

# Recommendations

## 1 Initial Setup

Items in this section are advised for all systems but may be difficult or require extensive preparation after the initial setup of the system.

## 1.1 Filesystem

The file system is generally a built-in layer used to handle the data management of the storage.

### 1.1.1 Configure Filesystem Kernel Modules

Several uncommon filesystem types are supported under Linux. Removing support for unneeded filesystem types reduces the local attack surface of the system. If a filesystem type is not needed it should be disabled. Native Linux file systems are designed to ensure that built-in security controls function as expected. Non-native filesystems can lead to unexpected consequences to both the security and functionality of the system and should be used with caution. Many filesystems are created for niche use cases and are not maintained and supported as the operating systems are updated and patched. Users of non-native filesystems should ensure that there is attention and ongoing support for them, especially in light of frequent operating system changes.

Standard network connectivity and Internet access to cloud storage may make the use of non-standard filesystem formats to directly attach heterogeneous devices much less attractive.

**Note:** This should not be considered a comprehensive list of filesystems. You may wish to consider additions to those listed here for your environment. For the current available file system modules on the system see `/usr/lib/modules/$(uname -r)/kernel/fs`

#### Start up scripts

Kernel modules loaded directly via `insmod` will ignore what is configured in the relevant `/etc/modprobe.d/*.conf` files. If modules are still being loaded after a reboot whilst having the correctly configured `blacklist` and `install` command, check for `insmod` entries in start up scripts such as `.bashrc`.

You may also want to check `/lib/modprobe.d/`. Please note that this directory should not be used for user defined module loading. Ensure that all such entries resides in `/etc/modprobe.d/*.conf` files.

#### Return values

Using `/bin/false` as the command in disabling a particular module serves two purposes; to convey the meaning of the entry to the user and cause a non-zero return value. The latter can be tested for in scripts. Please note that `insmod` will ignore what is configured in the relevant `/etc/modprobe.d/*.conf` files. The preferred way to load modules is with `modprobe`.

### *1.1.1.1 Ensure cramfs kernel module is not available (Automated)*

#### **Profile Applicability:**

- Level 1 - Server
- Level 1 - Workstation

#### **Description:**

The **cramfs** filesystem type is a compressed read-only Linux filesystem embedded in small footprint systems. A **cramfs** image can be used without having to first decompress the image.

#### **Rationale:**

Removing support for unneeded filesystem types reduces the local attack surface of the system. If this filesystem type is not needed, disable it.

#### **Audit:**

Run the following script to verify:

- **IF** - the **cramfs** kernel module is available in ANY installed kernel, verify:

- An entry including **/bin/true** or **/bin/false** exists in a file within the **/etc/modprobe.d/** directory
- The module is deny listed in a file within the **/etc/modprobe.d/** directory
- The module is not loaded in the running kernel

- **IF** - the **cramfs** kernel module is not available on the system, or pre-compiled into the kernel, no additional configuration is necessary

```

#!/usr/bin/env bash

{
    a_output=() a_output2=() a_output3=() l_dl="" l_mod_name="cramfs"
    l_mod_type="fs"
    l_mod_path="$(readlink -f /lib/modules/**/kernel/$l_mod_type | sort -u)"
    f_module_chk()
    {
        l_dl="y" a_showconfig=()
        while IFS= read -r l_showconfig; do
            a_showconfig+=("$l_showconfig")
            done < <(modprobe --showconfig | grep -P --
'\b(install|blacklist)\h+'${l_mod_chk_name//-/}_'\b')
            if ! lsmmod | grep "$l_mod_chk_name" &> /dev/null; then
                a_output+=(" - kernel module: \"$l_mod_name\" is not loaded")
            else
                a_output2+=(" - kernel module: \"$l_mod_name\" is loaded")
            fi
            if grep -Pq -- '\bbinstall\h+'${l_mod_chk_name//-/}_
/_}'\h+(\usr)?\bin\/(true|false)\b' <<< "${a_showconfig[*]}"; then
                a_output+=(" - kernel module: \"$l_mod_name\" is not loadable")
            else
                a_output2+=(" - kernel module: \"$l_mod_name\" is loadable")
            fi
            if grep -Pq -- '\bbblacklist\h+'${l_mod_chk_name//-/}_'\b' <<<
"${a_showconfig[*]}"; then
                a_output+=(" - kernel module: \"$l_mod_name\" is deny listed")
            else
                a_output2+=(" - kernel module: \"$l_mod_name\" is not deny listed")
            fi
        }
        for l_mod_base_directory in $l_mod_path; do
            if [ -d "$l_mod_base_directory/${l_mod_name//-/}/" ] && [ -n "$(ls -A
"$l_mod_base_directory/${l_mod_name//-/}/")" ]; then
                a_output3+=(" - \"$l_mod_base_directory\"")
                l_mod_chk_name="$l_mod_name"
                [[ "$l_mod_name" =~ overlay ]] && l_mod_chk_name="$l_mod_name:-2"
                [ "$l_dl" != "y" ] && f_module_chk
            else
                a_output+=(" - kernel module: \"$l_mod_name\" doesn't exist in
\"$l_mod_base_directory\"")
            fi
        done
        [ "${#a_output3[@]}" -gt 0 ] && printf '%s\n' "" " -- INFO --" " - module:
\"$l_mod_name\" exists in:" "${a_output3[@]}"
        if [ "${#a_output2[@]}" -le 0 ]; then
            printf '%s\n' "" "- Audit Result:" " ** PASS **" "${a_output[@]}"
        else
            printf '%s\n' "" "- Audit Result:" " ** FAIL **" " - Reason(s) for
audit failure:" "${a_output2[@]}"
            [ "${#a_output[@]}" -gt 0 ] && printf '%s\n' "- Correctly set:"
"${a_output[@]}"
        fi
    }
}

```

## Remediation:

Run the following script to unload and disable the `cramfs` module:

- **IF** - the `cramfs` kernel module is available in ANY installed kernel:

- Create a file ending in `.conf` with `install cramfs /bin/false` in the `/etc/modprobe.d/` directory
- Create a file ending in `.conf` with `blacklist cramfs` in the `/etc/modprobe.d/` directory
- Run `modprobe -r cramfs 2>/dev/null; rmmod cramfs 2>/dev/null` to remove `cramfs` from the kernel

- **IF** - the `cramfs` kernel module is not available on the system, or pre-compiled into the kernel, no remediation is necessary





```
#!/usr/bin/env bash

{
    a_output2=() a_output3=() l_dl="" l_mod_name="cramfs" l_mod_type="fs"
    l_mod_path="$(readlink -f /lib/modules/**/kernel/$l_mod_type | sort -u)"
    f_module_fix()
    {
        l_dl="y" a_showconfig=()
        while IFS= read -r l_showconfig; do
            a_showconfig+=("$l_showconfig")
            done < <(modprobe --showconfig | grep -P --
'\b(install|blacklist)\h+'${l_mod_chk_name//-/}_'\b')
            if lsmod | grep "$l_mod_chk_name" &> /dev/null; then
                a_output2+=(" - unloading kernel module: \"$l_mod_name\"")
                modprobe -r "$l_mod_chk_name" 2>/dev/null; rmmod "$l_mod_name"
2>/dev/null
            fi
            if ! grep -Pq -- '\binstall\h+'${l_mod_chk_name//-/}_
/_'\b'\h+(\usr)?\bin\/(true|false)\b' <<< "${a_showconfig[*]}"; then
                a_output2+=(" - setting kernel module: \"$l_mod_name\" to
\"$(readlink -f /bin/false)\"")
                printf '%s\n' "install $l_mod_chk_name $(readlink -f /bin/false)" >>
/etc/modprobe.d/"$l_mod_name".conf
            fi
            if ! grep -Pq -- '\bblacklist\h+'${l_mod_chk_name//-/}_'\b' <<<
"${a_showconfig[*]}"; then
                a_output2+=(" - denylisting kernel module: \"$l_mod_name\"")
                printf '%s\n' "blacklist $l_mod_chk_name" >>
/etc/modprobe.d/"$l_mod_name".conf
            fi
        }
        for l_mod_base_directory in $l_mod_path; do # Check if the module exists
on the system
            if [ -d "$l_mod_base_directory/${l_mod_name//-/}/" ] && [ -n "$(ls -A
"$l_mod_base_directory/${l_mod_name//-/}/")" ]; then
                a_output3+=(" - \"$l_mod_base_directory\"")
                l_mod_chk_name="$l_mod_name"
                [[ "$l_mod_name" =~ overlay ]] && l_mod_chk_name="${l_mod_name::-2}"
                [ "$l_dl" != "y" ] && f_module_fix
            else
                printf '%s\n' " - kernel module: \"$l_mod_name\" doesn't exist in
\"$l_mod_base_directory\""
            fi
        done
        [ "${#a_output3[@]}" -gt 0 ] && printf '%s\n' "" " -- INFO --" " - module:
\"$l_mod_name\" exists in:" "${a_output3[@]}"
        [ "${#a_output2[@]}" -gt 0 ] && printf '%s\n' "" "${a_output2[@]}" ||
printf '%s\n' "" " - No changes needed"
        printf '%s\n' "" " - remediation of kernel module: \"$l_mod_name\"
complete" ""
    }
}
```

## References:

1. NIST SP 800-53 Rev. 5: CM-7
2. STIG Finding ID: V-230498

### CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>4.8 Uninstall or Disable Unnecessary Services on Enterprise Assets and Software</u> Uninstall or disable unnecessary services on enterprise assets and software, such as an unused file sharing service, web application module, or service function.			
v7	<u>9.2 Ensure Only Approved Ports, Protocols and Services Are Running</u> Ensure that only network ports, protocols, and services listening on a system with validated business needs, are running on each system.			

### MITRE ATT&CK Mappings:

Techniques / Sub-techniques	Tactics	Mitigations
T1005, T1005.000	TA0005	M1050



### *1.1.1.2 Ensure freevxfs kernel module is not available (Automated)*

#### **Profile Applicability:**

- Level 1 - Server
- Level 1 - Workstation

#### **Description:**

The **freevxfs** filesystem type is a free version of the Veritas type filesystem. This is the primary filesystem type for HP-UX operating systems.

#### **Rationale:**

Removing support for unneeded filesystem types reduces the local attack surface of the system. If this filesystem type is not needed, disable it.

#### **Audit:**

Run the following script to verify:

- **IF** - the **freevxfs** kernel module is available in ANY installed kernel, verify:

- An entry including **/bin/true** or **/bin/false** exists in a file within the **/etc/modprobe.d/** directory
- The module is deny listed in a file within the **/etc/modprobe.d/** directory
- The module is not loaded in the running kernel

- **IF** - the **freevxfs** kernel module is not available on the system, or pre-compiled into the kernel, no additional configuration is necessary

```

#!/usr/bin/env bash

{
    a_output=() a_output2=() a_output3=() l_dl="" l_mod_name="freevxfs"
    l_mod_type="fs"
    l_mod_path="$(readlink -f /lib/modules/**/kernel/$l_mod_type | sort -u)"
    f_module_chk()
    {
        l_dl="y" a_showconfig=()
        while IFS= read -r l_showconfig; do
            a_showconfig+=("$l_showconfig")
            done < <(modprobe --showconfig | grep -P --
'\b(install|blacklist)\h+'${l_mod_chk_name//-/}_'\b')
            if ! lsmod | grep "$l_mod_chk_name" &> /dev/null; then
                a_output+=(" - kernel module: \"$l_mod_name\" is not loaded")
            else
                a_output2+=(" - kernel module: \"$l_mod_name\" is loaded")
            fi
            if grep -Pq -- '\binstall\h+'${l_mod_chk_name//-/}_
/_}'\h+(\usr)?\bin\/(true|false)\b' <<< "${a_showconfig[*]}"; then
                a_output+=(" - kernel module: \"$l_mod_name\" is not loadable")
            else
                a_output2+=(" - kernel module: \"$l_mod_name\" is loadable")
            fi
            if grep -Pq -- '\bblacklist\h+'${l_mod_chk_name//-/}_'\b' <<<
"${a_showconfig[*]}"; then
                a_output+=(" - kernel module: \"$l_mod_name\" is deny listed")
            else
                a_output2+=(" - kernel module: \"$l_mod_name\" is not deny listed")
            fi
        }
        for l_mod_base_directory in $l_mod_path; do
            if [ -d "$l_mod_base_directory/${l_mod_name//-/}/" ] && [ -n "$(ls -A
"$l_mod_base_directory/${l_mod_name//-/}/")" ]; then
                a_output3+=(" - \"$l_mod_base_directory\"")
                l_mod_chk_name="$l_mod_name"
                [[ "$l_mod_name" =~ overlay ]] && l_mod_chk_name="$l_mod_name:-2"
                [ "$l_dl" != "y" ] && f_module_chk
            else
                a_output+=(" - kernel module: \"$l_mod_name\" doesn't exist in
\"$l_mod_base_directory\"")
            fi
        done
        [ "${#a_output3[@]}" -gt 0 ] && printf '%s\n' "" " -- INFO --" " - module:
\"$l_mod_name\" exists in:" "${a_output3[@]}"
        if [ "${#a_output2[@]}" -le 0 ]; then
            printf '%s\n' "" "- Audit Result:" " ** PASS **" "${a_output[@]}"
        else
            printf '%s\n' "" "- Audit Result:" " ** FAIL **" " - Reason(s) for
audit failure:" "${a_output2[@]}"
            [ "${#a_output[@]}" -gt 0 ] && printf '%s\n' "- Correctly set:"
"${a_output[@]}"
        fi
    }
}

