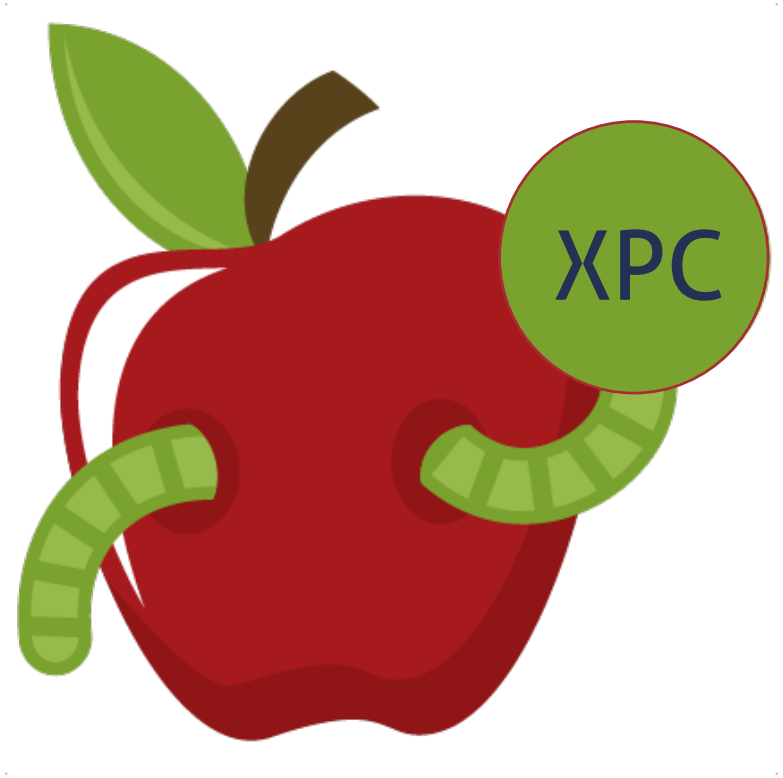
securing



Wojciech Reguła

Abusing & Securing XPC in macOS apps

WHOAMI

- Senior IT Security Consultant @ SecuRing
- Focused on iOS apps security
- Blogger https://wojciechregula.blog/
- iOS Security Suite creator

XPC VULNERABILITIES ARE EVERYWHERE



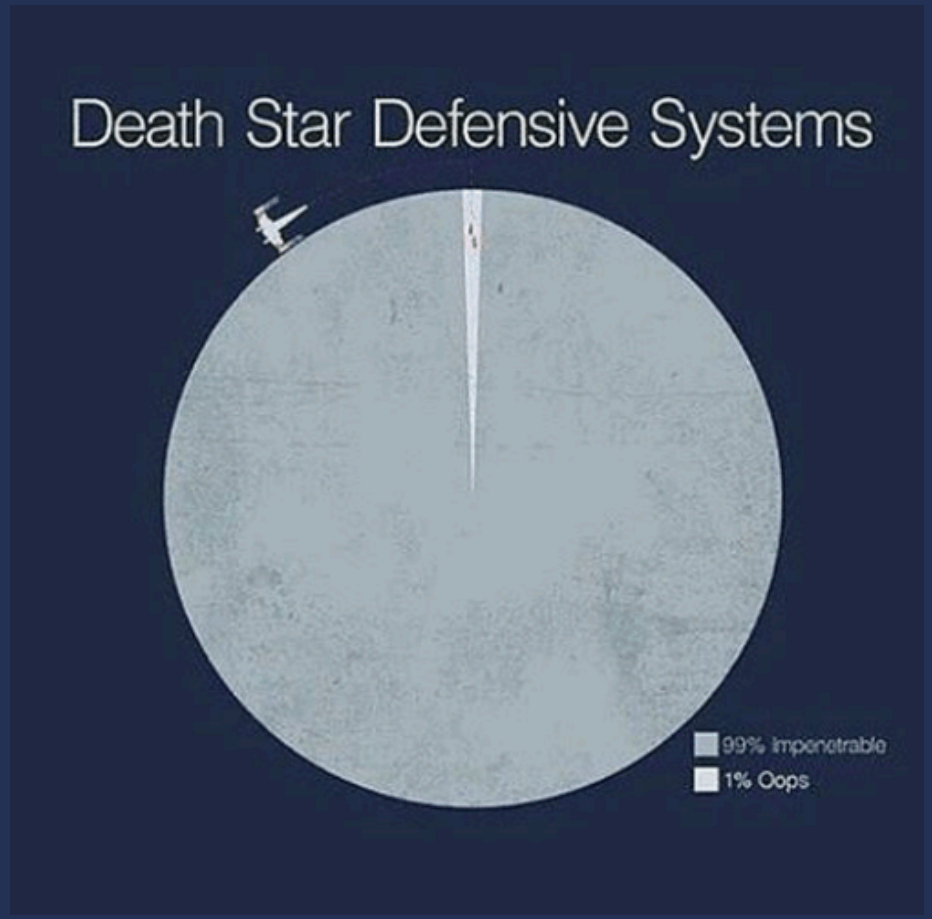







AGENDA

1. Introduction to XPC
2. Building secure Helper
3. Exploiting previous step
4. Hardening
5. IF bypass_available() { GOTO 2. }
6. Summary



