Living off the Orchard: Leveraging Apple Remote Desktop for Good and Evil | Mandiant

By Mandiant

Published: 2019-10-09 · Archived: 2026-04-05 21:58:41 UTC

Written by: Jake Nicastro, Willi Ballenthin

Attackers often make their lives easier by relying on pre-existing operating system and third party applications in an enterprise environment. Leveraging these applications assists them with blending in with normal network activity and removes the need to develop or bring their own malware. This tactic is often referred to as *Living Off The Land*. But what about when that land is an Apple orchard?

In recent enterprise macOS investigations, FireEye Mandiant identified the Apple Remote Desktop application as a lateral movement vector and as a source for valuable forensic artifacts.

Apple Remote Desktop (ARD) was first released in 2002 and is Apple's "desktop management system for software distribution, asset management, and remote assistance". An ARD deployment consists of administrator and client machines. While the administrator app must be downloaded from the macOS App Store, the client application is included natively as part of macOS. Client systems must be added to the client list on an administrator system manually, or they can be discovered via Bonjour if they are in the same local subnet as the administrator system. In a typical enterprise environment deployment, managers would be the ARD administrators and have the ability to view, manage, and remotely control their managed personnel's workstations via ARD.

Lateral Movement

Mandiant has observed attackers using the ARD screen sharing function to move laterally between systems. If remote desktop was not enabled on a target system, Mandiant observed attackers connecting to systems via SSH and executing a kickstart command to enable remote desktop management. This allowed remote desktop access to the target systems. The following is an example from the macOS Unified Log showing a kickstart command used by an attacker to enable remote desktop access for all users with all privileges:

```
2018-08-19 07:02:11.417255-0400 0x8ab7c0   Default     0x0                  9226
0    sudo:  [redacted] : TTY=ttys000 ; PWD=/Users/[redacted] ; USER=root ;
COMMAND=/System/Library/CoreServices/RemoteManagement/ARDAgent.app/Contents/Resources/
kickstart -activate -configure -allowAccessFor -allUsers -privs -all
```

Figure 1: Kickstart command example

During an investigation, you can use a few different artifacts to trace this activity. Execution of the kickstart command modifies the contents of the configuration file /Library/Application Support/Apple/Remote Desktop/RemoteManagement.launchd to contain the string “enabled”. SSH login activity can be found in the Apple System Logs or Audit Logs. Execution of the kickstart command can be found in the Unified Logs, as seen in Figure 1.

An ARD administrator has a substantial amount of power available to them, similar to compromising an administrator account in a Windows environment. By compromising an account that has access to ARD administrator system, an attacker can perform any of the following actions:

- Remotely control VNC-enabled machines, including in “Curtain Mode” which hides the remote actions from the local workstation’s screen
- Transfer files
- Remotely shut down or restart multiple machines simultaneously
- Schedule tasks
- Execute AppleScript and UNIX shell scripts

Apple’s ARD web page and the ARD help page contain more details about ARD’s capabilities.

ARD Reporting as a Forensic Force Multiplier

Along with remote system control functionality, Apple Remote Desktop’s asset management capabilities include conducting remote Spotlight searches, file searching, generating software version information reports, and more importantly, generating application usage and user history reports. The reporting process generally follows these steps:

1. Client systems compute reports and cache the data locally before transferring them to the administrator system (the default policy is to begin this at 12:00 AM local time, daily).
2. Data received from clients is cached on the administrator system. Alternatively, a macOS system with the administrator version of ARD installed can be set up as a “Task Server” for a centralized collection option.

