concern because it will indiscriminately run any tool or application, severely increasing the security risk. You should avoid the use of this function if possible. One alternative is to split your code into two parts—the application and a setuid tool.

macOS 10.1–10.7

Deprecated
High-level APIs

- **SMJobBless()**
  - **AuthorizationExecuteWithPrivileges()**

This function poses a security concern because it will indiscriminately run any tool or application, severely increasing the security risk. You should avoid the use of this function if possible. One alternative is to split your code into two parts—the application and a setuid tool.

Use a launchd-launched helper tool and/or the Service Management framework for this functionality.
There is no ‘UnBless’ 😞
Signing requirements

Client has requirements for Helper(s)

<!-- Arrows here -->

Helper has requirements for Client(s)

OS performs validation of the requirements ONLY on install & update of the Helper

⚠️ NO validation is performed on establishing XPC connection
1. Client has the Privileged Helper executable in the bundle

2. Signing requirements are met
   - Both client and Helper are signed
   - Privileged Helper has a plist file for launchd embedded into __TEXT section
   - Privileged Helper has Info.plist embedded
   - Client has signing requirements listed in its Info.plist
SMJobBless()

3. Obtain Authorization object:
call AuthorizationCreate()

```c
const AuthorizationRights *kNoRightsSpecified = NULL;
AuthorizationFlags flags = kAuthorizationFlagDefaults | 
                           kAuthorizationFlagInteractionAllowed | 
                           kAuthorizationFlagPreAuthorize | 
                           kAuthorizationFlagExtendRights;
self.lastErrorCode = AuthorizationCreate(kNoRightsSpecified,
                                       kAuthorizationEmptyEnvironment, flags, &authRef);
```

4. Call SMJobBless() with acquired Authorization object

```c
Boolean res = SMJobBless(kSMDomainSystemLaunchd,
                        label,
                        authRef,
                        &error);
```
SMJobBless()

5. OS validates code signing requirements in client and helper’s Info.plist and copies the executable from the bundle to /Library/PrivilegedHelperTools
5. Client can establish XPC connection to the Privileged Helper

```objective-c
17 - (BOOL)listener:(NSXPCLListener *)listener |shouldAcceptNewConnection:(NSXPCConnection *)newConnection
19 {
```
Apple’s Sample Code

<table>
<thead>
<tr>
<th>Title</th>
<th>Resource Type</th>
<th>Topic</th>
<th>Technology</th>
<th>Date</th>
</tr>
</thead>
</table>

Last change: New document.
Apple’s Sample Code

86  -(BOOL)listener:(NSXPCLListener *)listener shouldAcceptNewConnection:(NSXPCClone *)newConnection
87     // Called by our XPC listener when a new connection comes in. We configure the connection
88     // with our protocol and ourselves as the main object.
89     {
90         assert(listener == self.listener);
91         #pragma unused(listener)
92         assert(newConnection != nil);
93
94         newConnection.exportedInterface = [NSPCInterface interfaceWithProtocol:@protocol(HelperToolProtocol)];
95         newConnection.exportedObject = self;
96         [newConnection resume];
97
98         return YES;
99     }
Apple’s Sample Code

```c
- (BOOL)listener:(NSXPCListener *)listener shouldAcceptNewConnection:(NSXPCConnection *)newConnection
  // Called by our XPC listener when a new connection comes in. We configure the connection
  // with our protocol and ourselves as the main object.
  {
    assert(listener == self.listener);
    #pragma unused(listener)
    assert(newConnection != nil);

    newConnection.exportedInterface = [NSXPCInterface interfaceWithProtocol:@protocol(HelperToolProtocol)];
    newConnection.exportedObject = self;
    [newConnection resume];

    return YES;
  }
```
Issue #1

```objective-c
- (BOOL)listener:(NSXPCLlistener *)listener shouldAcceptNewConnection:(NSXPConnection *)newConnection

  // Called by our XPC listener when a new connection comes in. We configure the connection
  // with our protocol and ourselves as the main object.

  { //
    assert(listener == self.listener);
    #pragma unused(listener)
    assert(newConnection != nil);

    newConnection.exportedInterface = [NSXPCInterface interfaceWithProtocol:@protocol(HelperToolProtocol)];
    newConnection.exportedObject = self;
    [newConnection resume];

    return YES;
  }
```
First security issue, reported by Talos
Zero-Day Reports

