Title: Userland Persistence on Mac OS X

Subtitle: "It Just Works"

Primary Author Name: Josh Pitts

Primary Author Affiliation: Leviathan Security Group

Primary Author Email: the.midnite.runr@gmail.com

Keywords/Tags: OSX, mac, Apple, malware, userland, persistence, file infection, file injection, file patching

— Abstract: —

Userland file infection techniques for OS X are not new. However, they seem somewhat forgotten with all the kernel, boot, and rootkit infection techniques covered over the past seven or eight years. OS X userland remains a free-for all with very few publically available defenses for the home user or enterprise customer. This paper covers the injection of boot processes and core daemons, and available defenses.

— Content: —

Intro and Prior Work

Everything presented in the paper requires root permissions. OS X Userland persistence is not a new topic and has been researched heavily by Patrick Wardle and others.

According to the Joanna Rutkowska's "Introducing Stealth Malware Taxonomy" (pg. 3), the technique being used in this paper is considered "Type 1" Malware. The infection method used is called the "Pre-text Section Infection Method" by the author and the details are outlined here "Patching the Mach-o Format the Simple and Easy Way". This research is built off of prior Mach-O patching techniques from Pedro Vilaça and Roy G Biv though it works across all Mach-O executable binaries for both the LC_MAIN and LC_UNIXTHREAD binary types, as long as the pre-text section is large enough for the malicious payload. This method is included in the Backdoor-Factory (BDF) which supports x86 and x64 chipsets for Mach-O and the LC_MAIN and LC_UNIXTHREAD formats within a FAT file. When patching these processes one does not need to be concerned about signing as this is only a function of applications and enforced by imported dylibs on signed applications. The OS X kernel does not enforce code signing for executable binaries as of the writing of this paper. BDF reduces the number of load commands for each code signing library and thereby un-signs the binary as default behavior. This technique has been know for some time.

Finding Processes

Infecting boot processes and core daemons offers a run-time advantage as the process typically executes as root and, as the author learned, before other security implementations. To determine boot processes and core daemons, the author first looked at post boot processes after user login on a new OS X image for Mavericks and Yosemite by issuing the following command:

```
ps -xu root
```

For Mavericks and Yosemite this resulted in about 50 and 67 processes respectively.

However by inspecting the process status list there is a PID gap between *launchd*, the first process, and the next process. While this space turned out to be unimportant, it was determined that looking into the boot processes was important.

Dtrace is the perfect tool for looking at boot process through anonymous tracing; however Dtrace is broken on OS X. The fix requires patching the kernel.

To work around this, the author used the following metasploit command to build a payload:

```
msfvenom -p osx/x64/exec CMD=/exec.sh -f raw  \> /tmp/exec.bin
```

Which executes this script on the root of the filesystem:

```
#!/bin/bash
i="0"
while [ $i -lt 1000 ]
do
/bin/ps -xu root >> /Users/test/processoutput.txt
sleep .01
i=$[$i+1]
done
```

And the payload was patched this into *launchd* with the following commands using BDF:

```
$sudo ./backdoor.py -f launchd -s user_supplied_shellcode -U exec.bin
[output]
$sudo cp backdoored/launchd /sbin/launchd
$sudo reboot
```

This method found an additional 30 or so processes for Yosemite and Mavericks.

However, Dtrace is more accurate as the *ps* command is a point in time look and processes could be missed. The Dtrace execsnoop script solves this issue:

```
#!/bin/bash
i="0"
while [ $i -lt 1000 ]
do
/usr/bin/execsnoop -A >> /output.log
sleep .01
i=$[$i+1]
done
```

This method found an additional 24 and 25 processes, with a total of over 110 processes on both Yosemite and Mavericks.

Patching Processes

When it comes to network availability during the boot process, there are two states, pre-networking and post-networking. The author found that many processes launch pre-networking and prevent the normal execution of a standard reverse shell. Therefore, two payloads where developed - a delay reverse tcp shell and a beaconing reverse tcp shell. The delay payload works by waiting X seconds before launching the payload and doing so only once; the beaconing payload launches the payload every X seconds, repeating forever. For testing, the beaconing payload was employed with the default setting of 15 seconds.

