

# Nessus Compliance Checks Reference

April 11, 2016

---

## Table of Contents

Introduction .....	10
Prerequisites .....	10
Standards and Conventions.....	10
Tips on String Matching .....	10
Windows Configuration Audit Compliance File Reference.....	11
Check Type .....	11
Value Data.....	12
Data Types.....	12
Complex Expressions .....	12
The “check_type” Field .....	13
The “group_policy” Field.....	13
The “info” Field.....	14
The “debug” Field .....	15
ACL Format.....	15
File Access Control Checks .....	15
Registry Access Control Checks.....	17
Service Access Control Checks.....	18
Launch Permission Control Checks .....	19
Launch2 Permission Control Checks.....	21
Access Permission Control Checks .....	22
Custom Items .....	23
PASSWORD_POLICY.....	23
LOCKOUT_POLICY .....	24
KERBEROS_POLICY.....	25
AUDIT_POLICY .....	26
AUDIT_POLICY_SUBCATEGORY .....	27
AUDIT_POWERSHELL .....	29
AUDIT_FILEHASH_POWERSHELL.....	33
AUDIT_IIS_APPCMD.....	34
AUDIT_ALLOWED_OPEN_PORTS .....	36
AUDIT_DENIED_OPEN_PORTS .....	36
AUDIT_PROCESS_ON_PORT .....	37
AUDIT_USER_TIMESTAMPS .....	39



CHECK_ACCOUNT .....	40
CHECK_LOCAL_GROUP .....	42
ANONYMOUS_SID_SETTING .....	43
SERVICE_POLICY .....	44
GROUP_MEMBERS_POLICY .....	45
USER_GROUPS_POLICY .....	46
USER_RIGHTS_POLICY .....	46
FILE_CHECK .....	48
FILE_VERSION .....	49
FILE_PERMISSIONS .....	50
FILE_AUDIT .....	51
FILE_CONTENT_CHECK .....	53
FILE_CONTENT_CHECK_NOT .....	54
REG_CHECK .....	55
REGISTRY_SETTING .....	56
REGISTRY_PERMISSIONS .....	60
REGISTRY_AUDIT .....	61
REGISTRY_TYPE .....	62
SERVICE_PERMISSIONS .....	63
SERVICE_AUDIT .....	65
WMI_POLICY .....	66
Items.....	68
Predefined Policies.....	68
Forced Reporting.....	75
Conditions .....	75
Windows Content Audit Compliance File Reference .....	77
Check Type .....	77
Item Format.....	78
Command Line Examples.....	80
Target Test File .....	80
Example 1: Search for .tns documents that contain the word “Nessus” .....	81
Example 2: Search for .tns documents that contain the word “France” .....	81
Example 3: Search for .tns and .doc documents that contain the word “Nessus” .....	82
Example 4: Search for .tns and .doc documents that contain the word “Nessus” and have an 11 digit number in them .....	82
Example 5: Search for .tns and .doc documents that contain the word “Nessus” and have an 11 digit number in them, but only display last 4 bytes.....	83



Example 6: Search for .tns documents that contain the word “Correlation” in the first 50 bytes .....	84
Example 7: Controlling what is displayed in output .....	84
Example 8: Using the file name as a filter .....	85
Example 9: Using the inclusion/exclusion keywords .....	86
Auditing Different Types of File Formats .....	87
Performance Considerations .....	87
Cisco IOS Configuration Audit Compliance File Reference .....	88
Check Type .....	88
Keywords.....	88
Command Line Examples.....	91
Example 1: Search for a Defined SNMP ACL .....	91
Example 2: Make Sure the “finger” Service is Disabled .....	92
Example 3: Randomness Check to Verify SNMP Community Strings and Access Control are Sufficiently Random .....	93
Example 4: Context Check to Verify SSH Access Control .....	94
Conditions .....	95
Juniper Junos Configuration Audit Compliance File Reference.....	96
Check Type: CONFIG_CHECK .....	96
Keywords.....	96
CONFIG_CHECK Examples.....	98
Check Type: SHOW_CONFIG_CHECK .....	99
Keywords.....	99
SHOW_CONFIG_CHECK Examples .....	102
Conditions .....	103
Reporting.....	104
Check Point GAIa Configuration Audit Compliance File Reference .....	105
Check Type: CONFIG_CHECK .....	105
Keywords.....	105
CONFIG_CHECK Examples.....	107
Conditions .....	107
Reporting.....	108
Palo Alto Firewall Configuration Audit Compliance File Reference.....	109
AUDIT_XML.....	109
AUDIT_REPORTS.....	110
Keywords.....	112
Citrix XenServer Audit Compliance File Reference.....	113
Check Type: AUDIT_XE.....	114