(NS)XPC

- Fundamental IPC mechanism in Apple environment
- Built on top of Mach messages
- Dictionary based communication
- Possible multiple clients and one server
- You can send and serialize ObjC/Swift objects

Server

```
let listener = NSXPCListener.init(machServiceName: MACH_SERVICE_NAME)
let delegate = XPCDelegate()

listener.delegate = delegate
listener.resume()

RunLoop.main.run()
```

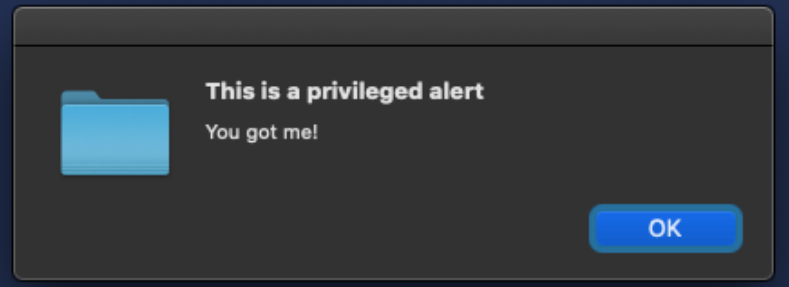
Client

```swift
@IBAction func spawnPrivilegedAlert(_ sender: Any) {
        let connection = NSXPCConnection(machServiceName: MACH_SERVICE_NAME, options: .privileged)
        connection.remoteObjectInterface = NSXPCInterface(with: SimpleXPCProtocol.self)
        connection.resume()

        let remoteObject = connection.remoteObjectProxyWithErrorHandler { (err) in
            print("Error \(err.localizedDescription)")
        } as? SimpleXPCProtocol

        remoteObject?.privilegedAlert()
    }
```

INTER-PROCESS COMMUNICATION

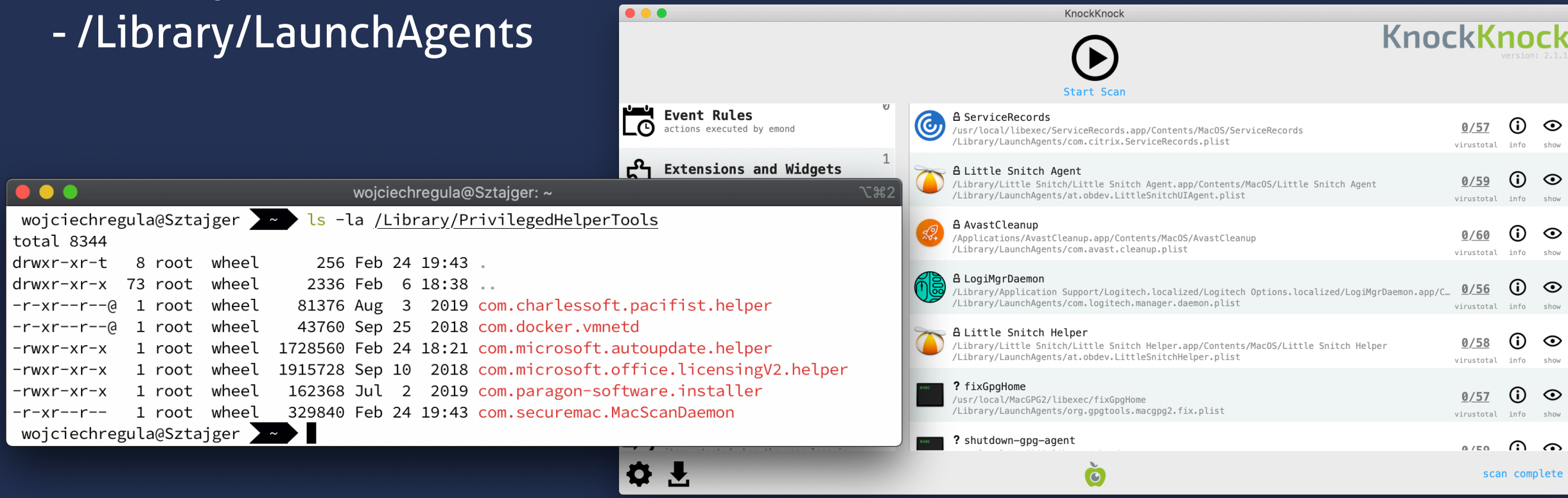
uid=501 (user)

uid=0 (root)



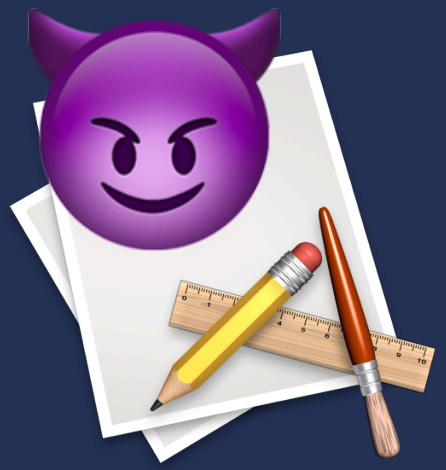
LOCATIONS OF THE XPC SERVERS

- /Library/PrivilegedHelperTools
- /Library/LaunchDaemons
- /Library/LaunchAgents



