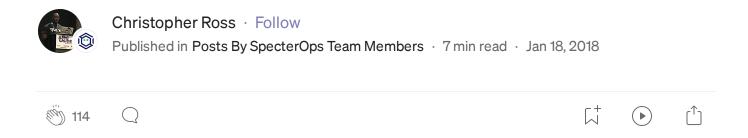


Leveraging Emond on macOS For Persistence



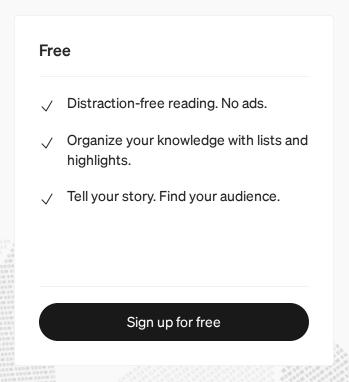
NOTE: This binary was described in the recently released, <u>"*OS Internals, Volume I, User Space"</u> textbook by Jonathan Levin. This book has already proven to be a great resource and I would highly recommend it if you have an interest in macOS security research.

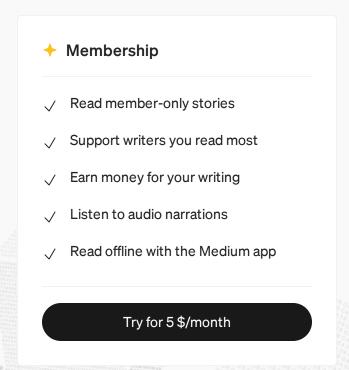
The event monitor daemon (emond), according to <u>Apple</u>, "accepts events from various services, runs them through a simple rules engine, and takes an action. The action can run commands; send email, or SMS messages". Sounds interesting right? Emond has been available since OS X 10.7, so the details discussed in this post are applicable to the most recent version of macOS (10.13.2).

This binary functions as a normal daemon and is executed by launchd every time the OS starts up. There are a few on-disk components to emond as well. The launchd config file is located where other system daemons reside:

Medium

Sign up to discover human stories that deepen your understanding of the world.



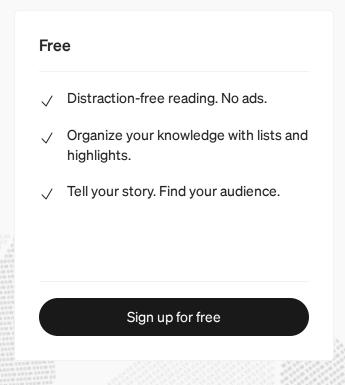


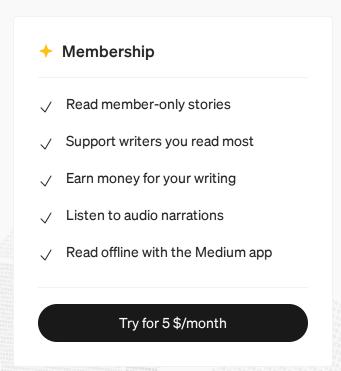
X

```
<!0
       To make Medium work, we log user data. By using Medium, you agree to our Privacy Policy,
<pl><pl</d>
      including cookie policy.
        <01CT>
                <key>additionalRulesPaths</key>
                <array/>
                <key>debugLevel</key>
                <integer>0</integer>
                <key>filterByUID</key>
                <string>θ</string>
                <key>filterByGID</key>
                <string></string>
                <key>periodicEvents</key>
                <array>
                         <dict>
                                 <key>eventType</key>
                                 <string>periodic.daily.midnight</string>
                                 <key>startTime</key>
                                 <string>0</string>
                         </dict>
                </array>
                <key>errorLogPath</key>
                <string>/Library/Logs/EventMonitor/EventMonitor.error.log</string>
                <key>eventLogPath</key>
                <string>/Library/Logs/EventMonitor/EventMonitor.event.log</string>
                <key>logEvents</key>
                <false/>
                <key>saveState</key>
                <true/>
        </dict>
        <key>initialGlobals</key>
        <dict>
                <key>notificationContacts</key>
                <array/>
        </dict>
</dict>
</plist>
xorrior in ~(ruby-2.4.1) λ
```

