



## Unix Configuration Audit Compliance File Reference

This section describes the built-in functions of the Unix compliance checks and the rationale behind each setting.

This section includes the following information:

- [Unix Configuration Check Type](#)
- [Unix Configuration Keywords](#)
- [Unix Configuration Custom Items](#)
- [Built-In Checks](#)
- [Conditions](#)
- [Global Settings](#)

### Unix Configuration Check Type

All Unix compliance checks must be bracketed with the “**check\_type**” encapsulation and the “Unix” designation. [All Compliance and Audit Files](#) contains an example Unix compliance check starting with the **check\_type** setting for “Unix” and is finished by the “**</check\_type>**” tag.

This is required to differentiate **.audit** files intended for Windows (or other platforms) compliance audits.

**Note:** The file is read over SSH into a memory buffer on the Nessus server, and then the buffer is processed to check for compliance/non-compliance.

### Unix Configuration Keywords

The following table indicates how each keyword in the Unix compliance checks can be used.

Keyword	Example Usage and Supported Settings
attr	This keyword is used in conjunction with FILE_CHECK and FILE_CHECK_NOT to audit the file attributes associated with a file. Please refer to the <code>chattr(1)</code> man page for details on configuring the file attributes of a file.



Keyword	Example Usage and Supported Settings
check_option	This keyword is used to allow a response to be NULL and still pass. Example: check_option: CAN_BE_NULL
check_unevenness	This keyword is used with FILE_CHECK and FILE_CHECK_NOT. File permissions are considered uneven if the <i>group</i> or <i>other</i> have additional permissions than owner or if <i>other</i> has additional permissions than <i>group</i> .
cmd	This keyword is required for use with CMD_EXEC to execute remote commands for the purpose of auditing a wide variety of items.
description	This keyword provides the ability to add a brief description of the check that is being performed. It is strongly recommended that the <b>description</b> field be unique and no distinct checks have the same description field. Tenable uses this field to automatically generate a unique plugin ID number based on the description field.  Example:  description: "Permission and ownership check for /etc/at.allow"
dont_echo_cmd	This keyword is used with "CMD_EXEC" Unix compliance check audits and tells the audit to omit the actual command run by the check from the output. Only the command's results are displayed.  Example:  dont_echo_cmd: YES
except	This keyword is used to exclude certain users, services and files from the check.  Example:  except: "guest"  Multiple user accounts can be piped together.  Example:



Keyword	Example Usage and Supported Settings
	<code>except: "guest"   "guest1"   "guest2"</code>
expect	<p>This keyword is used in combination with regex. It provides the ability to look for specific values within files.</p> <p>Example:</p> <pre>&lt;custom_item&gt; system: "Linux" type: FILE_CONTENT_CHECK description: "This check reports a problem when the log level setting in the sendmail.cf file is less than the value set in your security policy." file: "sendmail.cf" regex: ".*LogLevel=.*" expect: ".*LogLevel=9" &lt;/custom_item&gt;</pre>
file	<p>This keyword is used to describe the absolute or relative path of a file to be checked for permissions and ownership settings.</p> <p>Examples:</p> <pre>file: "/etc/inet/inetd.conf" file: "~/inetd.conf"</pre> <p>The file value can also be a glob.</p> <p>Example:</p> <pre>file: "/var/log/*"</pre> <p>This feature is particularly useful when all the files within a given directory need to be audited for permissions or contents using FILE_CHECK, FILE_CONTENT_CHECK, FILE_CHECK_NOT, or FILE_CONTENT_CHECK_NOT.</p>
file_required	<p>This keyword is used with FILE_CHECK, FILE_CHECK_NOT, FILE_CONTENT_CHECK, and FILE_CONTENT_CHECK NOT. The file_</p>



Keyword	Example Usage and Supported Settings
	required field can be set to specify if the audited file is required to be present or not. If this option is not set, it is assumed it is required.
file_type	<p>This keyword describes the type of file that is searched for. The following is the list of supported file types.</p> <ul style="list-style-type: none"><li>• b - block (buffered) special</li><li>• c - character (unbuffered) special</li><li>• d - directory</li><li>• p - named pipe (FIFO)</li><li>• f - regular file</li></ul> <p>Example:</p> <pre>file_type: "f"</pre> <p>One or more types of file types can be piped together in the same string.</p> <p>Example:</p> <pre>file_type: "c b"</pre>
gid	<p>This keyword is used with FILE_CHECK and FILE_CHECK_NOT to audit the numeric group ID associated with a file. Example: 500</p>
group	<p>This keyword is used to specify the group of a file; it is always used in conjunction with file keyword. The <b>group</b> keyword can have a value of "none" that helps with searching for files with no owner.</p> <p>Example:</p> <pre>group: "root"</pre> <p>Group can also be specified with a logical "OR" condition using the following syntax:</p> <pre>group: "root"    "bin"    "sys"</pre>



Keyword	Example Usage and Supported Settings
ignore	<p>This keyword tells the check to ignore designated files from the search. This keyword is available for the FILE_CHECK, FILE_CHECK_NOT, FILE_CONTENT_CHECK, and FILE_CONTENT_CHECK_NOT check types.</p> <p>Examples:</p> <pre># ignore single file ignore: "/root/test/2"</pre> <pre># ignore certain files from a directory ignore: "/root/test/foo*"</pre> <pre># ignore all files in a directory ignore: "/root/test/*"</pre>
info	<p>This keyword is used to add a more detailed description to the check that is being performed such as a regulation, URL, corporate policy or a reason why the setting is required. Multiple info fields can be added on separate lines to format the text as a paragraph. There is no preset limit to the number of info fields that can be used.</p> <p>Example:</p> <pre>info: "ref. CIS_AIX_Benchmark_v1.0.1.pdf ch 1, pg 28-29."</pre>
levels	<p>This keyword is used in conjunction with CHKCONFIG and is used to specify the run levels for which a service is required to be running. All the run levels must be described in a single string. For example, if service "sendmail" is required to be running at run level 1, 2 and 3, then the corresponding levels value in the CHKCONFIG check would be:</p> <pre>levels: "123"</pre>



Keyword	Example Usage and Supported Settings
json_transform	This keyword is used with FILE_CONTENT_CHECK and FILE_CONTENT_CHECK_NOT to evaluate JSON formatted data.
mask	<p>This keyword is the opposite of mode where one can specify permissions that should not be available for a particular user, group or other member. Unlike <b>mode</b> that checks for an exact permission value, <b>mask</b> audits are broader and will check if a file or directory is at a level that is equal to, or more secure than, what is specified by the <b>mask</b>. (Where mode may fail a file with a permission of 640 as not matching an audit expecting a value of 644, <b>mask</b> will see that 640 is “more secure” and will pass the audit as successful.)</p> <p>Example:</p> <p>mask: 022</p> <p>This would specify any permission is OK for owner and no write permissions for group and other member. A <b>mask</b> value of “7” would mean no permissions for that particular owner, group or other member.</p>
md5	<p>This keyword is used in FILE_CHECK and FILE_CHECK_NOT to make sure the MD5 of a file is actually set to whatever the policy sets.</p> <p>Example:</p> <pre>&lt;custom_item&gt; type: FILE_CHECK description: "/etc/passwd has the proper md5 set" required: YES file: "/etc/passwd" md5: "ce35dc081fd848763cab2cfd442f8c22" &lt;/custom_item&gt;</pre>
min_occurrences	<p>This keyword specifies the minimum number of specific values in <a href="#">FILE_CONTENT_CHECK</a> files.</p> <p>Example:</p>



Keyword	Example Usage and Supported Settings
	<code>min_occurrences: "3"</code>
mode	<p>This keyword describes the set of permissions for a file/folder under consideration. The mode keyword can be represented in string or octal format.</p> <p>Examples:</p> <pre>mode: "-rw-r--r--" mode: "644" mode: "7644"</pre>
name	<p>This keyword is used to identify process name in PROCESS_CHECK.</p> <p>Example:</p> <pre>name: "syslogd"</pre>
not_expect	<p>This keyword is used in combination with regex. It provides the ability to look for specific failing values in FILE_CONTENT_CHECK.</p>
not_regex	<p>This keyword is used with MACOSX_DEFAULTS_READ to evaluate all items found do not match the regex specified.</p>
operator	<p>This keyword is used in conjunction with RPM_CHECK and PKG_CHECK to specify the condition to pass or fail a check based on the version of the installed RPM package. It can take the following values:</p> <ul style="list-style-type: none"><li>• lt (less than)</li><li>• lte (less than or equal)</li><li>• gte (greater than equal)</li><li>• gt (greater than)</li><li>• eq (equal)</li></ul> <p>Example:</p>



Keyword	Example Usage and Supported Settings
	<code>operator: "lt"</code>
owner	<p>This keyword is used to specify the owner of a file; it is always used in conjunction with <code>file</code> keyword. The owner keyword can have a value of “none” that helps with searching for files with no owner.</p> <p>Example:</p> <p><code>owner: "root"</code></p> <p>Ownership can also be specified with a logical “OR” condition using the following syntax:</p> <p><code>owner: "root"    "bin"    "adm"</code></p>
pkg	<p>This keyword is used with <code>PKG_CHECK</code> to evaluate packages installed on a SunOS system. Example: <code>pkg: "SUNWcrman"</code></p>
ports	<p>This keyword is used with <code>AUDIT_ALLOWED_OPEN_PORTS</code> and <code>AUDIT_DENIED_OPEN_PORTS</code> to specify a single port, comma separated list, or regex range. The ports tag used with <code>AUDIT_PROCESS_ON_PORT</code> is used with a single port. Example: <code>ports: "80"</code>, <code>ports: "80, 443"</code>, <code>ports: "2[1-9]"</code></p>
port_type	<p>This keyword is used in with <code>AUDIT_ALLOWED_OPEN_PORTS</code>, <code>AUDIT_DENIED_OPEN_PORTS</code>, and <code>AUDIT_PROCESS_ON_PORT</code> to specify TCP or UDP. Example: <code>port_type: TCP</code> or <code>port_type: UDP</code></p>
reference	<p>This keyword provides a way to include cross-references in the <code>.audit</code>. The format is “<code>ref ref-id1,ref ref-id2</code>”.</p> <p>Example:</p> <p><code>reference: "CAT CAT II,800-53 IA-5,8500.2 IAIA-1,8500.2 IAIA-2,8500.2 IATS-1,8500.2 IATS-2"</code></p>
regex	<p>This keyword enables searching a file to match for a particular regex expression.</p> <p>Example:</p>





Keyword	Example Usage and Supported Settings
	<p>regex: ".*LogLevel=9\$"</p> <p>The following meta-characters require special treatment: + \ * ( ) ^</p> <p>Escape these characters out twice with two backslashes "\\" or enclose them in square brackets "[]" if you wish for them to be interpreted literally. Other characters such as the following need only a single backslash to be interpreted literally: . ? " ' "</p> <p>This has to do with the way that the compiler treats these characters.</p>
required	<p>This keyword is used to specify if the audited item is required to be present or not on the remote system. For example, if <b>required</b> is set to "NO" and the check <b>type</b> is "FILE_CHECK", then the check will pass if the file exists and permissions are as specified in the .audit file or if the file does not exist. On the other hand, if <b>required</b> was set to "YES", the above check would fail.</p>
rpm	<p>This keyword is used to specify the RPM to look for when used in conjunction with RPM_CHECK.</p> <p>Example:</p> <pre>&lt;custom_item&gt; type: RPM_CHECK description: "Make sure that the Linux kernel is BELOW version 2.6.0" rpm: "kernel-2.6.0-0" operator: "lt" required: YES &lt;/custom_item&gt;</pre>
search_ locations	<p>This keyword can be used to specify searchable locations within a file system.</p> <p>Example:</p> <p>search_locations: "/bin"</p>