TIME FOR EXPLOITATION

BUG #1 – NO CLIENT VALIDATION



Perform privileged action for me

OK, I don’t even verify you

exec

Malicious Application
uid= 501 (user)

Privileged XPC server
uid = 0 (root)

BUG #1 – NO CLIENT VALIDATION

Keybase.io

Bug found by Adam Chester (@_xpn_)
Methods exposed:
- remove
- move
- createDirectory
- addToPath
- kextInstall

BUG #1 – NO CLIENT VALIDATION

```
33  35      @try {
34  36        KBHelper *helper = [[KBHelper alloc] init];
35      -     [helper listen:service];
    37  +     [helper listen:service codeRequirement:codeRequirement];
36  38
37  39        dispatch_main();
38  40      } @catch(NSException *e) {
```

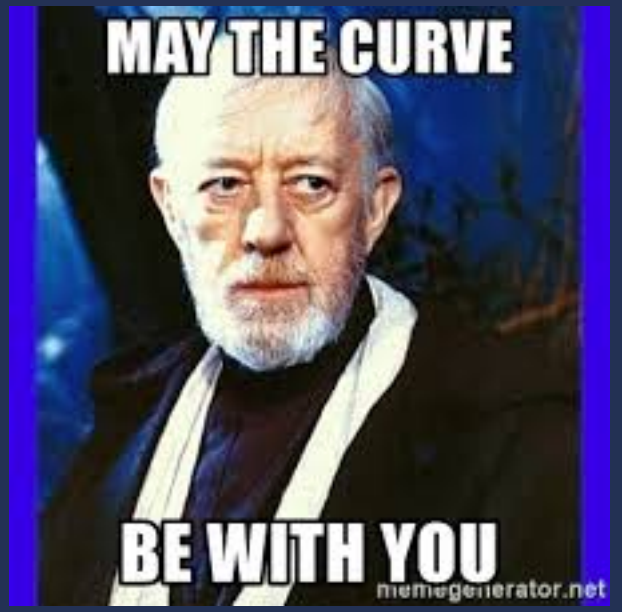
BUG #1 – FIX

```
class XPCDelegate: NSObject, NSXPCListenerDelegate, SimpleXPCProtocol {

    func listener(_ listener: NSXPCListener, shouldAcceptNewConnection newConnection: NSXPCConnection) -> Bool {

        if ConnectionVerifier.isValid(connection: newConnection) {
            newConnection.exportedInterface = NSXPCInterface(with: SimpleXPCProtocol.self)
            newConnection.exportedObject = self
            newConnection.resume()
            return true
        }
        return false
    }
}
```

BUG #2 – POOR SIGNATURE CHECK

BUG #2 – POOR SIGNATURE CHECK

- Checking only bundle ID
- Verifying only static code
- Using kSecCSBasicValidateOnly flag
- Too loose SecRequirement string
- and more...



```
loc_100001e19:
    SecRequirementCreateWithString(@"identifier com.securemac.MacScan3 and certificate leaf[subject.OU] = \"76W2LW5UNP\"", 0x0, &var_80);
    rax = SecCodeCheckValidity(var_70, 0x0, var_80);
    if (rax == 0x0) goto loc_100001f40;

loc_100001e44:
    r15 = [[LogController sharedInstance] retain];
    var_40 = @"errorExtraInfo";
    var_54 = 0x0;
    rax = [NSString stringWithFormat:@"SecCodeCheckValidity = %d", rax];
    rax = [rax retain];
    r13 = rax;
    var_38 = rax;
    rax = [NSDictionary dictionaryWithObjects:@"SecCodeCheckValidity = %d" forKeys:&var_40 count:0x1];
    rax = [rax retain];
    rbx = rax;
    [r15 logErrorWithType:0x9 andInfo:rax];
    goto loc_100001ee7;

loc_100001f40:
    rax = [NSXPCInterface interfaceWithProtocol:@protocol(MacScanDaemonProtocol)];
    rax = [rax retain];
    r12 = var_60;
    [r12 setExportedInterface:rax];
    [rax release];
    [r12 setExportedObject:r15];
```

Exposed methods:

- **uninstallDaemon**
- **terminateDaemonWithOptions**
- deleteQuarantineItemsWithInfo
- quarantineItemsWithInfo
- **cancelScan**
- startScanWithPaths
- getDaemonInfoWithOptions

BUG #2 – FIX

```
private static func verifyWithRequirementString(secCode: SecCode) -> Bool {
    let requirementString = "anchor apple generic and identifier \"\(MAIN_APP_BUNDLE_ID)\" and certificate
leaf[subject.CN] = \"\(SUBJECT_CN)\"" as NSString

    var secRequirement: SecRequirement? = nil
    if SecRequirementCreateWithString(requirementString as CFString, SecCSFlags(rawValue: 0), &secRequirement) !=
errSecSuccess {
        NSLog("Couldn't create the requirement string")
        return false
    }

    if SecCodeCheckValidity(secCode, SecCSFlags(rawValue: 0), secRequirement) != errSecSuccess {
        NSLog("NSXPC client does not meet the requirements")
        return false
    }

    return true
}
```