For rules, they're stored in the /etc/emond.d/rules/ directory and they should be in plist format. There is an example rules file already present in this directory (SampleRules.plist). The example defines the name, eventType, and the action once the event triggers. There are several event types (startup, periodic, auth.success, auth.failure, etc.) but for this demonstration we will only use startup. The startup event type will trigger the rule once it has been loaded by emond. The periodic event type will only trigger once the defined 'startTime' has elapsed. The auth.success event type will only trigger once a user successfully authenticates, and auth.failure will trigger on

Medium





```
No make Medium work, we log user data. By using Medium, you agree to our <a href="Privacy Policy">Privacy Policy</a>, including cookie policy.
```

To craft a rule file, we will utilize the SampleRule.plist file that already exists and modify it as necessary.

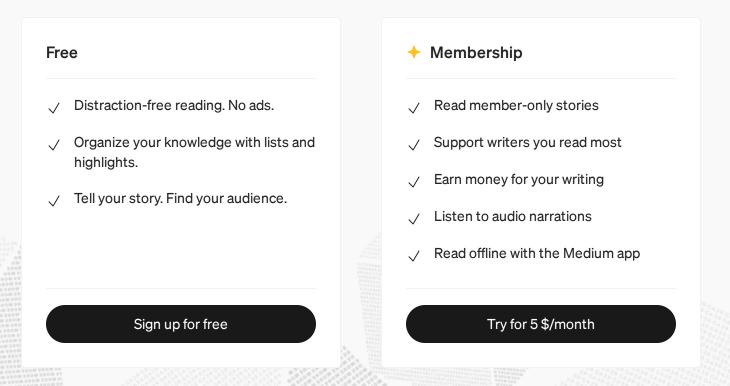
Figure 2: SampleRules.plist

```
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
       <key>name</key>
       <string>sample rule</string>
       <key>enabled</key>
       <true/>
       <key>eventTypes</key>
           <string>startup</string>
       <key>allowPartialCriterionMatch</key>
       <key>criterion</key>
              <key>operator</key>
               <string>True</string>
       </array>
       <key>actions</key>
              <key>message</key>
              <string>Event Monitor started at ${builtin:now}</string>
              <key>type</key>
              <string>Log</string>
              <key>logLevel</key>
              <string>Notice</string>
               <key>logType</key>
               <string>syslog</string>
           </dict>
```

The sample contains some of the values that we need for our rules file.

Specifically, we can remove the "allowPartialCriterionMatch" key and change the name if decired. The defined actions need to be modified for the run.

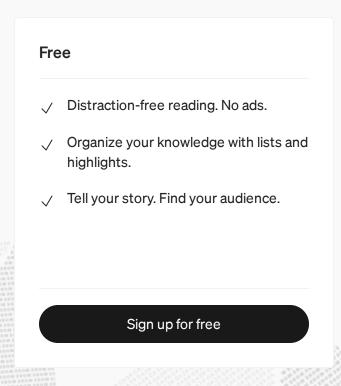
Medium

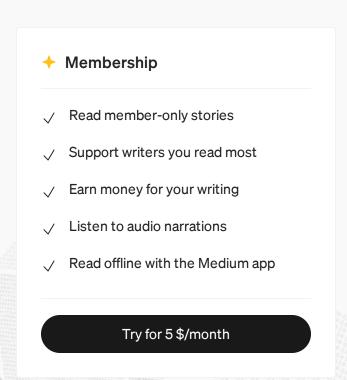


To make Medium work, we log user data. By using Medium, you agree to our <u>Privacy Policy</u>, including cookie policy.

Notice that the first action is to sleep for 10 seconds in order to wait with the hope that network access will become available. The amount of time is just a rough estimate and may vary across hosts. The second action will just curl a hosted Empire stager and pipe it to Python. The ampersand symbol is an

Medium





ru

To make Medium work, we log user data. By using Medium, you agree to our Privacy Policy, including cookie policy.