```

## Remediation:

Run the following script to unload and disable the `freevxfs` module:

- **IF** - the `freevxfs` kernel module is available in ANY installed kernel:

- Create a file ending in `.conf` with `install freevxfs /bin/false` in the `/etc/modprobe.d/` directory
- Create a file ending in `.conf` with `blacklist freevxfs` in the `/etc/modprobe.d/` directory
- Run `modprobe -r freevxfs 2>/dev/null; rmmod freevxfs 2>/dev/null` to remove `freevxfs` from the kernel

- **IF** - the `freevxfs` kernel module is not available on the system, or pre-compiled into the kernel, no remediation is necessary





```
#!/usr/bin/env bash

{
    a_output2=() a_output3=() l_dl="" l_mod_name="freevxfs" l_mod_type="fs"
    l_mod_path="$(readlink -f /lib/modules/**/kernel/$l_mod_type | sort -u)"
    f_module_fix()
    {
        l_dl="y" a_showconfig=()
        while IFS= read -r l_showconfig; do
            a_showconfig+=("$l_showconfig")
            done < <(modprobe --showconfig | grep -P --
'\b(install|blacklist)\h+'${l_mod_chk_name//-/}_'\b')
            if lsmod | grep "$l_mod_chk_name" &> /dev/null; then
                a_output2+=(" - unloading kernel module: \"$l_mod_name\"")
                modprobe -r "$l_mod_chk_name" 2>/dev/null; rmmod "$l_mod_name"
2>/dev/null
            fi
            if ! grep -Pq -- '\binstall\h+'${l_mod_chk_name//-/}_
/_'\b'\h+(\usr)?\bin\/(true|false)\b' <<< "${a_showconfig[*]}"; then
                a_output2+=(" - setting kernel module: \"$l_mod_name\" to
\"$(readlink -f /bin/false)\"")
                printf '%s\n' "install $l_mod_chk_name $(readlink -f /bin/false)" >>
/etc/modprobe.d/"$l_mod_name".conf
            fi
            if ! grep -Pq -- '\bblacklist\h+'${l_mod_chk_name//-/}_'\b' <<<
"${a_showconfig[*]}"; then
                a_output2+=(" - denylisting kernel module: \"$l_mod_name\"")
                printf '%s\n' "blacklist $l_mod_chk_name" >>
/etc/modprobe.d/"$l_mod_name".conf
            fi
        }
        for l_mod_base_directory in $l_mod_path; do # Check if the module exists
on the system
            if [ -d "$l_mod_base_directory/${l_mod_name//-/}/" ] && [ -n "$(ls -A
"$l_mod_base_directory/${l_mod_name//-/}/")" ]; then
                a_output3+=(" - \"$l_mod_base_directory\"")
                l_mod_chk_name="$l_mod_name"
                [[ "$l_mod_name" =~ overlay ]] && l_mod_chk_name="$l_mod_name:-2"
                [ "$l_dl" != "y" ] && f_module_fix
            else
                printf '%s\n' " - kernel module: \"$l_mod_name\" doesn't exist in
\"$l_mod_base_directory\""
            fi
        done
        [ "${#a_output3[@]}" -gt 0 ] && printf '%s\n' "" " -- INFO --" " - module:
\"$l_mod_name\" exists in:" "${a_output3[@]}"
        [ "${#a_output2[@]}" -gt 0 ] && printf '%s\n' "" "${a_output2[@]}" ||
printf '%s\n' "" " - No changes needed"
        printf '%s\n' "" " - remediation of kernel module: \"$l_mod_name\"
complete" ""
    }
}
```

## References:

1. NIST SP 800-53 Rev. 5: CM-7

**CIS Controls:**

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>4.8 Uninstall or Disable Unnecessary Services on Enterprise Assets and Software</u> Uninstall or disable unnecessary services on enterprise assets and software, such as an unused file sharing service, web application module, or service function.			
v7	<u>9.2 Ensure Only Approved Ports, Protocols and Services Are Running</u> Ensure that only network ports, protocols, and services listening on a system with validated business needs, are running on each system.			

**MITRE ATT&CK Mappings:**

Techniques / Sub-techniques	Tactics	Mitigations
T1005, T1005.000	TA0005	M1050

### *1.1.1.3 Ensure hfs kernel module is not available (Automated)*

#### **Profile Applicability:**

- Level 1 - Server
- Level 1 - Workstation

#### **Description:**

The **hfs** filesystem type is a hierarchical filesystem that allows you to mount Mac OS filesystems.

#### **Rationale:**

Removing support for unneeded filesystem types reduces the local attack surface of the system. If this filesystem type is not needed, disable it.

#### **Audit:**

Run the following script to verify:

- **IF** - the **hfs** kernel module is available in ANY installed kernel, verify:

- An entry including **/bin/true** or **/bin/false** exists in a file within the **/etc/modprobe.d/** directory
- The module is deny listed in a file within the **/etc/modprobe.d/** directory
- The module is not loaded in the running kernel

- **IF** - the **hfs** kernel module is not available on the system, or pre-compiled into the kernel, no additional configuration is necessary

```

#!/usr/bin/env bash

{
    a_output=() a_output2=() a_output3=() l_dl="" l_mod_name="hfs"
    l_mod_type="fs"
    l_mod_path="$(readlink -f /lib/modules/**/kernel/$l_mod_type | sort -u)"
    f_module_chk()
    {
        l_dl="y" a_showconfig=()
        while IFS= read -r l_showconfig; do
            a_showconfig+=("$l_showconfig")
            done < <(modprobe --showconfig | grep -P --
'\b(install|blacklist)\h+'${l_mod_chk_name//-/}_'\b')
            if ! lsmod | grep "$l_mod_chk_name" &> /dev/null; then
                a_output+=(" - kernel module: \"$l_mod_name\" is not loaded")
            else
                a_output2+=(" - kernel module: \"$l_mod_name\" is loaded")
            fi
            if grep -Pq -- '\bbinstall\h+'${l_mod_chk_name//-/}_
/_}'\h+(\usr)?\bin\/(true|false)\b' <<< "${a_showconfig[*]}"; then
                a_output+=(" - kernel module: \"$l_mod_name\" is not loadable")
            else
                a_output2+=(" - kernel module: \"$l_mod_name\" is loadable")
            fi
            if grep -Pq -- '\bbblacklist\h+'${l_mod_chk_name//-/}_'\b' <<<
"${a_showconfig[*]}"; then
                a_output+=(" - kernel module: \"$l_mod_name\" is deny listed")
            else
                a_output2+=(" - kernel module: \"$l_mod_name\" is not deny listed")
            fi
        }
        for l_mod_base_directory in $l_mod_path; do
            if [ -d "$l_mod_base_directory/${l_mod_name//-/}/" ] && [ -n "$(ls -A
"$l_mod_base_directory/${l_mod_name//-/}/")" ]; then
                a_output3+=(" - \"$l_mod_base_directory\"")
                l_mod_chk_name="$l_mod_name"
                [[ "$l_mod_name" =~ overlay ]] && l_mod_chk_name="$l_mod_name:-2)"
                [ "$l_dl" != "y" ] && f_module_chk
            else
                a_output+=(" - kernel module: \"$l_mod_name\" doesn't exist in
\"$l_mod_base_directory\"")
            fi
        done
        [ "${#a_output3[@]}" -gt 0 ] && printf '%s\n' "" " -- INFO --" " - module:
\"$l_mod_name\" exists in:" "${a_output3[@]}"
        if [ "${#a_output2[@]}" -le 0 ]; then
            printf '%s\n' "" "- Audit Result:" " ** PASS **" "${a_output[@]}"
        else
            printf '%s\n' "" "- Audit Result:" " ** FAIL **" " - Reason(s) for
audit failure:" "${a_output2[@]}"
            [ "${#a_output[@]}" -gt 0 ] && printf '%s\n' "- Correctly set:"
"${a_output[@]}"
        fi
    }
}

```

## Remediation:

Run the following script to unload and disable the `hfs` module:

- **IF** - the `hfs` kernel module is available in ANY installed kernel:

- Create a file ending in `.conf` with `install hfs /bin/false` in the `/etc/modprobe.d/` directory
- Create a file ending in `.conf` with `blacklist hfs` in the `/etc/modprobe.d/` directory
- Run `modprobe -r hfs 2>/dev/null; rmmod hfs 2>/dev/null` to remove `hfs` from the kernel

- **IF** - the `hfs` kernel module is not available on the system, or pre-compiled into the kernel, no remediation is necessary







```
#!/usr/bin/env bash

{
  a_output2=() a_output3=() l_dl="" l_mod_name="hfs" l_mod_type="fs"
  l_mod_path="$(readlink -f /lib/modules/**/kernel/$l_mod_type | sort -u)"
  f_module_fix()
  {
    l_dl="y" a_showconfig=()
    while IFS= read -r l_showconfig; do
      a_showconfig+=("$l_showconfig")
      done < <(modprobe --showconfig | grep -P --
'\b(install|blacklist)\h+' "${l_mod_chk_name//-/}_"' \b')
      if lsmod | grep "$l_mod_chk_name" &> /dev/null; then
        a_output2+=(" - unloading kernel module: \"$l_mod_name\"")
        modprobe -r "$l_mod_chk_name" 2>/dev/null; rmmod "$l_mod_name"
2>/dev/null
      fi
      if ! grep -Pq -- '\binstall\h+' "${l_mod_chk_name//-/}_"' \b' <<<
"${a_showconfig[*]}"; then
        a_output2+=(" - setting kernel module: \"$l_mod_name\" to
\"$(readlink -f /bin/false)\"")
        printf '%s\n' "install $l_mod_chk_name $(readlink -f /bin/false)" >>
/etc/modprobe.d/"$l_mod_name".conf
      fi
      if ! grep -Pq -- '\bblacklist\h+' "${l_mod_chk_name//-/}_"' \b' <<<
"${a_showconfig[*]}"; then
        a_output2+=(" - denylisting kernel module: \"$l_mod_name\"")
        printf '%s\n' "blacklist $l_mod_chk_name" >>
/etc/modprobe.d/"$l_mod_name".conf
      fi
    }
    for l_mod_base_directory in $l_mod_path; do # Check if the module exists
on the system
      if [ -d "$l_mod_base_directory/${l_mod_name//-/}/" ] && [ -n "$(ls -A
"$l_mod_base_directory/${l_mod_name//-/}/")" ]; then
        a_output3+=(" - \"$l_mod_base_directory\"")
        l_mod_chk_name="$l_mod_name"
        [[ "$l_mod_name" =~ overlay ]] && l_mod_chk_name="${l_mod_name::-2}"
        [ "$l_dl" != "y" ] && f_module_fix
      else
        printf '%s\n' " - kernel module: \"$l_mod_name\" doesn't exist in
\"$l_mod_base_directory\""
      fi
    done
    [ "${#a_output3[@]}" -gt 0 ] && printf '%s\n' "" " -- INFO --" " - module:
\"$l_mod_name\" exists in:" "${a_output3[@]}"
    [ "${#a_output2[@]}" -gt 0 ] && printf '%s\n' "" "${a_output2[@]}" ||
printf '%s\n' "" " - No changes needed"
    printf '%s\n' "" " - remediation of kernel module: \"$l_mod_name\"
complete" ""
  }
}
```

## References:

1. NIST SP 800-53 Rev. 5: CM-7

**CIS Controls:**

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>4.8 Uninstall or Disable Unnecessary Services on Enterprise Assets and Software</u> Uninstall or disable unnecessary services on enterprise assets and software, such as an unused file sharing service, web application module, or service function.			
v7	<u>9.2 Ensure Only Approved Ports, Protocols and Services Are Running</u> Ensure that only network ports, protocols, and services listening on a system with validated business needs, are running on each system.			