BUG #3 – USING PID TO GET CODE OBJECT

Kudos for Samuel Groß (@5aelo) and Ian Beer (@l41nbeer) 👏

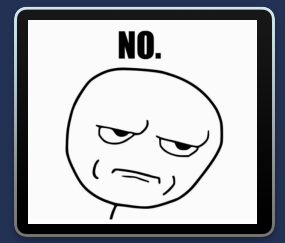
pid: 1337

Perform privileged action

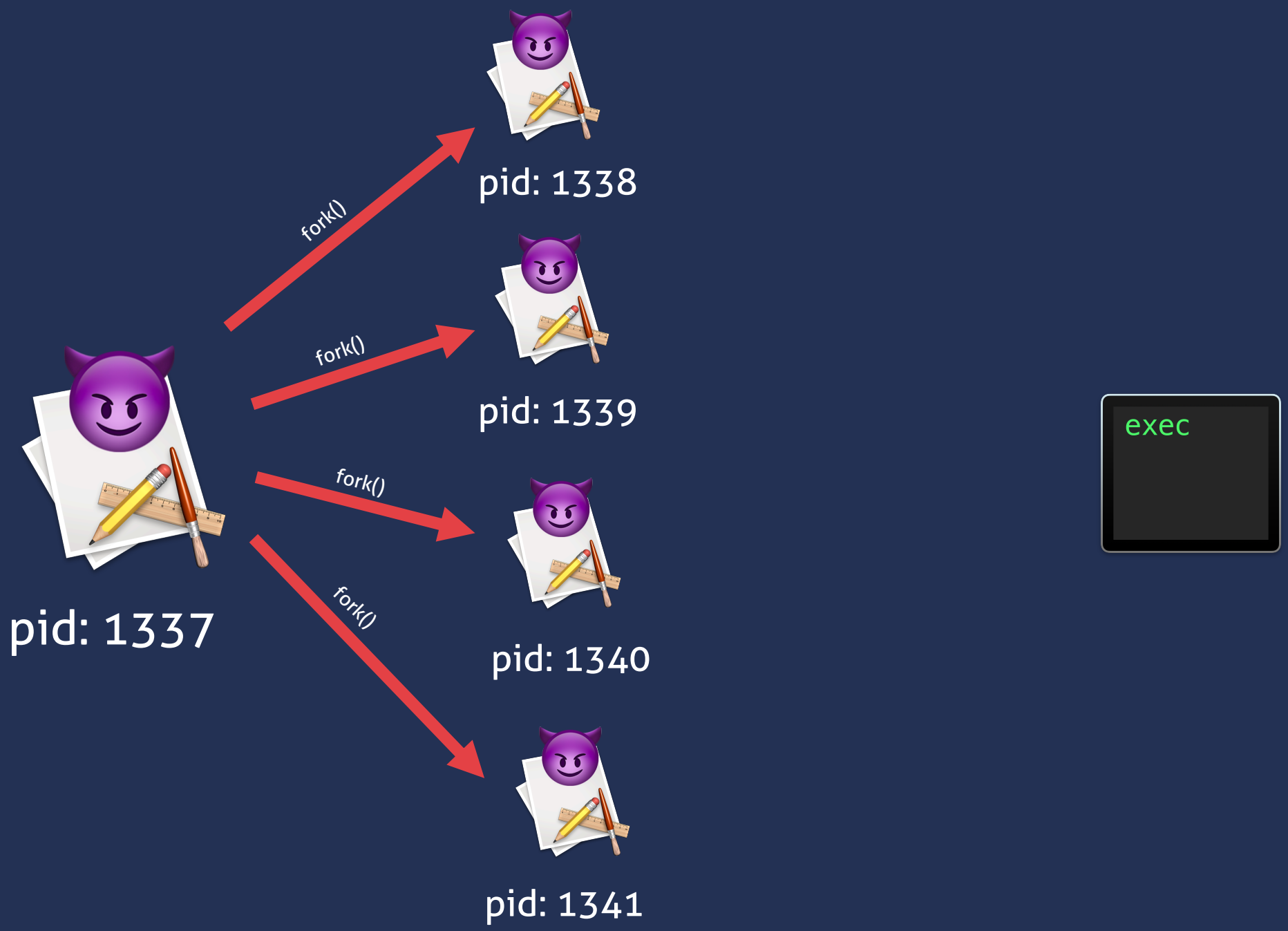
exec