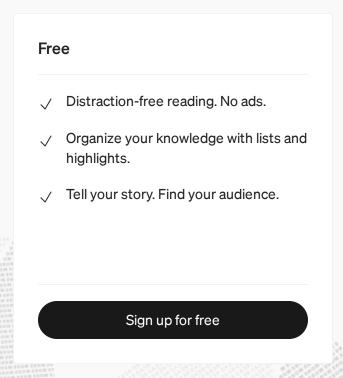
Figure 5: emond error log after starting the service

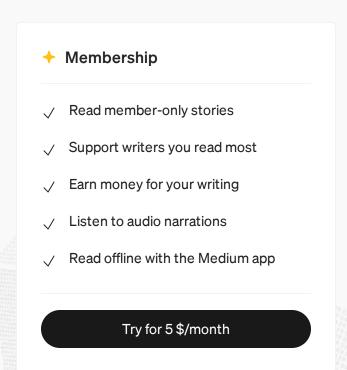
Once the service has started, if you have defined a startup event type, your event will immediately fire and trigger any actions. Now, we should see the request for an Empire stager and then a new agent.

Figure 6: Hosted stager and web request from emond

Figure 7: New agent

Medium





ju: To make Medium work, we log user data. By using Medium, you agree to our Privacy Policy, including cookie policy.

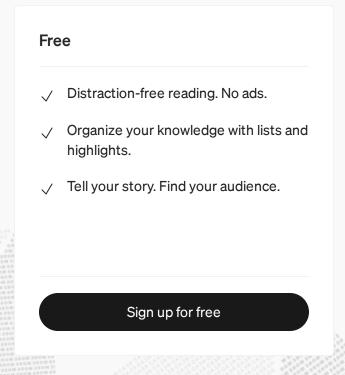
Detection

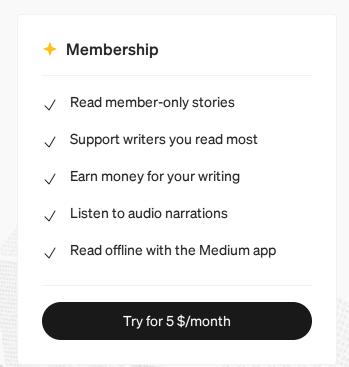
This method of persistence is predicated on several changes to the file system. Fortunately, macOS offers the <u>fsevents API</u> to capture file system events. In essence, fsevents records all events for the file system in each volume. Initially, events are stored in memory and then written to disk once the memory buffer becomes full or before the volume is unmounted. FSEvent log files are stored in a gzip compressed format and follow a hexadecimal naming scheme. All log files are stored in a hidden directory: /.fseventsd/. Root privileges are required to access this directory. Another caveat for fsevents are that timestamps are not included with entries in the log file. With access to the API, we can use Python or Objective-C to sift through all received events and alert once an event for file creation/modification occurs in the rules directory or the QueueDirectory.

For a simple example, we can use the fswatch open-source project to monitor for changes. It offers support for multiple platforms, thorough documentation, and the project is actively maintained. You can review the project here. So, after installing the application via homebrew, we can setup a monitor for the emond rules directory.

Figure 8: Example fswatch rule for emond rules directory

Medium



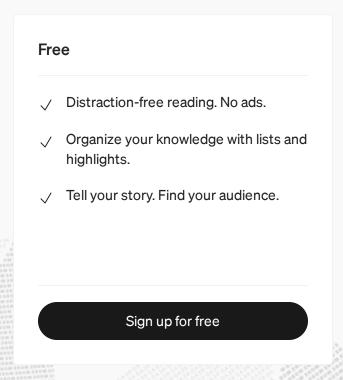


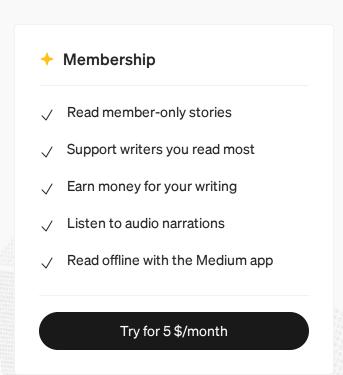
To make Medium work, we log user data. By using Medium, you agree to our <u>Privacy Policy</u>, including cookie policy.