Keywords.....	114
HP ProCurve Audit Compliance File Reference.....	117
Check Types .....	117
Keywords.....	118
FireEye Audit Compliance File Reference .....	120
Check Types .....	120
Keywords.....	120
Brocade Fabric OS (FOS) Compliance File Reference .....	123
Syntax .....	123
Huawei VRP Compliance File Reference.....	124
Syntax .....	124
Dell Force10 Compliance File Reference .....	125
Syntax .....	126
Fortinet FortiOS Audit Compliance File Reference.....	127
Syntax .....	127
no regex, expect, or not_expect .....	128
regex only.....	128
expect only .....	128
not_expect only.....	128
regex and expect .....	129
regex and not_expect .....	129
context .....	129
cmd .....	130
Amazon AWS Compliance File Reference .....	130
Audit File Syntax.....	130
Keywords.....	130
Debugging .....	132
Known Good Auditing.....	132
Use Cases .....	133
Miscellaneous.....	133
Adtran AOS Compliance File Reference.....	134
Syntax .....	134
SonicWALL SonicOS Compliance File Reference .....	135
Syntax .....	135
Extreme ExtremeXOS Compliance File Reference .....	136



Syntax .....	136
Red Hat Enterprise Virtualization (RHEV) Compliance File Reference .....	137
Syntax .....	137
Debugging .....	138
MongoDB Compliance File Reference .....	138
Syntax .....	139
OpenStack .....	140
Syntax .....	140
Example Queries .....	141
Salesforce Compliance File Reference .....	142
Setup Requirements .....	143
Adding a trusted IP range .....	143
Using a security token .....	143
Syntax .....	144
Example Queries .....	144
BlueCoat ProxySG Compliance File Reference .....	145
Syntax .....	145
Context.....	145
Database Configuration Audit Compliance File Reference .....	146
Check Type .....	146
Keywords.....	146
Command Line Examples.....	148
Example 1: Search for logins with no expiration date .....	148
Example 2: Check enabled state of unauthorized stored procedure.....	149
Example 3: Check database state with mixed result sql_types.....	149
Conditions .....	150
Unix Configuration Audit Compliance File Reference.....	151
Check Type .....	151
Keywords.....	151
Custom Items .....	157
AUDIT_XML.....	157
AUDIT_ALLOWED_OPEN_PORTS .....	158
AUDIT_DENIED_OPEN_PORTS .....	158
AUDIT_PROCESS_ON_PORT .....	159
CHKCONFIG.....	159



CMD_EXEC .....	159
FILE_CHECK .....	160
FILE_CHECK_NOT .....	161
FILE_CONTENT_CHECK .....	162
FILE_CONTENT_CHECK_NOT .....	163
GRAMMAR_CHECK .....	164
MACOSX_DEFAULTS_READ .....	164
PKG_CHECK .....	165
PROCESS_CHECK .....	166
RPM_CHECK .....	166
SVC_PROP .....	167
XINETD_SVC .....	168
Built-In Checks .....	168
Password Management .....	168
min_password_length .....	169
max_password_age .....	169
min_password_age .....	170
Root Access .....	171
root_login_from_console .....	171
Permissions Management .....	172
accounts_bad_home_permissions .....	172
accounts_bad_home_group_permissions .....	172
accounts_without_home_dir .....	172
active_accounts_without_home_dir .....	173
invalid_login_shells .....	173
login_shells_with_suid .....	174
login_shells_writeable .....	174
login_shells_bad_owner .....	174
Password File Management .....	175
passwd_file_consistency .....	175
passwd_zero_uid .....	175
passwd_duplicate_uid .....	176
passwd_duplicate_gid .....	176
passwd_duplicate_username .....	176
passwd_duplicate_home .....	177
passwd_shadowed .....	177