Keyword	Example Usage and Supported Settings
	<p>Multiple search locations can be piped together.</p> <p>Example:</p> <pre>search_locations: "/bin"   "/etc/init.d"   "/etc/rc0.d"</pre>
see_also	<p>This keyword allows to include links to a reference.</p> <p>Example:</p> <p>see_also:</p> <p><a href="https://benchmarks.cisecurity.org/tools2/linux/CIS_Redhat_Linux_5_Benchmark_v2.0.0.pdf">"https://benchmarks.cisecurity.org/tools2/linux/CIS_Redhat_Linux_5_Benchmark_v2.0.0.pdf"</a></p>
service	<p>This keyword is used in conjunction with CHKCONFIG, XINETD_SVC and SVC_PROP and is used to specify the service that is being audited.</p> <p>Example:</p> <pre>&lt;custom_item&gt; type: CHKCONFIG description: "2.1 Disable Standard Services – Check if cups is disabled" service: "cups" levels: "123456" status: OFF &lt;/custom_item&gt;</pre>
severity	<p>In any test, <b>&lt;item&gt;</b> or <b>&lt;custom_item&gt;</b>, a “<b>severity</b>” flag can be added and set to “LOW”, “MEDIUM”, or “HIGH”. By default, non-compliant results show up as “high”.</p> <p>Example:</p> <pre>severity: MEDIUM</pre>
solution	<p>This keyword provides a way to include “Solution” text if available.</p> <p>Example:</p>



Keyword	Example Usage and Supported Settings
	<code>solution: "Remove this file, if its not required"</code>
status	<p>This keyword is used in PROCESS_CHECK, CHKCONFIG and XINETD_SVC to determine if a service that is running on a given host should be running or disabled. The status keyword can take 2 values: "ON" or "OFF".</p> <p>Example:</p> <p><code>status: ON</code></p> <p><code>status: OFF</code></p>
system	<p>This keyword specifies the type of system the check is to be performed on.</p> <div><p>Note: The "system" keyword is only applicable to "custom_item" checks, not built-in "item" checks.</p></div> <p>The available values are the ones returned by the "uname" command on the target OS. For example, on Solaris the value is "SunOS", on macOS it is "Darwin", on FreeBSD it is "FreeBSD", etc.</p> <p>Example:</p> <p><code>system: "SunOS"</code></p>
timeout	<p>This keyword is used in conjunction with CMD_EXEC and specifies, in seconds, the amount of time that the specified command will be allowed to run before it times out. This keyword is useful in cases where a particular command, such as the Unix "find" command, requires extended periods of time to complete. If this keyword is not specified, the default timeout for CMD_EXEC audits is five minutes.</p> <p>Example:</p> <p><code>timeout: "600"</code></p>
type	<p>CHKCONFIG</p> <p>CMD_EXEC</p>



Keyword	Example Usage and Supported Settings
	FILE_CHECK  FILE_CHECK_NOT  FILE_CONTENT_CHECK  FILE_CONTENT_CHECK_NOT  GRAMMAR_CHECK  PKG_CHECK  PROCESS_CHECK  RPM_CHECK  SVC_PROP  XINETD_SVC
uid	This keyword is used with FILE_CHECK and FILE_CHECK_NOT to audit the numeric user ID associated with a file. Example: 0
value	<p>The <b>value</b> keyword is useful to check if a setting on the system confirms to the policy value.</p> <p>Example:</p> <p>value: "90..max"</p> <p>The value keyword can be specified as a range [number..max]. If the value lies between the specified number and "max", the check will pass.</p>
xsl_stmt	This keyword is used with AUDIT_XML to audit XML data with the use of XSL transforms. The xsl_stmt tag can be multiline or multiple individual tags.

## Unix Configuration Custom Items

A custom item is a complete check defined on the basis of the keywords defined above. This section contains a list of custom items. Each check starts with a "<custom\_item>" tag and ends with



“</custom\_item>”. Enclosed within the tags are lists of one or more keywords that are interpreted by the compliance check parser to perform the checks.

**Tip:** Custom audit checks may use “</custom\_item>” and “</item>” interchangeably for the closing tag.

This section includes the following information:

- [AUDIT\\_XML](#)
- [AUDIT\\_ALLOWED\\_OPEN\\_PORTS](#)
- [AUDIT\\_DENIED\\_OPEN\\_PORTS](#)
- [AUDIT\\_PROCESS\\_ON\\_PORT](#)
- [BANNER\\_CHECK](#)
- [CHKCONFIG](#)
- [CMD\\_EXEC](#)
- [FILE\\_CHECK](#)
- [FILE\\_CHECK\\_NOT](#)
- [FILE\\_CONTENT\\_CHECK](#)
- [FILE\\_CONTENT\\_CHECK\\_NOT](#)
- [FIND\\_CMD](#)
- [GRAMMAR\\_CHECK](#)
- [MACOSX\\_DEFAULTS\\_READ](#)
- [MACOSX\\_OSASCRIP](#)
- [PKG\\_CHECK](#)
- [PROCESS\\_CHECK](#)
- [RPM\\_CHECK](#)
- [SVC\\_PROP](#)
- [XINETD\\_SVC](#)



## AUDIT\_XML

The “AUDIT\_XML” audit check allows you to examine and audit the contents of an XML file by first applying XSL transforms, extracting relevant data, and then determine compliance based on the regex, expect, and not\_expect keywords. The check consists of four or more keywords, keywords type, description file, and xsl\_stmt directives (mandatory), which are followed by regex, expect, or not\_expect keywords to audit the content.

**Tip:** For information about the parameters commonly found in Unix custom items, see [Unix Configuration Keywords](#).

## Example

```
<custom_item>
type: AUDIT_XML
description: "1.14 - Ensure Oracle Database persistence plugin is set correctly -
'DatabasePersistencePlugin'"
file: "/opt/jboss-5.0.1.GA/server/all/deploy/ejb2-timer-service.xml"
xsl_stmt: "<xsl:template match=\"server\">"
xsl_stmt: "DatabasePersistencePlugin = <xsl:value-of select=\"/server/mbean
[@code='org.jboss.ejb.tx.timer.DatabasePersistencePolicy']/attribute
[@name='DatabasePersistencePlugin']/text()\"/>"
xsl_stmt: "</xsl:template>"
regex: "DatabasePersistencePlugin = .+"
not_expect: "org.jboss.ejb.tx.timer.GeneralPurposeDatabasePersistencePlugin"
</custom_item>
```

Note that the file keyword accepts wildcards. For example:

```
<custom_item>
type: AUDIT_XML
description: "1.14 - Ensure Oracle Database persistence plugin is set correctly -
'DatabasePersistencePlugin'"
file: "/opt/jboss-5.0.1.GA/server/all/deploy/ejb2-*.xml"
xsl_stmt: "<xsl:template match=\"server\">"
xsl_stmt: "DatabasePersistencePlugin = <xsl:value-of select=\"/server/mbean
[@code='org.jboss.ejb.tx.timer.DatabasePersistencePolicy']/attribute
[@name='DatabasePersistencePlugin']/text()\"/>"
```



```
xsl_stmt: "</xsl:template>"
regex: "DatabasePersistencePlugin = .+"
not_expect: "org.jboss.ejb.txtimer.GeneralPurposeDatabasePersistencePlugin"
</custom_item>
```

## AUDIT\_ALLOWED\_OPEN\_PORTS

The “AUDIT\_ALLOWED\_OPEN\_PORTS” audit check is used to define an open port based policy. Users can specify which ports can be open on a given system, and if any other ports apart from the specified ports are open, then it will be considered a failure. A comma separates more than one port, and the port value could also be a regex.

**Tip:** For information about the parameters commonly found in Unix custom items, see [Unix Configuration Keywords](#).

```
<custom_item>
type: AUDIT_ALLOWED_OPEN_PORTS
description: "Only allow port 80,443, 808[0-9] open on Web Server"
port_type: TCP
ports: "80,443, 808[0-9]"
</custom_item>
```

## AUDIT\_DENIED\_OPEN\_PORTS

The “AUDIT\_DENIED\_OPEN\_PORTS” audit check is used to define an open port based policy. Users can specify which ports cannot be open a given system, and if those ports open, then it will be considered a failure. A comma separates more than one port, and the port value could also be a regex.

**Tip:** For information about the parameters commonly found in Unix custom items, see [Unix Configuration Keywords](#).

```
<custom_item>
type: AUDIT_DENIED_OPEN_PORTS
description: "Do not allow port 23 (telnet) to be open"
port_type: TCP
ports: "23"
```



```
</custom_item>
```

## AUDIT\_PROCESS\_ON\_PORT

The “AUDIT\_PROCESS\_PORT” check allows users to verify whether the process running on a port is indeed an authorized process and not a backdoor process hiding in plain sight. More than one allowed process can be separated by a “|” (pipe) character.

**Tip:** For information about the parameters commonly found in Unix custom items, see [Unix Configuration Keywords](#).

```
<custom_item>
type: AUDIT_PROCESS_ON_PORT
description: "Make sure 'sshd' is running on port 22"
port_type: TCP
ports: "22"
name: "sshd|launchd"
</custom_item>
```

## BANNER\_CHECK

This policy item checks if the file content matches the content provided by normalizing the values to use common newline, escaping patterns, and stripping white space from the beginning and end of policy text.

### Usage

```
<custom_item>
type: BANNER_CHECK
description: ["description"]
file: ["path to file"]
content: ["banner content"]
is_substring: [YES|NO]
</custom_item>
```

The following are descriptions of the keywords:





- **file:** The path and filename for the banner to reside in.
- **content:** What you expect the banner to display. New lines in the banner are represented by adding an `\n` where the new line should be placed.
- **is\_substring:** An optional flag that supports the possibility of location specific information being placed in a banner. If set to YES, the expected banner can be a substring of the file content, and not require a full match.

**Tip:** For information about the parameters commonly found in Unix custom items, see [Unix Configuration Keywords](#).

## Example

```
<custom_item>
type      : BANNER_CHECK
description : "Banner is configured in /etc/issue"
file      : "/etc/issue"
content   : "*** No Unauthorized Access ***"
</custom_item>
```

## CHKCONFIG

The “CHKCONFIG” audit check allows interaction with the “**chkconfig**” utility on the remote Red Hat system being audited. This check consists of five mandatory keywords: **type**, **description**, **service**, **levels**, and **status**. This check also has the optional keyword “**check\_option**” to allow NULL responses. Example: **check\_option: CAN\_BE\_NULL**.

**Note:** The CHKCONFIG audit only works on Red Hat systems or a derivative of a Red Hat system such as Fedora.

**Tip:** For information about the parameters commonly found in Unix custom items, see [Unix Configuration Keywords](#).

## Example



```
<custom_item>
type: CHKCONFIG
description: "Make sure that xinetd is disabled"
service: "xinetd"
levels: "123456"
status: OFF
</custom_item>
```

## CMD\_EXEC

It is possible to execute commands on the remote host and to check that the output matches what is expected. This kind of check should be used with extreme caution, as it is not always portable across different flavors of Unix.

The `quiet` keyword tells Nessus not to show the output of the command that failed. It can be set to “YES” or “NO”. By default, it is set to “NO” and the result of the command is displayed. Similarly, the `dont_echo_cmd` keyword limits the results by outputting the command results, but not the command itself.