**MITRE ATT&CK Mappings:**

Techniques / Sub-techniques	Tactics	Mitigations
T1005, T1005.000	TA0005	M1050

### *1.1.1.4 Ensure hfsplus kernel module is not available (Automated)*

#### **Profile Applicability:**

- Level 1 - Server
- Level 1 - Workstation

#### **Description:**

The **hfsplus** filesystem type is a hierarchical filesystem designed to replace **hfs** that allows you to mount Mac OS filesystems.

#### **Rationale:**

Removing support for unneeded filesystem types reduces the local attack surface of the system. If this filesystem type is not needed, disable it.

#### **Audit:**

Run the following script to verify:

- **IF** - the **hfsplus** kernel module is available in ANY installed kernel, verify:

- An entry including **/bin/true** or **/bin/false** exists in a file within the **/etc/modprobe.d/** directory
- The module is deny listed in a file within the **/etc/modprobe.d/** directory
- The module is not loaded in the running kernel

- **IF** - the **hfsplus** kernel module is not available on the system, or pre-compiled into the kernel, no additional configuration is necessary

```

#!/usr/bin/env bash

{
    a_output=() a_output2=() a_output3=() l_dl="" l_mod_name="hfsplus"
    l_mod_type="fs"
    l_mod_path="$(readlink -f /lib/modules/**/kernel/$l_mod_type | sort -u)"
    f_module_chk()
    {
        l_dl="y" a_showconfig=()
        while IFS= read -r l_showconfig; do
            a_showconfig+=("$l_showconfig")
            done < <(modprobe --showconfig | grep -P --
'\b(install|blacklist)\h+'${l_mod_chk_name//-/}_'\b')
            if ! lsmod | grep "$l_mod_chk_name" &> /dev/null; then
                a_output+=(" - kernel module: \"$l_mod_name\" is not loaded")
            else
                a_output2+=(" - kernel module: \"$l_mod_name\" is loaded")
            fi
            if grep -Pq -- '\bbinstall\h+'${l_mod_chk_name//-/}_'\b' <<<
"${a_showconfig[*]}"; then
                a_output+=(" - kernel module: \"$l_mod_name\" is not loadable")
            else
                a_output2+=(" - kernel module: \"$l_mod_name\" is loadable")
            fi
            if grep -Pq -- '\bbblacklist\h+'${l_mod_chk_name//-/}_'\b' <<<
"${a_showconfig[*]}"; then
                a_output+=(" - kernel module: \"$l_mod_name\" is deny listed")
            else
                a_output2+=(" - kernel module: \"$l_mod_name\" is not deny listed")
            fi
        }
        for l_mod_base_directory in $l_mod_path; do
            if [ -d "$l_mod_base_directory/${l_mod_name//-/}/" ] && [ -n "$(ls -A
"$l_mod_base_directory/${l_mod_name//-/}/")" ]; then
                a_output3+=(" - \"$l_mod_base_directory\"")
                l_mod_chk_name="$l_mod_name"
                [[ "$l_mod_name" =~ overlay ]] && l_mod_chk_name="$l_mod_name:-2"
                [ "$l_dl" != "y" ] && f_module_chk
            else
                a_output+=(" - kernel module: \"$l_mod_name\" doesn't exist in
\"$l_mod_base_directory\"")
            fi
        done
        [ "${#a_output3[@]}" -gt 0 ] && printf '%s\n' "" " -- INFO --" " - module:
\"$l_mod_name\" exists in:" "${a_output3[@]}"
        if [ "${#a_output2[@]}" -le 0 ]; then
            printf '%s\n' "" "- Audit Result:" " ** PASS **" "${a_output[@]}"
        else
            printf '%s\n' "" "- Audit Result:" " ** FAIL **" " - Reason(s) for
audit failure:" "${a_output2[@]}"
            [ "${#a_output[@]}" -gt 0 ] && printf '%s\n' "- Correctly set:"
"${a_output[@]}"
        fi
    }
}

```

## Remediation:

Run the following script to unload and disable the `hfsplus` module:

- **IF** - the `hfsplus` kernel module is available in ANY installed kernel:

- Create a file ending in `.conf` with `install hfsplus /bin/false` in the `/etc/modprobe.d/` directory
- Create a file ending in `.conf` with `blacklist hfsplus` in the `/etc/modprobe.d/` directory
- Run `modprobe -r hfsplus 2>/dev/null; rmmod hfsplus 2>/dev/null` to remove `hfsplus` from the kernel

- **IF** - the `hfsplus` kernel module is not available on the system, or pre-compiled into the kernel, no remediation is necessary





```
#!/usr/bin/env bash

{
  a_output2=() a_output3=() l_dl="" l_mod_name="hfsplus" l_mod_type="fs"
  l_mod_path="$(readlink -f /lib/modules/**/kernel/$l_mod_type | sort -u)"
  f_module_fix()
  {
    l_dl="y" a_showconfig=()
    while IFS= read -r l_showconfig; do
      a_showconfig+=("$l_showconfig")
      done < <(modprobe --showconfig | grep -P --
'\b(install|blacklist)\h+'"$${l_mod_chk_name//-/}_"''\b')
      if lsmod | grep "$l_mod_chk_name" &> /dev/null; then
        a_output2+=(" - unloading kernel module: \"$l_mod_name\"")
        modprobe -r "$l_mod_chk_name" 2>/dev/null; rmmod "$l_mod_name"
2>/dev/null
      fi
      if ! grep -Pq -- '\binstall\h+'"$${l_mod_chk_name//-/}_"''\b' <<<
/"$a_showconfig[*]"; then
        a_output2+=(" - setting kernel module: \"$l_mod_name\" to
\"$(readlink -f /bin/false)\"")
        printf '%s\n' "install $l_mod_chk_name $(readlink -f /bin/false)" >>
/etc/modprobe.d/"$l_mod_name".conf
      fi
      if ! grep -Pq -- '\bblacklist\h+'"$${l_mod_chk_name//-/}_"''\b' <<<
/"$a_showconfig[*]"; then
        a_output2+=(" - denylisting kernel module: \"$l_mod_name\"")
        printf '%s\n' "blacklist $l_mod_chk_name" >>
/etc/modprobe.d/"$l_mod_name".conf
      fi
    }
    for l_mod_base_directory in $l_mod_path; do # Check if the module exists
on the system
      if [ -d "$l_mod_base_directory/$${l_mod_name//-/}_/" ] && [ -n "$(ls -A
"$l_mod_base_directory/$${l_mod_name//-/}_/")" ]; then
        a_output3+=(" - \"$l_mod_base_directory\"")
        l_mod_chk_name="$l_mod_name"
        [[ "$l_mod_name" =~ overlay ]] && l_mod_chk_name="$${l_mod_name::-2}"
        [ "$l_dl" != "y" ] && f_module_fix
      else
        printf '%s\n' " - kernel module: \"$l_mod_name\" doesn't exist in
\"$l_mod_base_directory\""
      fi
    done
    [ "${#a_output3[@]}" -gt 0 ] && printf '%s\n' "" " -- INFO --" " - module:
\"$l_mod_name\" exists in:" "${a_output3[@]}"
    [ "${#a_output2[@]}" -gt 0 ] && printf '%s\n' "" "${a_output2[@]}" ||
printf '%s\n' "" " - No changes needed"
    printf '%s\n' "" " - remediation of kernel module: \"$l_mod_name\"
complete" ""
  }
}
```

## References:

1. NIST SP 800-53 Rev. 5: CM-7

**CIS Controls:**

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>4.8 Uninstall or Disable Unnecessary Services on Enterprise Assets and Software</u> Uninstall or disable unnecessary services on enterprise assets and software, such as an unused file sharing service, web application module, or service function.			
v7	<u>9.2 Ensure Only Approved Ports, Protocols and Services Are Running</u> Ensure that only network ports, protocols, and services listening on a system with validated business needs, are running on each system.			

**MITRE ATT&CK Mappings:**

Techniques / Sub-techniques	Tactics	Mitigations
T1005, T1005.000	TA0005	M1050

### *1.1.1.5 Ensure jffs2 kernel module is not available (Automated)*

#### **Profile Applicability:**

- Level 1 - Server
- Level 1 - Workstation

#### **Description:**

The **jffs2** (journaling flash filesystem 2) filesystem type is a log-structured filesystem used in flash memory devices.

#### **Rationale:**

Removing support for unneeded filesystem types reduces the local attack surface of the system. If this filesystem type is not needed, disable it.

#### **Audit:**

Run the following script to verify:

- **IF** - the **jffs2** kernel module is available in ANY installed kernel, verify:

- An entry including **/bin/true** or **/bin/false** exists in a file within the **/etc/modprobe.d/** directory
- The module is deny listed in a file within the **/etc/modprobe.d/** directory
- The module is not loaded in the running kernel

- **IF** - the **jffs2** kernel module is not available on the system, or pre-compiled into the kernel, no additional configuration is necessary



```

#!/usr/bin/env bash

{
    a_output=() a_output2=() a_output3=() l_dl="" l_mod_name="jffs2"
    l_mod_type="fs"
    l_mod_path="$(readlink -f /lib/modules/**/kernel/$l_mod_type | sort -u)"
    f_module_chk()
    {
        l_dl="y" a_showconfig=()
        while IFS= read -r l_showconfig; do
            a_showconfig+=("$l_showconfig")
            done < <(modprobe --showconfig | grep -P --
'\b(install|blacklist)\h+'${l_mod_chk_name//-/}_'\b')
            if ! lsmod | grep "$l_mod_chk_name" &> /dev/null; then
                a_output+=(" - kernel module: \"$l_mod_name\" is not loaded")
            else
                a_output2+=(" - kernel module: \"$l_mod_name\" is loaded")
            fi
            if grep -Pq -- '\bbinstall\h+'${l_mod_chk_name//-/}_
/_}'\h+(\usr)?\bin\(/(true|false)\b' <<< "${a_showconfig[*]}"; then
                a_output+=(" - kernel module: \"$l_mod_name\" is not loadable")
            else
                a_output2+=(" - kernel module: \"$l_mod_name\" is loadable")
            fi
            if grep -Pq -- '\bbblacklist\h+'${l_mod_chk_name//-/}_'\b' <<<
"${a_showconfig[*]}"; then
                a_output+=(" - kernel module: \"$l_mod_name\" is deny listed")
            else
                a_output2+=(" - kernel module: \"$l_mod_name\" is not deny listed")
            fi
        }
        for l_mod_base_directory in $l_mod_path; do
            if [ -d "$l_mod_base_directory/${l_mod_name//-/}/" ] && [ -n "$(ls -A
"$l_mod_base_directory/${l_mod_name//-/}/")" ]; then
                a_output3+=(" - \"$l_mod_base_directory\"")
                l_mod_chk_name="$l_mod_name"
                [[ "$l_mod_name" =~ overlay ]] && l_mod_chk_name="$l_mod_name:-2)"
                [ "$l_dl" != "y" ] && f_module_chk
            else
                a_output+=(" - kernel module: \"$l_mod_name\" doesn't exist in
\"$l_mod_base_directory\"")
            fi
        done
        [ "${#a_output3[@]}" -gt 0 ] && printf '%s\n' "" " -- INFO --" " - module:
\"$l_mod_name\" exists in:" "${a_output3[@]}"
        if [ "${#a_output2[@]}" -le 0 ]; then
            printf '%s\n' "" "- Audit Result:" " ** PASS **" "${a_output[@]}"
        else
            printf '%s\n' "" "- Audit Result:" " ** FAIL **" " - Reason(s) for
audit failure:" "${a_output2[@]}"
            [ "${#a_output[@]}" -gt 0 ] && printf '%s\n' "- Correctly set:"
"${a_output[@]}"
        fi
    }
}

```

## Remediation:

Run the following script to unload and disable the `jffs2` module:

- **IF** - the `jffs2` kernel module is available in ANY installed kernel:

- Create a file ending in `.conf` with `install jffs2 /bin/false` in the `/etc/modprobe.d/` directory
- Create a file ending in `.conf` with `blacklist jffs2` in the `/etc/modprobe.d/` directory
- Run `modprobe -r jffs2 2>/dev/null; rmmod jffs2 2>/dev/null` to remove `jffs2` from the kernel

