Process injection

Breaking All macOS Security Layers
With a Single Vulnerability
@implementation AppDelegate

-(void)applicationDidFinishLaunching:(NSNotification *)aNotification {
    // Insert code here to initialize your application
}

-(void)applicationWillTerminate:(NSNotification *)aNotification {
    // Insert code here to tear down your application
}

-(BOOL)applicationSupportsSecureRestorableState:(NSApplication *)app {
    return YES;
}
@end
Hello!

I’m Thijs Alkemade

Security researcher at Computest
About me

> Thijs Alkemade (@xnyhps)
> Security researcher at Computest
> Computest research lab: Sector 7
> Other recent work includes:
  - 0click Zoom RCE at Pwn2Own Vancouver 2021
  - Winning Pwn2Own Miami 2022 with 5 ICS vulnerabilities
In this talk

1. macOS security model
2. CVE-2021-30873: process injection using saved states
3. Using process injection for:
   - Sandbox escape
   - Privilege escalation
   - SIP bypass
macOS security model

In macOS 12 Monterey
Old *NIX security model

- Users are security boundaries, processes are not
- File permissions: POSIX flags
- Attach debugger: target must run as same user
- root has full access
System Integrity Protection

Security policy applying to every process, including privileged code running unsandboxed.
Extends additional protections to system components on disk and at runtime.
System binaries can only be modified by Apple Installer and Software Update, and no longer permit runtime attachment or code injection.
SIP restrictions

>“Dangerous” operations require **entitlements:**
  - Loading a kernel extension
  - Modifying system files
  - Debugging system processes

>More and more restrictions in each macOS release
  - Debugging any app is now restricted
  - “Data vaults” with restricted file access
$ ls ~/Library.Mail/
ls: /Users/talkemade/Library/Mail/: Operation not permitted
$ sudo ls ~/Library.Mail/
ls: /Users/talkemade/Library/Mail/: Operation not permitted
$
$ codesign -dvvv --entitlements - /System/Applications/Mail.app/Executable=/System/Applications/Mail.app/Contents/MacOS/Mail Identifier=com.apple.mail Format=app bundle with Mach-O universal (x86_64 arm64e)

[...]

[Key] com.apple.rootless.storage.Mail
[Value]
  [Bool] true
Process injection

> Process A executing code “as” process B
> Many techniques are restricted by SIP
> Hardened runtime prevents it in apps:
  - No DYLD_* environment variables
  - Library validation
> But macOS is old, and large...

Platform Policy
Restricted processes

task_for_pid() / processor_set_tasks() fail with EPERM
Mach special ports are reset on exec(2)
dyld environment variables are ignored
dtrace probes unavailable
Process injection

- Common in third-party apps
- Abuse TCC permissions: access webcam, microphone, etc.
- Downgrade attacks often work
- What’s better than process injection in one app? Process injection everywhere!
CVE-2021-30873
Process injection in AppKit
Saved state feature

> Re-opening the windows of an app when relaunched
> Restores unsaved documents
> Works automatically, can be extended by developers
Saved state storage

> Stored in:
  - ~/Library/Saved Application
  - State/<ID>.savedState

> windows.plist
  - array of all windows, each with an encryption key

> data.data
  - custom format, AES-CBC encrypted serialized object per record
Serialization vulnerabilities

> Insecure deserialization can lead to RCE
  - Well known in C#, Java, Python, Ruby...
> Apple’s serialization is NSCoding
> Added NSSecureCoding in 10.8 (2012)
// Insecure
id obj = [decoder decodeObjectForKey:@"myKey"];
if (![obj isKindOfClass:[MyClass class]]) { /* ...fail... */ }

// Secure
id obj = [decoder decodeObjectOfClass:[MyClass class]
    forKey:@"myKey"];
Exploiting for process injection

1. Create a saved state using a malicious serialized object
2. Write it to the saved state directory of the other app
3. Launch other app
4. App automatically deserializes our object
5. Execute code in the other app!
What object to write?

> ysoserial-objective-c?
> Google Project Zero writeups?
Insecure deserialization with NSCoding

And defeating the hardened runtime by executing Python
Search for an object chain

- Disassemble -initWithCoder: methods
- Surprisingly, many classes do not support secure coding!
- ...but in most cases it only recursively decodes instance variables
Step 1: NSRuleEditor

> NSRuleEditor creates a binding to a keypath also from the archive:

```objc
ID NSRuleEditor::initWithCoder:(ID param_1,SEL param_2,ID unarchiver)
{
    ...
    id arrayOwner = [unarchiver decodeObjectForKey:@"NSRuleEditorBoundArrayOwner"];  
    ...
    if (arrayOwner) {
        keyPath = [unarchiver decodeObjectForKey:@"NSRuleEditorBoundArrayKeyPath"];
        [self bind:@"rows" toObject:arrayOwner withKeyPath:keyPath options:nil];
    }
    ...
}
```

> Result: call any zero-argument method on a deserialized object
Step 2: NSCustomImageRep

> NSCustomImageRep obtains an object and selector from the archive:

```objc
ID NSCustomImageRep::initWithCoder:(ID param_1,SEL param_2,ID unarchiver)
{
    ...
    self.drawObject = [unarchiver decodeObjectForKey:@"NSDrawObject"];
    id drawMethod = [unarchiver decodeObjectForKey:@"NSDrawMethod"];
    self.drawMethod = NSSelectorFromString(drawMethod);
    ...
}
```
Step 2: NSCustomImageRep

