Script Best Practices

Description

Tips and tricks for running Filesets with scripts

Don't put passwords in scripts

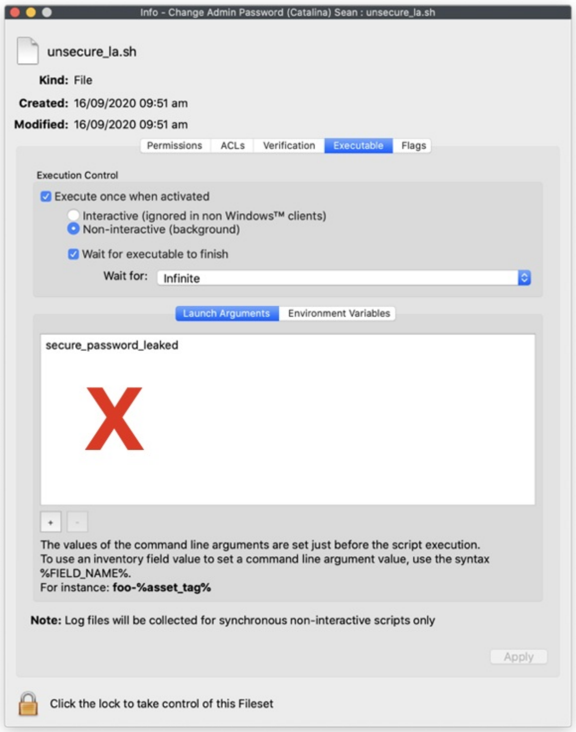
The scripts are stored locally on devices. For security reasons, usernames and passwords should not be included within the body of scripts.

For example:

Example: password in command

```
somecommand -u "USERNAME_HERE" -p "PASSSWORD_HERE"
```

Additionally, DO NOT use Launch Arguments to provide passwords to scripts. Launch Arguments are visible in the process list during script execution. Instead supply the username and password as Environment Variables:



unsecure_la.sh

Kind: File

Created: 16/09/2020 09:51 am

Modified: 16/09/2020 09:51 am

Execution Control

[✓] Execute once when activated

Interactive (ignored in non Windows™ clients)

[✓ Non-interactive (background)]

[✓] Wait for executable to finish

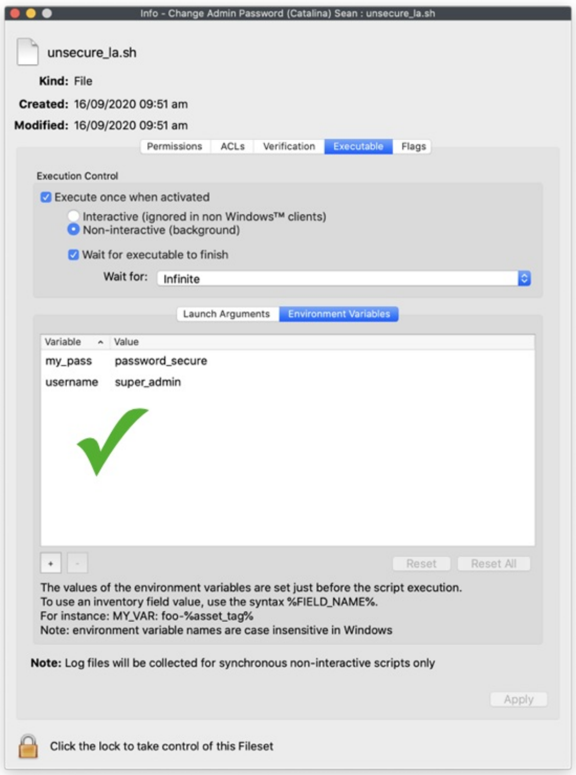
Wait for: Infinite

secure_password_leaked

The values of the command line arguments are set just before the script execution.
To use an inventory field value to set a command line argument value, use the syntax %FIELD_NAME%.
For instance: **foo-%asset_tag%**