3. Cached data is written to SQLite database on the administrator system

The cached data is stored in various subdirectories under the /private/var/db/RemoteManagement/ parent directory. The directory has the following structure:

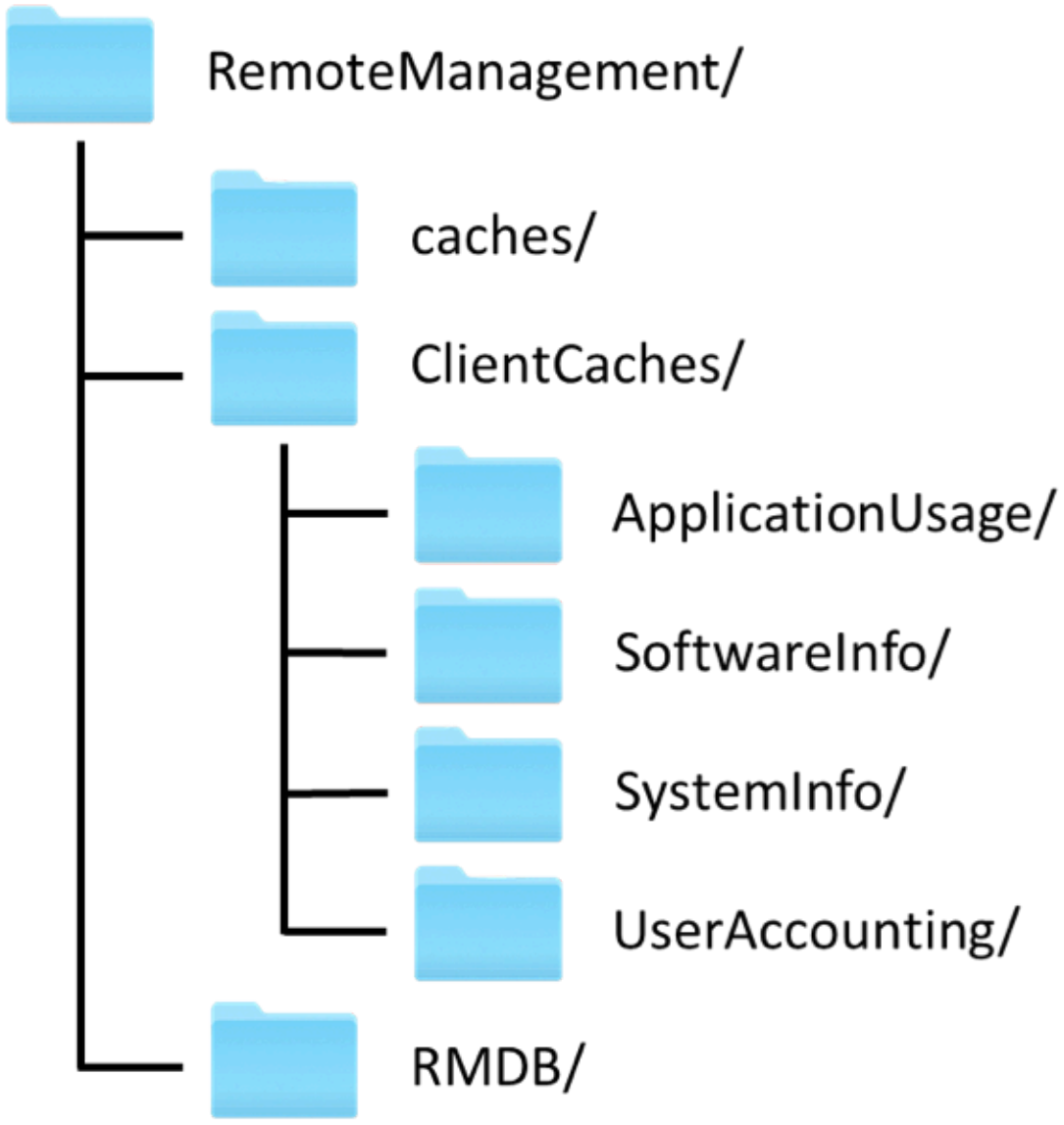

Figure 2: /private/var/db/RemoteManagement/ directory structure

This directory structure is present on all systems, but which files exist in which directories depends on whether the system is an ARD client or administrator system.

Artifacts from ARD Client Systems

There is one directory that is the focus for investigations on client systems: /private/var/db/RemoteManagement/caches/. This directory contains the following files, which are the local client data cache that is periodically reported to the administrator system. Do note, however, that these files are routinely deleted by the system, so they may not be present. These files are typically deleted from the client system once they are transmitted to the administrator system. Once transmitted, the data is stored on the administrator system.