The `nosudo` keyword lets the user tell Nessus not to use `sudo` to execute the command by setting it to “YES”. By default, it is set to “NO” and `sudo` is always used when configured to do so.

**Tip:** For information about the parameters commonly found in Unix custom items, see [Unix Configuration Keywords](#).

## Example

```
<custom_item>
type: CMD_EXEC
description: "Make sure that we are running FreeBSD 4.9 or higher"
cmd: "uname -a"
timeout: "600"
expect: "FreeBSD (4\.(9|[1-9][0-9])|[5-9]\.)"
dont_echo_cmd: YES
</custom_item>
```

## FILE\_CHECK



Unix compliance audits typically test for the existence and settings of a given file. The “FILE\_CHECK” audit uses four or more keywords to allow the specification of these checks. The keywords **type**, **description**, and **file** are mandatory and are followed by one or more checks. Current syntax supports checking for owner, group and file permissions.

It is possible to use globs in FILE\_CHECK (e.g., `/var/log/*`). However, note that globs will only be expanded to files, not to directories. If a glob is specified and one or more matched files must be ignored from the search, use the “ignore” keyword to specify the files to ignore.

The allowed keywords are:

- `uid`: Numeric User ID (e.g., 0)
- `gid`: Numeric Group ID (e.g., 500)
- `check_unevenness`: YES
- `system`: System type (e.g., Linux)
- `description`: Text description of the file check
- `file`: Full path and file to check (e.g., `/etc/sysconfig/sendmail`)
- `required`: Specifies whether a check match is required or not (e.g., YES or NO). If this option is not set, it is assumed it is required.
- `file_required`: Specifies whether a file is required to be present or not (e.g., YES or NO). If this option is not set, it is assumed it is required.
- `owner`: Owner of the file (e.g., root)
- `group`: Group owner of the file (e.g., bin)
- `mode`: Permission mode (e.g., 644)
- `mask`: File umask (e.g., 133)
- `md5`: The MD5 hash of a file (e.g., 88d3dbe3760775a00b900a850b170fcd)
- `ignore`: A file to ignore (e.g., `/var/log/secure`)
- `attr`: A file attribute (e.g., `----i-----`)

File permissions are considered uneven if the “group” or “other” have additional permissions than “owner” or if “other” has additional permissions than “group”.



**Tip:** For information about the parameters commonly found in Unix custom items, see [Unix Configuration Keywords](#).

## Examples:

```
<custom_item>
system: "Linux"
type: FILE_CHECK
description: "Permission and ownership check for /etc/default/cron"
file: "/etc/default/cron"
owner: "bin"
group: "bin"
mode: "-r--r--r--"
</custom_item>
```

```
<custom_item>
system: "Linux"
type: FILE_CHECK
description: "Permission and ownership check for /etc/default/cron"
file: "/etc/default/cron"
owner: "bin"
group: "bin"
mode: "444"
</custom_item>
```

```
<custom_item>
system: "Linux"
type: FILE_CHECK
description: "Make sure /tmp has its sticky bit set"
file: "/tmp"
mode: "1000"
</custom_item>
```

```
<custom_item>
type: FILE_CHECK
description: "/etc/passwd has the proper md5 set"
required: YES
```



```
file: "/etc/passwd"
md5: "ce35dc081fd848763cab2cfd442f8c22"
</custom_item>
```

```
<custom_item>
type: FILE_CHECK
description: "Ignore maillog in the file mode check"
required: YES
file: "/var/log/m*"
mode: "1000"
ignore: "/var/log/maillog"
</custom_item>
```

## FILE\_CHECK\_NOT

The “FILE\_CHECK\_NOT” audit consists of three or more keywords. The keywords **type**, **description**, and **file** are mandatory and are followed by one or more checks. Current syntax supports checking for owner, group and file permissions. Similar to the FILE\_CHECK audit, the “**ignore**” keyword can be used to ignore one or more files if a file glob is specified.

This function is the opposite of FILE\_CHECK. A policy fails if a file does not exist or if its mode is the same as the one defined in the check itself.

It is possible to use globs in FILE\_CHECK\_NOT (e.g., `/var/log/*`). However, note that globs will only be expanded to files, not to directories

The allowed keywords are:

- `uid`: Numeric User ID (e.g., 0)
- `gid`: Numeric Group ID (e.g., 500)
- `check_uneveness`: YES
- `system`: System type (e.g., Linux)
- `description`: Text description of the file check
- `file`: Full path and file to check (e.g., `/etc/sysconfig/sendmail`)



- `file_required`: File is required to be present or not. If this option is not set, it is assumed it is required.
- `owner`: Owner of the file (e.g., `root`)
- `group`: Group owner of the file (e.g., `bin`)
- `mode`: Permission mode (e.g., `644`)
- `mask`: File umask (e.g., `133`)
- `md5`: The MD5 hash of a file (e.g., `88d3dbe3760775a00b900a850b170fcd`)
- `ignore`: A file to ignore (e.g., `/var/log/secure`)
- `attr`: A file attribute (e.g., `----i-----`)

File permissions are considered uneven if the “group” or “other” have additional permissions than “owner” or if “other” has additional permissions than “group”.

**Tip:** For information about the parameters commonly found in Unix custom items, see [Unix Configuration Keywords](#).

## Examples

```
<custom_item>
type: FILE_CHECK_NOT
description: "Make sure /bin/bash does NOT belong to root"
file: "/bin/bash"
owner: "root"
</custom_item>
```

```
<custom_item>
type: FILE_CHECK_NOT
description: "Make sure that /usr/bin/ssh does NOT exist"
file: "/usr/bin/ssh"
</custom_item>
```

```
<custom_item>
type: FILE_CHECK_NOT
```



```
description: "Make sure /root is NOT world writeable"
file: "/root"
mode: "0777"
</custom_item>
```

## FILE\_CONTENT\_CHECK

As with testing the existence and settings of a file, the content of text files can also be analyzed. Regular expressions can be used to search one or more locations for existing content. Use the “**ignore**” keyword to ignore one or more files from the specified search location or locations.

The **string\_required** field can be set to specify if the audited string being searched for is required to be present or not. If this option is not set, it is assumed it is required. The **file\_required** field can be set to specify if the audited file is required to be present or not. If this option is not set, it is assumed it is required. Use the "json\_transform" tag to evaluate specific JSON-formatted data within a file.

**Tip:** For information about the parameters commonly found in Unix custom items, see [Unix Configuration Keywords](#).

## Examples

```
<custom_item>
system: "Linux"
type: FILE_CONTENT_CHECK
description: "This check reports a problem when the log level setting in the
sendmail.cf file is less than the value set in your security policy."
file: "sendmail.cf"
regex: ".*LogLevel=.*$"
expect: ".*LogLevel=9"
</custom_item>
```

```
<custom_item>
system: "Linux"
type: FILE_CONTENT_CHECK
file: "sendmail.cf"
search_locations: "/etc:/etc/mail:/usr/local/etc/mail/"
```



```
regex: ".*PrivacyOptions=.*"
expect: ".*PrivacyOptions=.*,novrfy,.*"
</custom_item>
```

```
<custom_item>
#System: "Linux"
type: FILE_CONTENT_CHECK
description: "FILE_CONTENT_CHECK"
file: "/root/test2/foo*"
# ignore single file
ignore: "/root/test/2"
# ignore all files in a directory
ignore: "/root/test/*"
#ignore certain files from a directory
ignore: "/root/test/foo*"
regex: "FOO"
expect: "FOO1"
file_required: NO
string_required: NO
</custom_item>
```

Add a “~” to a file parameter to configure FILE\_CONTENT\_CHECK to scan a user’s home directories for non-compliant content.

```
<custom_item>
system: "Linux"
type: FILE_CONTENT_CHECK
description: "Check all user home directories"
file: "~/.rhosts"
ignore: "/.foo"
regex: "\\+"
expect: "\\+"
</custom_item>
```

## FILE\_CONTENT\_CHECK\_NOT

This audit examines the contents of a file for a match with the regex description in the **regex** field. This function negates FILE\_CONTENT\_CHECK. That is, a policy fails if the regex does match in the file. Use the “**ignore**” keyword to ignore one or more files from the specified search location(s).





This policy item checks if the file contains the regular expression **regex** and that this expression does not match **expect**.

Both **regex** and **expect** must be specified in this check.

**Tip:** For information about the parameters commonly found in Unix custom items, see [Unix Configuration Keywords](#).

## Example

```
<custom_item>
type: FILE_CONTENT_CHECK_NOT
description: "Make sure NIS is not enabled on the remote host by making sure that '+::'
is not in /etc/passwd"
file: "/etc/passwd"
regex: "^\\+::"
expect: "^\\+::"
file_required: NO
string_required: NO
</custom_item>
```

## GRAMMAR\_CHECK

The “GRAMMAR\_CHECK” audit check examines the contents of a file and matches a loosely defined grammar (made up of one or multiple regex statements). If one line in the target file does not match any of the regex statements, then the test will fail.

**Tip:** For information about the parameters commonly found in Unix custom items, see [Unix Configuration Keywords](#).

## Example

```
<custom_item>
type: GRAMMAR_CHECK
description: "Check /etc/securetty contents are OK."
file: "/etc/securetty"
regex: "console"
regex: "vc/1"
```



```
regex: "vc/2"
regex: "vc/3"
regex: "vc/4"
regex: "vc/5"
regex: "vc/6"
regex: "vc/7"
</custom_item>
```

## MACOSX\_DEFAULTS\_READ

The "MACOSX\_DEFAULTS\_READ" audit check examines the default system values on macOS. This check behaves differently if you set certain properties.

- If you set `plist_user` to `all`, all user settings are audited, otherwise the specified user setting is audited. In other words, you can only audit all users or one specific user.
- If you set `byhost` to `YES` in addition to the `plist_user` property being set, the following query runs:

```
/usr/bin/defaults -currentHost read /Users/foo/Library/Preferences/ByHost/plist_name plist_item
```

- If you do not set `byhost` and you set `plist_user`, the following query runs:

```
/usr/bin/defaults -currentHost read /Users/foo/Library/Preferences/plist_name plist_item
```

- If you do not set `byhost` or `plist_user`, the following query runs:

```
/usr/bin/defaults -currentHost read plist_name plist_item
```

The following properties are supported:

Property	Description	Accepted Value
<code>plist_name</code>	The plist that you want to query.	Example: <code>com.apple.digihub</code>



plist_item	The plist item that you want to audit.	Example: com.apple.digihub.blank.cd.appeared
plist_option	If you set this property to CANNOT_BE_NULL, the check fails if the setting being audited is not set.	CANNOT_BE_NULL
byhost	If you set this property YES, the query results are generated by host.	YES
not_regex	Ensures that all found items do no match a specified regex.	Example: not_regex: ".* = 6"
managed_path	Specifies a custom path that contains the plist.	Example: managed_path: "/Library/Managed\ Preferences/"

**Tip:** For information about the parameters commonly found in Unix custom items, see [Unix Configuration Keywords](#).

## Examples

### Example 1:

```
<custom_item>
system: "Darwin"
type: MACOSX_DEFAULTS_READ
description: "Automatic actions must be disabled for blank CDs - 'action=1;'"
plist_user: "all"
plist_name: "com.apple.digihub"
plist_item: "com.apple.digihub.blank.cd.appeared"
regex: "\\s*action\\s*=\\s*1;"
plist_option: CANNOT_BE_NULL
</custom_item>
```



```
<custom_item>
system: "Darwin"
type: MACOSX_DEFAULTS_READ
description: "System must have a password-protected screen saver configured to DoD"
plist_user: "all"
plist_name: "com.apple.screensaver"
byhost: YES
plist_item: "idleTime"
regex: "[A-Za-z0-9_-]+\s*=\s*(900|[2-8][0-9][0-9]|1[8-9][0-9])$"
plist_option: CANNOT_BE_NULL
</custom_item>

<custom_item>
system: "Darwin"
type: MACOSX_DEFAULTS_READ
description: "System must have a password-protected screen saver configured to DoD"
plist_name: "com.apple.screensaver"
plist_item: "idleTime"
regex: "[A-Za-z0-9_-]+\s*=\s*(900|[2-8][0-9][0-9]|1[8-9][0-9])$"
plist_option: CANNOT_BE_NULL
</custom_item>
```

#### Example 2:

```
<custom_item>
system : "Darwin"
type : MACOSX_DEFAULTS_READ
description : "Use a custom managed_path"
plist_name : "com.apple.Terminal"
plist_item : "HasMigratedDefaults"
regex : "1"
managed_path : "/Library/Managed\ Preferences/"
</custom_item>
```

## MACOSX\_OSASCRIP

The **MACOSX\_OSASCRIP** audit check uses the `osascript` command to return configured payload data.