> NSCustomImageRep in `-draw` then calls the selector on the object:

```swift
void ___24-[NSCustomImageRep_draw]_block_invoke(long param_1)
{
    ...  
    [self.drawObject performSelector:self.drawMethod withObject:self];
    ...
}
```

> Result: call any method on a deserialized object (limited control over arguments)
Deserialization to arbitrary code execution

1. Call zero-argument methods on deserialized objects
2. Call any method on deserialized objects
3. Create objects not implementing NSCoder
4. Call zero-argument methods on arbitrary objects
5. Call any method on arbitrary objects
6. Evaluate AppleScript
Hardened runtime vs Python

> Hardened runtime:
  - No JIT memory pages allowed
  - No unsigned memory allowed
  - Library validation (all libraries signed by same developer or Apple)
  - No DYLD environment variables

> Solution:
  - Load Python.framework (signed by Apple)
  - import ctypes

> How to evaluate Python?
AppleScript-Objective-C bridge

- Load AppleScriptObjC.framework
- Can load scripts from unsigned bundles
- Can:
  - Create Objective-C objects
  - Call methods
  - Call C-functions
  - Can **not**: create non-object pointers, structs, C-strings
- Call Py_Main(0, NULL): reads script from stdin!
on initialize()
    tell current application to NSLog("Hello world from AppleScript!")

    current application's NSBundle's alloc's
    initWithPath_("/System/Library/Frameworks/Python.framework")'s load

    current application's Py_Initialize()

    tell current application to NSLog("Py_Main: %d",
        Py_Main(0, reference))
end initialize
Deserialization to arbitrary code execution

1. Call zero-argument methods on deserialized objects
2. Call any method on deserialized objects
3. Create objects not implementing NSCoder
4. Call zero-argument methods on arbitrary objects
5. Call any method on arbitrary objects
6. Evaluate AppleScript
7. Evaluate AppleScript with the AppleScript-Objective-C bridge
8. Evaluate Python
9. Import ctypes
10. Execute code equivalent to native code
Exploitation
Sandbox escape
Window: the app

Contents: openAndSavePanelService
Sandbox escape

> Open/save panel loaded its saved state from the same files as the app!
  - Write new object in the app’s own saved state directory
  - Open a panel
  - Sandbox escaped!

> Fixed in 11.3: no long shares directory

CoreFoundation
Available for: macOS Big Sur
Impact: A malicious application may be able to leak sensitive user information
Description: A validation issue was addressed with improved logic.
CVE-2021-30659: Thijs Alkemade of Computest
Exploitation

Privilege escalation to root
Privilege escalation

> Use the same technique as “Unauthd - Logic bugs FTW” by Ilias Morad

> First, find an app with entitlement:

com.apple.private.AuthorizationServices

containing:

system.install.apple-software
Privilege escalation

> Then, install this package to a RAM disk

> It runs a post-install script from the target disk as root
  - Target disk may not even have macOS!
  - Mounting a RAM disk does not require root

macOSPublicBetaAccessUtility.pkg
Installer package – 84 KB
Exploitation

SIP filesystem bypass
SIP filesystem bypass

- App from the macOS Big Sur beta installation dmg
- Has the entitlement:
  - com.apple.rootless.install.heritable
- Very powerful entitlement: access all SIP protected files!
  - Heritable as a bonus, so can spawn a reverse shell
SIP filesystem bypass: result

- Read mail, messages, Safari history, etc. of all users
- Grant ourselves permission for webcam, microphone, etc.
- Powerful persistence (SIP protected locations, delete MRT)
- Load a kernel extension without user approval
The fixes
The fixes

> In Monterey, an app can indicate that it only accepts secure serialized objects in its saved state
  - Already enabled for Apple’s apps
  - Existing apps may want to store objects that do not implement secure deserialization
  - Unclear if exploitable when apps don’t use custom serialized objects

> Reported December 4, 2020

> Sandbox escape fixed (CVE-2021-30659) in 11.3 (April 26, 2021)

> Fix introduced in macOS Monterey 12.0.1 (October 25, 2021)
  - Unclear if backported to Big Sur and Catalina
Conclusion
> macOS has a security boundary between processes

> Process injection vulnerabilities can be used to break those boundaries

> CVE-2021-30873 was a process injection vulnerability affecting AppKit apps

> We used it to escape the sandbox, privilege escalation, bypassing SIP

> Fixed by Apple in Monterey

**Conclusion**
macOS security keeps adding more and more defensive layers

Adding new layers to an established system is hard
- Code written 10+ years ago without security requirements is today’s attack surface

Effort of attackers may not increase with more layers
- Use the same bug for multiple layers or skip layers
References

> https://googleprojectzero.blogspot.com/2020/01/remote-iphone-exploitation-part-1.html
> https://googleprojectzero.blogspot.com/2022/03/forcedentry-sandbox-escape.html
> https://a2nkf.github.io/unauthd_Logic_bugs_FTW/
> https://github.com/frohoff/ysoserial
> https://github.com/pwntester/ysoserial.net