Note: Log files will be collected for synchronous non-interactive scripts only

Click the lock to take control of this Fileset


Info - Change Admin Password (Catalina) Sean : unsecure_la.sh

unsecure_la.sh

Kind: File
Created: 16/09/2020 09:51 am
Modified: 16/09/2020 09:51 am

Execution Control

- [x] Execute once when activated
  - ○ Interactive (ignored in non Windows™ clients)
  - ● Non-interactive (background)
  - [x] Wait for executable to finish
  - Wait for: Infinite

Variable	Value
my_pass	password_secure
username	super_admin

The values of the environment variables are set just before the script execution.
To use an inventory field value, use the syntax %FIELD_NAME%.
For instance: MY_VAR: foo-%asset_tag%
Note: environment variable names are case insensitive in Windows

Note: Log files will be collected for synchronous non-interactive scripts only

Click the lock to take control of this Fileset

During script execution, the Launch Argument is seen:

Example: Visible Password

```
$ ps -ef | grep secure
    0 73010   155   0  9:51am ??         0:00.01 /bin/zsh /var/scripts/532417/unsecure_la.sh
secure_password_leaked
```

Using the example Environment Variables from the image, they would be addressed as:

OS	Script Type	Command
macOS	shell	`somecommand -u $username -p $my_pass`
Windows	Powershell	`somecommand -u $Env:username -p $Env:my_pass`
	Batch	`somecommand -u %username% -p %my_pass%`
	Batch	In order to not transmit the password to a log file accessible on the device, add @echo off before the line containing %my_pass% and @echo on as the next line. Example: @echo off %SystemRoot%\System32\Reg.exe ADD "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon" /v "DefaultPassword" /d "%my_pass%" /t REG_SZ /f @echo on
macOS & Windows	Python	`import os` `os.getenv('username')` `os.getenv('my_pass')`

Keep Requirements Scripts Small

Requirements scripts are pulled from a fileset and sent before the remainder of the fileset.

It behaves this way because if a requirements script fails, there is no point in downloading and installing the remainder of the fileset.

A fileset whose requirements have failed will not even show up in the kiosk.

Where possible, avoid piping commands. This increases overhead on the scripts. If pipes are required, try to reduce the quantity of pipes. If nothing else, this makes the scripts easier to read.

```
$ time system_profiler SPHardwareDataType | grep "Model Identifier" | awk '{print $NF}'
MacBookPro11,4


$ system_profiler SPHardwareDataType | awk '/Model Identifier/ {print $NF}'
MacBookPro11,4
```

And other commands may achieve the same result more efficiently without the need to pipe.

```
$ time system_profiler SPHardwareDataType | grep "Model Identifier" | awk '{print $NF}'
MacBookPro11,4

real    0m0.192s
user    0m0.071s
sys     0m0.049s


$ time sysctl -n hw.model
MacBookPro11,4

real    0m0.004s
user    0m0.001s
sys     0m0.002s
```

Consider this for all scripts beyond just requirement scripts.

Log Script Output

By default, Fileset scripts built through the Scripts button are logged. All output is redirected to a unique file per script.

If desired, additional information could be redirected to an alternate file.

Redirecting output to the FileWave Client log, allows the viewing of those details via the 'Get Log' feature

Redirecting to the FileWave Client log is not possible on windows, since the log file is locked by the client writing to the file.

Redirecting Output

Output may be redirected using one of the following:

macOS:

```
echo "hello" >> /tmp/tmp_log_file.log
```

Windows:

```
echo "hello" | Out-File -Append -Encoding Ascii C:\Temp\my_temp_file.log
```

Better than just redirecting output, consider using the tee/Tee-Object command, such that the FileWave generated log and the redirected log both show the output.

macOS:

```
echo "hello" | tee -a /tmp/tmp_log_file.log
```

Windows:

```
echo "hello" | Tee-Object -Variable out | Out-File -InputObject $out -append -encoding Ascii
C:\Temp\my_temp_file.log
```

On macOS, all output can be redirected by using the following at the beginning of the file:

```
#!/bin/zsh
exec 1>>/var/log/fwcld.log
exec 2>>/var/log/fwcld.log


... rest of script
```

Testing Scripts

Scripts run by FileWave are run by root or System. As such, scripts should be tested using the same user context to prevent erroneous results. Many commands will yield the same result regardless, but this cannot be relied upon.