Where `payload_key` and `payload_type` are specified in the check parameters, this check type sends the following command to the target:

```
echo "$.NSUserDefaults.alloc.initWithSuiteName(PAYLOAD_TYPE).objectForKey(PAYLOAD_KEY).js" | /usr/bin/osascript -l JavaScript
```

## Usage

```
<custom_item>
type          : MACOSX_OSASCRIP
description   : ["description"]
expect       : ["response to evaluate"]
payload_key   : ["string value"]
payload_type  : ["string value"]
(optional) required : [YES|NO]
</custom_item>
```

The following properties are supported:

- `expect` – The response to evaluate. Valid text includes strings and regex.
  - Examples: "false", "true", "^0\$"
- `payload_key` – The key name referenced in ``objectForKey().js``.
  - Examples: "AutoBackup", "ShowSystemServices"
- `payload_type` – The type referenced in ``initWithSuiteName()``.
  - Examples: "com.apple.TimeMachine", "com.apple.locationmenu"
- `required` – Defaults to yes. Setting no results in a PASS result if the response is empty.

## Examples

```
<custom_item>
type          : MACOSX_OSASCRIP
description   : "Require Alpha Numeric Password Policy"
expect       : "true"
payload_key   : "requireAlphanumeric"
```



```
payload_type : "com.apple.mobiledevice.passwordpolicy"
</custom_item>
```

```
<custom_item>
type          : MACOSX_OSASCRIP
description   : "Require a minimum of 1 Complex Character Password Policy"
expect       : "([1-9]|[0-9]{2,})"
payload_key   : "minComplexChars"
payload_type  : "com.apple.mobiledevice.passwordpolicy"
</custom_item>
```

```
<custom_item>
type          : MACOSX_OSASCRIP
description   : "Disable iCloud Drive Document and Desktop Sync"
expect       : "false"
payload_key   : "allowCloudDesktopAndDocuments"
payload_type  : "com.apple.applicationaccess"
</custom_item>
```

## PKG\_CHECK

The "PKG\_CHECK" audit check performs a **pkgchk** against a SunOS system. The **pkg** keyword is used to specify the package to look for and the operator keyword specifies the condition to pass or fail the check based on the version of the installed package.

**Tip:** For information about the parameters commonly found in Unix custom items, see [Unix Configuration Keywords](#).

## Examples

```
<custom_item>
system: "SunOS"
type: PKG_CHECK
description: "Make sure SUNWcrman is installed"
pkg: "SUNWcrman"
required: YES
</custom_item>
```



```
<custom_item>
system: "SunOS"
type: PKG_CHECK
description: "Make sure SUNWcrman is installed and is greater than 9.0.2"
pkg: "SUNWcrman"
version: "9.0.2"
operator: "gt"
required: YES
</custom_item>
```

## PROCESS\_CHECK

As with file checks, an audited Unix platform can be tested for running processes. The implementation runs the **ps** command to obtain a list of running processes.

```
<custom_item>
system: "Linux"
type: PROCESS_CHECK
name: "auditd"
status: OFF
</custom_item>
```

```
<custom_item>
system: "Linux"
type: PROCESS_CHECK
name: "syslogd"
status: ON
</custom_item>
```

## RPM\_CHECK

The “RPM\_CHECK” audit check is used to check the version numbers of installed RPM packages on the remote system. This check consists of four mandatory keywords (**type**, **description**, **rpm**, and **operator**) and one optional keyword (**required**). The **rpm** keyword is used to specify the package to look for and the **operator** keyword specifies the condition to pass or fail the check based on the version of the installed RPM package.



"RPM\_CHECK" uses various iterations of `rpm -qa` to capture and normalize. It then uses the data to evaluate packages.

**Note:** Using the RPM checks is not portable across Linux distributions. Therefore, using RPM\_CHECK is not considered portable.

## Examples

These examples assume that you have installed **iproute-2.4.7-10**.

```
<custom_item>
type: RPM_CHECK
description: "RPM check for iproute-2.4.7-10 - should pass"
rpm: "iproute-2.4.7-10"
operator: "gte"
</custom_item>
```

```
<custom_item>
type: RPM_CHECK
description: "RPM check for iproute-2.4.7-10 should fail"
rpm: "iproute-2.4.7-10"
operator: "lt"
required: YES
</custom_item>
```

```
<custom_item>
type: RPM_CHECK
description: "RPM check for iproute-2.4.7-10 should fail"
rpm: "iproute-2.4.7-10"
operator: "gt"
required: NO
</custom_item>
```

```
<custom_item>
type: RPM_CHECK
description: "RPM check for iproute-2.4.7-10 should pass"
rpm: "iproute-2.4.7-10"
operator: "eq"
```





```
required: NO
</custom_item>
```

## SVC\_PROP

The “SVC\_PROP” audit check lets one interact with the **svccprop -p** tool on a Solaris 10 system. This can be used to query properties associated with a specific service. The **service** keyword is used to specify the service that is being audited. The **property** keyword specifies the name of the property that we want to query. The **value** keyword is the expected value of the property. The expected value can also be a regex.

The **svccprop\_option** field can be set to specify if the audited string being searched for is required to be present or not. This field access CAN\_BE\_NULL or CANNOT\_BE\_NULL as arguments.

**Tip:** For information about the parameters commonly found in Unix custom items, see [Unix Configuration Keywords](#).

## Examples

```
<custom_item>
type: SVC_PROP
description: "Check service status"
service: "cde-ttdbserver:tcp"
property: "general/enabled"
value: "false"
</custom_item>
```

```
<custom_item>
type: SVC_PROP
description: "Make sure FTP logging is set"
service: "svc:/network/frp:default"
property: "inetd_start/exec"
regex: ".*frpd.*-1"
</custom_item>
```

```
<custom_item>
type: SVC_PROP
```



```
description: "Check if ipfilter is enabled - can be missing or not found"
service: "network/ipfilter:default"
property: "general/enabled"
value: "true"
svcprop_option: CAN_BE_NULL
</custom_item>
```

## XINETD\_SVC

The “XINETD\_SVC” audit check is used to audit the startup status of xinetd services. The check consists of four mandatory keywords (**type**, **description**, **service**, and **status**).

**Note:** This only works on Red Hat systems or a derivative of Red Hat system such as Fedora.

**Tip:** For information about the parameters commonly found in Unix custom items, see [Unix Configuration Keywords](#).

## Example

```
<custom_item>
type: XINETD_SVC
description: "Make sure that telnet is disabled"
service: "telnet"
status: OFF
</custom_item>
```

## Built-In Checks

The checks that could not be covered by the checks described above are required to be written as custom names in NASL. All such checks fall under the “built-in” category. Each check starts with a **<item>** tag and ends with **</item>**. Enclosed within the tags are lists of one or more keywords that are interpreted by the compliance check parser to perform the checks. The following is a list of available checks.

**Note:** The **system** keyword is not available for the built-in checks and will result in a syntax error if used.

**Note:** All built-in and **custom\_item** checks support the following optional keywords:



- **info** – Allows you to add a more detailed description to the check that is being performed. Multiple info fields are allowed with no preset limit. The info content must be enclosed in double-quotes.

Example: *info: "Verifies login authentication configuration."*

- **solution** – Allows you to add solution text to fix a compliance failure.

Example: *solution: "Modify the configuration to add missing line"*

- **see\_also** – Allows you to include links that might provide helpful information about a check.

Example: *see\_also: "http://www.fireeye.com/support"*

- **reference** – Allows you to include cross references for audit checks.

Example: *reference: "PCI/2.2.3,SANS-CSC/1"*

This section includes the following information:

- [Password Management](#)
- [Root Access](#)
- [Permissions Management](#)
- [Password File Management](#)
- [Group File Management](#)
- [Root Environment](#)
- [File Permissions](#)
- [Suspicious File Content](#)
- [Unnecessary Files](#)
- [Docker Containers](#)

## Password Management

In the examples in this section, <min> and <max> are used to represent an integer value and not a string to use in the audit value data. In cases where the exact minimum or maximum value is not known, substitute the strings “Min” or “Max” for the integer value.

This section includes the following information:

- [min\\_password\\_length](#)
- [max\\_password\\_age](#)
- [min\\_password\\_age](#)

## min\_password\_length

This built-in check ensures that the minimum password length enforced on the remote system is in the range `<min>..<max>`. Having a minimum password length forces users to choose more complex passwords.

Operating System	Implementation
Linux	The minimum password length is defined as <code>PASS_MIN_LEN</code> in <code>/etc/login.defs</code> .
Solaris	The minimum password length is defined as <code>PASSLENGTH</code> in <code>/etc/default/passwd</code> . <div> <b>Note:</b> This also controls the password maximum length.           </div>
HP-UX	The minimum password length is defined as <code>MIN_PASSWORD_LENGTH</code> in <code>/etc/default/security</code> .
macOS	The minimum password length is defined as “minChar” in the local policy, defined using the command <code>pwpolicy</code> .

## Usage

```
<item>
name: "min_password_length"
description: "This check examines the system configuration for the minimum password
length that the passwd program will accept. The check reports a problem if the minimum
length is less than the length specified in your policy."
value: "<min>..<max>"
</item>
```

## Example



```
<item>
name: "min_password_length"
description: "Make sure that each password has a minimum length of 6 chars or more"
value: "6..65535"
</item>
```

## max\_password\_age

This built-in function ensures that the maximum password age (e.g., the time when users are forced to change their passwords) is in the defined range.

Having a maximum password age prevents users from keeping the same password for multiple years. Changing passwords often helps prevent an attacker possessing a password from using it indefinitely.

Operating System	Implementation
Linux	The variable <code>PASS_MAX_DAYS</code> is defined in <code>/etc/login.defs</code> .
Solaris	The variable <code>MAXWEEKS</code> in <code>/etc/default/passwd</code> defines the maximum number of weeks a password can be used.
HP-UX	This value is controlled by the variable <code>PASSWORD_MAXDAYS</code> in <code>/etc/default/security</code> .
macOS	The option “maxMinutesUntilChangePassword” of the password policy (as set through the <code>pwpolicy</code> tool) can be used to set this value.

## Usage

```
<item>
name: "max_password_age"
description: "This check reports agents that have a system default maximum password age greater than the specified value and agents that do not have a maximum password age setting."
value: "<min>..<max>"
</item>
```



## Example

```
<item>
name: "max_password_age"
description: "Make sure a password can not be used for more than 21 days"
value: "1..21"
</item>
```

## min\_password\_age

This built-in function ensures that the minimum password age (e.g., the time required before users are permitted to change their passwords) is in the defined range.