• November 2018
<table>
<thead>
<tr>
<th>REPORT ID</th>
<th>TITLE</th>
<th>REPORT DATE</th>
<th>CVE NUMBER</th>
<th>CVSS SCORE</th>
</tr>
</thead>
<tbody>
<tr>
<td>TALOS-2018-0705</td>
<td>CleanMyMac X moveItemAtPath privilege escalation vulnerability</td>
<td>2019-01-02</td>
<td>CVE-2018-4032</td>
<td>7.1</td>
</tr>
<tr>
<td>TALOS-2018-0707</td>
<td>CleanMyMac X removeItemAtPath Privilege Escalation Vulnerability</td>
<td>2019-01-02</td>
<td>CVE-2018-4034</td>
<td>7.1</td>
</tr>
<tr>
<td>TALOS-2018-0706</td>
<td>CleanMyMac X moveToTrashItemAtPath privilege escalation vulnerability</td>
<td>2019-01-02</td>
<td>CVE-2018-4033</td>
<td>7.1</td>
</tr>
<tr>
<td>TALOS-2018-0709</td>
<td>CleanMyMac X removeKextAtPath privilege escalation vulnerability</td>
<td>2019-01-02</td>
<td>CVE-2018-4036</td>
<td>7.1</td>
</tr>
<tr>
<td>TALOS-2018-0710</td>
<td>CleanMyMac X removeDiagnosticLogs privilege escalation vulnerability</td>
<td>2019-01-02</td>
<td>CVE-2018-4037</td>
<td>7.1</td>
</tr>
</tbody>
</table>
Stumbled upon Talos’es Zero-Day reports

Contacted Talos for details, they answer the same day

We release a patched update v. 4.2.0

Talos reports insufficient fix

Tyler Bohan (Talos) delivers a talk at OffensiveCon19

We release a patch v. 4.3.0
Tyler Bohan: ‘OSX XPC Revisited - 3rd Party Application Flaws’ at OffensiveCon19

Clean My Mac X

- Optimization and malware detection tool
- No authentication or authorization checks
- Method list
  - "moveItemAtPath:toPath:withReply:"
  - "removeKextAtPath:withReply:"
  - "enableLaunchdAgentAtPath:withReply:"
  - "startStartupItem:withReply:"
  - "repairPermissionsWithReply:"
  - "runPeriodicScript:withReply:"
  - "removeDiagnosticLogsWithReply:"

https://www.youtube.com/watch?v=KPzhTqwf0bA
Tyler Bohan: ‘OSX XPC Revisited - 3rd Party Application Flaws’ at OffensiveCon19

https://www.youtube.com/watch?v=KPzhTqwf0bA
Fix

459  -(BOOL)listener:(NSXPCLListener *)listener shouldAcceptNewConnection:(NSXPCConnection *)newConnection
460  {
461     CLErrorRef errors = NULL;
462     SecCodeRef code = NULL;
463     SecRequirementRef requirement = NULL;
464
465     NSDictionary *attributes = @{
466         (__bridge_KeyedUnmanaged<NSString>)kSecGuestAttributePid:
467             @(connection.processIdentifier) },
468         ((__bridge(cfDictionaryRef))attributes,
469             kSecCSDefaultFlags, &code);
470
471     NSString *entitlement = @"anchor trusted and certificate leaf [subject.CN] = \
472         "Developer ID Application: MacPaw Inc. (AAAAAAAAAA)\";"
473
474     SecRequirementCreateWithStringAndErrors((__bridge(cfStringRef))entitlement,
475         kSecCSDefaultFlags, &errors, &requirement);
476
477     OSStatus status = SecCodeCheckValidity(code, kSecCSDefaultFlags, requirement);
478
479     if (errSecSuccess != status)
480     {
481         return NO;
482     }
Fix #1