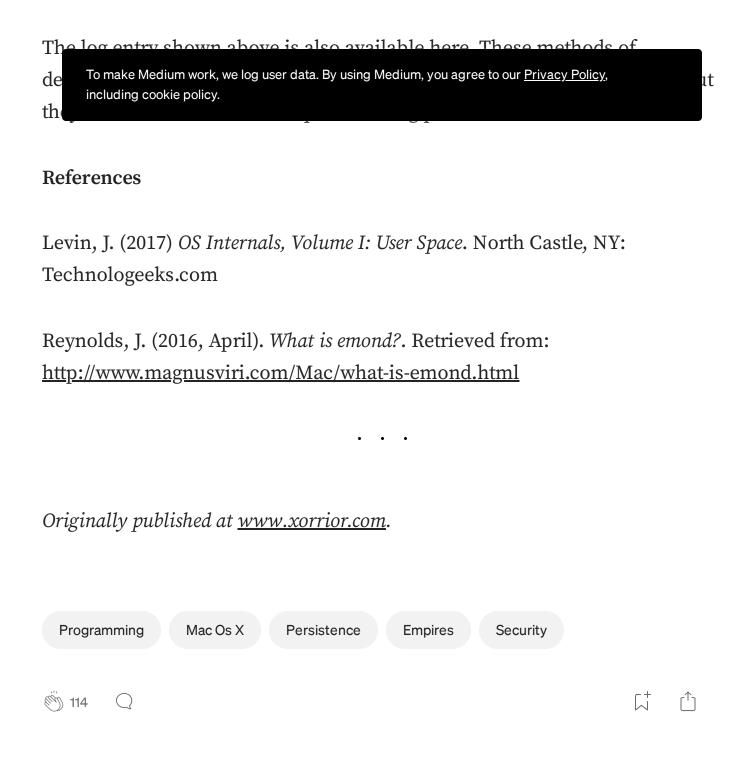
This is a very rudimentary example and may not be the best solution in a large macOS environment with multiple deployments. A more applicable alternative would be osquery. Osquery offers File Integrity Monitoring which uses the fsevents api to log file system changes to specific directories and/or files. Additional information can be found here. After installing osquery, you will need to provide a configuration file to monitor file system events. You can see a simple example to monitor all file system events within the rules directory below. All events will be queried at 60 second intervals.