passwd_invalid_gid.....	178
Group File Management.....	178
group_file_consistency .....	178
group_zero_gid .....	178
group_duplicate_name .....	179
group_duplicate_gid .....	179
group_duplicate_members.....	179
group_nonexistant_users .....	180
Root Environment .....	180
dot_in_root_path_variable.....	180
writeable_dirs_in_root_path_variable.....	181
File Permissions.....	181
find_orphan_files .....	181
find_world_writeable_files .....	182
find_world_writeable_directories .....	183
find_world_readable_files.....	183
find_suid_sgid_files.....	184
home_dir_localization_files_user_check.....	185
home_dir_localization_files_group_check .....	185
Suspicious File Content .....	186
admin_accounts_in_ftpusers .....	186
Unnecessary Files .....	186
find_pre-CIS_files .....	186
Conditions .....	187
Unix Content Audit Compliance File Reference .....	188
Check Type .....	188
Item Format.....	188
Command Line Examples.....	191
Target Test File .....	191
Example 1: Search files for properly formatted VISA credit card numbers .....	191
Example 2: Search files for a properly formatted AMEX credit card number .....	192
Auditing Different Types of File Formats.....	193
Performance Considerations .....	193
NetApp Data ONTAP .....	193
Required User Privileges .....	194
Check Type: CONFIG_CHECK .....	195





Keywords.....	195
CONFIG_CHECK Examples.....	196
Conditions .....	197
Reporting.....	198
<b>IBM iSeries Configuration Audit Compliance File Reference .....</b>	<b>198</b>
Required User Privileges .....	198
Check Type .....	199
Keywords.....	199
Custom Items.....	200
AUDIT_SYSTEMVAL .....	200
SHOW_SYSTEMVAL .....	201
Conditions .....	201
<b>VMware vCenter/ESXi Configuration Audit Compliance File Reference .....</b>	<b>202</b>
Requirements.....	202
Supported Versions .....	202
Check Type .....	203
AUDIT_VM .....	203
AUDIT_ESX .....	203
AUDIT_VCENTER.....	203
Keywords.....	204
Additional Notes .....	206
<b>For Further Information .....</b>	<b>206</b>
<b>About Tenable Network Security .....</b>	<b>207</b>
<b>Appendix A: Example Unix Compliance File .....</b>	<b>208</b>
<b>Appendix B: Example Windows Compliance File.....</b>	<b>215</b>
<b>Appendix C: XSL Transform to .audit Conversion .....</b>	<b>217</b>
Step 1: Install xsltproc.....	217
Step 2: Identify the XML File to Use .....	217
Step 3: Become Familiar with XSL Transforms and XPath .....	217
Step 4: Create the XSLT Transform .....	218
Step 5: Verify the XSLT Transform Works .....	218
Step 6: Copy the XSLT to the .audit.....	219
Step 7: Final Audit .....	219

---

## Introduction

This document describes the syntax used to create custom `.audit` files that can be used to audit the configuration of Unix, Windows, database, SCADA, IBM iSeries, and Cisco systems against a compliance policy as well as search the contents of various systems for sensitive content.



This guide is intended to assist with the manual creation and understanding of compliance audit file syntax. Please reference the Nessus Compliance Checks for a higher-level view of how Tenable compliance checks work.



Nessus supports SCADA system auditing; however, this functionality is outside of the scope of this document. Please reference the Tenable SCADA information page [here](#) for more information.

## Prerequisites

This document assumes some level of knowledge about the Nessus vulnerability scanner along with a detailed understanding of the target systems being audited. For more information on how Nessus can be configured to perform local Unix and Windows patch audits, please refer to the Nessus User Guide available at <https://docs.tenable.com/nessus/>.

## Standards and Conventions

Throughout the documentation, filenames, daemons, and executables are indicated with a **`courier`** font.

Command line options and keywords are also indicated with the **`courier`** font. Command line examples may or may not include the command line prompt and output text from the results of the command. Command line examples will display the command being run in **`courier`** to indicate what the user typed while the sample output generated by the system will be indicated in `courier` (not bold). Following is an example running of the Unix **`pwd`** command:

```
# pwd
/home/test/
#
```



Important notes and considerations are highlighted with this symbol and grey text boxes.



Tips, examples, and best practices are highlighted with this symbol and white on blue text.

## Tips on String Matching

As a general rule, where possible it's best and the most accurate (along with being easier to write and troubleshoot) if you confine the matching to a single line of the message. There are several ways to handle special cases:



### Quote Usage:

Single quotes and double quotes are interchangeable when surrounding audit fields, except in the following two cases:

1. In Windows compliance checks where special fields such as CRLF, etc. must be interpreted literally, use single-quotes. Any embedded fields that are to be interpreted as strings must be escaped out.