- **IF** - the `jffs2` kernel module is not available on the system, or pre-compiled into the kernel, no remediation is necessary





```
#!/usr/bin/env bash

{
  a_output2=() a_output3=() l_dl="" l_mod_name="jffs2" l_mod_type="fs"
  l_mod_path="$(readlink -f /lib/modules/**/kernel/$l_mod_type | sort -u)"
  f_module_fix()
  {
    l_dl="y" a_showconfig=()
    while IFS= read -r l_showconfig; do
      a_showconfig+=("$l_showconfig")
      done < <(modprobe --showconfig | grep -P --
'\b(install|blacklist)\h+' "${l_mod_chk_name//-/}_"' \b')
      if lsmod | grep "$l_mod_chk_name" &> /dev/null; then
        a_output2+=(" - unloading kernel module: \"$l_mod_name\"")
        modprobe -r "$l_mod_chk_name" 2>/dev/null; rmmod "$l_mod_name"
2>/dev/null
      fi
      if ! grep -Pq -- '\binstall\h+' "${l_mod_chk_name//-/}_"' \b' <<<
"${a_showconfig[*]}"; then
        a_output2+=(" - setting kernel module: \"$l_mod_name\" to
\"$(readlink -f /bin/false)\"")
        printf '%s\n' "install $l_mod_chk_name $(readlink -f /bin/false)" >>
/etc/modprobe.d/"$l_mod_name".conf
      fi
      if ! grep -Pq -- '\bblacklist\h+' "${l_mod_chk_name//-/}_"' \b' <<<
"${a_showconfig[*]}"; then
        a_output2+=(" - denylisting kernel module: \"$l_mod_name\"")
        printf '%s\n' "blacklist $l_mod_chk_name" >>
/etc/modprobe.d/"$l_mod_name".conf
      fi
    }
    for l_mod_base_directory in $l_mod_path; do # Check if the module exists
on the system
      if [ -d "$l_mod_base_directory/${l_mod_name//-/}/" ] && [ -n "$(ls -A
"$l_mod_base_directory/${l_mod_name//-/}/")" ]; then
        a_output3+=(" - \"$l_mod_base_directory\"")
        l_mod_chk_name="$l_mod_name"
        [[ "$l_mod_name" =~ overlay ]] && l_mod_chk_name="${l_mod_name::-2}"
        [ "$l_dl" != "y" ] && f_module_fix
      else
        printf '%s\n' " - kernel module: \"$l_mod_name\" doesn't exist in
\"$l_mod_base_directory\""
      fi
    done
    [ "${#a_output3[@]}" -gt 0 ] && printf '%s\n' "" " -- INFO --" " - module:
\"$l_mod_name\" exists in:" "${a_output3[@]}"
    [ "${#a_output2[@]}" -gt 0 ] && printf '%s\n' "" "${a_output2[@]}" ||
printf '%s\n' "" " - No changes needed"
    printf '%s\n' "" " - remediation of kernel module: \"$l_mod_name\"
complete" ""
  }
}
```

## References:

1. NIST SP 800-53 Rev. 5: CM-7

### CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>4.8 Uninstall or Disable Unnecessary Services on Enterprise Assets and Software</u> Uninstall or disable unnecessary services on enterprise assets and software, such as an unused file sharing service, web application module, or service function.			
v7	<u>9.2 Ensure Only Approved Ports, Protocols and Services Are Running</u> Ensure that only network ports, protocols, and services listening on a system with validated business needs, are running on each system.			

### MITRE ATT&CK Mappings:

Techniques / Sub-techniques	Tactics	Mitigations
T1005, T1005.000	TA0005	M1050

### *1.1.1.6 Ensure overlayfs kernel module is not available (Automated)*

#### **Profile Applicability:**

- Level 2 - Server
- Level 2 - Workstation

#### **Description:**

**overlayfs** is a Linux filesystem that layers multiple filesystems to create a single unified view which allows a user to "merge" several mount points into a unified filesystem.

#### **Rationale:**

The **overlayfs** has known CVE's: CVE-2023-32629, CVE-2023-2640, CVE-2023-0386. Disabling the **overlayfs** reduces the local attack surface by removing support for unnecessary filesystem types and mitigates potential risks associated with unauthorized execution of setuid files, enhancing the overall system security.

#### **Impact:**

**WARNING: If Container applications such as Docker, Kubernetes, Podman, Linux Containers (LXC), etc. are in use proceed with caution and consider the impact on containerized workloads, as disabling the **overlayfs** may severely disrupt containerization.**

#### **Audit:**

Run the following script to verify:

- **IF** - the **overlayfs** kernel module is available in ANY installed kernel, verify:

- An entry including **/bin/true** or **/bin/false** exists in a file within the **/etc/modprobe.d/** directory
- The module is deny listed in a file within the **/etc/modprobe.d/** directory
- The module is not loaded in the running kernel

- **IF** - the **overlayfs** kernel module is not available on the system, or pre-compiled into the kernel, no additional configuration is necessary

```

#!/usr/bin/env bash

{
    a_output=() a_output2=() a_output3=() l_dl="" l_mod_name="overlayfs"
    l_mod_type="fs"
    l_mod_path="$(readlink -f /lib/modules/**/kernel/$l_mod_type | sort -u)"
    f_module_chk()
    {
        l_dl="y" a_showconfig=()
        while IFS= read -r l_showconfig; do
            a_showconfig+=("$l_showconfig")
            done < <(modprobe --showconfig | grep -P --
'\b(install|blacklist)\h+'${l_mod_chk_name//-/}_'\b')
            if ! lsmod | grep "$l_mod_chk_name" &> /dev/null; then
                a_output+=(" - kernel module: \"$l_mod_name\" is not loaded")
            else
                a_output2+=(" - kernel module: \"$l_mod_name\" is loaded")
            fi
            if grep -Pq -- '\bbinstall\h+'${l_mod_chk_name//-/}_
/_}'\h+(\usr)?\bin\/(true|false)\b' <<< "${a_showconfig[*]}"; then
                a_output+=(" - kernel module: \"$l_mod_name\" is not loadable")
            else
                a_output2+=(" - kernel module: \"$l_mod_name\" is loadable")
            fi
            if grep -Pq -- '\bbblacklist\h+'${l_mod_chk_name//-/}_'\b' <<<
"${a_showconfig[*]}"; then
                a_output+=(" - kernel module: \"$l_mod_name\" is deny listed")
            else
                a_output2+=(" - kernel module: \"$l_mod_name\" is not deny listed")
            fi
        }
        for l_mod_base_directory in $l_mod_path; do
            if [ -d "$l_mod_base_directory/${l_mod_name//-/}/" ] && [ -n "$(ls -A
"$l_mod_base_directory/${l_mod_name//-/}/")" ]; then
                a_output3+=(" - \"$l_mod_base_directory\"")
                l_mod_chk_name="$l_mod_name"
                [[ "$l_mod_name" =~ overlay ]] && l_mod_chk_name="$l_mod_name:-2)"
                [ "$l_dl" != "y" ] && f_module_chk
            else
                a_output+=(" - kernel module: \"$l_mod_name\" doesn't exist in
\"$l_mod_base_directory\"")
            fi
        done
        [ "${#a_output3[@]}" -gt 0 ] && printf '%s\n' "" " -- INFO --" " - module:
\"$l_mod_name\" exists in:" "${a_output3[@]}"
        if [ "${#a_output2[@]}" -le 0 ]; then
            printf '%s\n' "" "- Audit Result:" " ** PASS **" "${a_output[@]}"
        else
            printf '%s\n' "" "- Audit Result:" " ** FAIL **" " - Reason(s) for
audit failure:" "${a_output2[@]}"
            [ "${#a_output[@]}" -gt 0 ] && printf '%s\n' "- Correctly set:"
"${a_output[@]}"
        fi
    }
}

```

## Remediation:

Run the following script to unload and disable the `overlayfs` module:

- **IF** - the `overlayfs` kernel module is available in ANY installed kernel:

- Create a file ending in `.conf` with `install overlayfs /bin/false` in the `/etc/modprobe.d/` directory
- Create a file ending in `.conf` with `blacklist overlayfs` in the `/etc/modprobe.d/` directory
- Run `modprobe -r overlayfs 2>/dev/null; rmmod overlayfs 2>/dev/null` to remove `overlayfs` from the kernel

- **IF** - the `overlayfs` kernel module is not available on the system, or pre-compiled into the kernel, no remediation is necessary

```

#!/usr/bin/env bash

{
    a_output2=() a_output3=() l_dl="" l_mod_name="overlayfs" l_mod_type="fs"
    l_mod_path="$(readlink -f /lib/modules/**/kernel/$l_mod_type | sort -u)"
    f_module_fix()
    {
        l_dl="y" a_showconfig=()
        while IFS= read -r l_showconfig; do
            a_showconfig+=("$l_showconfig")
            done < <(modprobe --showconfig | grep -P --
'\b(install|blacklist)\h+'${l_mod_chk_name//-/}_'\b')
            if lsmod | grep "$l_mod_chk_name" &> /dev/null; then
                a_output2+=(" - unloading kernel module: \"$l_mod_name\"")
                modprobe -r "$l_mod_chk_name" 2>/dev/null; rmmod "$l_mod_name"
2>/dev/null
            fi
            if ! grep -Pq -- '\binstall\h+'${l_mod_chk_name//-/}_
/_'\b'\h+(\usr)?\bin\/(true|false)\b' <<< "${a_showconfig[*]}"; then
                a_output2+=(" - setting kernel module: \"$l_mod_name\" to
\"$(readlink -f /bin/false)\"")
                printf '%s\n' "install $l_mod_chk_name $(readlink -f /bin/false)" >>
/etc/modprobe.d/"$l_mod_name".conf
            fi
            if ! grep -Pq -- '\bblacklist\h+'${l_mod_chk_name//-/}_'\b' <<<
"${a_showconfig[*]}"; then
                a_output2+=(" - denylisting kernel module: \"$l_mod_name\"")
                printf '%s\n' "blacklist $l_mod_chk_name" >>
/etc/modprobe.d/"$l_mod_name".conf
            fi
        }
        for l_mod_base_directory in $l_mod_path; do # Check if the module exists
on the system
            if [ -d "$l_mod_base_directory/${l_mod_name//-/}/" ] && [ -n "$(ls -A
"$l_mod_base_directory/${l_mod_name//-/}/")" ]; then
                a_output3+=(" - \"$l_mod_base_directory\"")
                l_mod_chk_name="$l_mod_name"
                [[ "$l_mod_name" =~ overlay ]] && l_mod_chk_name="${l_mod_name::-2}"
                [ "$l_dl" != "y" ] && f_module_fix
            else
                printf '%s\n' " - kernel module: \"$l_mod_name\" doesn't exist in
\"$l_mod_base_directory\""
            fi
        done
        [ "${#a_output3[@]}" -gt 0 ] && printf '%s\n' "" " -- INFO --" " - module:
\"$l_mod_name\" exists in:" "${a_output3[@]}"
        [ "${#a_output2[@]}" -gt 0 ] && printf '%s\n' "" "${a_output2[@]}" ||
printf '%s\n' "" " - No changes needed"
        printf '%s\n' "" " - remediation of kernel module: \"$l_mod_name\"
complete" ""
    }
}

```



## References:

1. NIST SP 800-53 Rev. 5: CM-7
2. <https://docs.kernel.org/filesystems/overlayfs.html>
3. [https://wiki.archlinux.org/title/Overlay\\_filesystem](https://wiki.archlinux.org/title/Overlay_filesystem)
4. <https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=overlayfs>

## CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<b><u>4.8 Uninstall or Disable Unnecessary Services on Enterprise Assets and Software</u></b> Uninstall or disable unnecessary services on enterprise assets and software, such as an unused file sharing service, web application module, or service function.		●	●
v7	<b><u>9.2 Ensure Only Approved Ports, Protocols and Services Are Running</u></b> Ensure that only network ports, protocols, and services listening on a system with validated business needs, are running on each system.		●	●

## MITRE ATT&CK Mappings:

Techniques / Sub-techniques	Tactics	Mitigations
T1005, T1005.000	TA0005	M1050

### 1.1.1.7 Ensure squashfs kernel module is not available (Automated)

#### Profile Applicability:

- Level 2 - Server
- Level 2 - Workstation

#### Description:

The **squashfs** filesystem type is a compressed read-only Linux filesystem embedded in small footprint systems. A **squashfs** image can be used without having to first decompress the image.

#### Rationale:

Removing support for unneeded filesystem types reduces the local attack surface of the system. If this filesystem type is not needed, disable it.

#### Impact:

As Snap packages utilize **squashfs** as a compressed filesystem, disabling **squashfs** will cause Snap packages to fail.

**Snap** application packages of software are self-contained and work across a range of Linux distributions. This is unlike traditional Linux package management approaches, like APT or RPM, which require specifically adapted packages per Linux distribution on an application update and delay therefore application deployment from developers to their software's end-user. Snaps themselves have no dependency on any external store ("App store"), can be obtained from any source and can be therefore used for upstream software deployment.