File	Description
AppUsage.plist	plist file containing application usage data
AppUsage.tmp	Binary plist file containing application usage data, often the same as or less thorough than AppUsage.plist
asp.cache	Binary plist of system information
filesystem.cache	Database containing an index of the entire file system, including users and groups
sysinfo.cache	Binary plist containing system information, some of which is also present in asp.cache
UserAcct.tmp	Binary plist containing user login activity

Table 1: ARD cache files

In our experience, the most useful information available from these files is application usage and user activity.

Application Usage

The RemoteManagement/caches/AppUsage.plist file contains one key per application, where each key is the full path of the application, such as file:///Applications/Calculator.app/.

Each application key contains a dictionary that includes a “runData” array and a “Name” string, which is the friendly name of the application, such as “Calculator”, as seen in Figure 3.

```
<?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>file:///Applications/Calculator.app/</key>
   <dict>
      <key>runData</key>
      <array>
         <dict>
            <key>wasQuit</key>
            <true/>
            <key>Frontmost</key>
            <real>130.760330</real>
            <key>Launched</key>
            <real>566327037.983282</real>
            <key>runLength</key>
            <real>1563.459526</real>
            <key>userName</key>
            <string>exampleuser </string>
         </dict>
         <dict>
            <key>wasQuit</key>
            <true/>
            <key>Frontmost</key>
            <real>303.270443</real>
            <key>Launched</key>
            <real>566512918.471135</real>
            <key>runLength</key>
            <real>2100.610710</real>
            <key>userName</key>
            <string>exampleuser </string>
```

Figure 3: AppUsage.plist structure

Each “runData” array contains at least one dictionary consisting of the following keys and values:

Key	Value Format	Description
wasQuit	Boolean: true or false	Indicator of whether or not the application was quit prior to the last report time. This field may not exist if the value is not “true”.
Frontmost	Number of seconds	Total duration which the application was “frontmost” on the screen
Launched	macOS absolute timestamp	Time the application was launched
runLength	Number of seconds	Duration the application was run
username	String	User who launched the application

Table 2: AppUsage.plist runData keys and values

Of the two application usage cache artifacts, RemoteManagement/caches/AppUsage.plist usually contains the same or more content than RemoteManagement/caches/AppUsage.tmp.

User Activity

The RemoteManagement/caches/UserAcct.tmp file is a binary plist that contains user activity that can be correlated with other artifacts on a macOS systems, such as the Apple System Logs or Audit Logs. The file contains keys with the short name of each user logged on the system.

Each key contains a dictionary that includes a “uid” string with the user’s UID, and an array for each login type: console, tty, or SSH. Each login-type array contains at least one dictionary consisting of the following keys and values:

Key	Value Format	Description
inTime	macOS absolute timestamp	Time the user logged in
outTime	macOS absolute timestamp	Time the user logged out
host	String	Originating host for remote login. This field has been observed to not be consistently present.

Table 3: UserAcct.tmp keys and values

Artifacts From ARD Administrator Systems

The data outlined in Table 1 is reported to the administrator system daily. The files are then stored in the RemoteManagement/ClientCaches/ directory. Each file is renamed to the MAC address of the reporting system and placed into the appropriate subdirectory, as seen in Table 4. The subdirectories contain the following:

Subdirectory	Data Contained in Each File
ApplicationUsage/	AppUsage.plist files
SoftwareInfo/	Filesystem.cache files
SystemInfo/	Sysinfo.cache files
UserAccounting/	UserAcct.tmp files

Table 4: /private/var/db/RemoteManagement/ClientCaches/ subdirectories

Additionally, there is a plist file, RemoteManagement/ClientCaches/cacheAccess.plist that contains keys of MAC addresses with values of more MAC addresses. The purpose and context for this file has yet to be determined.