Having a minimum password age prevents users from changing passwords too often in an attempt to override the maximum password history. Some users do this to cycle back to their original password, circumventing password change requirements.

Operating System	Implementation
Linux	The variable PASS_MIN_DAYS is defined in <b>/etc/login.defs</b> .
Solaris	The variable MINWEEKS in <b>/etc/default/passwd</b> defines the maximum number of weeks a password can be used.
HP-UX	This value is controlled by the variable PASSWORD_MINDAYS in <b>/etc/default/security</b> .
macOS	This option is not supported.

## Usage

```
<item>
name: "min_password_age"
description: "This check reports agents and users with password history settings that are less than a specified minimum number of passwords."
value: "<min>..<max>"
</item>
```



## Example

```
<item>
name: "min_password_age"
description: "Make sure a password cannot be changed before 4 days while allowing the
user to change at least after 21 days"
value: "4..21"
</item>
```

## Root Access

### root\_login\_from\_console

This built-in function ensures that the “root” user can only directly log into the remote system through the physical console.

The rationale behind this check is that good administrative practices disallow the direct use of the root account so that access can be traced to a specific person. Instead, use a generic user account (member of the wheel group on BSD systems) then use “su” (or sudo) to elevate privileges to perform administrative tasks.

Operating System	Implementation
Linux and HP-UX	Make sure that <b>/etc/securetty</b> exists and only contains “console”.
Solaris	Make sure that <b>/etc/default/login</b> contains the line <b>CONSOLE=/dev/console</b> .
macOS	This option is not supported.

## Usage

```
<item>
name: "root_login_from_console"
description: "This check makes sure that root can only log in from the system console
(not remotely)."
```

---

```
</item>
```

## Permissions Management

The topics in this section describe the following checks related to managing permissions:

- [accounts\\_bad\\_home\\_permissions](#)
- [accounts\\_bad\\_home\\_group\\_permissions](#)
- [accounts\\_without\\_home\\_dir](#)
- [active\\_accounts\\_without\\_home\\_dir](#)
- [invalid\\_login\\_shells](#)
- [login\\_shells\\_with\\_suid](#)
- [login\\_shells\\_writeable](#)
- [login\\_shells\\_bad\\_owner](#)

### accounts\_bad\_home\_permissions

This built-in function ensures that the home directory of each non-privileged user belongs to the user and that third party users (either belonging to the same group or “everyone”) may not write to it. It is generally recommended that user home directories are set to mode 0755 or stricter (e.g., 0700). This test succeeds if each home directory is configured properly and fails otherwise. Either of the keywords **mode** or **mask** may be used here to specify desired permission levels for home directories. The mode keyword will accept home directories matching exactly a specified level and the mask keyword will accept home directories that are at the specified level or more secure. If no “mask” tag is found, a default mask of 022 (755) will be applied.

This check can be modified using the following tags:

- **use\_valid\_shells** : [YES | NO] - This reads in shells from /etc/shells and only uses that list against /etc/passwd to determine interactive users
- **ignore\_user** : “username1 username2” - A space separated list of users to ignore
- **ignore\_shell**: “shell1 shell2” - A space separated list of shells to ignore.





If third parties can write to the home directory of a user, they can force the user to execute arbitrary commands by tampering with the `~/.profile`, `~/.cshrc`, `~/.bashrc` files.

If files need to be shared among users of the same group, it is usually recommended that a dedicated directory writeable to the group be used, not a user's home directory.

For any misconfigured home directories, run `chmod 0755 <user directory>` and change the ownership accordingly.

To force the check to ignore a directory, use `ignore`.

## Usage

```
<item>
name: "accounts_bad_home_permissions"
description: "This check reports user accounts that have home directories with
incorrect user or group ownerships."
mask: "027"
ignore: "/example/path"
</item>
```

### accounts\_bad\_home\_group\_permissions

This built-in function is operationally similar to `accounts_bad_home_permissions`, but ensures that the user home directories are group owned by the user's primary group.

## Usage

```
<item>
name: "accounts_bad_home_group_permissions"
description: "This check makes sure user home directories are group owned by the user's
primary group."
</item>
```

### accounts\_without\_home\_dir

This built-in function ensures that every user has a home directory. It passes if a valid directory is attributed to each user and fails otherwise. Note that home directory ownership or permissions are not tested by this check.



It is generally recommended that each user on a system have a home directory defined as some tools may need to read from it or write to it (for instance, **sendmail** checks for a **~/ .forward** file). If a user does not need to log in, a non-existent shell (e.g., **/bin/false**) should be defined instead. On many systems, a user with no home directory will still be granted login privileges but their effective home directory is **/**.

## Usage

```
<item>
name: "accounts_without_home_dir"
description: "This check reports user accounts that do not have home directories."
</item>
```

### active\_accounts\_without\_home\_dir

This built-in function ensures that every active user (users that are not non-interactive) has a home directory. It passes if a valid directory is attributed to each user and fails otherwise. Note that home directory ownership or permissions are not tested by this check.

It is generally recommended that each active user on a system have a home directory defined as some tools may need to read from it or write to it (for instance, **sendmail** checks for a **~/ .forward** file). If an active user does not need to log in, a non-existent shell (e.g., **/bin/false**) should be defined instead. On many systems, an active user with no home directory will still be granted login privileges but their effective home directory is **/**.

## Usage

```
<item>
name: "active_accounts_without_home_dir"
description: "This check reports active user accounts that do not have home
directories."
</item>
```

### invalid\_login\_shells

This built-in function ensures that each user has a valid shell as defined in **/etc/shells**.



The `/etc/shells` file is used by applications such as Sendmail and FTP servers to determine if a shell is valid on the system. While it is not used by the login program, administrators can use this file to define which shells are valid on the system. The `invalid_login_shells` check can verify that all users in the `/etc/passwd` file are configured with valid shells as defined in the `/etc/shells` file.

This avoids unsanctioned practices such as using `/sbin/passwd` as a shell to let users change their passwords. If you do not want a user to be able to log in, create an invalid shell in `/etc/shells` (e.g., `/nonexistent`) and set it for the desired users.

If you have users without a valid shell, define a valid shell for them.

## Usage

```
<item>
name: "invalid_login_shells"
description: "This check reports user accounts with shells which do not exist or is not
listed in /etc/shells."
</item>
```

### login\_shells\_with\_suid

This built-in function makes sure that no shell has “set-uid” capabilities.

A “setuid” shell means that whenever the shell is started, the process itself will have the privileges set to its permissions (a setuid “root” shell grants super-user privileges to anyone for instance).

Having a “setuid” shell defeats the purpose of having UIDs and GIDs and makes access control much more complex.

Remove the SUID bit of each shell that is “setuid”.

## Usage

```
<item>
name: "login_shells_with_suid"
description: "This check reports user accounts with login shells that have setuid or
setgid privileges."
</item>
```



## login\_shells\_writeable

This built-in function makes sure that no shell is world/group writeable.

If a shell is world writeable (or group writeable) then non-privileged users can replace it with any program. This enables a malicious user to force other users of that shell to execute arbitrary commands when they log in.

Ensure the permissions of each shell are set appropriately.

## Usage

```
<item>
name: "login_shells_writeable"
description: "This check reports user accounts with login shells that have group or
world write permissions."
</item>
```

## login\_shells\_bad\_owner

This built-in function ensures that every shell belongs to the “root” or “bin” users.

As for shells with invalid permissions, if a user owns a shell used by other users, then they can modify it to force third party users to execute arbitrary commands when they log in.

Only “root” and/or “bin” should be able to modify system-wide binaries.

## Usage

```
<item>
name: "login_shells_bad_owner"
description: "This check reports user accounts with login shells that are not owned by
root or bin."
</item>
```

## Password File Management

The topics in this section describe the following checks related to managing password files:

- 
- [passwd\\_file\\_consistency](#)
  - [passwd\\_zero\\_uid](#)
  - [passwd\\_duplicate\\_uid](#)
  - [passwd\\_duplicate\\_gid](#)
  - [passwd\\_duplicate\\_username](#)
  - [passwd\\_duplicate\\_home](#)
  - [passwd\\_shadowed](#)
  - [passwd\\_invalid\\_gid](#)

## passwd\_file\_consistency

This built-in function ensures that each line in **/etc/passwd** has a valid format (e.g., seven fields separated by colon). If a line is malformed, it is reported and the check fails.

Having a malformed **/etc/passwd** file can break several user-management tools. It may also indicate a break-in or a bug in a custom user-management application. It may also show that someone attempted to add a user with an invalid name (in the past, it was popular to create a user named “toor:0:0” to obtain root privileges).

If the test is considered non-compliant, the administrator must remove or fix the offending lines from **/etc/passwd**.

## Usage

```
<item>
name: "passwd_file_consistency"
description: "This check makes sure /etc/passwd is valid."
</item>
```

## passwd\_zero\_uid

This built-in function ensures that only one account has a UID of “0” in **/etc/passwd**. This is intended to be reserved for the “root” account but it is possible to add additional accounts with UID 0 that would have the same privileged access. This test succeeds if only one account has a UID of zero and fails otherwise.



A UID of “0” grants root privileges on the system. A root user can perform anything they want to on the system, which typically includes snooping the memory of other processes (or of the kernel), read and write any file on the system and so on. Because this account is so powerful, its use must be restrained to the bare minimum and it must be well protected.

Good administrative practices dictate that each UID be unique (hence the “U” in UID). Having two (or more) accounts with “root” privileges negates the accountability a system administrator may have towards the system. In addition, many systems restrict the direct login of root to the console only so that administrative use can be tracked. Typically, systems administrators have to first log in to their own account and use the `su` command to become root. An additional UID 0 account evades this restriction.

If “root” access needs to be shared among users, use a tool like **sudo** or **calife** instead (or RBAC on Solaris). There should only be one account with a UID of “0”.

## Usage

```
<item>
name: "passwd_zero_uid"
description: "This check makes sure that only ONE account has a uid of 0."
</item>
```

### passwd\_duplicate\_uid

This built-in function ensures that every account listed in `/etc/passwd` has a unique UID. This test succeeds if every UID is unique and fails otherwise.

Each user on a Unix system is identified by its User ID (UID), a number comprised between 0 and 65535. If two users share the same UID, then they are not only granted the same privileges, but the system will consider them as being the same person. This defeats any kind of accountability since it is impossible to tell which actions have been performed by each user (typically, the system will do a reverse look up on the UID and will use the first name of the accounts sharing the UID when displaying logs).

Security standards such as the CIS benchmarks forbid sharing a UID among users. If users need to share files, then use groups instead.

Give each user on the system a unique ID.



## Usage

```
<item>
name: "passwd_duplicate_uid"
description: "This check makes sure that every UID in /etc/passwd is unique."
</item>
```

### passwd\_duplicate\_gid

This built-in function ensures that the primary group ID (GID) of each user is unique. The test succeeds if every user has a unique GID and fails otherwise.

Security standards recommend creating one group per user (typically with the same name as the username). With this setup, files created by the user are typically “secure by default” as they belong to its primary group, and therefore can only be modified by the user itself. If the user wants the file to be owned by the other members of a group, he will have to explicitly use the **chgrp** command to change ownership.

Another advantage of this approach is that it unifies group membership management into a single file (**/etc/group**), instead of a mix between **/etc/passwd** and **/etc/group**.

For each user, create a group with the same name. Manage group ownership through **/etc/group** only.

