

Forum:[Hack Forums](#)

Thread: [MOS Legendary DMG Mount Loader / MacOS App / 100% SEXY & CLEAN](#)

Hackers - October 19, 2025, 7:14 pm

MOS Legendary DMG Mount Loader Dev Notes & Sneak Preview

I am still working on things, but wanted to share a sneak preview.

1. Source file is a .dmg, rather than a .app or Unix executable. (example: Burp Suite CE app, but with original .dmg)
2. The builder extracts the app name & icon from inside the dmg.
3. The builder encrypts the dmg using a secret key. It uses a basic bin file (temporarily) during build (sh build.sh blah.dmg)
4. Builder then generates 'blah', the unix executable version. This is basically the stub program which decrypts the dmg to tmp, mounts, deletes tmp data, then runs the app inside.
5. The builder then takes blah (unix executable) and creates a final .app package as well with same name & icon as the original.
6. The builder also creates a new .dmg (dmg wrapping an App which is wrapping a dmg wrapping an App! Woo, tongue twister).
7. In the future (next steps/notes below), it will continue to create the two separate dist/ versions (App & DMG), but also allow for different inputs too (DMG, App, Unix, etc), and also allow for multi-file binding to run all.

But why .dmg files?

I just wanted to try something new, and I had a little trouble with handling the raw App originally (not single file).

I then realized that the DMG mounting & loading added an extra layer of security/privacy in runtime, so I stuck with it to get it fully working.

One final touch I added today was extracting the icon & file name. I need to check for edge cases, but this function here finds the app package inside the dmg and pulls that data.

It uses hdiutil to mount the original dmg in readonly mode to get the info. This is another benefit of working with dmg, so a real App is not exposed or executable in your project dir. By using a dmg as a dev, you could protect yourself from accidents. For example, AI loves to test things, but using a .dmg can help if you are working with sensitive or other intense data/programs.

```
Code def extract_app_info_from_dmg(dmg_path, output_dir):  
    """Extract app name and icon from DMG"""  
    import subprocess
```

```

import tempfile
import shutil

app_name = "DMGLauncher"
icon_path = None

try:
    # Create temporary mount point
    with tempfile.TemporaryDirectory() as temp_dir:
        mount_point = os.path.join(temp_dir, "dmg_mount")
        os.makedirs(mount_point, exist_ok=True)

        # Mount the DMG
        mount_cmd = ["hdiutil", "attach", dmg_path, "-readonly", "-mountpoint", mount_point,
"-nobrowse", "-quiet"]
        result = subprocess.run(mount_cmd, capture_output=True, text=True)

        if result.returncode == 0:
            # Find the first .app bundle
            find_cmd = ["find", mount_point, "-name", "*.app", "-type", "d", "-maxdepth", "2"]
            find_result = subprocess.run(find_cmd, capture_output=True, text=True)

            if find_result.returncode == 0 and find_result.stdout.strip():
                app_path = find_result.stdout.strip().split('\n')[0]
                app_name = os.path.basename(app_path).replace('.app', '')

                # Look for icon files - check all .icns files in Resources
                resources_dir = os.path.join(app_path, "Contents", "Resources")
                icon_candidates = []

                if os.path.exists(resources_dir):
                    for file in os.listdir(resources_dir):
                        if file.endswith('.icns'):
                            icon_candidates.append(os.path.join(resources_dir, file))

                # Also check common icon names
                icon_candidates.extend([
                    os.path.join(app_path, "Contents", "Resources", "AppIcon.icns"),
                    os.path.join(app_path, "Contents", "Resources", f'{app_name}.icns'),
                    os.path.join(app_path, "Contents", "Resources", "icon.icns"),
                ])

                for icon_candidate in icon_candidates:
                    if os.path.exists(icon_candidate):
                        # Copy icon to a persistent location
                        persistent_icon_path = os.path.join(output_dir, "app_icon.icns")
                        shutil.copy2(icon_candidate, persistent_icon_path)
                        icon_path = persistent_icon_path
                        break

            # Unmount the DMG
            subprocess.run(["hdiutil", "detach", mount_point, "-quiet"], capture_output=True)

```

except Exception as e:

```
print(f"Warning: Could not extract app info from DMG: {e}")
```

```
return app_name, icon_path
```

That was mostly just for ease of use & quality of life in the dev/compile flow.

Edge / Use Cases

I wanted to make sure big files, and complicated programs still work, so I used Burp Suite Community Edition for testing too:

It's basically forcing everything into a file I called "secure_runtime" (the unix executable in scans below), which is packaged into the App/MacOS/ folder, which lets us fully control icons and other data. We could then take our final app and create a new dmg if we want. This would be more familiar to users who may want to drag our program into Applications or another folder of theirs.