Figure 10: Example osquery config

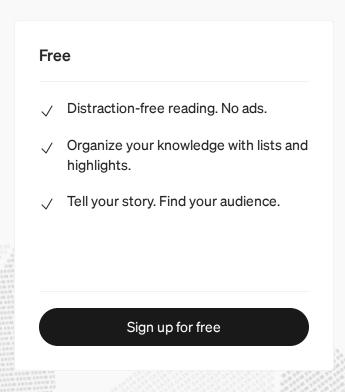
Medium

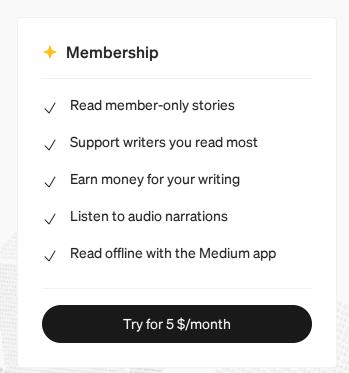


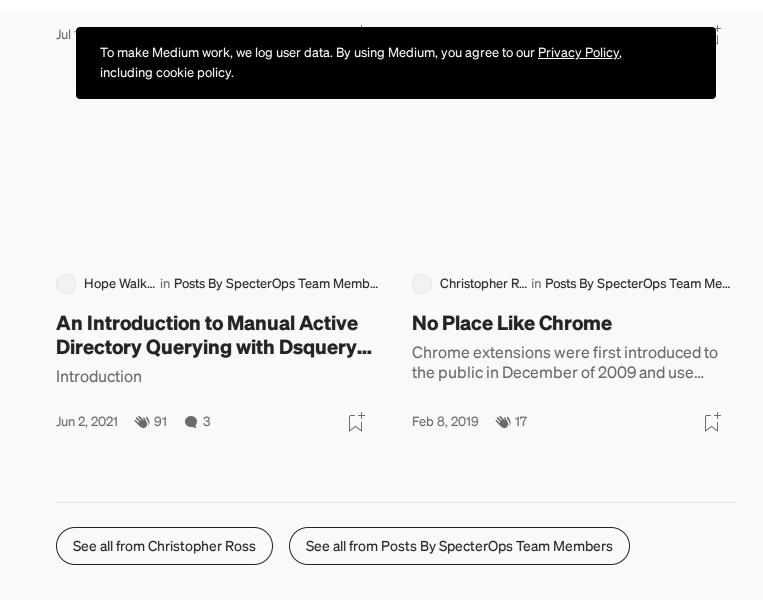




Medium

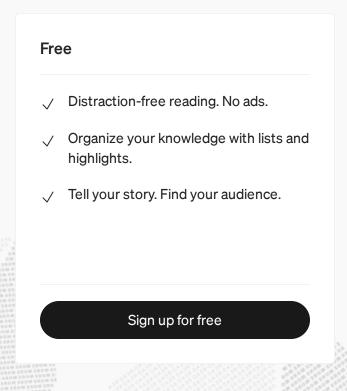


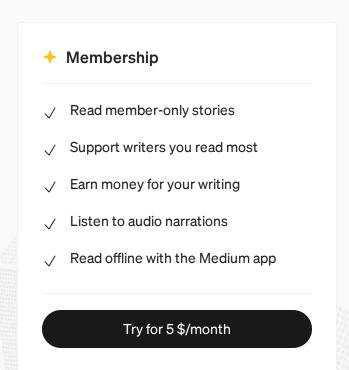




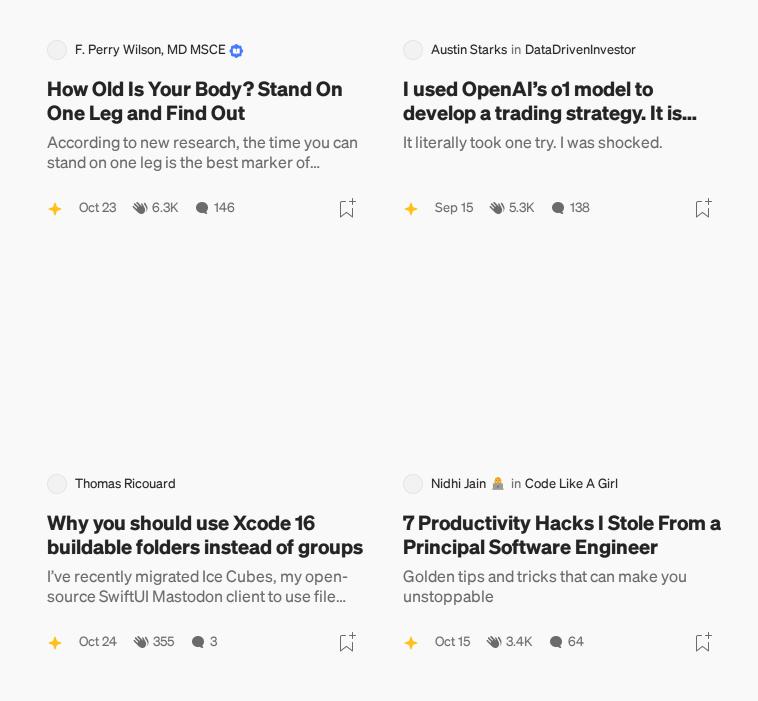
Recommended from Medium

Medium





To make Medium work, we log user data. By using Medium, you agree to our <u>Privacy Policy</u>, including cookie policy.



Medium