With the pre-networking issue solved, over 200 boot and core processes needed testing. Two scripts were created to patch (infect) these binaries with regular payloads and beaconing payloads. Testing each binary by hand resulted in a time consuming endeavor. An additional script was developed to automate this process using python, VMFusion, vmrun, and BDF. A time lapse video of the process is located here.

Results

	Yosemite	Mavericks
Boot Processes	118	111
Pre-Networking	55	24
Post-Networking	36	59
Not Viable	27	28

Detailed results for Yosemite and Mavericks including KnockKnock analysis. The 'Not Viable' processes are a result of binaries that either could not be patched because of a small pre-text section, the OS would not boot after patching the binary, or the process was properly sandboxed and would not connect out. The reader should make note that most of the tested binaries were viable for patching/file infection.

The most interesting startup processes noted are:

- /sbin/launchd
- /usr/libexec/xpcproxy
- /usr/sbin/sshd
- /usr/bin/awk

The first process to launch after the kernel is launchd. It holds the equivalent responsibility to /sbin/init on Linux. Since *launchd* executes before other security implementations, such as antivirus, whitelisting, and application based network filters, it can avoid detection. Further, removing an infected *launchd* binary is troublesome; without *launchd*, OS X will not successfully boot. An example of an infection process is demonstrated in a video, with the script available on github.

The XPC framework is the OS X interprocess communication technology and is implemented via xpcproxy. Xpcproxy usually launches multiple instances when an application is launched and executes as root. See the xpcproxy demo video for an example.

The sshd daemon executes as root only when the service is enabled and when any packet is routed to the service port. This is a setting in its plist file called "Sockets". To find all the services that use "Sockets" execute the following command:

```
grep -r -i "<key>Sockets</key>" /System/Library/Launch*
```

The most surprising boot process is perhaps /usr/bin/awk. Launchd executes an ntp script '/bin/sh /usr/libexec/ntpd-wrapper' which contains awk. This script is executed with root privileges, therefore so is awk. See the demo video for both the pathcing of awk and sshd.

Defenses

The majority of the videos previously mentioned had anti-virus installed on the demo machines. Anti-virus is not equipped to handle malicious code patched into valid binaries. Whitelisting is the preferred solution. Until recently there have been no solutions available to everyday consumers. In November 2014, Google Santa (not an official Google product) was released, which occurred after this topic was submitted to ShmooCon CFP. Santa works by loading a kext (driver) and having a daemon (santad) that conducts the actual checking of binaries. If a binary is executed before santad and if santad itself is infected, both will be given a pass as demonstrated in this video. Network filters such as Little Snitch fall under same type of attack, if the process is executed before the first Little Snitch daemon then the network traffic will not be filtered.

KnockKnock by Patrick Wardle provided the best results of all the tests, though it missed about half of the processes. This was due to the script not checking everything on disk and only reviewing known persistence locations. For example, KnockKnock overlooks the 'Socket' plist launch setting and binaries that were executed by scripts, such as awk, and simply did not check the binary such as launchd. Patrick Wardle updated KnockKnock before the conference to check currently running processes; however, the update will not catch processes like awk or perhaps sshd. Note: KnockKnock is not a whitelisting solution, but rather an OS X persistence location checking script. It executes as needed by the user, or via jobs, but does not provide continuous monitoring.

Solutions

Recommendations:

- For Apple to make available an OS X Enterprise Edition where only signed applications can execute
- For Apple to include a TPM to verify the kernel and boot process
- For hashing tools to check the entire disk, there are simply too many places to hide to not check everything

Since the conference, Yelp Security Engineering has released an update to their OSXCollector tool with an option to check everything on disk. Initial tests provided full coverage of the disk partition and an expected long runtime of approximately 50 minutes on a new VM.