459  - (BOOL)listener:(NSXPCLListener *)listener shouldAcceptNewConnection:(NSXPCConnection *)newConnection
460  {
461      CFErrorRef errors = NULL;
462      SecCodeRef code = NULL;
463      SecRequirementRef requirement = NULL;
464
465      NSDictionary *attributes = @
466          {
467              ((__bridge NSString *)kSecGuestAttributePid:
468                  @(connection.processIdentifier) ),
469              SecCodeCopyGuestWithAttributes(0, ((__bridge CFDictionaryRef)attributes,
470                  kSecCSDefaultFlags, &code);
471
472      NSString *entitlement = @"anchor trusted and certificate leaf [subject.CN] = "
473          \"Developer ID Application: MacPaw Inc. (AAAAAAAAAAA)\";
474
475      SecRequirementCreateWithStringAndErrors((__bridge CFStringRef)entitlement,
476          kSecCSDefaultFlags, &errors, &requirement);
477
478          OSStatus status = SecCodeCheckValidity(code, kSecCSDefaultFlags, requirement);
479
480              if (errSecSuccess != status)
481                  {
482                      return NO;
483                  }
484          }
on hackerone
MacPaw launched a private h1 program for our other product Setapp

CleanMyMac desktop client is added to the scope

March 2018

May 2019
Client’s requirements

<table>
<thead>
<tr>
<th>Tools owned after installation</th>
<th>Dictionary</th>
<th>(1 item)</th>
</tr>
</thead>
</table>

Bundle identifier

Signing identity (team id)
Client’s requirements

Privileged helper’s executable can be replaced with old version

Lack of validation for "SMJobBless"ed helper allows installing vulnerable CMMX agent's version under the latest CMMX version's identity [LPE2ROOT]
What’s the fuss about old versions?
What’s the fuss about old versions?

Hardened Runtime introduced in Mojave:
• libraries signing validation == protect from dylib injection
• remove get-task-allow from entitlements == protect from attaching with debugger
  (and other things)
Issue #2: steps

**Preconditions:** Privileged Helper is not authorized yet. A malicious executable is present on the user’s computer.

1. Download an app version, vulnerable to dylib injection

2. Replace the Privileged Helper executable in the installed app with the vulnerable one

3. User authorizes the Helper

4. Perform a dylib injection into the Helper—it is run as root!
What about code signing?

Replacing the Privileged Helper in the signed bundle doesn’t change anything, because

OS validates the signature only when app is quarantined

After the first launch no signature validation is performed on Mojave.

Time-to-time signature checks were announced in Catalina.
Fix #2

<table>
<thead>
<tr>
<th>Tools owned after installation</th>
<th>Dictionary</th>
<th>(1 item)</th>
</tr>
</thead>
</table>

Version check
Privileged Helper’s requirements

<table>
<thead>
<tr>
<th>▼ Clients allowed to add and remove tool</th>
<th>Array (1 item)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item 0</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>anchor apple generic and certificate leaf[subject.OU] = &quot;AAAAAAAAAAAAA&quot;</td>
</tr>
</tbody>
</table>

Signing identity (team id)
# Privileged Helper’s requirements

<table>
<thead>
<tr>
<th>▼Clients allowed to add and remove tool</th>
<th></th>
<th>Array</th>
<th>(1 item)</th>
</tr>
</thead>
</table>
| Item 0                              |  | String | anchor apple generic and certificate leaf[subject.OU] = "AAAAAAAAAAAA"

old 🐞 client versions can connect

[Vladimir Metnew (metnew)](#)

Unprivileged user can abuse privileged helper via previous codesigned vulnerable version of CMMX for LPE to root

- State: Resolved (Closed)
- Severity: Critical (9 – 10)
Issue #3: steps

**Preconditions:** Privileged Helper is authorized. A malicious executable is present on the user’s computer.

- Download an old app version, vulnerable to dylib injection
- Launch client executable with a dylib injection
- Call privileged helper’s methods from the injected code
  - In our case it leads to LPE to root
Issue #3: steps

**Preconditions:** Privileged Helper is authorized. A malicious executable is present on the user’s computer.

- Download an old app version, vulnerable to dylib injection
- Launch client executable with a dylib injection
- Call privileged helper’s methods from the injected code
  - In our case it leads to LPE to root

**Takeaway:** Dylib injection does NOT break the code signature
Fix #3

```objective-c
NSString *entitlement = @"anchor trusted and \
certificate leaf [subject.CN] = "Developer ID Application: MacPaw Inc. (AAAAAAAAAAA)" and \
info [CFBundleShortVersionString] >= "10.10.10";"
```
Privileged Helper’s requirements

- **Clients allowed to add and remove tool**: Array (1 item)
  - Item 0: String

  anchor apple generic and certificate leaf[subject.OU] = "AAAAAAAAAAA"