For example:

```
expect: 'First line\r\nSecond line\r\nJohn\'s Line'
```

2. Double-quotes are required when using the FileContent “include\_paths” and exclude\_paths”.

If using strings in any field type (description, value\_data, regex, etc.) that contain single or double quotes, there are two ways to handle them:

- a. Use the opposite quote type for the outermost enclosing quotes.

For example:

```
expect: "This is John's Line"
```

```
expect: 'We are looking for a double-quote-".*'
```

- b. Escape out any embedded quotes with a backslash (double-quotes only).

For example:

```
expect: "\"Text to be searched\""
```

3. Escaping a single character can be done so it matches the literal character rather than the normal regex interpretation of any single character.

For example:

```
expect: "Find this line\\. Even if it has periods\\."
```

## Windows Configuration Audit Compliance File Reference

The basis for Windows **.audit** compliance files is a specially formatted text file. Entries in the file can invoke a variety of “custom item” checks such as registry setting checks, as well as more generic ones such as local security policy setting checks. Examples are used throughout this guide for clarification.

### Check Type

All Windows compliance checks must be bracketed with the **check\_type** encapsulation with the “Windows” designation and also specify version “2”:

```
<check_type:"Windows" version:"2">
```

An example Windows compliance check can be seen in “Appendix B”, starting with the **check\_type** setting for “Windows” and version “2”, and is finished by the “**</check\_type>**” tag.

This is required to differentiate Windows **.audit** files from those intended for Unix (or other platforms).

## Value Data

The `.audit` file syntax contains keywords that can be assigned various value types to customize your checks. This section describes these keywords and the format of the data that can be entered.

### Data Types

The following types of data can be entered for the checks:

Data Type	Description
DWORD	0 to 2,147,483,647
RANGE [X..Y]	Where X is a DWORD or MIN and Y is a DWORD or MAX

Examples:

```
value_data: 45
value_data: [11..9841]
value_data: [45..MAX]
```

In addition, numbers can be specified with plus (+) or minus (-) to indicate their “sign” and be specified as hexadecimal values. Hexadecimal and signs can be combined. The following are valid examples (without the corresponding label in parentheses) within a `REGISTRY_SETTING` audit for a `POLICY_DWORD`:

```
value_data: -1    (signed)
value_data: +10   (signed)
value_data: 10    (unsigned)
value_data: 2401649476 (unsigned)
value_data: [MIN..+10] (signed range)
value_data: [20..MAX] (unsigned range)
value_data: 0x800010AB (unsigned hex)
value_data: -0x10 (signed hex)
```

### Complex Expressions

Complex expressions can be used for the `value_data` field by using:

- `||`: conditional OR
- `&&`: conditional AND
- `|`: binary OR (bit operation)
- `&`: binary AND (bit operation)
- `( and )`: to delimitate complex expressions

Examples:

```
value_data: 45 || 10
value_data: (45 || 10) && ([9..12] || 37)
```

## The “check\_type” Field

This check type is different than the “**check\_type**” field specified above that is used at the beginning of each audit file to denote the generic audit type (Windows, FileContent, Unix, Database, Cisco). It is optional and can be performed against Windows **value\_data** values to determine the type of check to be performed. The following settings are available:

- **CHECK\_EQUAL**: compare the remote value against the policy value (default if **check\_type** is missing)
- **CHECK\_EQUAL\_ANY**: checks that each element of **value\_data** is at least present once in the system list
- **CHECK\_NOT\_EQUAL**: checks the remote value is different than the policy value
- **CHECK\_GREATER\_THAN**: checks the remote value is greater than the policy value
- **CHECK\_GREATER\_THAN\_OR\_EQUAL**: checks the remote value is greater or equal than the policy value
- **CHECK\_LESS\_THAN**: checks the remote value is less than the policy value
- **CHECK\_LESS\_THAN\_OR\_EQUAL**: checks the remote value is less or equal than the policy value
- **CHECK\_REGEX**: checks that the remote value match the regex in the policy value (only works with **POLICY\_TEXT** and **POLICY\_MULTI\_TEXT**)
- **CHECK\_SUBSET**: checks that the remote ACL is a subset of the policy ACL (only works with ACLs)
- **CHECK\_SUPERSET**: checks that the remote ACL is a superset of the policy ACL (only works with deny rights ACLs)

Following is an example audit to check to make sure that the account name “Guest” does not exist for any Guest account.

```
<custom_item>
  type: CHECK_ACCOUNT
  description: "Accounts: Rename guest account"
  value_type: POLICY_TEXT
  value_data: "Guest"
  account_type: GUEST_ACCOUNT
  check_type: CHECK_NOT_EQUAL
</custom_item>
```

If any other value besides “Guest” is present, the test will pass. If “Guest” is found, the audit will fail.