#### Audit:

Run the following script to verify:

- **IF** - the **squashfs** kernel module is available in ANY installed kernel, verify:

- An entry including **/bin/true** or **/bin/false** exists in a file within the **/etc/modprobe.d/** directory
- The module is deny listed in a file within the **/etc/modprobe.d/** directory
- The module is not loaded in the running kernel

- **IF** - the **squashfs** kernel module is not available on the system, or pre-compiled into the kernel, no additional configuration is necessary

```

#!/usr/bin/env bash

{
    a_output=() a_output2=() a_output3=() l_dl="" l_mod_name="squashfs"
    l_mod_type="fs"
    l_mod_path="$(readlink -f /lib/modules/**/kernel/$l_mod_type | sort -u)"
    f_module_chk()
    {
        l_dl="y" a_showconfig=()
        while IFS= read -r l_showconfig; do
            a_showconfig+=("$l_showconfig")
            done < <(modprobe --showconfig | grep -P --
'\b(install|blacklist)\h+'${l_mod_chk_name//-/}_'\b')
            if ! lsmod | grep "$l_mod_chk_name" &> /dev/null; then
                a_output+=(" - kernel module: \"$l_mod_name\" is not loaded")
            else
                a_output2+=(" - kernel module: \"$l_mod_name\" is loaded")
            fi
            if grep -Pq -- '\bbinstall\h+'${l_mod_chk_name//-/}_
/_}'\h+(\usr)?\bin\/(true|false)\b' <<< "${a_showconfig[*]}"; then
                a_output+=(" - kernel module: \"$l_mod_name\" is not loadable")
            else
                a_output2+=(" - kernel module: \"$l_mod_name\" is loadable")
            fi
            if grep -Pq -- '\bbblacklist\h+'${l_mod_chk_name//-/}_'\b' <<<
"${a_showconfig[*]}"; then
                a_output+=(" - kernel module: \"$l_mod_name\" is deny listed")
            else
                a_output2+=(" - kernel module: \"$l_mod_name\" is not deny listed")
            fi
        }
        for l_mod_base_directory in $l_mod_path; do
            if [ -d "$l_mod_base_directory/${l_mod_name//-/}/" ] && [ -n "$(ls -A
"$l_mod_base_directory/${l_mod_name//-/}/")" ]; then
                a_output3+=(" - \"$l_mod_base_directory\"")
                l_mod_chk_name="$l_mod_name"
                [[ "$l_mod_name" =~ overlay ]] && l_mod_chk_name="$l_mod_name:-2"
                [ "$l_dl" != "y" ] && f_module_chk
            else
                a_output+=(" - kernel module: \"$l_mod_name\" doesn't exist in
\"$l_mod_base_directory\"")
            fi
        done
        [ "${#a_output3[@]}" -gt 0 ] && printf '%s\n' "" " -- INFO --" " - module:
\"$l_mod_name\" exists in:" "${a_output3[@]}"
        if [ "${#a_output2[@]}" -le 0 ]; then
            printf '%s\n' "" "- Audit Result:" " ** PASS **" "${a_output[@]}"
        else
            printf '%s\n' "" "- Audit Result:" " ** FAIL **" " - Reason(s) for
audit failure:" "${a_output2[@]}"
            [ "${#a_output[@]}" -gt 0 ] && printf '%s\n' "- Correctly set:"
"${a_output[@]}"
        fi
    }
}

```

**Note:** On operating systems where `squashfs` is pre-build into the kernel:

- This is considered an acceptable "passing" state
- The kernel **should not** be re-compiled to remove `squashfs`
- This audit will return a passing state with "module: "squashfs" doesn't exist in ..."

### Remediation:

Run the following script to unload and disable the `udf` module:

- **IF** - the `squashfs` kernel module is available in ANY installed kernel:

- Create a file ending in `.conf` with `install squashfs /bin/false` in the `/etc/modprobe.d/` directory
- Create a file ending in `.conf` with `blacklist squashfs` in the `/etc/modprobe.d/` directory
- Run `modprobe -r squashfs 2>/dev/null; rmmod squashfs 2>/dev/null` to remove `squashfs` from the kernel

- **IF** - the `squashfs` kernel module is not available on the system, or pre-compiled into the kernel, no remediation is necessary





```
#!/usr/bin/env bash

{
  a_output2=() a_output3=() l_dl="" l_mod_name="squashfs" l_mod_type="fs"
  l_mod_path="$(readlink -f /lib/modules/**/kernel/$l_mod_type | sort -u)"
  f_module_fix()
  {
    l_dl="y" a_showconfig=()
    while IFS= read -r l_showconfig; do
      a_showconfig+=("$l_showconfig")
      done < <(modprobe --showconfig | grep -P --
'\b(install|blacklist)\h+' "${l_mod_chk_name//-/}_"' \b')
      if lsmod | grep "$l_mod_chk_name" &> /dev/null; then
        a_output2+=(" - unloading kernel module: \"$l_mod_name\"")
        modprobe -r "$l_mod_chk_name" 2>/dev/null; rmmod "$l_mod_name"
2>/dev/null
      fi
      if ! grep -Pq -- '\binstall\h+' "${l_mod_chk_name//-/}_"' \b' <<<
"${a_showconfig[*]}"; then
        a_output2+=(" - setting kernel module: \"$l_mod_name\" to
\"$(readlink -f /bin/false)\"")
        printf '%s\n' "install $l_mod_chk_name $(readlink -f /bin/false)" >>
/etc/modprobe.d/"$l_mod_name".conf
      fi
      if ! grep -Pq -- '\bblacklist\h+' "${l_mod_chk_name//-/}_"' \b' <<<
"${a_showconfig[*]}"; then
        a_output2+=(" - denylisting kernel module: \"$l_mod_name\"")
        printf '%s\n' "blacklist $l_mod_chk_name" >>
/etc/modprobe.d/"$l_mod_name".conf
      fi
    }
    for l_mod_base_directory in $l_mod_path; do # Check if the module exists
on the system
      if [ -d "$l_mod_base_directory/${l_mod_name//-/}/" ] && [ -n "$(ls -A
"$l_mod_base_directory/${l_mod_name//-/}/")" ]; then
        a_output3+=(" - \"$l_mod_base_directory\"")
        l_mod_chk_name="$l_mod_name"
        [[ "$l_mod_name" =~ overlay ]] && l_mod_chk_name="${l_mod_name::-2}"
        [ "$l_dl" != "y" ] && f_module_fix
      else
        printf '%s\n' " - kernel module: \"$l_mod_name\" doesn't exist in
\"$l_mod_base_directory\""
      fi
    done
    [ "${#a_output3[@]}" -gt 0 ] && printf '%s\n' "" " -- INFO --" " - module:
\"$l_mod_name\" exists in:" "${a_output3[@]}"
    [ "${#a_output2[@]}" -gt 0 ] && printf '%s\n' "" "${a_output2[@]}" ||
printf '%s\n' "" " - No changes needed"
    printf '%s\n' "" " - remediation of kernel module: \"$l_mod_name\"
complete" ""
  }
}
```

## References:

1. NIST SP 800-53 Rev. 5: CM-7

### CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>4.8 Uninstall or Disable Unnecessary Services on Enterprise Assets and Software</u> Uninstall or disable unnecessary services on enterprise assets and software, such as an unused file sharing service, web application module, or service function.			
v7	<u>9.2 Ensure Only Approved Ports, Protocols and Services Are Running</u> Ensure that only network ports, protocols, and services listening on a system with validated business needs, are running on each system.			

### MITRE ATT&CK Mappings:

Techniques / Sub-techniques	Tactics	Mitigations
T1005, T1005.000	TA0005	M1050

### *1.1.1.8 Ensure udf kernel module is not available (Automated)*

#### **Profile Applicability:**

- Level 2 - Server
- Level 2 - Workstation

#### **Description:**

The **udf** filesystem type is the universal disk format used to implement ISO/IEC 13346 and ECMA-167 specifications. This is an open vendor filesystem type for data storage on a broad range of media. This filesystem type is necessary to support writing DVDs and newer optical disc formats.

#### **Rationale:**

Removing support for unneeded filesystem types reduces the local attack surface of the system. If this filesystem type is not needed, disable it.

#### **Impact:**

Microsoft Azure requires the usage of **udf**.

**udf** **should not** be disabled on systems run on Microsoft Azure.

#### **Audit:**

Run the following script to verify:

- **IF** - the **udf** kernel module is available in ANY installed kernel, verify:

- An entry including **/bin/true** or **/bin/false** exists in a file within the **/etc/modprobe.d/** directory
- The module is deny listed in a file within the **/etc/modprobe.d/** directory
- The module is not loaded in the running kernel

- **IF** - the **udf** kernel module is not available on the system, or pre-compiled into the kernel, no additional configuration is necessary

```

#!/usr/bin/env bash

{
    a_output=() a_output2=() a_output3=() l_dl="" l_mod_name="udf"
    l_mod_type="fs"
    l_mod_path="$(readlink -f /lib/modules/**/kernel/$l_mod_type | sort -u)"
    f_module_chk()
    {
        l_dl="y" a_showconfig=()
        while IFS= read -r l_showconfig; do
            a_showconfig+=("$l_showconfig")
            done < <(modprobe --showconfig | grep -P --
'\b(install|blacklist)\h+'${l_mod_chk_name//-/}_'\b')
            if ! lsmod | grep "$l_mod_chk_name" &> /dev/null; then
                a_output+=(" - kernel module: \"$l_mod_name\" is not loaded")
            else
                a_output2+=(" - kernel module: \"$l_mod_name\" is loaded")
            fi
            if grep -Pq -- '\bbinstall\h+'${l_mod_chk_name//-/}_'\b' <<< "${a_showconfig[*]}"; then
                a_output+=(" - kernel module: \"$l_mod_name\" is not loadable")
            else
                a_output2+=(" - kernel module: \"$l_mod_name\" is loadable")
            fi
            if grep -Pq -- '\bbblacklist\h+'${l_mod_chk_name//-/}_'\b' <<<
"${a_showconfig[*]}"; then
                a_output+=(" - kernel module: \"$l_mod_name\" is deny listed")
            else
                a_output2+=(" - kernel module: \"$l_mod_name\" is not deny listed")
            fi
        }
        for l_mod_base_directory in $l_mod_path; do
            if [ -d "$l_mod_base_directory/${l_mod_name//-/}/" ] && [ -n "$(ls -A
"$l_mod_base_directory/${l_mod_name//-/}/")" ]; then
                a_output3+=(" - \"$l_mod_base_directory\"")
                l_mod_chk_name="$l_mod_name"
                [[ "$l_mod_name" =~ overlay ]] && l_mod_chk_name="$l_mod_name:-2"
                [ "$l_dl" != "y" ] && f_module_chk
            else
                a_output+=(" - kernel module: \"$l_mod_name\" doesn't exist in
\"$l_mod_base_directory\"")
            fi
        done
        [ "${#a_output3[@]}" -gt 0 ] && printf '%s\n' "" " -- INFO --" " - module:
\"$l_mod_name\" exists in:" "${a_output3[@]}"
        if [ "${#a_output2[@]}" -le 0 ]; then
            printf '%s\n' "" "- Audit Result:" " ** PASS **" "${a_output[@]}"
        else
            printf '%s\n' "" "- Audit Result:" " ** FAIL **" " - Reason(s) for
audit failure:" "${a_output2[@]}"
            [ "${#a_output[@]}" -gt 0 ] && printf '%s\n' "- Correctly set:"
"${a_output[@]}"
        fi
    }
}

```



## Remediation:

Run the following script to unload and disable the `udf` module:

- **IF** - the `udf` kernel module is available in ANY installed kernel:

- Create a file ending in `.conf` with `install udf /bin/false` in the `/etc/modprobe.d/` directory
- Create a file ending in `.conf` with `blacklist udf` in the `/etc/modprobe.d/` directory
- Run `modprobe -r udf 2>/dev/null; rmmod udf 2>/dev/null` to remove `udf` from the kernel