- **Unprivileged user can abuse privileged helper via previous codesigned vulnerable version of CMMX for LPE to root**

  Unprivileged user can abuse privileged helper via previous codesigned vulnerable version of CMMX for LPE to root

  State: Resolved (Closed)  Severity: Critical (9 – 10)  Edit

- **Signing identity (team id)**

- **Other apps of the same vendor can connect**

  Insufficient fix for #674518 potentially allows bypassing Agent’s checks via other MacPaw apps and leaked CMMX test builds

  State: Resolved (Closed)  Severity: Medium (4 – 6.9)  Edit
Fix #4

```objective-c
NSString *entitlement = @"anchor trusted and \
certificate leaf [subject.CN] = "Developer ID Application: MacPaw Inc. (AAAAAAAAAAA)" and \ninfo [CFBundleShortVersionString] >= "10.10.10" and \
identifier "com.macpaw.CleanMyMac4";```

SecCodeRef code = NULL;
NSDictionary *attributes = @{
    kSecGuestAttributePid: @(connection.processIdentifier) 
};
SecCodeCopyGuestWithAttributes(0, attributes, kSecCSDefaultFlags, &code);
anyone can impersonate the client due to pid checks logic performed by OS

Vladimir Metnew (metnew)

Arbitrary app can abuse privileged helper to gain root privileges due to racy checks by pid in validation logic of XPC messaging

State: Resolved (Closed) Severity: Critical (9 ~ 10)
Issue #5: anyone can impersonate the client due to racy pid checks performed by OS

```swift
NSDictionary *attributes = @{ (__bridge NSString *)&kSecGuestAttributePid:
    @(connection.processIdentifier) };

status = SecCodeCopyGuestWithAttributes(0,
    (__bridge CFDictionaryRef)attributes,
    kSecCSDefaultFlags,
    &code);
```
#import <Foundation/Foundation.h>
#import <xc/xc.h>
#import Darwin.POSIX.spawn;

extern char **environ;

int main()
{
    // selector `removeItemAtPath:withReply:`
    NSdata *someEncodedString = @"RyBsaXWMTag2gAAAAAAAAB/ERxy2V1vdvVJ3dGVtQXQRYXRo0ndpGhSZ8eseToA3ZADk8APwCgZgAAAAAABVERe3AhQAbQbwACB8aAB1A0wAbABVA4AbQBhAQMaCABH4H4A==" options:0;

    int RACE_COUNT = 100;
    int pids[RACE_COUNT];

    for (int i = 0; i < RACE_COUNT; i++) {
        int pid = fork();
        if (pid == 0) {
            xpc_connection_t connection = xpc_connection_create_mach_service("com.mcpaw.CleanMyMac4.Agent", NULL, XPC_CONNECTION_MACH_SERVICE_PRIVILEGED);
            // xpc connection and message setup boilerplate code
            xpc_connection_send_message(connection, message);

            // setup spawn boilerplate code
            posix_spawn(NULL, target_binary, NULL, ..., ..., ...);
        }
    }

    // keep the children alive
    sleep(10);

    for (int i = 0; i < RACE_COUNT; i++) {
        pids[i] && kill(pids[i], 9);
    }

    printf("\n\n\nEND OF THE CYCLE\n\n\n");
}
Fix #5

```swift
    audit_token_t auditToken = connection.auditToken;
    NSData *tokenData = [NSData dataWithBytes:&auditToken length:sizeof(audit_token_t)];
    attributes = @[ (__bridge NSString *)kSecGuestAttributeAudit: tokenData ];
```
The APIs are private 😞

```objective-c
@interface NSXPCConnection (AuditToken)
@property (nonatomic, readonly) audit_token_t auditToken;
@end