The Gold Mine

All the aforementioned data, with the exception of the filesystem.cache files, is added to the main SQLite database RemoteManagement/RMDB/rmdb.sqlite3 (“RMDB”). The RMDB exists on all ARD systems but is only populated on the administrator system. It houses a wealth of information about the systems in the ARD network over a significant timespan. Mandiant has observed data for application usage timestamps from over a year prior to when we acquired a database on a live system.

The RMDB file contains five tables: ApplicationName, ApplicationUsage, PropertyNameMap, SystemInformation, and UserUsage. The following sections detail each table within the database:

ApplicationName

This table is an index for the applications on each system, where each application is assigned an item sequence number (“ItemSeq”) per system. This data is used for correlation in the ApplicationUsage table.

Column	Value Format	Description
ComputerID	String	Client MAC address, no separators
AppName	String	Friendly application name
AppURL	String	Application URL path (i.e. file:///Applications/Calculator.app)
ItemSeq	Integer	ID number for each application, per ComputerID, used for the AppName table
LastUpdated	macOS absolute timestamp	Last report time of the client

Table 5: ApplicationName table columns

ApplicationUsage

The AppName table is unique in the fact the “Frontmost” and “LaunchTime” values in the table are swapped. The research at the time of this blog post was verified on MacOS 10.14 (Mojave).

Column	Value Format	Description
ComputerID	String	Client MAC address, no separators
FrontMost	macOS absolute timestamp	Application launch time
LaunchTime	Number of seconds to 6 decimal places	Total duration the application was “frontmost” on screen
RunLength	Number of seconds to 6 decimal places	Total duration the application was running
ItemSeq	Integer	ItemSeq number for the respective ComputerID, referenced in the ApplicationName table
LastUpdated	macOS absolute timestamp	Last report time of the client
UserName	String	User who launched the application
RunState	Integer	“1” for “running”, or “0” for “terminated” at the time of the last report

Table 6: ApplicationUsage table columns

PropertyNameMap

This table is used as a reference for the SystemInformation table.

Column	Value Format	Description
ObjectName	String	Various elements of a macOS system, such as Mac_HardDriveElement, Mac_USBDeviceElement, Mac_SystemInfoElement
PropertyName	String	Property names for each element, such as ProductName, ProductID, VendorID, VendorName for Mac_USBDeviceElement
PropertyMapID	Integer	ID number for each property, per element

Table 7: PropertyNameMap table columns

SystemInformation

There is a substantial amount of system information collected in this table. This table can be leveraged to extract USB device information, IP addresses, hostnames, and more, of all the reported client systems.

Column	Value Format	Description
ComputerID	String	Client MAC address, with colon separators
ObjectName	String	Elements of a macOS system outlined in the PropertyNameMap table
PropertyName	String	Properties per element outlined in the PropertyNameMap table
ItemSeq	Integer	ID number for each element, i.e. if there are 4 Mac_USBDeviceElement data sets, each one will have an ItemSeq number, 0-3, to group the properties together
Value	String	Data for the respective property
LastUpdated	yyyy-mm-ddThh:mm:ssZ	24 hour local time, last report time of the client. Example: 2019-08-07T02:11:34Z

Table 8: SystemInformation table columns

UserUsage

This table contains the user login activity for all the reported client systems.

Column	Description of Value
ComputerID	Client MAC address, no separators
LastUpdated	macOS absolute timestamp, last report time of the client
UserName	Short name of the user
LoginType	Console, tty, or ssh
inTime	macOS absolute timestamp, time the user logged in
outTime	macOS absolute timestamp, time the user logged out
Host	Originating host for remote login. This field has been observed to not be consistently present.

Table 9: UserUsage table columns

Filesystem Cache

The RemoteManagement/ClientCaches/filesystem.cache file is a database that indexes the files and directories found on a macOS computer's file system. Rather than using SQLite like the RMDB, ARD uses a custom database implementation to track this information. Fortunately, the database file format is fairly simple, consisting of a file header, six tables, and entries that point to string values. By interpreting the information in the filesystem cache file, an investigator can recreate the directory structure of an ARD-enable system. Mandiant uses this technique to identify and demonstrate the existence of attacker-created files.