Windows

E.g. Running the following command will provide a different output, when ran on a 32bit Windows environment as opposed to a 64bit Windows environment:

```
 (Get-ItemProperty "HKLM:\SOFTWARE\Microsoft\Windows NT\CurrentVersion").EditionId
```

64bit:

```
 (Get-ItemProperty "HKLM:\SOFTWARE\Microsoft\Windows NT\CurrentVersion").EditionId
Professional
```

32bit:

```
 (Get-ItemProperty "HKLM:\SOFTWARE\Microsoft\Windows NT\CurrentVersion").EditionId
Enterprise
```

Similarly, the user executing a script can have an impact on the outcome. Username itself is a relatively obvious example:

```
$Env:UserName
```

When ran locally through a shell, it should report the name of the current user. However, when ran through FileWave, it should report the System name.

Prior to FileWave 15.5, the Windows client was 32bit, but since then, a 64bit client is supplied. Either way, the user executing any scripts is the System user. As such, all tests should be ran in that same context:

Version	User	Bit
FileWave 15.5+	System	64
FileWave 15.4.x-	System	32

Fileset Properties has the option to specify 32 or 64bit, setting the executing environment.

Although FileWave 15.5.0 now provides 64bit options by default, current scripts will automatically be set as 32bit

Custom Fields and Blocker Scripts in 15.5.0 are still currently 32bit only. This should be addressed in an upcoming release.

PsTools: This relies on downloading and installing, onto the test machine, PsTools.

Running Environment

Prior to FileWave 15.5

Take a look at Getting a CMD prompt as SYSTEM in Windows Vista and Windows Server 2008 for details about running scripts as System. Note, that by default, this will start an executable as 64-bit, for native 64-bit OS.

From a device with the PsTools installed, start by opening a Command Shell as an Administrator. From that shell, another command should be run to open yet another shell, but this time in the chosen environment.

The below example shows launching the 32bit version of PowerShell as the System user:

```
PSEXEC  -i -s -d C:\Windows\SysWOW64\windowsPowerShell\v1.0\powershell.exe
```

To open a command shell in that same environment would use the following:

```
PSEXEC -i -s -d %windir%\SysWoW64\cmd.exe
```

Similarly, when attempting to run some commands, it may be necessary to ensure Windows is using the correct version of a binary with the 'sysnative' redirect. An example would be Bitlocker's 'manage-bde.exe'. To use this in a Fileset, try the following:

```
C:\Windows\sysnative\manage-bde.exe -status
```

If you have a requirement to run a particular command through the 64-bit version of Powershell this can be achieved as follows:

```
If ( [IntPtr]::Size * 8 -ne 64 )
{
    C:\Windows\SysNative\WindowsPowerShell\v1.0\PowerShell.exe -File $MyInvocation.MyCommand.Path
}
Else
{
    # Add code here
}
```

Testing scripts designed to be ran with the 64bit FileWave Client, still requires the above actions, to ensure that the System account has been targeted, but this time with a 64bit application.

Opening a 64bit version of PowerShell as the System user:

```
PSEXEC  -i -s -d C:\Windows\System32\windowsPowerShell\v1.0\powershell.exe
```

Example 32bit to 64bit

The below demonstrates running a 64bit script, but from the 32bit FileWave Client, which will create a new administrator. Additionally, the FileSet > Get Info > Environment Variables are being used to supply the name and password to the script.