## The “group\_policy” Field

The “**group\_policy**” field can be used to provide a short text string that describes the audit. The **group\_policy** must be included in an audit file, and should be inserted after the **check\_type** field.

```
<check_type: "Windows" version:"2">
<group_policy: "Audit file for Windows 2008">
```



```
...  
</group_policy>  
</check_type>
```

## The “info” Field

The optional “**info**” field can be used to label each audit field with one or more external references. For example, this field will be used to place references from NIST CCE tags as well as CIS specific audit requirements. These external references are printed out in the final audit performed by Nessus and will be displayed in the Nessus report or through SecurityCenter’s user interface.

Following is an example password audit policy that has been augmented to list references to a fictitious corporate policy:

```
<custom_item>  
  type: PASSWORD_POLICY  
  description: "Password History: 24 passwords remembered"  
  value_type: POLICY_DWORD  
  value_data: [22..MAX] || 20  
  password_policy: ENFORCE_PASSWORD_HISTORY  
  info: "Corporate Policy 102-A"  
</custom_item>
```

If multiple policy references are required for a single audit, the string specified by the “**info**” keyword can make use of the “\n” separator to specify multiple strings. For example, consider the following audit:

```
<custom_item>  
  type: CHECK_ACCOUNT  
  description: "Accounts: Rename Administrator account"  
  value_type: POLICY_TEXT  
  value_data: "Administrator"  
  account_type: ADMINISTRATOR_ACCOUNT  
  check_type: CHECK_NOT_EQUAL  
  info: 'Ron Gula Mambo Number 5\nCCE-60\nTenable Best Practices Policy 1005-a'  
</custom_item>
```

When run with the **nas1** command line tool, this audit function produces the following output:

```
# /opt/nessus/bin/nas1 -t 192.168.20.16 ./compliance_check.nbin  
  
Windows Compliance Checks, version 2.0.0  
  
Which file contains your security policy : ./test_v2.audit  
SMB login : Administrator  
SMB password :  
SMB domain (optional) :  
"Accounts: Rename Administrator account": [FAILED]  
  
Ron Gula Mambo Number 5  
CCE-60  
Tenable Best Practices Policy 1005-a
```

```
Remote value: "Administrator"
Policy value: "administrator"
```

## The “debug” Field

The optional “**debug**” field can be used to troubleshoot Windows content compliance checks. The debug keyword outputs information about the content scan being conducted, such as file(s) being processed, scanned and whether any results were found. Due to the large amount of output this keyword should only be used for troubleshooting purposes. For example:

```
<item>
  debug
  type: FILE_CONTENT_CHECK
  description: "TNS File that Contains the word Nessus"
  file_extension: "tns"
  expect: "Nessus"
</item>
```

## ACL Format

This section describes the syntax used to determine if a file or folder has the desired ACL setting.

### File Access Control Checks

#### Usage

```
<file_acl: ["name"]>

  <user: ["user_name"]>
    acl_inheritance: ["value"]
    acl_apply: ["value"]
    (optional) acl_allow: ["rights value"]
    (optional) acl_deny: ["rights value"]
  </user>

</acl>
```

A file Access Control List (ACL) is identified by the keyword **file\_acl**. The ACL name must be unique to be used with a file permissions item. A file ACL can contain one or multiple user entry.