The database header, identified by the magic value "hdix", contains metadata about the database, such as the total number of indexed folders, files, and symlinks. Pointers from this header lead to the six tables: "main", "names" (file names), "kinds" (file extensions), "versions" (macOS app bundle version infos), "users", and "groups". Entries in the "main" table contain references to entries in the other tables; by walking these references, an investigator can recover full file system paths and metadata.

In practice, the filesystem.cache file may be tens of megabytes in size, tracking dozens or hundreds of thousands of file system entries. Figure 4 shows truncated content of a parsed file system cache file; these entries are for the artifacts discussed in this article!

```
- dir:  path: /var/db/emondClients/
                ext:                  user: 0.root          group: 0.wheel
- dir:  path: /var/db/RemoteManagement/
                ext:                  user: 0.root          group: 0.wheel
- dir:  path: /var/db/RemoteManagement/ClientCaches/
                ext:                  user: 0.root          group: 0.wheel
- dir:  path: /var/db/RemoteManagement/ClientCaches/SoftwareInfo/
                ext:                  user: 0.root          group: 0.wheel
- dir:  path: /var/db/RemoteManagement/ClientCaches/ApplicationUsage/
                ext:                  user: 0.root          group: 0.wheel
- dir:  path: /var/db/RemoteManagement/ClientCaches/UserAccounting/
                ext:                  user: 0.root          group: 0.wheel
- dir:  path: /var/db/RemoteManagement/ClientCaches/SystemInfo/
                ext:                  user: 0.root          group: 0.wheel
- file: path: /var/db/RemoteManagement/ClientCaches/cacheAccess.plist
                ext: .plist           user: 0.root          group: 0.wheel
- dir:  path: /var/db/RemoteManagement/RMDB/
                ext:                  user: 0.root          group: 0.wheel
- file: path: /var/db/RemoteManagement/RMDB/rmdb.sqlite3
                ext: .sqlite3         user: 0.root          group: 0.wheel
- dir:  path: /var/db/RemoteManagement/caches/
                ext:                  user: 0.root          group: 0.wheel
- file: path: /var/db/RemoteManagement/caches/AppUsage.plist
                ext: .plist           user: 0.root          group: 0.wheel
- file: path: /var/db/RemoteManagement/caches/sysinfo.cache
                ext: .cache           user: 0.root          group: 0.wheel
- file: path: /var/db/RemoteManagement/caches/AppUsage.tmp
                ext: .tmp             user: 0.root          group: 0.wheel
- file: path: /var/db/RemoteManagement/caches/UserAcct.tmp
                ext: .tmp             user: 0.root          group: 0.wheel
- file: path: /var/db/RemoteManagement/caches/filesystem.cache
                ext: .cache           user: 0.root          group: 0.wheel
- file: path: /var/db/RemoteManagement/caches/asp.cache
                ext: .cache           user: 0.root          group: 0.wheel
- dir:  path: /var/db/GPURestartReporter/
                ext:                  user: 0.root          group: 0.wheel
```

