Forklift <=3.3.9 and <=3.4 Local Privilege Escalations on macOS (CVE-2020-15349/CVE-2020-27192)

I have started to have a look at my local installed helpers on macOS. These helpers are used as an interface for applications to perform privileged operations on the system. Thus, it is quite a nice attack surface to search for Local Privilege Escalations.

Forklift is an advanced dual pane file manager for macOS. It is well known under macOS power users.

As part of my investigation I identified vulnerabilities in Forklift allowing local privilege escalation.

By now all vulnerabilities are fixed by the vendor I can release the details: https://binarynights.com/versionhistory

Forklift 3.3.9 Local Privilege Escalation CVE-2020-15349

When installing the Forklift, a new helper called *com.binarynights.ForkLiftHelper* for Mac OS X is automatically installed to the */Library/PrivilegedHelperTools/* directory.
Analyzing this helper, which handles XPC messages, resulted in different ways of escalating privileges from user to *root* on Mac OS X.

When accepting XPC calls the `HelperTool listener:shouldAcceptNewConnection` respectively `(BOOL)listener:(NSXPCListener *)listener shouldAcceptNewConnection:(NSXPCConnection *)connection` function by default. This function is used to perform the initial steps for establishing an XPC connection. Usually, this function performs the authorization of the caller—however, the function of the *com.binarynights.ForkLiftHelper* does not implement any authorization checks. Thus, it is possible to call any exposed functions over XPC unauthorized.

The following functions are exposed over XPC to the caller:

```
@protocol _TtP4main21ForkLiftHelperProtcol_
- (void)changePermissions:(NSString *)arg1 permissions:(long long)arg2 reply:(void (^)(NSError *))arg3;
- (void)changeOwner:(NSString *)arg1 owner:(long long)arg2 group:(long long)arg3 reply:(void (^)(NSError *))arg4;
- (void)calculateDirectorySize:(NSString *)arg1 reply:(void (^)(NSNumber *, NSError *))arg2;
- (void)createDirectory:(NSString *)arg1 reply:(void (^)(NSError *))arg2;
- (void)deleteItem:(NSString *)arg1 reply:(void (^)(NSError *))arg2;
- (void)moveItem:(NSString *)arg1 targetPath:(NSString *)arg2 reply:(void (^)(NSError *))arg3;
- (void)copyItemAbort:(NSString *)arg1;
- (void)copyItemProgress:(NSString *)arg1 reply:(void (^)(NSNumber *, NSError *))arg2;
- (void)copyItem:(NSString *)arg1 targetPath:(NSString *)arg2 UUID:(NSString *)arg3 reply:(void (^)(NSError *))arg4;
- (void)moveToTrash:(NSString *)arg1 reply:(void (^)(NSError *))arg2;
- (void)getHelperVersion:(void (^)(NSString *))arg1;
@end
```

Setuid Exploitation

The first method to exploit this helper is as follows:

Step 1: Interpreter Target chown

Copy, for example, the python interpreter to a user-controllable directory and call the XPC function *changeOwner* with the parameter `@"<path>" owner:0 group:0`. This will change the ownership of the python interpreter to *root*.

Step 2: Set SUID flag

Second, it is possible to set the SUID bit to the interpreter issuing an XPC request to the *changePermissions* function with the following parameter `@"<path>" permissions:2541`. The permissions *2541* represent in octal the number *4755*, which sets the SUID flag on the binary.

Step 3: LPE

The following command will spawn a shell as root:

```
$/tmp/python_copied -c 'import pty; import os; os.setuid(0);pty.spawn("/bin/bash")'
# id
uid=0(root) [...]
```

Full Exploit

Please make sure to first have a python interpreter copied to */tmp* with the name *python_copied*.
A working exploit can be found here.