## Usage

```
<item>
name: "passwd_duplicate_gid"
description: "This check makes sure that every GID in /etc/passwd is unique."
</item>
```

### passwd\_duplicate\_username

This built-in function ensures that each username in **/etc/passwd** is unique. It succeeds if that is the case and fails otherwise.

Duplicate user names in **/etc/passwd** create problems since it is unclear which account's privileges are being used.



The **adduser** command will not let you create a duplicate username. Such a setup typically means that the system has been compromised, tools to handle user management are buggy or the **/etc/passwd** file was manually edited.

Delete duplicate usernames or modify them to be different.

## Usage

```
<item>
name: "passwd_duplicate_username"
description: "This check makes sure that every username in /etc/passwd is unique."
</item>
```

### passwd\_duplicate\_home

This built-in function ensures that each non-system user (whose UID is greater than 100) in **/etc/passwd** has a unique home directory.

Each username in **/etc/passwd** must have a unique home directory. If users share the same home directory, then one can force the other to execute arbitrary commands by modifying the startup files (**.profile**, etc.) or by putting rogue binaries in the home directory itself. In addition, a shared home directory defeats user accountability.

Compliance requirements mandate that each user have a unique home directory.

## Usage

```
<item>
name: "passwd_duplicate_home"
description: "(arbitrary user comment)"
</item>
```

### passwd\_shadowed

This built-in check ensures that every password in **/etc/passwd** is “shadowed” (i.e., that it resides in another file).

Since **/etc/passwd** is world-readable, storing users’ password hashes in it permits anyone with access to it the ability to run password cracking programs on it. Attempts to guess a user’s password





through a brute force attack (repeated login attempts, trying different passwords each time) are usually detected in system log files. If the **/etc/passwd** file contains the password hashes, the file could be copied offline and used as input to a password cracking program. This permits an attacker the ability to obtain user passwords without detection.

Most modern Unix systems have shadowed password files. Consult your system documentation to learn how to enable shadowed passwords on your system.

## Usage

```
<item>
name: "passwd_shadowed"
description: "(arbitrary user comment)"
</item>
```

### passwd\_invalid\_gid

This built-in function ensures that each group ID (GID) listed in **/etc/passwd** exists in **/etc/group**. It succeeds if each GID is properly defined and fails otherwise.

Every time a group ID is defined in **/etc/passwd**, it should immediately be listed in **/etc/group**. Otherwise, the system is in an inconsistent state and problems may arise.

Consider the following scenario: a user ("bob") has a UID of 1000 and GID of 4000. The GID is not defined in **/etc/group**, which means that the primary group of the user does not grant him any privileges today. A few months later, the system administrator edits **/etc/group** and adds the group "admin" and selects the "unused" GID #4000 to identify it. Now, user "bob" by default belongs to the "admin" group even though this was not intended.

Edit **/etc/group** to add the missing GIDs.

## Usage

```
<item>
name: "passwd_invalid_gid"
description: "This check makes sure that every GID defined in /etc/passwd exists in
/etc/group."
</item>
```



## Group File Management

The topics in this section describe the following checks related to managing group files:

- [group\\_file\\_consistency](#)
- [group\\_zero\\_gid](#)
- [group\\_duplicate\\_name](#)
- [group\\_duplicate\\_gid](#)
- [group\\_duplicate\\_members](#)
- [group\\_nonexistent\\_users](#)

### group\_file\_consistency

This built-in function ensures that each line in **/etc/group** has a valid format (e.g., three items separated by colon and a list of users). If a line is malformed, it is reported and the check fails.

Having a malformed **/etc/group** file may break several user-management tools. It may also indicate a break-in or a bug in a custom user-management application. It may also show that someone attempted to add a user with an invalid group name.

Edit the **/etc/group** file to fix the badly formed lines.

## Usage

```
<item>
name: "group_file_consistency"
description: "This check makes sure /etc/group is valid."
</item>
```

### group\_zero\_gid

This built-in function ensures that only one group has a group ID (GID) of 0. It passes if only one group has a GID of 0 and fails otherwise.

A GID of “0” means that the users who are members of this group are also members root’s primary group. This grants them root privileges on any files with root group permissions.

If you want to define a group of administrators, create an “admin” group instead.



## Usage

```
<item>
name: "group_zero_gid"
description: "This check makes sure that only ONE group has a gid of 0."
</item>
```

### group\_duplicate\_name

This built-in check ensures that each group name is unique. It succeeds if that is the case and fails otherwise.

Duplicate group names in **/etc/group** create problems, since it is unclear which group privileges are being used. This means that a duplicate group name may end up having members or privileges it should not have had in the first place.

Delete or rename duplicate group names.

## Usage

```
<item>
name: "group_duplicate_name"
description: "This check makes sure that every group name in /etc/group is unique."
</item>
```

### group\_duplicate\_gid

Each group on a Unix system is identified by its group ID (GID), a number comprised between 0 and 65535. If two groups share the same GID, then they are not only granted the same privileges, but the system will consider them as being the same group. This defeats the purpose of using groups to segregate user privileges.

Security standards forbid sharing a GID among groups. If two groups need to have the same privileges, they should have the same users.

Delete the duplicate groups or assign one of the duplicates a new unique GID.

## Usage



```
<item>
name: "group_duplicate_gid"
description: "(arbitrary user comment)"
</item>
```

## group\_duplicate\_members

This built-in function ensures that each member of a group is only listed once. It passes if each member is unique and fails otherwise.

Each member of a group should only be listed once. While being listed multiple times does not cause a problem to the underlying operating system, it makes the system administrator's life more difficult as revoking privileges becomes more complex. For instance, if the group "admin" has the members "alice, bob, charles, daniel, bob" then "bob" will need to be removed twice if his privileges were to be revoked.

Ensure that each member is listed only once.

## Usage

```
<item>
name: "group_duplicate_members"
description: "This check makes sure that every member of a group is listed once."
</item>
```

## group\_nonexistent\_users

This check ensures that each member of a group actually exists in **/etc/passwd**.

Having non-existent users in **/etc/group** implies incomplete administration practices. The user does not exist either because it has been mistyped or because it has not been removed from the group when the user has been removed from the system.

It is not recommended to have "ghost" users stay in **/etc/group**. If a user with the same username were to be added at a later time, the user may have group privileges that should not be granted.

Remove non-existent users from **/etc/group**.

## Usage



```
<item>
name: "group_nonexistant_users"
description: "This check makes sure that every member of a group actually exists."
</item>
```

## Root Environment

### dot\_in\_root\_path\_variable

This check ensures the following in the executable path of the root user:

- current working directory not present - `.` (dot)
- directories starts with a slash - `/`
- path contains double colon - `::`
- path has a trailing colon - `:\$` (end of line)

Ensuring this prevents a malicious user from escalating privileges to superuser by forcing an administrator logged in as root from running a Trojan horse that may be installed in the current working directory.

## Usage

```
<item>
name: "dot_in_root_path_variable"
description: "This check makes sure that root's $PATH variable does not contain any
relative path."
</item>
```

### writable\_dirs\_in\_root\_path\_variable

This check reports all the world/group writeable directories in root users PATH variable. All directories returned by this check should be carefully examined and unnecessary world/group writeable permissions on directories should be removed as follows:

```
# chmod go-w path/to/directory
```

## Usage



```
<item>
name: "writeable_dirs_in_root_path_variable"
description: "This check makes sure that root's $PATH variable does not contain any
writeable directory."
</item>
```

## File Permissions

The topics in this section describe the following checks related to managing file permissions:

- [find\\_orphan\\_files](#)
- [find\\_world\\_writeable\\_files](#)
- [find\\_world\\_writeable\\_directories](#)
- [find\\_world\\_readable\\_files](#)
- [find\\_suid\\_sgid\\_files](#)
- [home\\_dir\\_localization\\_files\\_user\\_check](#)
- [home\\_dir\\_localization\\_files\\_group\\_check](#)

### find\_orphan\_files

This check reports all files that are un-owned on the system.

By default, the search is done recursively under the “/” directory. This can make this check extremely slow to execute depending on the number of files present on the remote system. However, if needed, the default base directory to search for can be changed by using the optional keyword `basedir`. It is also possible to skip certain files within a base directory from being searched using another optional keyword `ignore`.

This check can be modified to report files that have no user or group found specifically. This is used with the `find_option` tag. Valid values are `nouser`, `nogroup`, and `both`. The `both` setting is default if no `find_option` tag is specified.

Due to the nature of the check, it is normal for it to keep running for a couple of hours, depending on the type of system being scanned. The check can be modified with the `timeout` tag with a value between 1 and 7,200 seconds to control processing time for this check.



## Usage

```
<item>
name: "find_orphan_files"
description: "This check finds all the files which are 'orphaned' (ie: whose owner is
an invalid UID or GID)."
# Globs allowed (? and *)
(optional) basedir: "<directory>"
(optional) ignore: "<directory>"
(optional) find_option: ["nouser", "nogroup", "both"]
(optional) timeout: "[1 - 7200]"
</item>
```

## Examples

```
<item>
name: "find_orphan_files"
description: "This check finds all the files which are 'orphaned' (ie: whose owner is
an invalid UID or GID)."
# Globs allowed (? and *)
basedir: "/tmp"
ignore: "/tmp/foo"
ignore: "/tmp/b*"
</item>
```

```
<item>
name: "find_orphan_files"
description: "Only find files that have no group"
basedir: "/tmp"
find_option: "nogroup"
</item>
```

```
<item>
name: "find_orphan_files"
description: "Only find files that have no user"
basedir: "/tmp"
find_option: "nouser"
```



```
</item>
```

## find\_world\_writeable\_files

This check reports all the files that are world writeable on the remote system. Ideally, there should be no world writeable files on the remote system, for example, the result from this check should show nothing. However, in some cases, depending on organizational needs, there may be a requirement for having world writeable files. All items returned from this check must be carefully audited and files that do not necessarily need world writeable attributes should be removed as follows:

```
# chmod o-w world_writeable_file
```

By default, the search is done recursively under the “/” directory. This can make this check extremely slow to execute depending on the number of files present on the remote system. However, if needed, the default base directory to search for can be changed by using the optional keyword **basedir**. It is also possible to skip certain files within a base directory from being searched using another optional keyword **ignore**.

Due to the nature of the check, it is normal for it to keep running for a couple of hours, depending on the type of system being scanned. The check can be modified with the **timeout** tag with a value between 1 and 7,200 seconds to control processing time for this check.

## Usage

```
<item>
name: "find_world_writeable_files"
description: "This check finds all the files which are world writeable and whose sticky
bit is not set."
# Globs allowed (? and *)
(optional) basedir: "<directory>"
(optional) ignore: "<directory>"
(optional) timeout: "[1 - 7200]"
</item>
```

## Example





```
<item>
name: "find_world_writeable_files"
description: "Search for world-writable files"
# Globs allowed (? and *)
basedir: "/tmp"
ignore: "/tmp/foo"
ignore: "/tmp/bar"
</item>
```

## find\_world\_writeable\_directories

This check reports all the directories that are world writeable and whose sticky bit is not set on the remote system. Checking that the sticky bit is set for all world writeable directories ensures that only the owner of file within a directory can delete the file. This prevents any other user from accidentally or intentionally deleting the file.

By default, the search is done recursively under the “/” directory. This can make this check extremely slow to execute depending on the number of files present on the remote system. However, if needed, the default base directory to search for can be changed by using the optional keyword **basedir**. It is also possible to skip certain files within a base directory from being searched using another optional keyword **ignore**.