- **IF** - the `udf` kernel module is not available on the system, or pre-compiled into the kernel, no remediation is necessary





```
#!/usr/bin/env bash

{
  a_output2=() a_output3=() l_dl="" l_mod_name="udf" l_mod_type="fs"
  l_mod_path="$(readlink -f /lib/modules/**/kernel/$l_mod_type | sort -u)"
  f_module_fix()
  {
    l_dl="y" a_showconfig=()
    while IFS= read -r l_showconfig; do
      a_showconfig+=("$l_showconfig")
      done < <(modprobe --showconfig | grep -P --
'\b(install|blacklist)\h+'${l_mod_chk_name//-/}_'\b')
      if lsmod | grep "$l_mod_chk_name" &> /dev/null; then
        a_output2+=(" - unloading kernel module: \"$l_mod_name\"")
        modprobe -r "$l_mod_chk_name" 2>/dev/null; rmmod "$l_mod_name"
2>/dev/null
      fi
      if ! grep -Pq -- '\binstall\h+'${l_mod_chk_name//-/}_
/_'\b'\h+(\usr)?\bin\/(true|false)\b' <<< "${a_showconfig[*]}"; then
        a_output2+=(" - setting kernel module: \"$l_mod_name\" to
\"$(readlink -f /bin/false)\"")
        printf '%s\n' "install $l_mod_chk_name $(readlink -f /bin/false)" >>
/etc/modprobe.d/"$l_mod_name".conf
      fi
      if ! grep -Pq -- '\bblacklist\h+'${l_mod_chk_name//-/}_'\b' <<<
"${a_showconfig[*]}"; then
        a_output2+=(" - denylisting kernel module: \"$l_mod_name\"")
        printf '%s\n' "blacklist $l_mod_chk_name" >>
/etc/modprobe.d/"$l_mod_name".conf
      fi
    }
    for l_mod_base_directory in $l_mod_path; do # Check if the module exists
on the system
      if [ -d "$l_mod_base_directory/${l_mod_name//-/}/" ] && [ -n "$(ls -A
"$l_mod_base_directory/${l_mod_name//-/}/")" ]; then
        a_output3+=(" - \"$l_mod_base_directory\"")
        l_mod_chk_name="$l_mod_name"
        [[ "$l_mod_name" =~ overlay ]] && l_mod_chk_name="${l_mod_name::-2}"
        [ "$l_dl" != "y" ] && f_module_fix
      else
        printf '%s\n' " - kernel module: \"$l_mod_name\" doesn't exist in
\"$l_mod_base_directory\""
      fi
    done
    [ "${#a_output3[@]}" -gt 0 ] && printf '%s\n' "" " -- INFO --" " - module:
\"$l_mod_name\" exists in:" "${a_output3[@]}"
    [ "${#a_output2[@]}" -gt 0 ] && printf '%s\n' "" "${a_output2[@]}" ||
printf '%s\n' "" " - No changes needed"
    printf '%s\n' "" " - remediation of kernel module: \"$l_mod_name\"
complete" ""
  }
}
```

## References:

1. NIST SP 800-53 Rev. 5: CM-7

**CIS Controls:**

Controls Version	Control	IG 1	IG 2	IG 3
v8	<b><u>4.8 Uninstall or Disable Unnecessary Services on Enterprise Assets and Software</u></b> Uninstall or disable unnecessary services on enterprise assets and software, such as an unused file sharing service, web application module, or service function.			
v7	<b><u>9.2 Ensure Only Approved Ports, Protocols and Services Are Running</u></b> Ensure that only network ports, protocols, and services listening on a system with validated business needs, are running on each system.			

**MITRE ATT&CK Mappings:**

Techniques / Sub-techniques	Tactics	Mitigations
T1005, T1005.000	TA0005	M1050

### *1.1.1.9 Ensure usb-storage kernel module is not available (Automated)*

#### **Profile Applicability:**

- Level 1 - Server
- Level 2 - Workstation

#### **Description:**

USB storage provides a means to transfer and store files ensuring persistence and availability of the files independent of network connection status. Its popularity and utility has led to USB-based malware being a simple and common means for network infiltration and a first step to establishing a persistent threat within a networked environment.

#### **Rationale:**

Restricting USB access on the system will decrease the physical attack surface for a device and diminish the possible vectors to introduce malware.

#### **Impact:**

Disabling the **usb-storage** module will disable any usage of USB storage devices.

If requirements and local site policy allow the use of such devices, other solutions should be configured accordingly instead. One example of a commonly used solution is **USBGuard**.

#### **Audit:**

Run the following script to verify:

- **IF** - the **usb-storage** kernel module is available in ANY installed kernel, verify:

- An entry including **/bin/true** or **/bin/false** exists in a file within the **/etc/modprobe.d/** directory
- The module is deny listed in a file within the **/etc/modprobe.d/** directory
- The module is not loaded in the running kernel

- **IF** - the **usb-storage** kernel module is not available on the system, or pre-compiled into the kernel, no additional configuration is necessary

```

#!/usr/bin/env bash

{
    a_output=() a_output2=() a_output3=() l_dl="" l_mod_name="usb-storage"
    l_mod_type="drivers"
    l_mod_path="$(readlink -f /lib/modules/**/kernel/$l_mod_type | sort -u)"
    f_module_chk()
    {
        l_dl="y" a_showconfig=()
        while IFS= read -r l_showconfig; do
            a_showconfig+=("$l_showconfig")
            done < <(modprobe --showconfig | grep -P --
'\b(install|blacklist)\h+'${l_mod_chk_name//-/}_}''\b')
            if ! lsmod | grep "$l_mod_chk_name" &> /dev/null; then
                a_output+=(" - kernel module: \"$l_mod_name\" is not loaded")
            else
                a_output2+=(" - kernel module: \"$l_mod_name\" is loaded")
            fi
            if grep -Pq -- '\bbinstall\h+'${l_mod_chk_name//-/}_}''\b' <<<
"/_}"'\h+(\usr)?\bin\/(true|false)\b' <<< "${a_showconfig[*]}"; then
                a_output+=(" - kernel module: \"$l_mod_name\" is not loadable")
            else
                a_output2+=(" - kernel module: \"$l_mod_name\" is loadable")
            fi
            if grep -Pq -- '\bbblacklist\h+'${l_mod_chk_name//-/}_}''\b' <<<
"${a_showconfig[*]}"; then
                a_output+=(" - kernel module: \"$l_mod_name\" is deny listed")
            else
                a_output2+=(" - kernel module: \"$l_mod_name\" is not deny listed")
            fi
        }
        for l_mod_base_directory in $l_mod_path; do
            if [ -d "$l_mod_base_directory/${l_mod_name//-/}/" ] && [ -n "$(ls -A
"$l_mod_base_directory/${l_mod_name//-/}/")" ]; then
                a_output3+=(" - \"$l_mod_base_directory\"")
                l_mod_chk_name="$l_mod_name"
                [[ "$l_mod_name" =~ overlay ]] && l_mod_chk_name="$l_mod_name::-2"
                [ "$l_dl" != "y" ] && f_module_chk
            else
                a_output+=(" - kernel module: \"$l_mod_name\" doesn't exist in
\"$l_mod_base_directory\"")
            fi
        done
        [ "${#a_output3[@]}" -gt 0 ] && printf '%s\n' "" " -- INFO --" " - module:
\"$l_mod_name\" exists in:" "${a_output3[@]}"
        if [ "${#a_output2[@]}" -le 0 ]; then
            printf '%s\n' "" "- Audit Result:" " ** PASS **" "${a_output[@]}"
        else
            printf '%s\n' "" "- Audit Result:" " ** FAIL **" " - Reason(s) for
audit failure:" "${a_output2[@]}"
            [ "${#a_output[@]}" -gt 0 ] && printf '%s\n' "- Correctly set:"
"${a_output[@]}"
        fi
    }
}

```

## Remediation:

Run the following script to unload and disable the `usb-storage` module:

- **IF** - the `usb-storage` kernel module is available in ANY installed kernel:

- Create a file ending in `.conf` with `install usb-storage /bin/false` in the `/etc/modprobe.d/` directory
- Create a file ending in `.conf` with `blacklist usb-storage` in the `/etc/modprobe.d/` directory
- Run `modprobe -r usb-storage 2>/dev/null; rmmod usb-storage 2>/dev/null` to remove `usb-storage` from the kernel

- **IF** - the `usb-storage` kernel module is not available on the system, or pre-compiled into the kernel, no remediation is necessary

```
#!/usr/bin/env bash

{
    a_output2=() a_output3=() l_dl="" l_mod_name="usb-storage"
l_mod_type="drivers"
    l_mod_path="$(readlink -f /lib/modules/**/kernel/$l_mod_type | sort -u)"
    f_module_fix()
    {
        l_dl="y" a_showconfig=()
        while IFS= read -r l_showconfig; do
            a_showconfig+=("$l_showconfig")
            done < <(modprobe --showconfig | grep -P --
'\b(install|blacklist)\h+' "${l_mod_chk_name//-/}_"' \b')
            if lsmod | grep "$l_mod_chk_name" &> /dev/null; then
                a_output2+=(" - unloading kernel module: \"$l_mod_name\"")
                modprobe -r "$l_mod_chk_name" 2>/dev/null; rmmod "$l_mod_name"
2>/dev/null
            fi
            if ! grep -Pq -- '\binstall\h+' "${l_mod_chk_name//-/}_"' \b' <<<
"${a_showconfig[*]}"; then
                a_output2+=(" - setting kernel module: \"$l_mod_name\" to
\"$(readlink -f /bin/false)\")
                printf '%s\n' "install $l_mod_chk_name $(readlink -f /bin/false)" >>
/etc/modprobe.d/"$l_mod_name".conf
            fi
            if ! grep -Pq -- '\bblacklist\h+' "${l_mod_chk_name//-/}_"' \b' <<<
"${a_showconfig[*]}"; then
                a_output2+=(" - denylisting kernel module: \"$l_mod_name\"")
                printf '%s\n' "blacklist $l_mod_chk_name" >>
/etc/modprobe.d/"$l_mod_name".conf
            fi
        }
        for l_mod_base_directory in $l_mod_path; do # Check if the module exists
on the system
            if [ -d "$l_mod_base_directory/${l_mod_name//-/}_/" ] && [ -n "$(ls -A
"$l_mod_base_directory/${l_mod_name//-/}_/")" ]; then
                a_output3+=(" - \"$l_mod_base_directory\"")
                l_mod_chk_name="$l_mod_name"
                [[ "$l_mod_name" =~ overlay ]] && l_mod_chk_name="$l_mod_name:-2}"
                [ "$l_dl" != "y" ] && f_module_fix
            else
                printf '%s\n' " - kernel module: \"$l_mod_name\" doesn't exist in
\"$l_mod_base_directory\""
            fi
        done
        [ "${#a_output3[@]}" -gt 0 ] && printf '%s\n' "" " -- INFO --" " - module:
\"$l_mod_name\" exists in:" "${a_output3[@]}"
        [ "${#a_output2[@]}" -gt 0 ] && printf '%s\n' "" "${a_output2[@]}" ||
printf '%s\n' "" " - No changes needed"
        printf '%s\n' "" " - remediation of kernel module: \"$l_mod_name\"
complete" ""
    }
}
```

## References:






1. NIST SP 800-53 Rev. 5: SI-3

### Additional Information:

An alternative solution to disabling the usb-storage module may be found in USBGuard.

Use of USBGuard and construction of USB device policies should be done in alignment with site policy.

### CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	10.3 <u>Disable Autorun and Autoplay for Removable Media</u> Disable autorun and autoplay auto-execute functionality for removable media.			
v7	13.7 <u>Manage USB Devices</u> If USB storage devices are required, enterprise software should be used that can configure systems to allow the use of specific devices. An inventory of such devices should be maintained.			

### MITRE ATT&CK Mappings:

Techniques / Sub-techniques	Tactics	Mitigations
T1052, T1052.001, T1091, T1091.000, T1200, T1200.000	TA0001, TA0010	M1034



### *1.1.1.10 Ensure unused filesystems kernel modules are not available (Manual)*

#### **Profile Applicability:**

- Level 1 - Server
- Level 1 - Workstation

#### **Description:**

Filesystem kernel modules are pieces of code that can be dynamically loaded into the Linux kernel to extend its filesystem capabilities, or so-called base kernel, of an operating system. Filesystem kernel modules are typically used to add support for new hardware (as device drivers), or for adding system calls.

#### **Rationale:**

While loadable filesystem kernel modules are a convenient method of modifying the running kernel, this can be abused by attackers on a compromised system to prevent detection of their processes or files, allowing them to maintain control over the system. Many rootkits make use of loadable filesystem kernel modules in this way.

Removing support for unneeded filesystem types reduces the local attack surface of the system. If this filesystem type is not needed, disable it. The following filesystem kernel modules have known CVE's and should be made unavailable if no dependencies exist:

- **afs** - CVE-2022-37402
- **ceph** - CVE-2022-0670
- **cifs** - CVE-2022-29869
- **exfat** CVE-2022-29973
- **ext** CVE-2022-1184
- **fat** CVE-2022-22043
- **fscache** CVE-2022-3630
- **fuse** CVE-2023-0386
- **gfs2** CVE-2023-3212
- **nfs\_common** CVE-2023-6660
- **nfsd** CVE-2022-43945
- **smbfs\_common** CVE-2022-2585

#### **Impact:**

This list may be quite extensive and covering all edges cases is difficult. Therefore, it's crucial to carefully consider the implications and dependencies before making any changes to the filesystem kernel module configurations.