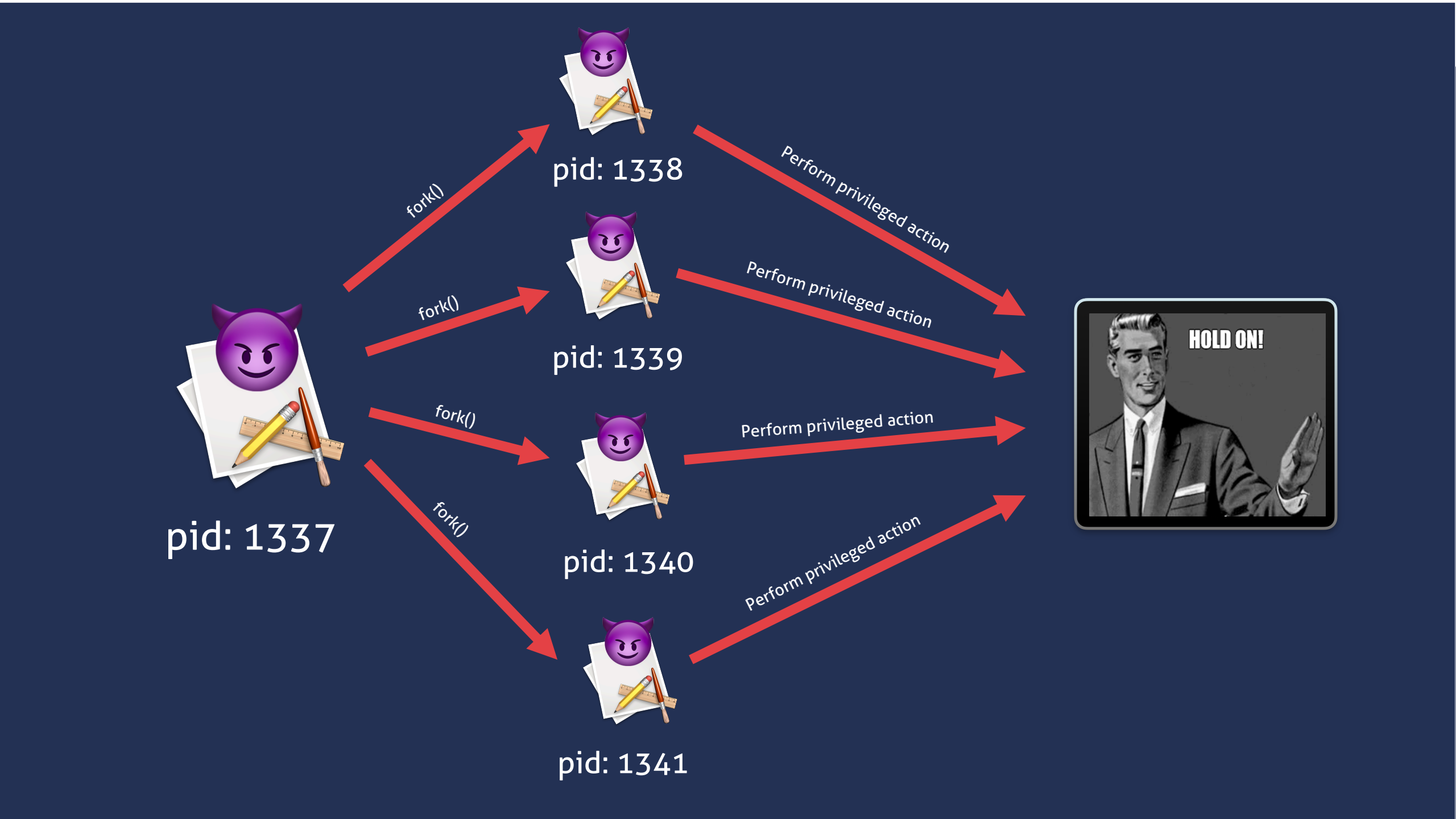
pid: 1337

INVALIDATED



1. Get process identifier of the client
2. Create code object basing on that PID
3. Perform signature check
4. isValid() -> false
5. Invalidating the XPC connection