LaunchAgent Exploitation

The second method to exploit this helper is as follows:

Writing a new Launch Agent

Due to the XPC call *moveItem* it is possible to move any item as *root*. Thus, it is possible to write a *plist* file to the */tmp* directory and then copy it to the */Library/LaunchDaemons/* directory. The following parameters are used for the *moveItem* function `@"/tmp/com.sample.Load.plist" targetPath:@"/Library/LaunchDaemons/com.sample.Load.plist"`

Full Exploit

The exploit can be found here and automatically adds a new launch agent which is triggered at a restart:

Recommendation

It is recommended to enforce authorization when calling the XPC helper.
These authorization checks should contain:
– The caller is a valid signed application
– The caller application was well hardened against DYLIB injection attacks (runtime flag)
– The caller application has the correct TeamID from the Software Company
An excellent resource for the authorization checks can be found at https://blog.obdev.at/what-we-have-learned-from-a-vulnerability/.
The following example *shouldAcceptNewConnecton* functions displays the different stages of the correct authorization checks:

Partial Fix

er if the caller is compiled with the runtime flag (e.g., *0x1000)*. Thus, it is possible to load an older version of Forklift,

This problem was also noticed by Csaba Fitzl (Twitter: @theevilbit). Huge shoutout to him as I learned a lot from his excellent resources!

Entitlements LPE CVE-2020-27192

Now the current measures that are in place are checking whether the Forklift application itself is calling the helper.
It is possible to bypass this using a technique called *Dylib* injection. The operating system has a load order where the OS will try to load *dylib* libraries from different resources from the file system or sometimes even by environment variables.

Forklift is compiled with the following runtime flags:

```
codesign -dvvvv /Applications/ForkLift.app/Contents/MacOS/ForkLift
Executable=/Applications/ForkLift.app/Contents/MacOS/ForkLift
Identifier=com.binarynights.ForkLift-3
Format=app bundle with Mach-O thin (x86_64)
CodeDirectory v=20500 size=27271 flags=0x10000(runtime) hashes=843+5 location=embedded
```

The system will prevent such *dylib* loading from happening. However, there is a catch which disables these hardening features which is called the entitlement. Entitlements can bypass this loading prevention and unfortunately forklift uses the entitlement to disable library validation by default.
The following displays that:

```
jtool2 --ent /Applications/ForkLift.app/Contents/MacOS/ForkLift
Note: 15214 symbols detected in this file! This will take a little while - generate a companion file or use '-q' for quick mode..
<?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>com.apple.security.application-groups</key>
    <array>
        <string>J3CP9BBBN6.com.binarynights.ForkLift</string>
    </array>
    <key>com.apple.security.automation.apple-events</key>
    <true/>
    <key>com.apple.security.cs.disable-library-validation</key> <------
    <true/>
    <key>com.apple.security.personal-information.photos-library</key>
    <true/>
</dict>
</plist>
```

This is the same for the helper:

```
jtool2 --ent /Library/PrivilegedHelperTools/com.binarynights.ForkLiftHelper
<?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>com.apple.security.automation.apple-events</key>
    <true/>
    <key>com.apple.security.cs.disable-library-validation</key>    <-------
    <true/>
    <key>com.apple.security.personal-information.photos-library</key>
    <true/>
</dict>
</plist>
```

Thus, it is possible to inject a library at load time of forklift which connects to the XPC helper service and exploits the same vulnerability as previously.
This attack is described in detail at the following blogposts:
– https://objective-see.com/blog/blog_0x56.html
– https://theevilbit.github.io/posts/dyld_insert_libraries_dylib_injection_in_macos_osx_deep_dive/
– https://secret.club/2020/08/14/macos-entitlements.html

Recommendation

It is recommended to not have the entitlement *com.apple.security.cs.disable-library-validation* in your application. Further, it is recommended to check whether the entitlements are securely set.

Disclosure Timeline

– 08.06.2020 Initial Contact for Mail Address
– 08.06.2020 Response to send the same mail unencrypted
– 09.06.2020 Follow Up question for encryption
– 10.06.2020 Clearance of Encrypted Disclosure
– 12.06.2020 Disclosure of the LPE
– 15.06.2020 Partial Fix for CVE-2020-15349 (LPE)
– 16.06.2020 Disclosure of Entitlements LPE
– 19.06.2020 Disclosure of only partial fix of CVE-2020-15349
– 19.08.2020 Fix of CVE-2020-27192 and Final Fix for CVE-2020-15349

Final Words

I have to add that the communication with the vendor was very responsive, and even when they had a hard time implementing all recommended controls, they were all the time very professional! Thus, I will for sure further use this excellent file viewer.

Cheers,
Birk

‹