Figure 4: Screenshot of filesystem.cache contents, listing ARD artifacts

On a macOS system, the program “build_hd_index” traverses the file system and indexes the files and directories into filesystem.cache. Figure 5 shows a portion of the documentation for this tool; as expected, the default output directory is [/private]/var/db/RemoteManagement/caches/.

```
usage:
    build_hd_index         # Build searchable index of local hard drives
        --topdir /
        --exclude Volumes --exclude Developer
        --out /foo.txt
        --bundlecontents
        --symlinks
        --maxdepth
        --progress
        --verbose
        --debug 1
        --nice
        --help

        All arguments may be replaced with single char -t, -e, -b, -s, etc.

        With no arguments, indexes all fixed hard drives, starting at
        /, does not follow symlinks, excludes bundles, goes to
        arbitrary path depth, and saves the output file in:
        /var/db/RemoteManagement/caches/filesystem.cache
        --topdir            # where to begin traversing (default is /)
        --exclude foo       # Paths to be excluded (relative to topdir)
        --exclude bar       # Specify as many as you like.
        --bundlecontents    # Whether to go inside bundles (default is no!)
        --symlinks          # Treat symlinks as files and descend into symlinked directories (default is no!)
        #                  # Note: takes care to avoid loops introduced by following symlinks.

        --maxdepth 1        # How many levels to descend from / or from --topdir if given
        --out /foo.txt      # Alternate file in which to save the index.
        # Verbosity options
        --progress          # List paths, but no more than about once every 2 seconds.
        --verbose           # List every file path that is added to index.
        --debug 1           # debug level: (default 0 => off), 1 ,2, 3, or 4
        --nice 10           # Set process priority. Uses nice values (-20 highest priority, 20 lowest).
        --help              # Show more detailed help
```

Figure 5: Documentation for build_hd_index

Ironically, internet message board posts going back to at least 2007 complain of the performance impact of this tool. A post by “Anonymous” indicates that “build_hd_index” was designed to support file indexing on OS X Panther (2003), which didn’t have Spotlight. Now, 16 years later, we can exploit these artifacts during an incident response.

Introducing: ARDvark

It was evident that if this artifact exists in a future investigation, leveraging its wealth of data will be critical to identifying attacker activities. In some scenarios, investigators may be able to generate reports directly from an ARD administrator system, but this may not always be the case. If not, then investigators would have to rely on manually acquiring and extracting information from the RMDB file on the ARD administrator system. ARDvark is a tool that extracts all user activity and application usage recorded in the RMDB and outputs the data in an analyst-friendly format.

ARDvark will also process the AppUsage.plist and UserAcct.tmp files found on ARD client systems under /private/var/db/RemoteManagement/caches/. Additionally, ARDvark has the capability to parse the filesystem.cache files to produce a file system listing, as well as all users and groups present on the respective system. Please see the FireEye Github for more information.

Detecting and Preventing ARD Abuse

To detect suspicious ARD usage, organizations can monitor for anomalous modification of the /Library/Application Support/Apple/Remote Desktop/RemoteManagement.launchd file to identify remote desktop access enablement where ARD is not used. Analyzing the Unified Logs for evidence of unexpected kickstart commands during threat hunting missions can uncover suspicious ARD usage as well.

Mitigating ARD abuse is reliant upon the principle of least privilege. Mandiant recommends allowing as few remote control privileges as possible, and only allowing administrator privileges to necessary accounts. Apple provides guidance on setting privileges, and authenticating without using local accounts with ARD in the help page and in the ARD user guide. ARD administrators can then routinely generate reports in the ARD application to ensure no changes are made to administration privilege settings.

A Bushel of Evidence

Application usage artifacts for macOS are few and far between. To date, some of the best artifacts for application usage include CoreAnalytics files and the Spotlight database, but none of these artifacts provide the exact time of execution of all applications. While ARD artifacts are not present across every macOS system, if ARD is deployed in an enterprise environment it may provide some of the most valuable data for investigators which you would not uncover otherwise.

User login activity typically exists in the Apple System Logs and Audit Logs, but short log retention is frequently an issue when the average attacker dwell time in 2018 was 78 days. The RMDB provides a potential source of application usage and user login information that is over a year old, long outliving typical log retention times.

The system information available in the RMDB includes IP addresses, USB device information, and more which may be useful to investigators. Also, the file system cache files that are collected contain an extensive file listing of multiple macOS systems, which allows investigators to identify files or users of interest on other systems without having to collect data from the suspect system directly.

ARD is an excellent example of how remote administration tools provide an attack surface for abuse while simultaneously providing a vast amount of data to help piece together malicious activity, all from a single system. If your organization utilizes ARD, consider reviewing the information available through the reporting functionality during threat hunting and future investigative purposes, as the artifact doesn't fall far from the tree.

Posted in

- Threat Intelligence
- Security & Identity