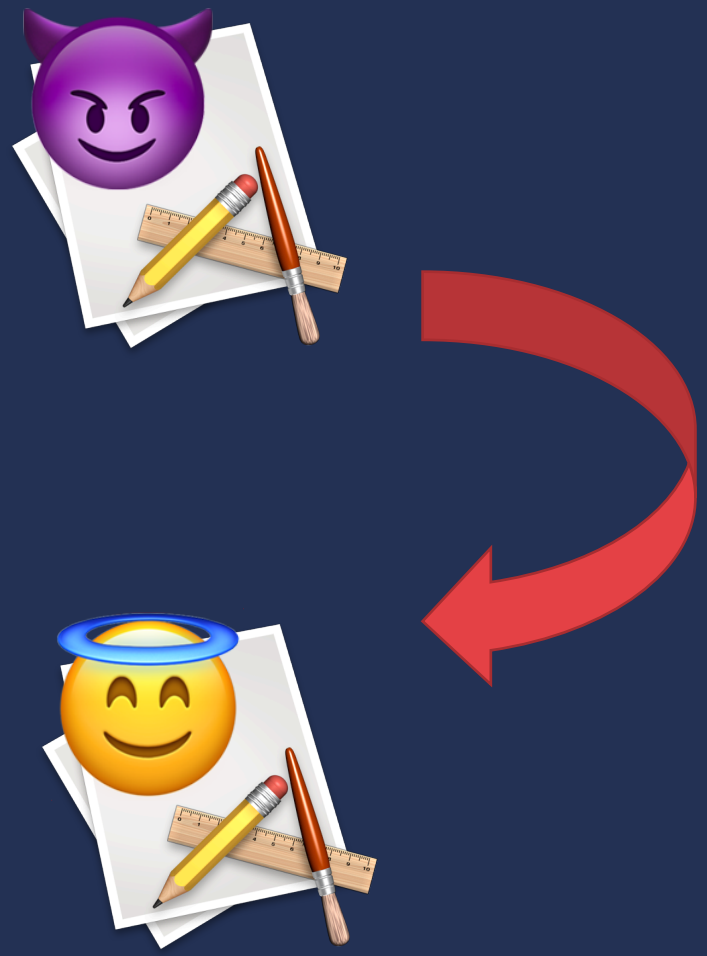

fork()
fork()
fork()
fork()
pid: 1337
pid: 1338
pid: 1339
pid: 1340
pid: 1341
exec


pid: 1337
fork()
fork()
fork()
fork()
pid: 1338
pid: 1339
pid: 1340
pid: 1341
Perform privileged action
Perform privileged action
Perform privileged action
Perform privileged action
HOLD ON!

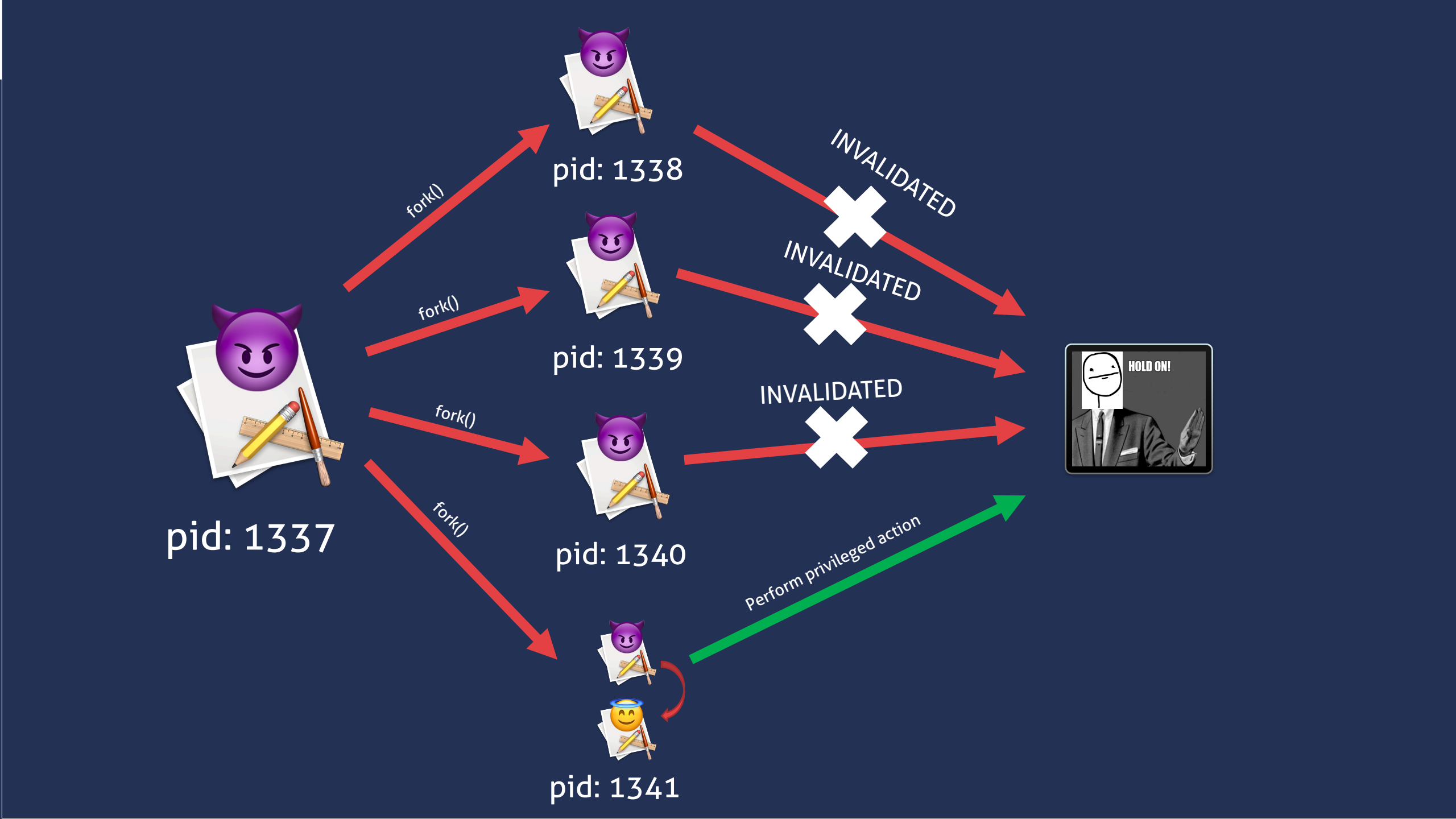
XPC connections queue

~~connection1~~	~~connection2~~	~~connection3~~	connection4

1. Get process identifier of the client
2. Create code object based on that PID 😇
3. Perform signature check
4. isValid() -> true
5. Establish valid connection



Change process' image to the legit executable using posix_spawn()


pid: 1337
fork()
fork()
fork()
fork()
pid: 1338
pid: 1339
pid: 1340
pid: 1341
INVALIDATED
INVALIDATED
INVALIDATED
Perform privileged action
HOLD ON!