**Audit:**

Run the following script to:

- Look at the filesystem kernel modules available to the currently running kernel.
- Exclude mounted filesystem kernel modules that don't currently have a CVE
- List filesystem kernel modules that are not fully disabled, or are loaded into the kernel

Review the generated output

```

#!/usr/bin/env bash

{
  a_output=(); a_output2=(); a_modprobe_config=(); a_excluded=(); a_available_modules=()
  a_ignore=("xfs" "vfat" "ext2" "ext3" "ext4")
  a_cve_exists=("afs" "ceph" "cifs" "exfat" "ext" "fat" "fscache" "fuse" "gfs2" "nfs_common"
  "nfsd" "smbfs_common")
  f_module_chk()
  {
    l_out2=""; grep -Pq -- "\b$l_mod_name\b" <<< "${a_cve_exists[*]}" && l_out2=" <- CVE
exists!"
    if ! grep -Pq -- '\bblacklist\b' "$l_mod_name" <<< "${a_modprobe_config[*]"; then
      a_output2+=(" - Kernel module: \"$l_mod_name\" is not fully disabled $l_out2")
    elif ! grep -Pq -- '\binstall\b' "$l_mod_name" <<< "${a_modprobe_config[*]"; then
      a_output2+=(" - Kernel module: \"$l_mod_name\" is not fully disabled $l_out2")
    fi
    if lsmod | grep "$l_mod_name" &> /dev/null; then # Check if the module is currently loaded
      l_output2+=(" - Kernel module: \"$l_mod_name\" is loaded" "")
    fi
  }
  while IFS= read -r -d $'\0' l_module_dir; do
    a_available_modules+=("${basename "$l_module_dir"}")
    done < <(find "$(readlink -f /lib/modules/"$(uname -r)"/kernel/fs)" -mindepth 1 -maxdepth 1 -
type d ! -empty -print0)
    while IFS= read -r l_exclude; do
      if grep -Pq -- "\b$l_exclude\b" <<< "${a_cve_exists[*]"; then
        a_output2+=(" - ** WARNING: kernel module: \"$l_exclude\" has a CVE and is currently
mounted! **")
      elif
        grep -Pq -- "\b$l_exclude\b" <<< "${a_available_modules[*]"; then
          a_output+=(" - Kernel module: \"$l_exclude\" is currently mounted - do NOT unload or
disable")
        fi
        ! grep -Pq -- "\b$l_exclude\b" <<< "${a_ignore[*]"; && a_ignore+=("$l_exclude")
      done < <(findmnt -knD | awk '{print $2}' | sort -u)
      while IFS= read -r l_config; do
        a_modprobe_config+=("$l_config")
        done < <(modprobe --showconfig | grep -P '^h*(blacklist|install)')
        for l_mod_name in "${a_available_modules[@]"; do # Iterate over all filesystem modules
          [[ "$l_mod_name" =~ overlay ]] && l_mod_name="$l_mod_name:-2"
          if grep -Pq -- "\b$l_mod_name\b" <<< "${a_ignore[*]"; then
            a_excluded+=(" - Kernel module: \"$l_mod_name\"")
          else
            f_module_chk
          fi
        done
        [ "${#a_excluded[@]}" -gt 0 ] && printf '%s\n' "" -- INFO -- \
"The following intentionally skipped" \
"${a_excluded[@]}"
        if [ "${#a_output2[@]}" -le 0 ]; then
          printf '%s\n' "" -- No unused filesystem kernel modules are enabled "${a_output[@]}" ""
        else
          printf '%s\n' "" -- Audit Result: --" " ** REVIEW the following **" "${a_output2[@]}"
          [ "${#a_output[@]}" -gt 0 ] && printf '%s\n' "" -- Correctly set: --" "${a_output[@]}" ""
        fi
      }
}

```

**WARNING:** disabling or denylisting filesystem modules that are in use on the system may be FATAL. It is extremely important to thoroughly review this list.

## Remediation:

- **IF** - the module is available in the running kernel:

- Unload the filesystem kernel module from the kernel
- Create a file ending in **.conf** with install filesystem kernel modules **/bin/false** in the **/etc/modprobe.d/** directory
- Create a file ending in **.conf** with deny list filesystem kernel modules in the **/etc/modprobe.d/** directory

**WARNING:** unloading, disabling or denylisting filesystem modules that are in use on the system maybe FATAL. It is extremely important to thoroughly review the filesystems returned by the audit before following the remediation procedure.

*Example of unloading the **gfs2** kernel module:*

```
# modprobe -r gfs2 2>/dev/null
# rmmod gfs2 2>/dev/null
```

*Example of fully disabling the **gfs2** kernel module:*

```
# printf '%s\n' "blacklist gfs2" "install gfs2 /bin/false" >>
/etc/modprobe.d/gfs2.conf
```

## Note:

- Disabling a kernel module by modifying the command above for each unused filesystem kernel module
- The example **gfs2** must be updated with the appropriate module name for the command or example script below to run correctly.

**Below is an example Script that can be modified to use on various filesystem kernel modules manual remediation process:**

*Example Script*

```

#!/usr/bin/env bash





{
    a_output2=(); a_output3=(); l_dl="" # Initialize arrays and clear
variables
    l_mod_name="gfs2" # set module name
    l_mod_type="fs" # set module type
    l_mod_path="$(readlink -f /lib/modules/**/kernel/$l_mod_type | sort -u)"
    f_module_fix()
    {
        l_dl="y" # Set to ignore duplicate checks
        a_showconfig=() # Create array with modprobe output
        while IFS= read -r l_showconfig; do
            a_showconfig+=("$l_showconfig")
            done < <(modprobe --showconfig | grep -P --
'\b(install|blacklist)\h+'${l_mod_name//-/}_'\b')
            if lsmod | grep "$l_mod_name" &> /dev/null; then # Check if the module
is currently loaded
                a_output2+=(" - unloading kernel module: \"${l_mod_name}\"")
                modprobe -r "${l_mod_name}" 2>/dev/null; rmmod "${l_mod_name}"
2>/dev/null
            fi
            if ! grep -Pq -- '\binstall\h+'${l_mod_name//-/}_
/_'\b'\h+(\|/usr)?\|/bin\|(true|false)\b' <<< "${a_showconfig[*]}"; then
                a_output2+=(" - setting kernel module: \"${l_mod_name}\" to
\"$(readlink -f /bin/false)\")
                printf '%s\n' "install $l_mod_name $(readlink -f /bin/false)" >>
/etc/modprobe.d/"${l_mod_name}.conf
            fi
            if ! grep -Pq -- '\bblacklist\h+'${l_mod_name//-/}_'\b' <<<
"${a_showconfig[*]}"; then
                a_output2+=(" - denylisting kernel module: \"${l_mod_name}\"")
                printf '%s\n' "blacklist $l_mod_name" >>
/etc/modprobe.d/"${l_mod_name}.conf
            fi
        }
        for l_mod_base_directory in $l_mod_path; do # Check if the module exists
on the system
            if [ -d "$l_mod_base_directory/${l_mod_name//-/}/" ] && [ -n "$(ls -A
"$l_mod_base_directory/${l_mod_name//-/}/")" ]; then
                a_output3+=(" - \"${l_mod_base_directory}\"")
                [[ "$l_mod_name" =~ overlay ]] && l_mod_name="${l_mod_name::-2}"
                [ "$l_dl" != "y" ] && f_module_fix
            else
                echo -e " - kernel module: \"${l_mod_name}\" doesn't exist in
\"${l_mod_base_directory}\""
            fi
        done
        [ "${#a_output3[@]}" -gt 0 ] && printf '%s\n' "" " -- INFO --" " - module:
\"${l_mod_name}\" exists in:" "${a_output3[@]}"
        [ "${#a_output2[@]}" -gt 0 ] && printf '%s\n' "" "${a_output2[@]}" ||
printf '%s\n' "" " - No changes needed"
        printf '%s\n' "" " - remediation of kernel module: \"${l_mod_name}\"
complete" ""
    }
}

```

## References:

1. <https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=filesystem>

## CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<b><u>4.8 Uninstall or Disable Unnecessary Services on Enterprise Assets and Software</u></b> Uninstall or disable unnecessary services on enterprise assets and software, such as an unused file sharing service, web application module, or service function.			
v7	<b><u>9.2 Ensure Only Approved Ports, Protocols and Services Are Running</u></b> Ensure that only network ports, protocols, and services listening on a system with validated business needs, are running on each system.			

## 1.2 Package Management

Patch management procedures may vary widely between enterprises. Large enterprises may choose to install a local updates server that can be used in place of their distributions servers, whereas a single deployment of a system may prefer to get updates directly. Updates can be performed automatically or manually, depending on the site's policy for patch management. Organizations may prefer to test patches against their environment on a non-production system before rolling out to production.

Outdated software is vulnerable to cyber criminals and hackers. Software updates help reduce the risk to your organization. The release of software update notes often reveals the patched exploitable entry points to the public. Public knowledge of these exploits can make your organization more vulnerable to malicious actors attempting to gain entry to your system's data.

Software updates often offer new and improved features and speed enhancements.

For the purpose of this benchmark, the requirement is to ensure that a patch management process is defined and maintained, the specifics of which are left to the organization.

## 1.2.1 Configure Package Repositories

Patch management procedures may vary widely between enterprises. Large enterprises may choose to install a local updates server that can be used in place of their distributions servers, whereas a single deployment of a system may prefer to get updates directly. Updates can be performed automatically or manually, depending on the site's policy for patch management. Organizations may prefer to test patches against their environment on a non-production system before rolling out to production.

Outdated software is vulnerable to cyber criminals and hackers. Software updates help reduce the risk to your organization. The release of software update notes often reveals the patched exploitable entry points to the public. Public knowledge of these exploits can leave your organization more vulnerable to malicious actors attempting to gain access to your system's data.

**Note:** Creation of an appropriate patch management policy is left to the organization.



### *1.2.1.1 Ensure GPG keys are configured (Manual)*

**Profile Applicability:**

- Level 1 - Server
- Level 1 - Workstation

**Description:**

Most package managers implement GPG key signing to verify package integrity during installation.

**Rationale:**

It is important to ensure that updates are obtained from a valid source to protect against spoofing that could lead to the inadvertent installation of malware on the system.

## Audit:

Verify GPG keys are configured correctly for your package manager:

```
# apt-key list
```

## Note:

- **apt-key list** is deprecated. Manage keyring files in **trusted.gpg.d** instead (see apt-key(8)).
- With the deprecation of **apt-key** it is recommended to use the **Signed-By** option in **sources.list** to require a repository to pass apt-secure(8) verification with a certain set of keys rather than all trusted keys apt has configured.

## - OR -

1. Run the following script and verify GPG keys are configured correctly for your package manager:

```
#!/usr/bin/env bash

{
  for file in /etc/apt/trusted.gpg.d/*.{gpg,asc}
  /etc/apt/sources.list.d/*.{gpg,asc} ; do
    if [ -f "$file" ]; then
      echo -e "File: $file"
      gpg --list-packets "$file" 2>/dev/null | awk '/keyid/ &&
!seen[$NF]++ {print "keyid:", $NF}'
      gpg --list-packets "$file" 2>/dev/null | awk '/Signed-By:/ {print
"signed-by:", $NF}'
      echo -e
    fi
  done
}
```

2. REVIEW and VERIFY to ensure that GPG keys are configured correctly for your package manager IAW site policy.













## Remediation:

Update your package manager GPG keys in accordance with site policy.

## References:

1. NIST SP 800-53 Rev. 5: SI-2
2. <https://manpages.debian.org/stretch/apt/sources.list.5.en.html>

## CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<b><u>7.3 Perform Automated Operating System Patch Management</u></b> Perform operating system updates on enterprise assets through automated patch management on a monthly, or more frequent, basis.			
v8	<b><u>7.4 Perform Automated Application Patch Management</u></b> Perform application updates on enterprise assets through automated patch management on a monthly, or more frequent, basis.			
v7	<b><u>3.4 Deploy Automated Operating System Patch Management Tools</u></b> Deploy automated software update tools in order to ensure that the operating systems are running the most recent security updates provided by the software vendor.			
v7	<b><u>3.5 Deploy Automated Software Patch Management Tools</u></b> Deploy automated software update tools in order to ensure that third-party software on all systems is running the most recent security updates provided by the software vendor.			