Due to the nature of the check, it is normal for it to keep running for a couple of hours, depending on the type of system being scanned. The check can be modified with the `timeout` tag with a value between 1 and 7,200 seconds to control processing time for this check.

**Note:** File globs are not supported on AIX, Solaris, and other Unix systems where the `find` command does not support the `-path` flag.

## Usage

```
<item>
name: "find_world_writeable_directories"
description: "This check finds all the directories which are world writeable and whose
sticky bit is not set."
# Globs allowed (? and *)
(optional) basedir: "<directory>"
(optional) ignore: "<directory>"
```



```
(optional) timeout: "[1 - 7200]"
</item>
```

## Example

```
<item>
name: "find_world_writeable_directories"
description: "This check finds all the directories which are world writeable and whose
sticky bit is not set."
# Globs allowed (? and *)
basedir: "/tmp"
ignore: "/tmp/foo"
ignore: "/tmp/b*"
</item>
```

### find\_world\_readable\_files

This check reports all the files that are world readable. Checking for readable files, for example in user home directories, ensures that no sensitive files are accessible by other users (e.g., private SSH keys).

By default, the search is done recursively under the “/” directory. This can make this check extremely slow to execute depending on the number of files present on the remote system. However, if needed, the default base directory to search for can be changed by using the optional keyword **basedir**. It is also possible to skip certain files within a base directory from being searched using another optional keyword **ignore**.

Due to the nature of the check, it is normal for it to keep running for a couple of hours, depending on the type of system being scanned. The check can be modified with the **timeout** tag with a value between 1 and 7,200 seconds to control processing time for this check.

## Usage

```
<item>
name: "find_world_readable_files"
description: "This check finds all the files in a directory with world readable
permissions."
```



```
# Globs allowed (? and *)
(optional) basedir: "<directory>"
(optional) ignore: "<directory>"
(optional) timeout: "[1 - 7200]"
</item>
```

## Example

```
<item>
name: "find_world_readable_files"
description: "This check finds all the files in a directory with world readable
permissions."
basedir: "/home"
ignore: "/home/tmp"
</item>
```

### find\_suid\_sgid\_files

This check reports all files with the SUID/SGID bit set. All files reported by this check should be carefully audited, especially shell scripts and home grown/in-house executables, for example executables that are not shipped with the system. SUID/SGID files present the risk of escalating privileges of a normal user to the ones possessed by the owner or the group of the file. If such files/scripts do need to exist then they should be specially examined to check if they allow creating file with elevated privileges.

This check can be modified to report files that are SUID (-4000) or SGID (-2000) specifically.

`find_option: [suid | sgid | both]`

The **both** setting is default if no **find\_option** tag is specified.

By default, the search is done recursively under the '/' directory. This can make this check extremely slow to execute depending on the number of files present on the remote system. However, if needed, the default base directory to search for can be changed by using the optional keyword **basedir**. It is also possible to skip certain files within a base directory from being searched using another optional keyword **ignore**.



Due to the nature of the check, it is normal for it to keep running for a couple of hours, depending on the type of system being scanned. The check can be modified with the `timeout` tag with a value between 1 and 7,200 seconds to control processing time for this check.

## Usage

```
<item>
name: "find_suid_sgid_files"
description: "This check finds all the files which have their SUID or SGID bit set."
# Globs allowed (? and *)
(optional) basedir: "<directory>"
(optional) ignore: "<directory>"
(optional) timeout: "[1 - 7200]"
</item>
```

## Example

```
<item>
name: "find_suid_sgid_files"
description: "Search for SUID/SGID files"
# Globs allowed (? and *)
basedir: "/"
ignore: "/usr/sbin/ping"
</item>
```

## home\_dir\_localization\_files\_user\_check

This built-in checks whether a localization file within a user's home directory is either owned by the user or the root.

One or more files could be listed using the "file" token. However if the "file" token is missing the check by default looks for the following files:

- **.login**
- **.cschrc**
- **.logout**
- **.profile**



- `.bash_profile`
- `.bashrc`
- `.bash_logout`
- `.env`
- `.dtpprofile`
- `.dispatch`
- `.emacs`
- `.exrc`

## Example

```
<item>
name: "home_dir_localization_files_user_check"
description: "Check file .foo/.foo2"
file: ".foo"
file: ".foo2"
file: ".foo3"
</item>
```

### home\_dir\_localization\_files\_group\_check

This built-in checks whether a localization file within a user's home directory is group owned by the user's primary group or root.

One or more files could be listed using the "file" token. However if the "file" token is missing the check by default looks for the following files:

- `.login`
- `.cschrc`
- `.logout`
- `.profile`
- `.bash_profile`



- `.bashrc`
- `.bash_logout`
- `.env`
- `.dtpprofile`
- `.dispatch`
- `.emacs`
- `.exrc`

## Example

```
<item>
name: "home_dir_localization_files_group_check"
description: "Check file .foo/.foo2"
file: ".foo"
file: ".foo2"
file: ".foo3"
</item>
```

## Suspicious File Content

### admin\_accounts\_in\_ftpusers

This check audits if all admin accounts, users with UID less than 500, are present in `/etc/ftpusers`, `/etc/ftpd/ftpusers`, or `/etc/vsftpd.ftpusers`.

## Usage

```
<item>
name: "admin_accounts_in_ftpusers"
description: "This check makes sure every account whose UID is below 500 is present in
/etc/ftpusers."
</item>
```

## Unnecessary Files



## find\_pre-CIS\_files

This check is tailored towards a specific Center for Internet Security (CIS) requirement to pass the certification for Red Hat CIS benchmark. This check is particularly useful for someone who might have configured/hardened a Red Hat system based on the CIS Red Hat benchmark. The CIS benchmark tool provides a backup script to backup all the system files that may be modified during system hardening process and these files are suffixed with a keyword **-preCIS**. These files should be removed once all the benchmark recommendations are successfully applied and the system has been restored to its working condition. This check ensures that no **preCIS** files exist on the remote system.

By default, the search is done recursively under the "/" directory. This can make this check extremely slow to execute depending on the number of files present on the remote system. However, if needed, the default base directory to search for can be changed by using the optional keyword **basedir**. It is also possible to skip certain files within a base directory from being searched using another optional keyword **ignore**.

Due to the nature of the check, it is normal for it to keep running for a couple of hours, depending on the type of system being scanned. The check can be modified with the `timeout` tag with a value between 1 and 7,200 seconds to control processing time for this check.

## Usage

```
<item>
name: "find_preCIS_files"
description: "Find and list all files created by CIS backup script."
# Globs allowed (? and *)
(optional) basedir: "<directory>"
(optional) ignore: "<directory>"
(optional) timeout: "[1 - 7200]"
</item>
```

## Docker Containers

### list\_docker\_container\_packages



This check returns a listing of docker containers and packages for review. This built-in does not perform evaluations and is used for inventory purposes only.

## Usage

```
<item>
name: "list_docker_container_packages"
description: "This check returns docker containers and their packages for review"
</item>
```

## Conditions

It is possible to define **if/then/else** logic in the Unix policy. This allows the end-user to use a single file that is able to handle multiple configurations. For instance, the same policy file can check the settings for Postfix and Sendmail by using the proper **if/then/else** syntax.

The syntax to perform conditions is the following:

```
<if>
<condition type: "or">
<Insert your audit here>
</condition>
<then>
<Insert your audit here>
</then>
<else>
<Insert your audit here>
</else>
</if>
```

## Example

```
<if>
<condition type: "or">
<custom_item>
type: FILE_CHECK
description: "Check that at.allow exists"
file: "/etc/at.allow"
```





```
</custom_item>
<custom_item>
type: FILE_CHECK
description: "Check that at.deny exists"
file: "/etc/at.deny"
</custom_item>
</condition>

<then>
<report type:"PASSED">
description: "Make sure 'at' is secured with an allow or deny list"
</report>
</then>

<else>
<report type:"FAILED">
description: "Make sure 'at' is secured with an allow or deny list"
</report>
</else>
</if>
Whether the condition fails or passes never shows up in the report because it is a
“silent” check.
```

Conditions can be of type **and** or **or**.

## Caveats

The Unix compliance plugin can use a system tag to control if a particular check applies to the target OS. Using a system tag inside the `<condition></condition>` block is not recommended as it can cause false logic flow. The check content is evaluated before the system tag; therefore, a conditional may pass to the `<then>` section and not actually apply.

---

## Unix Content Audit Compliance File Reference

---

Unix Content **.audit** checks differ from Unix Configuration **.audit** checks in that they are designed to search a Unix file system for specific file types containing sensitive data rather than enumerate system configuration settings. They include a range of options to help the auditor narrow down the search parameters and more efficiently locate and display non-compliant data.

This section includes the following information:

- [Check Type](#)
- [Item Format](#)
- [Unix Content Command Line Examples](#)

### Check Type

All Unix content compliance checks must be bracketed with the **check\_type** encapsulation and the “FileContent” designation. This is very similar to all other **.audit** files. The basic format of a content check file is as follows:

```
<check_type: "FileContent">
<item>
</item>
<item>
</item>
<item>
</item>
</check_type>
```

The actual checks for each item are not shown. The following sections show how various keywords and parameters can be used to populate a specific content item audit.

### Item Format

Each of these items is used to audit a wide variety of file formats, with a wide variety of data types. The following table provides a list of supported data types. In the next section are numerous examples of how these keywords can be used together to audit various types of file content.



Keyword	Required	Description
type	yes	This must always be set to FILE_CONTENT_CHECK.
description	yes	This keyword provides the ability to add a brief description of the check that is being performed. It is strongly recommended that the <b>description</b> field be unique and no distinct checks have the same description field. Tenable uses this field to automatically generate a unique plugin ID number based on the description field.
file_extension	yes	This lists all desired extensions to be searched for by Nessus. The extensions are listed without their ".", in quotations and separated by pipes. When additional options such as <b>regex</b> and <b>expect</b> are not included in the audit, files with the file_extension specified are displayed in the audit output.
regex	no	<p>This keyword holds the regular expression used to search for complex types of data. If the regular expression matches, the first matched content will be displayed in the vulnerability report.</p> <div><p><b>Note:</b> The <b>regex</b> keyword must be run with the <b>expect</b> keyword described below.</p><p>Unlike Compliance Checks, File Content Compliance Check <b>regex</b> and <b>expect</b> do not have to match the same data string(s) within the searched file. File Content checks simply require that both the <b>regex</b> and <b>expect</b> statements match data within the &lt;max_size&gt; bytes of the file searched.</p></div>
expect	no	The <b>expect</b> statement is used to list one or more simple patterns that must be in the document in order for it to match. For example, when searching for Social Security numbers, the word "SSN", "SS#", or "Social" could be required.