Avast Cleanup

Exposed methods:
- removeWithFile
- copyWithSource

```
int __T028com_avast_cleanup_engine_xpc15ServiceDelegateC12authenticate33_D362805DE5EDCFC355566E7B9AB676EALLSbSo15NSXPCConnectionCFTf4gd_n() {
    r15 = [[_swift_rt_swift_getInitializedObjCClass() runningApplicationWithProcessIdentifier: rdi processIdentifier] retain];
    if (r15 == 0x0) goto loc_100003497;

loc_1000032e0:
    rax = [r15 bundleURL];
    rax = [rax retain];
    if (rax == 0x0) goto loc_10000348f;

loc_100003300:
    r14 = static Foundation.URL._unconditionallyBridgeFromObjectiveC(rax);
    var_30 = 0x0;
    [r14 retain];
    rbx = Foundation.URL._bridgeToObjectiveC(r14);
    rax = SecStaticCodeCreateWithPath(rbx, 0x0, &var_30);
```

Malwarebytes

Exposed methods:
- startDatabaseUpdate
- restoreApplicationLauncherWithCompletion
- **uninstallProduct**
- installProductUpdate
- **startProductUpdateWith**
- buildPurchaseSiteURLWithCompletion
- triggerLicenseRelatedChecks
- buildRenewalLinkWith
- cancelTrialWith
- startTrialWith
- applyLicenseWith
- controlProtectionWithRawFeatures
- confirmScanJobWith
- resumeScanJob
- **pauseScanJob**
- **stopScanJob**
- startScanJob
- disposeOperationBy
- pingWithTag

```
+(struct __SecCode *)initCodeObjectFrom:(int)arg2 {
    var_1C = arg2;
    rax = CFNumberCreate(**_kCFAllocatorDefault, 0x3, &var_1C);
    var_18 = rax;
    if (rax != 0x0) {
            rax = CFDictionaryCreate(**_kCFAllocatorDefault, *_kSecGuestAttributePid, &var_18, 0x1, 0x0, 0x0);
            if (rax != 0x0) {
                    var_10 = 0x0;
                    rbx = rax;
                    rax = SecCodeCopyGuestWithAttributes(0x0, rax, 0x0, &var_10);
                    if ((rax != 0x0) && (0x0 != 0x0)) {
                            CFRelease(0x0);
                    }
                    CFRelease(rbx);
            }
            rdi = var_18;
            if (rdi != 0x0) {
                    CFRelease(rdi);
            }
    }
    return 0x0;
}
```

BUG #3 – FIX

```objc
@interface NSXPCConnection(PrivateAuditToken)

@property (nonatomic, readonly) audit_token_t auditToken;

@end


@interface AuditTokenHack : NSObject

+(NSData *)getAuditTokenDataFromNSXPCConnection:(NSXPCConnection *)connection;

@end

@implementation AuditTokenHack

+ (NSData *)getAuditTokenDataFromNSXPCConnection:(NSXPCConnection *)connection {
    audit_token_t auditToken = connection.auditToken;
    return [NSData dataWithBytes:&auditToken length:sizeof(audit_token_t)];
}

@end
```

BUG #3 – FIX

```
private static func prepareCodeReferencesFromAuditToken(connection: NSXPCConnection, secCodeOptional: inout
SecCode?, secStaticCodeOptional: inout SecStaticCode?) -> Bool {
    let auditTokenData = AuditTokenHack.getAuditTokenData(from: connection)

    let attributesDictrionary = [
        kSecGuestAttributeAudit : auditTokenData
    ]

    if SecCodeCopyGuestWithAttributes(nil, attributesDictrionary as CFDictionary, SecCSFlags(rawValue: 0),
&secCodeOptional) != errSecSuccess {
        NSLog("Couldn't get SecCode with the audit token")
        return false
    }

    guard let secCode = secCodeOptional else {
        NSLog("Couldn't unwrap the secCode")
        return false
    }

    SecCodeCopyStaticCode(secCode, SecCSFlags(rawValue: 0), &secStaticCodeOptional)

    guard let _ = secStaticCodeOptional else {
        NSLog("Couldn't unwrap the secStaticCode")
        return false
    }

    return true
}
```

BUG #4 – NO CODE INJECTION PREVENTION

BUG #4 – NO CODE INJECTION PREVENTION

1. Get audit token of the client
2. Create code object based on that token
3. Perform signature check
4. isValid() -> false
5. Invalidating the XPC connection

BUG #4 – NO CODE INJECTION PREVENTION

- **Apple's signature checking API doesn't detect:**
  - code injected with DYLD_INSERT_LIBRARIES
  - or via task_for_pid() neither…