## MITRE ATT&CK Mappings:

Techniques / Sub-techniques	Tactics	Mitigations
T1195, T1195.001, T1195.002	TA0001	M1051

### *1.2.1.2 Ensure package manager repositories are configured (Manual)*

**Profile Applicability:**

- Level 1 - Server
- Level 1 - Workstation

**Description:**

Systems need to have package manager repositories configured to ensure they receive the latest patches and updates.

**Rationale:**

If a system's package repositories are misconfigured important patches may not be identified or a rogue repository could introduce compromised software.

**Audit:**

Run the following command and verify package repositories are configured correctly:

```
# apt-cache policy
```













**Remediation:**

Configure your package manager repositories according to site policy.

**References:**

1. NIST SP 800-53 Rev. 5: SI-2

## CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<b><u>7.3 Perform Automated Operating System Patch Management</u></b> Perform operating system updates on enterprise assets through automated patch management on a monthly, or more frequent, basis.			
v8	<b><u>7.4 Perform Automated Application Patch Management</u></b> Perform application updates on enterprise assets through automated patch management on a monthly, or more frequent, basis.			
v7	<b><u>3.4 Deploy Automated Operating System Patch Management Tools</u></b> Deploy automated software update tools in order to ensure that the operating systems are running the most recent security updates provided by the software vendor.			
v7	<b><u>3.5 Deploy Automated Software Patch Management Tools</u></b> Deploy automated software update tools in order to ensure that third-party software on all systems is running the most recent security updates provided by the software vendor.			

## MITRE ATT&CK Mappings:

Techniques / Sub-techniques	Tactics	Mitigations
T1195, T1195.001, T1195.002	TA0001	M1051

## 1.2.2 Configure Package Updates

### *1.2.2.1 Ensure updates, patches, and additional security software are installed (Manual)*

#### **Profile Applicability:**

- Level 1 - Server
- Level 1 - Workstation

#### **Description:**

Periodically patches are released for included software either due to security flaws or to include additional functionality.

#### **Rationale:**

Newer patches may contain security enhancements that would not be available through the latest full update. As a result, it is recommended that the latest software patches be used to take advantage of the latest functionality. As with any software installation, organizations need to determine if a given update meets their requirements and verify the compatibility and supportability of any additional software against the update revision that is selected.

#### **Audit:**

Verify there are no updates or patches to install:

```
# apt update  
# apt -s upgrade
```

#### **Remediation:**

Run the following command to update all packages following local site policy guidance on applying updates and patches:

```
# apt update  
  
# apt upgrade  
- OR -  
# apt dist-upgrade
```

#### **References:**













1. NIST SP 800-53 Rev. 5: SI-2

## Additional Information:

Site policy may mandate a testing period before installation onto production systems for available updates.

- `upgrade` - is used to install the newest versions of all packages currently installed on the system from the sources enumerated in `/etc/apt/sources.list` - OR - `/etc/apt/sources.list.d/ubuntu.sources`. Packages currently installed with new versions available are retrieved and upgraded; under no circumstances are currently installed packages removed, or packages not already installed retrieved and installed. New versions of currently installed packages that cannot be upgraded without changing the install status of another package will be left at their current version. An update must be performed first so that apt knows that new versions of packages are available.
- `dist-upgrade` - in addition to performing the function of `upgrade`, also intelligently handles changing dependencies with new versions of packages; apt has a "smart" conflict resolution system, and it will attempt to upgrade the most important packages at the expense of less important ones if necessary. So, `dist-upgrade` command may remove some packages. The `/etc/apt/sources.list` - OR - `/etc/apt/sources.list.d/ubuntu.sources` file contains a list of locations from which to retrieve desired package files. See also `apt_preferences(5)` for a mechanism for overriding the general settings for individual packages.

## CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<b><u>7.3 Perform Automated Operating System Patch Management</u></b> Perform operating system updates on enterprise assets through automated patch management on a monthly, or more frequent, basis.			
v8	<b><u>7.4 Perform Automated Application Patch Management</u></b> Perform application updates on enterprise assets through automated patch management on a monthly, or more frequent, basis.			
v7	<b><u>3.4 Deploy Automated Operating System Patch Management Tools</u></b> Deploy automated software update tools in order to ensure that the operating systems are running the most recent security updates provided by the software vendor.			
v7	<b><u>3.5 Deploy Automated Software Patch Management Tools</u></b> Deploy automated software update tools in order to ensure that third-party software on all systems is running the most recent security updates provided by the software vendor.			



**MITRE ATT&CK Mappings:**

<b>Techniques / Sub-techniques</b>	<b>Tactics</b>	<b>Mitigations</b>
T1195, T1195.001	TA0005	M1051

## 1.3 Mandatory Access Control

Mandatory Access Control (MAC) provides an additional layer of access restrictions to processes on top of the base Discretionary Access Controls. By restricting how processes can access files and resources on a system the potential impact from vulnerabilities in the processes can be reduced.

**Impact:** Mandatory Access Control limits the capabilities of applications and daemons on a system, while this can prevent unauthorized access the configuration of MAC can be complex and difficult to implement correctly preventing legitimate access from occurring.

### 1.3.1 Configure AppArmor

AppArmor provides a Mandatory Access Control (MAC) system that greatly augments the default Discretionary Access Control (DAC) model. Under AppArmor MAC rules are applied by file paths instead of by security contexts as in other MAC systems. As such it does not require support in the filesystem and can be applied to network mounted filesystems for example. AppArmor security policies define what system resources applications can access and what privileges they can do so with. This automatically limits the damage that the software can do to files accessible by the calling user. The user does not need to take any action to gain this benefit. For an action to occur, both the traditional DAC permissions must be satisfied as well as the AppArmor MAC rules. The action will not be allowed if either one of these models does not permit the action. In this way, AppArmor rules can only make a system's permissions more restrictive and secure.

#### References:

1. AppArmor Documentation: <http://wiki.apparmor.net/index.php/Documentation>
2. Ubuntu AppArmor Documentation: <https://help.ubuntu.com/community/AppArmor>
3. SUSE AppArmor Documentation: <https://www.suse.com/documentation/apparmor/>

### 1.3.1.1 Ensure AppArmor is installed (Automated)

**Profile Applicability:**

- Level 1 - Server
- Level 1 - Workstation

**Description:**

AppArmor provides Mandatory Access Controls.

**Rationale:**

Without a Mandatory Access Control system installed only the default Discretionary Access Control system will be available.

**Audit:**

Run the following command to verify that **apparmor** is installed:

```
# dpkg-query -s apparmor &>/dev/null && echo "apparmor is installed"
apparmor is installed
```

Run the following command to verify that **apparmor-utils** is installed:

```
# dpkg-query -s apparmor-utils &>/dev/null && echo "apparmor-utils is installed"
apparmor-utils is installed
```

**Remediation:**







Install AppArmor.

```
# apt install apparmor apparmor-utils
```

**References:**

1. NIST SP 800-53 Rev. 5: AC-3

## CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<b>3.3 <u>Configure Data Access Control Lists</u></b> Configure data access control lists based on a user's need to know. Apply data access control lists, also known as access permissions, to local and remote file systems, databases, and applications.			
v7	<b>14.6 <u>Protect Information through Access Control Lists</u></b> Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.			

## MITRE ATT&CK Mappings:

Techniques / Sub-techniques	Tactics	Mitigations
T1068, T1068.000, T1565, T1565.001, T1565.003	TA0003	M1026

### 1.3.1.2 Ensure AppArmor is enabled in the bootloader configuration (Automated)

#### Profile Applicability:

- Level 1 - Server
- Level 1 - Workstation

#### Description:

Configure AppArmor to be enabled at boot time and verify that it has not been overwritten by the bootloader boot parameters.

**Note:** This recommendation is designed around the grub bootloader, if LILO or another bootloader is in use in your environment enact equivalent settings.

#### Rationale:

AppArmor must be enabled at boot time in your bootloader configuration to ensure that the controls it provides are not overridden.

#### Audit:

Run the following command to verify that all **linux** lines have the **apparmor=1** parameter set:

```
# grep "^s*linux" /boot/grub/grub.cfg | grep -v "apparmor=1"
```

Nothing should be returned.

Run the following command to verify that all **linux** lines have the **security=apparmor** parameter set:

```
# grep "^s*linux" /boot/grub/grub.cfg | grep -v "security=apparmor"
```

Nothing should be returned.

#### Remediation:

Edit **/etc/default/grub** and add the **apparmor=1** and **security=apparmor** parameters to the **GRUB\_CMDLINE\_LINUX=** line

```
GRUB_CMDLINE_LINUX="apparmor=1 security=apparmor"
```







Run the following command to update the **grub2** configuration:

```
# update-grub
```

#### References:

1. NIST SP 800-53 Rev. 5: AC-3

## CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<b>3.3 <u>Configure Data Access Control Lists</u></b> Configure data access control lists based on a user's need to know. Apply data access control lists, also known as access permissions, to local and remote file systems, databases, and applications.			
v7	<b>14.6 <u>Protect Information through Access Control Lists</u></b> Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.			

## MITRE ATT&CK Mappings:

Techniques / Sub-techniques	Tactics	Mitigations
T1068, T1068.000, T1565, T1565.001, T1565.003	TA0003	M1026

### *1.3.1.3 Ensure all AppArmor Profiles are in enforce or complain mode (Automated)*

#### **Profile Applicability:**

- Level 1 - Server
- Level 1 - Workstation

#### **Description:**

AppArmor profiles define what resources applications are able to access.

#### **Rationale:**

Security configuration requirements vary from site to site. Some sites may mandate a policy that is stricter than the default policy, which is perfectly acceptable. This item is intended to ensure that any policies that exist on the system are activated.

#### **Audit:**

Run the following command and verify that profiles are loaded, and are in either enforce or complain mode:

```
# apparmor_status | grep profiles
```

Review output and ensure that profiles are loaded, and in either enforce or complain mode:

```
37 profiles are loaded.  
35 profiles are in enforce mode.  
2 profiles are in complain mode.  
4 processes have profiles defined.
```

Run the following command and verify no processes are unconfined

```
# apparmor_status | grep processes
```

Review the output and ensure no processes are unconfined:

```
4 processes have profiles defined.  
4 processes are in enforce mode.  
0 processes are in complain mode.  
0 processes are unconfined but have a profile defined.
```



## Remediation:

Run the following command to set all profiles to enforce mode:

```
# aa-enforce /etc/apparmor.d/*
```

### - OR -

Run the following command to set all profiles to complain mode:







```
# aa-complain /etc/apparmor.d/*
```

**Note:** Any unconfined processes may need to have a profile created or activated for them and then be restarted.

## References:

1. NIST SP 800-53 Rev. 5: AC-3

## CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<b>3.3 <u>Configure Data Access Control Lists</u></b> Configure data access control lists based on a user's need to know. Apply data access control lists, also known as access permissions, to local and remote file systems, databases, and applications.			
v7	<b>14.6 <u>Protect Information through Access Control Lists</u></b> Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.			

## MITRE ATT&CK Mappings:

Techniques / Sub-techniques	Tactics	Mitigations
T1497	TA0005	

### 1.3.1.4 Ensure all AppArmor Profiles are enforcing (Automated)

#### Profile Applicability:

- Level 2 - Server
- Level 2 - Workstation

#### Description:

AppArmor profiles define what resources applications are able to access.

#### Rationale:

Security configuration requirements vary from site to site. Some sites may mandate a policy that is stricter than the default policy, which is perfectly acceptable. This item is intended to ensure that any policies that exist on the system are activated.

#### Audit:

Run the following commands and verify that profiles are loaded and are not in complain mode:

```
# apparmor_status | grep profiles
```

Review output and ensure that profiles are loaded, and in enforce mode:

```
34 profiles are loaded.
34 profiles are in enforce mode.
0 profiles are in complain mode.
2 processes have profiles defined.
```

Run the following command and verify that no processes are unconfined:

```
apparmor_status | grep processes
```

Review the output and ensure no processes are unconfined:

```
2 processes have profiles defined.
2 processes are in enforce mode.
0 processes are in complain mode.
0 processes are unconfined but have a profile defined.
```

#### Remediation:

Run the following command to set all profiles to enforce mode:

```
# aa-enforce /etc/apparmor.d/*
```

**Note:** Any unconfined processes may need to have a profile created or activated for them and then be restarted

#### References:

1. NIST SP 800-53 Rev. 5: AC-3

## CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<b>3.3 <u>Configure Data Access Control Lists</u></b> Configure data access control lists based on a user's need to know. Apply data access control lists, also known as access permissions, to local and remote file systems, databases, and applications.			
v7	<b>14.6 <u>Protect Information through Access Control Lists</u></b> Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.			

## MITRE ATT&CK Mappings:

Techniques / Sub-techniques	Tactics	Mitigations
T1068, T1068.000, T1565, T1565.001, T1565.003	TA0005	M1048