How to Perform Situational Awareness Attacks, Part 1

By **tokyoneon** 09/15/2018 12:46 am

The first few minutes after gaining access to a MacBook are critical — but where do we begin? Using tools built into macOS, we can develop an in-depth understanding of running background processes, detect antivirus software, locate sensitive files, and fingerprint other devices on the network. All of this can be done without installing additional software or modifying any files.

What Is Situational Awareness?

During most red team engagements, after compromising a target, pentester's will often find they need to learn as much about the device and its network surroundings as possible. This is commonly referred to as "situational awareness." This is the act of gathering hardware, software, and network information about the target. This information can be used to further compromise the target, their online accounts, and pivot to other devices and services within the network.

Our goal as penetration testers is to learn as much about our newly compromised macOS device as possible without alerting the target to our presence. Generally, using tools built into the operating system to perform information gathering will help us evade detection. There are many tools in macOS that we can use to fingerprint the device, the network, and Wi-Fi networks it's connected to. The first (and possibly the most important) tool we'll be talking about is system_profiler.

- **Don't Miss: How to Configure a Backdoor on Anyone's MacBook**

1 Discover Hardware & Software Details

The system_profiler tool was designed to print system hardware and software configurations. It features the ability to export information in XML format and supports several degrees of output verbosity.

In most cases, system_profiler will produce over 55,000 lines of data pertaining to the target macOS device. This data includes very specific hardware details, firewall settings, Wi-Fi adapter details, startup items, and detailed application info, to name just a few.

System_profiler can be used without root privileges and is, therefore, an attacker's greatest tool for quickly discovering hardware and software specifications.

The following system_profiler commands can be executed using a Terminal or from a Netcat backdoor. Use the **--help** argument to view the available options.

```
system_profiler --help

Usage: system_profiler [-listDataTypes]
      system_profiler [-xml] [-timeout n] [-detailLevel n]
      system_profiler [-xml] [-timeout n] [dataType1 ... dataTypeN]

  -detailLevel n    specifies the level of detail for the report
                    mini = short report (contains no identifying or personal information)
                    basic = basic hardware and network information
                    full = all available information

  -listDataTypes    lists all the available datatypes

  -xml              generates xml output instead of plain text
                    if redirected to a file with the extension ".spx"
                    the file can be opened in System Profiler.app

  -timeout          specifies the maximum time to spend gathering information
                    the default is 180 seconds, 0 means no timeout

  Redirect stderr to /dev/null to suppress progress and error messages.
```

The system_profiler "Datatypes" represent different components of the macOS system. For example, using the **SPFirewallDataType** argument will print the device's firewall configuration.

```
system_profiler SPFirewallDataType

Firewall:

    Firewall Settings:

      Mode: Block all incoming connections
      Firewall Logging: Yes
      Stealth Mode: No
```

We've now learned the device has the firewall enabled and is blocking all incoming connections. This small bit of information is critical to an attacker planning their next move and trying to establish persistence.

There's a **-listDataTypes** argument that can be used to view all of the available Datatypes.

```
system_profiler -listDataTypes

Available Datatypes:
SPParallelATADataType
SPUniversalAccessDataType
SPApplicationsDataType
SPAudioDataType
SPBluetoothDataType
SPCameraDataType
SPCardReaderDataType
SPComponentDataType
SPiBridgeDataType
SPDeveloperToolsDataType
SPDiagnosticsDataType
SPDisabledSoftwareDataType
SPDiscBurningDataType
SPEthernetDataType
SPExtensionsDataType
SPFibreChannelDataType
SPFireWireDataType
SPFirewallDataType
SPFontsDataType
SPFrameworksDataType
SPDisplaysDataType
SPHardwareDataType
SPHardwareRAIDDataType
SPInstallHistoryDataType
SPNetworkLocationDataType
SPLogsDataType
SPManagedClientDataType
SPMemoryDataType
SPNVMeDataType
SPNetworkDataType
SPPCIDataType
SPParallelSCSIDataType
SPPowerDataType
SPPrefPaneDataType
SPPrintersSoftwareDataType
SPPrintersDataType
SPConfigurationProfileDataType
SPRawCameraDataType
SPSASDataType
SPSerialATADataType
SPSPIDataType
SPSmartCardsDataType
SPSoftwareDataType
```