Two Fileset Environment Variables are being supplied, for the user 'rstephens' with a password of 'filewave'

Variable	Value
username	rstephens
password	filewave

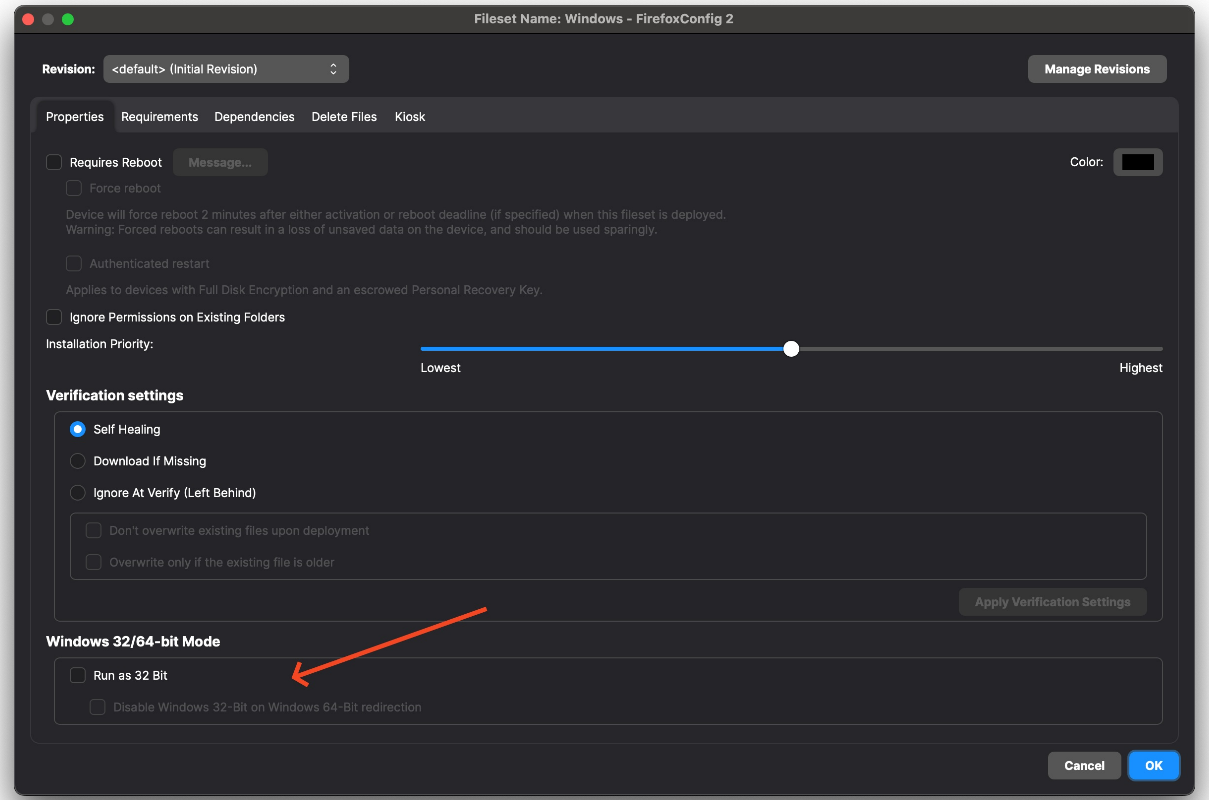
Those Parameters may then be referenced from within the script and have them defined for the launching of the 64bit executable within this 32bit script.

```
Param (
    [string]$MyUsername = $Env:username,
    [string]$MyPassword = $Env:password
)


If ( [IntPtr]::Size * 8 -ne 64 )
{
    C:\Windows\SysNative\WindowsPowerShell\v1.0\PowerShell.exe -File $MyInvocation.MyCommand.Path -MyUsername
$MyUsername -MyPassword $MyPassword
}
Else
{
    (New-LocalUser -AccountNeverExpires:$true -Password ( ConvertTo-SecureString -AsPlainText -Force $MyPassword)
 -Name $MyUsername | Add-LocalGroupMember -Group administrators)
}
```

Troubleshooting PowerShell scripts

As a best practice, always check the "Disable Windows 32-bit on Windows 64-bit redirection" checkbox in the Properties tab for your fileset priot to FileWave 15.5.0. From 15.5.0 and beyond make sure to uncheck the new "Run as 32 Bit" checkbox so that the Fileset will be truely 64-bit. This ensures that any scripts in the fileset will be run in a 64-bit session and built-in Windows executables triggered by those scripts will call the 64-bit versions. A common reason for why your script might not be performing the expected results could be due to 64-bit Only Modules or Cmdlets: Some PowerShell modules or cmdlets are available only for the 64-bit version of PowerShell. If a script relies on these 64-bit modules, it must run in a 64-bit shell.