LuLu

Exposed methods:

- loadKext
- getPreferences
- updatePreferences
- getRules
- **addRule**
- updateRule
- **deleteRule**
- importRules
- alertReply

```
122       //init signing req string
-     requirementString = [NSString stringWithFormat:@"anchor trusted and certificate leaf [subject.CN] = \"%@\"", SIGNING_AUTH];
123   +     requirementString = [NSString stringWithFormat:@"anchor trusted and identifier \"%@\" and certificate leaf [subject.CN] = \"%@\" and info [CFBundleSho
      rtVersionString] >= \"1.2.0\"", INSTALLER_ID, SIGNING_AUTH];
124
125       //step 1: create task ref
126       // uses NSXPCConnection's (private) 'auditToken' iVar
```

Gatekeeper
macOS Catalina

NEW

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

Prepare Your Software for Notarization

Notarization requires Xcode 10 or later. Building a new app for notarization requires macOS 10.13.6 or later. Stapling an app requires macOS 10.12 or later.

Apple's notary service requires you to adopt the following protections:

- Enable code-signing for all of the executables you distribute.
- Enable the Hardened Runtime capability for your app and command line targets, as described in Enable hardened runtime.
- Use a “Developer ID” application, kernel extension, or installer certificate for your code-signing signature. (Don't use a Mac Distribution or local development certificate.) For more information, see Create, export, and delete signing certificates.
- Include a secure timestamp with your code-signing signature. (The Xcode distribution workflow includes a secure timestamp by default. For custom workflows, include the `--timestamp` option when running the `codesign` tool.)
- Don’t include the `com.apple.security.get-task-allow` entitlement with the value set to any variation of `true`. If your software hosts third-party plug-ins and needs this entitlement to debug the plug-in in the context of a host executable, see Avoid the Get-Task-Allow Entitlement.
- Link against the macOS 10.9 or later SDK.

BYPASSING APPS WITH HARDENED RUNTIME

- SecRequirement(Certificate[teamID]) -> inject to another app from the same company
- SecRequirement(Certificate[teamID] + bundleID) -> inject to older version of the app without hardened runtime 😉
- Also, look for following problematic entitlements:
  - com.apple.security.get-task-allow
  - com.apple.security.disable-library-validation

DEMO 

HTTPS://VIMEO.COM/397568495

BUG #4 – FIX

```
if let signingFlags = signingFlagsOptional {
    let hardenedRuntimeFlag: UInt32 = 0x10000
    if (signingFlags & hardenedRuntimeFlag) != hardenedRuntimeFlag {
        NSLog("Hardened runtime is not set for the sender")
        return false
    }
```

...

```
let problematicEntitlements = [
    "com.apple.security.get-task-allow",
    "com.apple.security.cs.disable-library-validation",
    "com.apple.security.cs.allow-dyld-environment-variables"
]

for problematicEntitlement in problematicEntitlements {
    if let presentEntitlement = entitlements.object(forKey: problematicEntitlement) {
        if presentEntitlement as! Int == 1 {
            NSLog("The sender has \(problematicEntitlement) entitlement set to true")
            return false
        }
    }
```

SUMMARY

SECURING XPC SERVICES - GUIDELINE

1. [Client] Turn on hardened runtime and notarize your clients
2. [Server] Verify if the clients have been properly signed using restrictive SecRequirement string
3. [Server] Create Sec(Static)Code objects from audit tokens
4. [Server] Check if the clients have the hardened runtime capability turned on
5. [Server] Make sure if the clients are not signed with entitlements that may allow bypassing the hardened runtime

OPEN SOURCING SECURE PRIVILEGED HELPER EXAMPLE

https://github.com/securing

OPEN SOURCING THE SECURE HELPER

```
public static func isValid(connection: NSXPCConnection) -> Bool {
        var secCodeOptional: SecCode? = nil
        var secStaticCodeOptional: SecStaticCode? = nil

        if !prepareCodeReferencesFromAuditToken(connection: connection, secCodeOptional: &secCodeOptional,
secStaticCodeOptional: &secStaticCodeOptional) {
            return false
        }

        if !verifyHardenedRuntimeAndProblematicEntitlements(secStaticCode: secStaticCodeOptional!) {
            return false
        }

        if !verifyWithRequirementString(secCode: secCodeOptional!) {
            return false
        }

        return true
    }
```

REFERENCES

- https://hackerone.com/reports/397478
- https://bugs.chromium.org/p/project-zero/issues/detail?id=1223
- https://saelo.github.io/presentations/warcon18_dont_trust_the_pid.pdf
- https://blog.obdev.at/what-we-have-learned-from-a-vulnerability/
- https://developer.apple.com/documentation/xcode/notarizing_macos_software_before_distribution
- https://github.com/erikberglund/SwiftPrivilegedHelper
- https://developer.apple.com/videos/play/wwdc2019/701/

Contact
Wojciech Reguła

wojciech.regula@securing.pl

@_r3ggi

wojciech-regula

SecuRing
Kalwaryjska 65/6
30-504 Kraków, Poland

info@securing.pl
tel. +48 124252575
http://www.securing.biz/