Keyword	Required	Description
		<p>Multiple patterns are listed in quotes and separated with pipe characters.</p> <p>Simple pattern matching is also supported in this keyword with the period. When matching the string “C.T”, the <b>expect</b> statement would match “CAT”, “CaT”, “COT”, “C T” and so on.</p> <div><p><b>Note:</b> The <b>expect</b> keyword may be run standalone for single pattern matching, however, if the <b>regex</b> keyword is used, <b>expect</b> is required.</p><p>Unlike Compliance Checks, File Content Compliance Check <b>regex</b> and <b>expect</b> do not have to match the same data string(s) within the searched file. File Content checks simply require that both the <b>regex</b> and <b>expect</b> statements match data within the &lt;max_size&gt; bytes of the file searched.</p></div>
file_name	no	<p>Whereas the <b>file_extension</b> keyword is required, this keyword can further refine the list of files to be analyzed. By providing a list of patterns, files can be discarded or matched.</p> <p>For example, this makes it very easy to search for any type of file name that has terms in its name such as “employee”, “customer”, or “salary”.</p>
max_size	no	<p>For performance, an audit may only want to look at the first part of each file. This can be specified in bytes with this keyword. The number of bytes can be used as an argument. Also supported is an extension of “K” or “M” for kilobytes or megabytes respectively.</p>
only_show	no	<p>This keyword supports revealing a specific number of characters specified by policy. When matching sensitive data such as credit card numbers, your organization may</p>



Keyword	Required	Description
		<p>require that only a limited number of digits be made visible in the report. The default is 4 or half of the matched string, whichever is smaller. For example, if a matched string is 10 characters long and <code>only_show</code> is set to 4, only the last 4 characters are shown. If the matched string is 6 characters long, only 3 characters will be shown.</p> <div><b>Note:</b> When you match against US Social Security numbers (SSNs), the specified number of digits are revealed <i>in front</i> of the string (for example, 123-XX-XXXX).</div>
<code>regex_replace</code>	no	<p>This keyword controls which pattern in the regular expression is shown in the report. When searching for complex data patterns, such as credit card numbers, it is not always possible to get the first match to be the desired data. This keyword provides more flexibility to capture the desired data with greater accuracy.</p>
<code>include_paths</code>	no	<p>This keyword allows for directory or drive inclusion within the search results. This keyword may be used in conjunction with, or independently of the <code>exclude_paths</code> keyword. This is particularly helpful for cases where only certain drives or folders must be searched on a multi-drive system. Paths are double-quoted and separated by the pipe symbol where multiple paths are required.</p> <p>Only drive letters or folder names can be specified with the <code>include_paths</code> keyword. File names cannot be included in the <code>include_paths</code> value string.</p>
<code>exclude_paths</code>	no	<p>This keyword allows for drive, directory, or file exclusion from search results. This keyword may be used either in conjunction with, or independently of the <code>include_paths</code> keyword. This is particularly helpful in cases where a particular drive, directory, or file must be excluded from</p>




Keyword	Required	Description
		search results. Paths are double-quoted and separated by the pipe symbol where multiple paths are required.
see_also	no	This keyword allows to include links to a reference.  Example:  see_also: "example.com"
solution	no	This keyword provides a way to include “Solution” text if available.  Example:  solution: "Remove this file if it's not required"
reference	no	This keyword provides a way to include cross-references in the .audit. The format is “ref ref-id1,ref ref-id2”.  Example:  reference: "CAT CAT II,800-53 IA-5,8500.2 IAIA-1,8500.2 IAIA-2,8500.2 IATS-1,8500.2 IATS-2"
luhn	no	Setting <b>luhn</b> to YES forces the plugin to only report credit card numbers that are Luhn algorithm verified.

## Usage

```
<item>
type: FILE_CONTENT_CHECK
description: ["value data"]
file_extension: ["value data"]
(optional) regex: ["value data"]
(optional) expect: ["value data"]
(optional) file_name: ["value data"]
(optional) max_size: ["value data"]
```

---



```
(optional) only_show: ["value data"]
(optional) regex_replace: ["value data"]
(optional) luhn: ["value data"]
</item>
```

## Unix Content Command Line Examples

In this section, we will create a fake text document with a `.tns` extension and then run several simple to complex `.audit` files against it. As we go through each example, we will try each supported case of the File Content parameters.

We will also use the `nasl` command line binary. For each of the `.audit` files, you can easily drop these into your scan policies, but for quick audits of one system, this way is very efficient. The command we will execute each time from the `/opt/nessus/bin` directory will be:

```
# ./nasl -t <IP> /opt/nessus/lib/nessus/plugins/ unix_file_content_compliance_
check.nbin
```

The `<IP>` is the IP address of the system you will be auditing.

With Nessus, when running the `.nbin` (or any other plugin), it will prompt you for the credentials of the target system, plus the location of the `.audit` file.

This section includes the following information:

- [Target Test File](#)
- [Search Files for Properly Formatted VISA Credit Card Numbers](#)
- [Search for AMEX Credit Card Numbers](#)
- [Auditing Different Types of File Formats](#)
- [Performance Considerations](#)

## Target Test File

The target file we will be using contains the following content:

```
abcdefghijklmnopqrstuvwxyz
01234567890
```



```
Tenable Network Security
SecurityCenter
Nessus
Passive Vulnerability Scanner
Log Correlation Engine
AB12CD34EF56
Nessus
```

Please take this data and copy it to any Unix system you have credentialed access to. Name the file "Tenable\_Content.tns".

## Search Files for Properly Formatted VISA Credit Card Numbers

Following is a simple `.audit` file that looks for a list of file types that contain a properly formatted VISA credit card number. This audit does not use the Luhn algorithm to verify they are valid.

```
<item>
type: FILE_CONTENT_CHECK
description: "Determine if a file contains a properly formatted VISA credit card
number."
file_extension: "pdf" | "doc" | "xls" | "xlsx" | "xls" | "xlsb" | "xml" | "xlt" |
"xltm" | "docx" | "docm" | "dotx" | "dot" | "txt"
regex: "([^\0-9-]|^)(4[0-9]{3}(|-|)([0-9]{4})(|-|)([0-9]{4})(|-|)([0-9]{4}))([^\0-
9-]|$)"
regex_replace: "\3"
expect: "VISA" | "credit" | "Visa" | "CCN"
#luhn: YES
include_paths : "/home/mehul/foo"
max_size : "50K"
only_show : "4"
</item>
```

When running this command, the following output is expected:

```
Path: /home/brave/cc.txt      ('XXXXXXXXXXXX1111', 'XXXXXXXXXXXX1881')
Path: /home/snout/foo/email.txt ('XXXXXXXXXXXX4931', 'XXXXXXXXXXXX4932',
'XXXXXXXXXXXX4934', 'XXXXXXXXXXXX4935', 'XXXXXXXXXXXX4936')
Path: /home/twins/mylist.txt  ('XXXXXXXXXXXX4931', 'XXXXXXXXXXXX4932',
'XXXXXXXXXXXX4934', 'XXXXXXXXXXXX4935', 'XXXXXXXXXXXX4936')
```





```
Path: /root/cc.txt      ('XXXXXXXXXXXX1270', 'XXXXXXXXXXXX4023', 'XXXXXXXXXXXX5925',  
'XXXXXXXXXXXX4932')  
Path: /root/cc1.txt     ('XXXXXXXXXXXX5925')
```

These results show that we found a match. The report says we “failed” because we found data we consider an issue. For example, if you are doing an audit for a credit card number and had a positive match of the credit card number on the public computer, although the match is positive, it is logged as a failure for compliance reasons.

## Search for AMEX Credit Card Numbers

Following is a simple **.audit** file that looks for a list of file types that contain a properly formatted AMEX credit card number.

```
<item>  
type: FILE_CONTENT_CHECK  
file_extension: 'pdf', 'doc', 'xls', 'xlsx', 'xls', 'xlsb', 'xml', 'xltx', 'xltm',  
'docx', 'docm', 'dotx', 'dot', 'txt'  
exclude_paths: '/root/unix_file_content_test_files/non'  
regex: ([^0-9-]|^)([0-9]{3}-[0-9]{2}-[0-9]{4})([^0-9-]|$)  
regex_replace: \3  
only_show: 4  
expect: 'American Express', 'CCAX', 'amex', 'credit', 'AMEX', 'CCN'  
max_size: 51200  
</item>
```

The output we get this time is as follows:

```
No files were found to be in violation.
```

We were able to “pass” the audit because none of the files we audited contained an AMEX credit card number.

## Auditing Different Types of File Formats

Any file extension may be audited; however, files such as **.zip** and **.gz** are not decompressed on the fly. If your file has compression or some sort of encoding in the data, pattern searching may not be possible.



For documents that store data in Unicode format, the parsing routines of the `.nbm` file will string out all “NULL” bytes that are encountered.

Last, support for various types of PDF file formats is included. Tenable has written an extensive PDF analyzer that extracts raw strings for matching. Users should only concern themselves for what sort of data they want to look for in a PDF file.

## Performance Considerations

There are several trade-offs that any organization needs to consider when modifying the default `.audit` files and testing them on live networks:

- Which extensions should we search for?
- How much data should be scanned?

The `.audit` files do not require the `max_size` keyword. In this case, Nessus attempts to retrieve the entire file and will continue unless it has a match on a pattern. Since these files traverse the network, there is more network traffic with these audits than with typical scanning or configuration auditing.

If multiple Nessus scanners are being managed by Tenable Security Center, the data only needs to travel from the scanned Unix host to the scanner performing the vulnerability audit.

## Performance Considerations

File content auditing is a very resource intensive operation. All file systems need to be crawled, every directory visited, and every file of interest analyzed. There are several trade-offs that any organization needs to consider when modifying the default `.audit` files and testing them on live networks.

## Start small and build up

When modifying a published audit, or creating a custom audit, it pays to start small. Starting small on a focused set of files eases the manual work, resources, and reduces the overall time it takes. Use the following steps:

1. Plant a file that should be found.
2. Place terms in the planted file that match terms that the policy expects.



3. Place the actual value that the policy regular expression should match.
4. Update the audit file to include only the path where you planted the file.
5. If there are multiple policies, make sure all policies include the path.
6. Run a scan with the new audit file to get it working to find the planted file.
7. Pull the include restriction back to get more directories scanned.
8. Repeat the scan.

This method helps with understanding what it takes to find files, and the resources and time it takes for a small scan to complete.

## Which extensions should we search for?

The types of files being scanned is a factor in overall performance.

File types that are based on text encoding are the quickest and easiest to find content in. These file types tend to be text files, XML files, and JSON files.

File types that are compressed containers for other files must go through additional processing. Most productivity documents, such as Microsoft Word and Microsoft Excel, are collections of XML and other files in a zip archive. These complex documents must be uncompressed, and then evaluated. If any of the uncompressed parts of the document are too large, they may be skipped for evaluation.

Standard compressed archives, such as .zip and .gz, are not uncompressed and are not evaluated.

Binary file types are the hardest to work with. If there is a supported decoding of the binary data available in the file content plugins, there is a possibility for evaluation. If the binary files go beyond encoding and into encryption, it is not feasible to evaluate the files. The most notable of these types of files are PDF files, which have binary encoded objects for maintaining consistent presentation and security.

## How much data should be scanned?

If a `max_size` is not specified, Tenable Nessus will attempt to retrieve entire files. Since these files traverse the network, there is more network traffic with these audits than with typical scanning or



configuration auditing. By specifying a `max_size` in the audit files, the amount of data transferred over the network can be limited.

If multiple Tenable Nessus scanners are being managed by Tenable Security Center, the data only needs to travel from the scanned host to the scanner performing the vulnerability audit.

## How many policies are being evaluated?

A single audit file is a container of one or more policies that the file content scans will evaluate. While it is tempting to "shoot the works" and add a bunch of audit files to a scan, it is also a hindrance that can suffer performance impact.

For each file that is found as a target to be evaluated, that file has to be compared against each policy that was added to understand if the file applies to the policy and if the content should be evaluated.

It is beneficial to select audits that are targeted to what you are evaluating.

## Can an agent be used?

One of the largest bottle necks for file content scanning is the network communication and bandwidth. If the targets that are being scanned are able to have a Tenable Nessus Agent installed, then installing and using the agent for file content scanning will drastically speed up the scanning process. If agent scanning is not available in your environment, the Powershell File Enumeration would be another good option to reduce network traffic.