Next Steps

1. Update builder to accept .app, unix executable, or .dmg for input, for flexibility. Mostly done
2. Update builder to allow more than one input file (ex: Burp.app, Blah.dmg, and Payload...) at a time, to bind together into one final prog.
3. Update stub to launch more than 1 encrypted bin at same time.
4. Release POC for binder / loader.

Scan Results

Concept results so far. This is a mix of scans, mostly because Virus Total can't handle .app (it's not really a single file) very well. I wanted to show you guys a few different final file types. These all come from the same builder. Regardless of whether the final payload prog is in a zip, app, or unix form, I am seeing 0-1 detects in testing. This will help me document the process for blue teamers to defend against. I always upload to VT to help them detect hackers. The MOS Legendary software series is whitehat in nature. Test programs are used to understand the effectiveness and privacy accomplished by the MOS builders.

Original 21/62

<https://www.virustotal.com/gui/file/90596fdc4ae84e8248f406378efb9765a625a93d1404204043d6888fb6ae6dae?nocache=1>

Original 28/62

<https://www.virustotal.com/gui/file/e5126f74d430ff075d6f7edcae0c95b81a5e389bf47e4c742618a042f378a3fa/detection>

versus...

Wrapped 0/65

<https://www.virustotal.com/gui/file/679000272d6eb9b236956f26bc9bf3f798f4554732adc46f810b9e304b22f5fd?nocache=1>

Wrapped 1/50

<https://www.virustotal.com/gui/file/15365ebf94246dc0de55ca7b23a326e250655b1d3758c6f82803721242f9c732/behavior>

Wrapped 1/63

<https://www.virustotal.com/gui/file/15365ebf94246dc0de55ca7b23a326e250655b1d3758c6f82803721242f9c732/details>

You will notice that the .zip is the only one at 0. Microsoft Defender kept flagging things, and I noticed this in MOS Legendary Static Crypt too.

I researched this further, such as with a .dmg wrapping my App (which wraps an original dmg). Even when I do this with Burp, I see a Microsoft flag in static analysis:

<https://www.virustotal.com/gui/file/d8de02f6d98e35051b2c3536512ccb1db71a9d4446b4399ed4837f6db9d5c41d?nocache=1>

There will be more work to be done on the base program - Program:Script/Wacapew.A!ml

However, I did find a solve besides .zip - using a dmg. This works, and also normalizes the experience for users. I would need to update this to either match the original DMG's installer dynamically, or to at least look better.

That did work well on the Burp test!

Trying it with a payload again, but we're back to a MS detection: Trojan:Script/Wacatac.B!ml

<https://www.virustotal.com/gui/file/ebdf1458a5e1786a56277355781b81d60a516493a35c68a8578df0cf934251cb?nocache=1>

So we've swapped detects! More for another day...

I set up the builder to create both, for flexibility, but am stopping here at this point.

MOS Legendary Series

This dmg method started from my original POC here (XOR, drop to tmp, and run):
<https://hackforums.net/showthread.php?tid=6311006>

I am going to get the simple version done ASAP and release that as MOS Legendary DMG Mount Loader (done, just needs documentation and a git repo).
That will only work with the dmg original files, and only for 1 being wrapped.

The next phase will be released as POC for a program: MOS Legendary Program Binder, or a similar name.

The MOS Legendary series is something I'm working on for learning macOS specific features, program types, etc for research & curiosity.

How to Detect It

Honestly, there's nothing really special happening yet. Besides some very simple encryption, this is all just with basic tmp files. I am clearing out data when it's not needed during the process of embedded bytes -> encrypted tmp -> dmg decrypted & mounted -> run... I haven't added any other features yet.

I feel like competent devs could replicate this flow pretty easily.

For blue teamers looking to detect:

- > encrypted bytes to tmp
- > decrypted & matching dmg format
- > mounting a dmg, quietly
- > launching from the dmg, no user interaction

MeW11 - October 20, 2025, 7:15 am

amazing post! thank you for this my friend!

Hackers - October 21, 2025, 5:11 am

(Oct 20, 2025 07:15 AM)MeW11 Wrote: [\[CITE\]](#) amazing post! thank you for this my friend!
No problem, hope it inspires people to try things

Stanley - April 4, 2026, 8:46 am

I'm Stanley HF's helper bot. The OP (original poster) of this thread wants everyone to see this page and has elected to bump it to the top of the showthread listing. Since HF doesn't want a ton of LQP (Low Quality Posts) this is the best method of bumping your content. There is a small ?yte fee for each bump. Any OP can bump any of their threads on HF from the linked icon on the top of their first post.

If you haven't already read the first post of this thread we encourage you to do so.

Thanks,
Stanley

Hackers - April 14, 2026, 8:31 am

By far one of my more advanced projects (to figure out at the time), but simple to understand and use after the fact... I want to revisit this more and figure out double/triple program binding in 1 app a

bit better.