MacOs System Identification & Triage Using Terminal

Scenario: A Mac computer has been handed to you with no documentation. Your task is to use the terminal to identify the system, its operating environment, and any connected devices so that further security analysis can be performed.

Before any investigation or troubleshooting can begin, any analyst must first determine what kind of system they are working on. Different operating systems, kernels, and hardware architecture behave differently.

System identification must follow this order:

1. Operating System – Identify whether the system is macOS, Linux, Windows, or another OS. This determines which tools, logs and commands are available
2. Kernel - Determine the kernel type and version (e.g. Darwin, Linux). Security vulnerabilities, drivers and system behaviour depend on the kernel.
3. Hardware Architecture – Identify whether the system is running on Intel (x86_64) or Apple Silicon (ARM64). This affects binaries, drivers and exploit compatibility
4. Shell Environment - Determine which shell is in use (bash, zsh, sh, etc), as command behaviour, scripting, and environment variables depend on the shell.
5. User and Privileges – Identify the logged-in user and their permission level to understand what actions can be performed and whether administrative access is available.
6. Network Interface - Identify how the system connects to networks, including Wi-Fi, Ethernet, VPNs, and USB network adapters.
7. Connected Devices (USB & Internal Hardware) – Identify all hardware devices connected to the system, including both external peripherals (Such as printers, storage devices, and USB hubs).

```
bash-3.2$ sw_vers
ProductName:    macOS
ProductVersion: 11.7.10
BuildVersion:   20G14[redacted]
bash-3.2$ uname -a
Darwin [redacted] 20.6.0 Darwin Kernel Version 20.6.0: Thu Jul  6 22:12:47 PDT 2023; root:xnu-
7195.141.[redacted]~1/RELEASE_X86_64 x86_64
bash-3.2$ echo $SHELL
/bin/bash
bash-3.2$ ps -p $$ -o comm=
bash
bash-3.2$ whoami
[redacted]
bash-3.2$ id
uid=501([redacted]) gid=20(staff) groups=20(staff),101(access_bpf),701(com.apple.sharepoint.group.1),12
(everyone),61(localaccounts),79(_appserverusr),80(admin),81(_appserveradm),98(_lpadmin),33(_appst
ore),100(_lpoperator),204(_developer),250(_analyticsusers),395(com.apple.access_ftp),398(com.appl
e.access_screensharing),399(com.apple.access_ssh),400(com.apple.access_remote_ae)
bash-3.2$
```

1. Identifying the operating system of the Mac using the “sw_vers” command

The Mac outputs:

```
ProductName:    macOS
ProductVersion: 11.7.10
BuildVersion:   20G1[redacted]
```

OS: System is running macOS, specifically version 11.7.10 (Big Sur).

2. Identifying the kernel using the command “uname -a”

```
Darwin 192.168.1.[redacted] 20.6.0 Darwin Kernel Version 20.6.0: Thu Jul  6 22:12:47 PDT 2023; root:xnu-
7195.141.[redacted]/RELEASE_X86_64 x86_64
```

Kernel: System is running Darwin Kernel 20.6.0

3. Identifying Hardware Architecture using the command “uname -a”

We must identify whether the system is intel (x86_64) or Apple Silicon (arm64), which affects software and exploit compatibility.

Mac output:

```
X86_64 x86_64
```

CPU architecture: System is running x86_74 (intel).

4. Identifying the Shell

We first identify the default shell being used when a user is logged in: “echo $SHELL” command.

The Mac outputs:

```
/bin/bash
```

This Mac’s default shell is bash.

If we want to identify which shells are available, we have to read the file which contains all the shells using: “cat /etc/shells”.

The /etc/shells file was also checked to verify that only legitimate shell programs are permitted as login shells, which help detect shell-based persistence mechanisms.

The Mac outputs:

```
# List of acceptable shells for chpass(1).
# Ftpd will not allow users to connect who are not using
# one of these shells.

/bin/bash
/bin/csh
/bin/dash
/bin/ksh
/bin/sh
/bin/tcsh
/bin/zsh
```

The available shells in the /etc/shells file are: bash, csh, dash, ksh, sh, tcsh, zsh.

The /etc/shells file was checked to verify that only legitimate shell programs are permitted as login shells, which help detect shell-based persistence mechanisms.

To identify the current shell being used we use "ps -p $$ -o comm="

Mac outputs:

```
bash
```

The shell currently being used in this session is bash.