Fileset Name: Windows - FirefoxConfig 2

Revision: <default> (Initial Revision)

Requires Reboot: ☐
Force reboot: ☐

Device will force reboot 2 minutes after either activation or reboot deadline (if specified) when this fileset is deployed.
Warning: Forced reboots can result in a loss of unsaved data on the device, and should be used sparingly.

Authenticated restart: ☐

Applies to devices with Full Disk Encryption and an escrowed Personal Recovery Key.

Ignore Permissions on Existing Folders: ☐

Installation Priority: Lowest — Highest

Verification settings: Self Healing ●, Download If Missing ○, Ignore At Verify (Left Behind) ○

Don't overwrite existing files upon deployment: ☐
Overwrite only if the existing file is older: ☐

Windows 32/64-bit Mode

Run as 32 Bit: ☐

Disable Windows 32-Bit on Windows 64-Bit redirection: ☐

macOS

On macOS, running commands as sudo is not necessarily the same as actually becoming root.

Root vs As Root

E.g. Run the following commands to evaluate the local variable $HOME, once using sudo and once as root.

```
$ whoami
auser
$ sudo echo $HOME
/Users/auser
$ sudo su -
$ whoami
root
$ echo $HOME
/var/root
```

Paths

Similarly, the paths used to locate executable files will differ, since FileWave is a service ran as root and is not the root account. On an example device:

User Account	Root Account	FileWave Client

`% echo $PATH | tr ":" "\n"`
`/opt/homebrew/bin`
`/opt/homebrew/sbin`
	`% echo $PATH | tr ":" "\n"`
	`/usr/local/bin`
	`/System/Cryptexes/App/usr/bin`
		`/usr/bin`
		`/bin`
		`/usr/sbin`

/var/root/.cask/bin /usr/local/sbin /usr/bin /bin /usr/sbin /sbin	/usr/bin /bin /usr/sbin /sbin /Applications/VMware Fusion.app/Contents/Public /Library/Apple/usr/bin /var/run/com.apple.security.cryptexd/codex.system/bootstrap/usr/local/bin /var/run/com.apple.security.cryptexd/codex.system/bootstrap/usr/bin /var/run/com.apple.security.cryptexd/codex.system/bootstrap/usr/appleinternal/bin	/sbin

As such, consider always using the full path within a script to an executable, to be explicit, and ensure the executable is found.

For example, it can be seen from the above that homebrew is installed.

```
 % ls -al /usr/local/bin/brew
lrwxrwxrwx  1 root  _developer  28 Mar 23  2023 /usr/local/bin/brew -> /usr/local/homebrew/bin/brew
```

Running the following command would work as the user or root account, but would fail through FileWave, since the FileWave Client does not search /usr/local at all for executables:

`brew -v`

To ensure the script works and targets the correct brew, the full path should be entered:

`/usr/local/bin/brew -v`

Plist

It is common to see plist files edited with the 'defaults' command. However, this command is unique when it comes to ownership and permissions of files. The 'defaults' command will both take ownership and change permissions of files when used to write to plist files:

```
$ whoami
root
$ ls -al /tmp/example_plist.plist
-rw-r--r--  1 rstephens  staff  66 Feb 28 10:03 /tmp/example_plist.plist
$ defaults write /tmp/example_plist Label example_plist
$ ls -al /tmp/example_plist.plist
-rw-------  1 root  wheel  66 Feb 28 10:05 /tmp/example_plist.plist
```

As such, ensure to add a repair to scripts to reset permissions and ownership after the command has been used or consider using the following command instead (Note the full path is required if /usr/libexec is not in the paths list:

```
/usr/libexec/PlistBuddy
```

Related Content

- Custom Fields
- Filesets / Payloads
- Fileset / Payload Script Exit Code Status

Revision #13
Created 2 July 2023 14:43:06 by Josh Levitsky
Updated 4 November 2024 13:43:14 by Sean Holden