xpc_connection_get_audit_token();
```
Privileged operations implementation on and SETAPP
<table>
<thead>
<tr>
<th>Application</th>
<th>API</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SMJobBless()</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AuthorizationExecuteWithPrivileges()</td>
<td></td>
</tr>
</tbody>
</table>

# bugs reported*: 5

* as for March 2020
Application

API

SMJobBless()

AuthorizationExecuteWithPrivileges()

# bugs reported*

5

0

* as for March 2020
Summary & Takeaways
Takeaways for developers

1. Think about security in your project/company. A good start is creating a security@yourcompany.com email handle.

2. Have one source of truth for Client’s signing requirements and one for Privileged Helper’s, e.g. put them in Preprocessor Macros and use it in:
   - `Info.plist` file
   - `listener:shouldAcceptNewConnection`:

3. In signing requirements check at least for:
   - signing identity
   - bundle identifier
   - minimum version

4. In `SecCodeCopyGuestWithAttributes` use 📝 audit token to obtain code reference for signature validation, not the pid

5. In order to be a good citizen remember to unregister the Privileged Helper via `launchctl` or `SMJobRemove` API, remove the executable from `/Library/PrivilegedHelperTools` and the auto generated .plist from `/Library/LaunchDaemons`
Example set up requirements for Privileged Helper

1. Add User-Defined Build Settings:

<table>
<thead>
<tr>
<th>Setting</th>
<th>ObjCCommandLineTool</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLIENT_IDENTIFIER</td>
<td>com.yourcompany.yourapp</td>
</tr>
<tr>
<td>CLIENT_MIN_VERSION</td>
<td>10.10.10</td>
</tr>
<tr>
<td>CLIENT_SIGNING.IDENTITY</td>
<td>Developer ID Application: YourCompany Inc. (AAAAAAAAAAA)</td>
</tr>
</tbody>
</table>

2. Use them to create a macro definition

```plaintext
CLIENT_REQUIREMENTS="@"anchor trusted and certificate leaf [subject.CN] = "$(CLIENT_SIGNING_IDENTITY)" and info[CFBundleShortVersionString] >= "$CLIENT_MIN_VERSION" and identifier "$(CLIENT_IDENTIFIER)"
```
3. Use your Build Settings in Info.plist client requirements:

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Information Property List</td>
<td>Dictionary</td>
<td>(3 items)</td>
</tr>
<tr>
<td>▼ Clients allowed to add and remove...</td>
<td>Array</td>
<td>(1 item)</td>
</tr>
<tr>
<td>Item 0</td>
<td>String</td>
<td>anchor trusted and certificate leaf [subject.CN] = &quot;$(CLIENT_SIGNING.IDENTITY)&quot; and info[CFBundleShortVersionString] &gt;= &quot;$CLIENT_MIN_VERSION&quot; and identifier &quot;$CLIENT_IDENTIFIER&quot;</td>
</tr>
<tr>
<td>Bundle identifier</td>
<td>String</td>
<td>$PRODUCT_BUNDLE_IDENTIFIER</td>
</tr>
<tr>
<td>InfoDictionary version</td>
<td>String</td>
<td>0.0</td>
</tr>
</tbody>
</table>

4. Use the Macro Definition from 2. in code to validate incoming connection:

```objective-c
- (BOOL)listener:(NSXPCLeader *)listener shouldAcceptNewConnection:(NSXPCConnection *)newConnection
{
    NSString *requirements = CLIENT_REQUIREMENTS;
    SecRequirementRef requirement = NULL;
    OSSstatus status = errSecSuccess;
    SecRequirementCreateWithStringAndErrors((__bridge CFStringRef)entitlement, kSecCSDefaultFlags, NULL, &requirement);
    status = SecCodeCheckValidity(code, kSecCSDefaultFlags, requirement);
    if (status != errSecSuccess)
    {
        return NO;
    }
    ...
```
Summary/Wishlist
1. We need the documentation

There is no easily available Apple’s documentation about securing XPC connection with Privileged Helpers

2. We need Code Samples

Apple’s code samples are not secure

3. Using pid to check the signature of a process is not secure. It should be clearly stated in docs

Checks by pid are racy by nature

4. Audit token should not be private

It is the most secure way, but it is not available to 3rd party developers

5. There should be some Uninstallation API

When the app is being removed, the Helpers are usually forgotten in /Library/PrivilegedHelperTools
Further Reading

1. project-zero ‘Issue 1223: MacOS/iOS userspace entitlement checking is racy’ by Ian Beer

2. OffensiveCon19 'OSX XPC Revisited - 3rd Party Application Flaws' by Tyler Bohan

3. Apple Developer Forums 'XPC restricted to processes with the same code signing?'


5. ‘No Privileged Helper Tool Left Behind’ by Erik Berglund
Call to Action

If I could ask you to do 1 thing, let it be:
Call to Action 🧙‍♀️

If I could ask you to do 1 thing, let it be:

reporting to Apple, that audit tokens should be made available for 3rd party developers:
Thank you! 😎