5. Identifying User and User Privileges using "whoami" & "id" command

After identifying the shell environment, the next step is to determine which user is currently logged in and what of level access they have. Different users have different permissions, and many system-level actions require administrative or root privileges.

Mac outputs:

```
bash-3.2$ whoami
[redacted]
bash-3.2$ id
uid=501([redacted]) gid=20(staff) groups=20(staff),101(access_bpf),701(com.apple.sharepoint.group.1),12
(everyone),61(localaccounts),79(_appserverusr),80(admin),81(_appserveradm),98(_lpadmin),33(_appst
ore),100(_lpoperator),204(_developer),250(_analyticsusers),395(com.apple.access_ftp),398(com.appl
e.access_screensharing),399(com.apple.access_ssh),400(com.apple.access_remote_ae)
```

We confirm that the "whoami" output is user: ****

Using the "id" command we also confirm that the user is not the root account, as the user's unique identifier (UID) is 501 (the root account has UID 0). The output also shows that the user is a member of the admin group, meaning they have administrative privileges and can perform privileged actions using sudo.

6. Identifying Network interface using “ifconfig” command

After identifying the user and the privileges, the next step is to determine how the system is connected to networks. This includes physical interfaces such as WI-FI and Ethernet, as well as virtual interfaces used by VPNs, tunnels, and virtual machines. Network interfaces are important because they define how data enters and leaves the system and whether the machine is exposed to external networks.

Mac outputs:

en0 interface is up and active:

The en0 and has been assigned IPv4 and IPv6 addresses, indicating that the system is currently connected to a network via Wi-Fi and has obtained network configuration through DHCP and IPv6 autoconfiguration.

From the active En0 interface we can confirm the following information about the Mac system:

IPv4 address: 192.168.1.xxx
IPv6 address: fe80: 18ea: xxx: xxx: xxx: xxx
Mac address: c4: b3: xx: xx: xx: xx

adwl0 (Apple Wireless Direct Link) interface is up and active:

adwl0 interface is used for Airdrop, Airplay, Bluetooth, Device discovery and Apple peer to peer networking.

The awdl0 interface is currently active on the system, indicating that Apple's peer to peer wireless networking is enabled and the Mac is capable of discovering and communicating directly with nearby Apple devices using features such as AirDrop and Airplay.

llw0 (Low- Latency Wireless) interface being up and active:

llw0 interface currently active, indicating that the system has Apple's low latency peer to peer wireless channel enabled for features such as Sidecar, Handoff, and real-time device-to-device communication.

Examining all network interfaces allow an analyst to identify every communication channel available on the system. By mapping these interfaces, it becomes possible or detect unexpected or unauthorised connections that could be used for data exfiltration, remote access, or command-and-control traffic.

7. Identifying Connected Devices (USB & Bluetooth) using "system_profiler SPUSBDataType" & "system_profiler SPBluetoothDataType" command

USB interface Enumeration (Physical Attack Surface)

Command used: System_profiler SPUSBDataType

```
bash-3.2$ system_profiler SPUSBDataType
2026-01-17 14:19:03.526 system_profiler[89358:7755729] SPUSBDevice: IOCreatePlugInInterfaceForServic
e failed 0xe00002be
2026-01-17 14:19:03.527 system_profiler[89358:7755729] SPUSBDevice: IOCreatePlugInInterfaceForServic
e failed 0xe00002be
USB:

    USB 3.0 Bus:

      Host Controller Driver: [redacted]
      PCI Device ID: [redacted]
      PCI Revision ID: [redacted]
      PCI Vendor ID: [redacted]

        Apple Internal Keyboard / Trackpad:

          Product ID: [redacted]
          Vendor ID: [redacted]
          Version: [redacted]
          Serial Number: [redacted]
          Manufacturer: [redacted]
          Location ID: [redacted]

        Bluetooth USB Host Controller:

          Product ID: [redacted]
          Vendor ID: [redacted]
          Version: [redacted]
          Manufacturer: [redacted]
          Location ID: [redacted]
```

This command was used to identify all devices connected via the USB subsystem in order to determine which hardware components are capable of interacting directly with the operating system.

The output of “system_profiler SPUSBDataType identified a single USB 3.0 bus containing only the following components:

- **USB Host Controller Driver**, responsible for managing USB communications between the operating system and connected devices.
- **Apple Internal Keyboard**, a permanent integrated Human Interface Device
- **Bluetooth USB Host Controller**, an internal component providing Bluetooth functionality via the USB bus.

No external USB peripherals, removable storage devices, or third-party USB hubs were present at the time of inspection.