SPStartupItemDataType
SPStorageDataType
SPSyncServicesDataType
SPThunderboltDataType
SPUSBDataType
SPNetworkVolumeDataType
SPWWANDataType
SPAirPortDataType

Multiple Datatypes can be used simultaneously. Below, I'm printing the MacBook's OS version and network info.

system_profiler SPSoftwareDataType SPNetworkDataType

```
Software:

    System Software Overview:

      System Version: macOS 10.13.6 (17G65)
      Kernel Version: Darwin 17.7.0
      Boot Volume: macOS
      Boot Mode: Normal
      Computer Name: tokyoneon’s MacBook Air
      User Name: tokyoneon (tokyoneon)
      Secure Virtual Memory: Enabled
      System Integrity Protection: Enabled
      Time since boot: 1:27

Network:

    Wi-Fi:

      Type: AirPort
      Hardware: AirPort
      BSD Device Name: en0
      IPv4 Addresses: 192.168.1.98
      IPv4:
        AdditionalRoutes:
          DestinationAddress: 192.168.1.98
          SubnetMask: 255.255.255.255
          DestinationAddress: 169.254.0.0
          SubnetMask: 255.255.0.0
        Addresses: 192.168.1.98
        ARPResolvedHardwareAddress: xx:xx:xx:xx:xx:xx
        ARPResolvedIPAddress: 192.168.1.1
        Configuration Method: DHCP
        ConfirmedInterfaceName: en0
        Interface Name: en0
```

```
    Network Signature: IPv4.Router=192.168.1.1;IPv4.RouterHardwareAddress=xx:xx:xx:xx:xx:xx
    Router: 192.168.1.1
    Subnet Masks: 255.255.255.0
  IPv6:
    Configuration Method: Automatic
  DNS:
    Server Addresses: 192.168.1.1
  DHCP Server Responses:
    Domain Name Servers: 192.168.1.1
    Lease Duration (seconds): 0
    DHCP Message Type: 0x05
    Routers: 192.168.1.1
    Server Identifier: 192.168.1.1
    Subnet Mask: 255.255.255.0
  Ethernet:
    MAC Address: xx:xx:xx:xx:xx:xx
    Media Options:
    Media Subtype: Auto Select
  Proxies:
    Exceptions List: *.local, 169.254/16
    FTP Passive Mode: Yes
  Service Order: 0

Bluetooth PAN:

  Type: Ethernet
  Hardware: Ethernet
  BSD Device Name: en2
  IPv4:
    Configuration Method: DHCP
  IPv6:
    Configuration Method: Automatic
  Proxies:
    Exceptions List: *.local, 169.254/16
    FTP Passive Mode: Yes
  Service Order: 1

Thunderbolt Bridge:

  Type: Ethernet
  Hardware: Ethernet
  BSD Device Name: bridge0
  IPv4:
    Configuration Method: DHCP
  IPv6:
    Configuration Method: Automatic
  Proxies:
    Exceptions List: *.local, 169.254/16
```

```
    FTP Passive Mode: Yes
  Service Order: 2
```

When using the system_profiler without any arguments, it will use all of the available Datatypes. This will produce an enormous amount of data and can take several minutes to complete.

2 Identify Devices on the Network

The Address Resolution Protocol, known commonly as ARP, translates physical (MAC) addresses into IP addresses. Computers cache ARP information in "ARP tables," which aid routers and devices on the network in quickly locating each other.

The **arp** command can be used to print the macOS device's ARP table and discover devices on the network without performing a single Nmap scan.

```
arp -i en0 -l -a

Neighbor              Linklayer Address  Expire(O) Expire(I)     Netif  Refs Prbs
192.168.1.1           xx:xx:xx:xx:xx:xx  1m36s     1m36s         en0    1
192.168.1.79           xx:xx:xx:xx:xx:xx  expired   1m18s         en0    1
192.168.1.102          xx:xx:xx:xx:xx:xx  expired   1m20s         en0    1
```

The **-i** argument is used to specifies the Wi-Fi interface while **-l** prints the output data in a more human-readable format. To print all of the ARP table entries, use the **-a** argument.

We've discovered several devices on the network. The MAC addresses have been redacted but this information can be used to identify operating systems and hardware details