Security interpretation

The presence of only internally integrated USB components indicates that the system was operating with a **minimal USB attack surface**. As no external or removable USB devices were connected, the risk of USB-based-threats such as unauthorised mass storage access or firmware-level attacks- was significantly reduced.

Bluetooth Interface Enumeration (Wireless Attack Surface)

Command used: System_profiler SPBluetoothDataType

```
Apple Bluetooth Software Version: 8.0.5d7
Hardware, Features, and Settings:
    Name: [redacted]
    Address: [redacted]
    Bluetooth Low Energy Supported: Yes
    Handoff Supported: Yes
    Instant Hot Spot Supported: Yes
    Manufacturer: [redacted]
    Transport: USB
    Chipset: [redacted]
    Firmware Version: v169 c4825
    Bluetooth Power: On
    Discoverable: Off
    Connectable: Yes
```

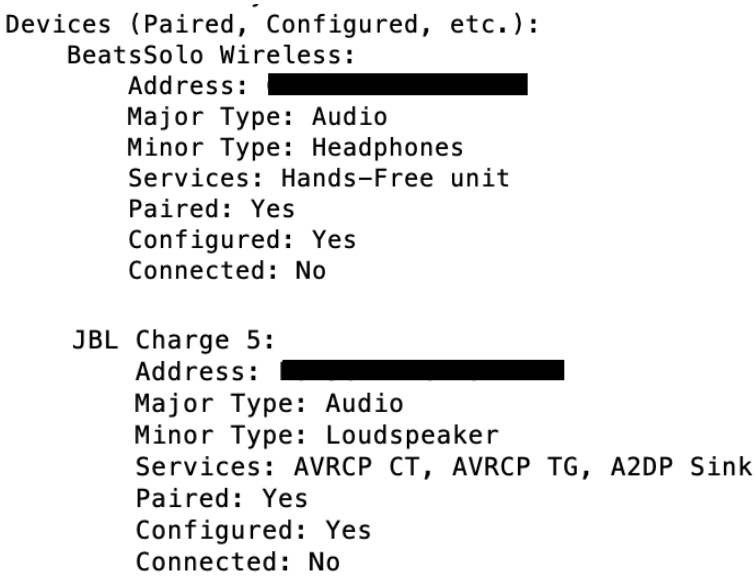
The screenshot above shows the status of the operating system’s Bluetooth software stack and the underlying controller configuration. The Bluetooth controller is internally integrated and connected via the USB subsystem, confirming that it is managed by a kernel driver rather than exposed as an external peripheral.

Bluetooth power was enabled at the time of inspection; however, the device was not discoverable, which reduces exposure to unsolicited pairing attempts. While the controller

was marked as connectable, this state only permits connections from previously authorised devices and does not allow anonymous access.

Overall, this configuration indicates that Bluetooth functionality was enabled but restricted to trusted interactions, with no open discovery or unauthorised connection capability present at the time of analysis.

Paired devices:



The Bluetooth software stack reported two paired and configured devices shown in the screenshot above, both of which were not actively connected at the time of inspection. This indicates that while trust relationships with these devices exist at the operating system level, no Bluetooth active sessions were present during analysis.

From a security perspective, inactive paired devices, present a low risk than active connections, as no real-time data transfer or control channels are established. However, persistent pairing records may allow automatic reconnection if the device return within range, and therefore should be periodically reviewed and removed if no longer required.

Security Learning Outcomes & System Conclusions – Checklist

System Identification & Environment

✓ Identified the operating system, kernel, and hardware architecture, shell environment, user privilege level to establish a complete bassline understanding of the Mac
✓ Identified the current shell in use and verified that only legitimate shells were present, confirming that no unauthorised or malicious programs was masquerading a shell to intercept user activity.

✓ Confirmed the system was not operating as the root user, reducing the risk of unrestricted system-level access, and limiting the impact of potential misuse or compromise.
✓ Analysed the user context to confirm administrative privileges were available without
✓ Examined all active network interfaces to understand how the system communicates externally and identify potential exposure points
✓ Performed full USB diagnostics to identify all physical devices capable of interacting with the operating system and confirm that no unauthorised or foreign peripherals were present
✓ Confirmed that no external USB devices or active Bluetooth connections were present that could pose a security risk
✓ Determined that both physical and wireless attack surfaces were minimal and securely configured at the time of analysis
✓ Concluded that the Mac was operating in a low-risk state, providing a secure baseline for further security investigation if required