Associated Types	Allowed Types
<b>acl_inheritance</b>	<ul style="list-style-type: none"><li>not inherited</li><li>inherited</li><li>not used</li></ul>
<b>acl_apply</b>	<ul style="list-style-type: none"><li>this folder only</li><li>this object only</li><li>this folder and files</li><li>this folder and subfolders</li><li>this folder, subfolders and files</li></ul>



	<ul style="list-style-type: none"><li>• files only</li><li>• subfolders only</li><li>• subfolders and files only</li></ul>
<b>acl_allow</b> <b>acl_deny</b>	<p>These settings are optional.</p> <p>Generic rights are:</p> <ul style="list-style-type: none"><li>• full control</li><li>• modify</li><li>• read &amp; execute</li><li>• read</li><li>• write</li><li>• list folder contents</li></ul> <p>Advanced rights are:</p> <ul style="list-style-type: none"><li>• full control</li><li>• traverse folder / execute file</li><li>• list folder / read data</li><li>• read attributes</li><li>• read extended attributes</li><li>• create files / write data</li><li>• create folders / append data</li><li>• write attributes</li><li>• write extended attributes</li><li>• delete subfolder and files</li><li>• delete</li><li>• read permissions</li><li>• change permissions</li><li>• take ownership</li></ul>

Here is an example file access control **.audit** text:

```
<file acl: "ASU1">

<user: "Administrators">
  acl_inheritance: "not inherited"
  acl_apply: "This folder, subfolders and files"
  acl_allow: "Full Control"
</user>

<user: "System">
  acl_inheritance: "not inherited"
  acl_apply: "This folder, subfolders and files"
  acl_allow: "Full Control"
</user>

<user: "Users">
  acl_inheritance: "not inherited"
  acl_apply: "this folder only"
  acl_allow: "list folder / read data" | "read attributes" | "read extended
    attributes" | "create files / write data" | "create folders / append data" |
    "write attributes" | "write extended attributes" | "read permissions"
</user>
```





```
</acl>
```

## Registry Access Control Checks

### Usage

```
<registry_acl: ["name"]>

  <user: ["user_name"]>
    acl_inheritance: ["value"]
    acl_apply: ["value"]
    (optional) acl_allow: ["rights value"]
    (optional) acl_deny: ["rights value"]
  </user>

</acl>
```

A registry ACL is identified by the keyword **registry\_acl**. The ACL name must be unique to be used with a registry permissions item. A registry ACL can contain one or multiple user entry.

Associated Types	Allowed Types
<b>acl_inheritance</b>	<ul style="list-style-type: none"><li>• not inherited</li><li>• inherited</li><li>• not used</li></ul>
<b>acl_apply</b>	<ul style="list-style-type: none"><li>• this key only</li><li>• this key and subkeys</li><li>• subkeys only</li></ul>
<b>acl_allow</b> <b>acl_deny</b>	<p>These settings are optional and are used to define the rights a user has on the object.</p> <p>Generic rights are:</p> <ul style="list-style-type: none"><li>• full control</li><li>• read</li></ul> <p>Advanced rights are:</p> <ul style="list-style-type: none"><li>• full control</li><li>• query value</li><li>• set value</li><li>• create subkey</li><li>• enumerate subkeys</li><li>• notify</li><li>• create link</li><li>• delete</li><li>• write dac</li><li>• write owner</li><li>• read control</li></ul>

Here is an example registry access control list **.audit** text:

```
<registry_acl: "SOFTWARE ACL">

  <user: "Administrators">
    acl_inheritance: "not inherited"
    acl_apply: "This key and subkeys"
    acl_allow: "Full Control"
  </user>

  <user: "CREATOR OWNER">
    acl_inheritance: "not inherited"
    acl_apply: "Subkeys only"
    acl_allow: "Full Control"
  </user>

  <user: "SYSTEM">
    acl_inheritance: "not inherited"
    acl_apply: "This key and subkeys"
    acl_allow: "Full Control"
  </user>

  <user: "Users">
    acl_inheritance: "not inherited"
    acl_apply: "This key and subkeys"
    acl_allow: "Read"
  </user>

</acl>
```

## Service Access Control Checks

### Usage

```
<service_acl: ["name"]>

  <user: ["user_name"]>
    acl_inheritance: ["value"]
    acl_apply: ["value"]
    (optional) acl_allow: ["rights value"]
    (optional) acl_deny: ["rights value"]
  </user>

</acl>
```

A service ACL is identified by the keyword **service\_acl**. The ACL name must be unique to be used with a service permissions item. A service ACL can contain one or multiple user entry.

Associated Types	Allowed Types
<b>acl_inheritance</b>	<ul style="list-style-type: none"><li>not inherited</li><li>inherited</li><li>not used</li></ul>

<b>acl_apply</b>	<ul style="list-style-type: none"> <li>this object only</li> </ul>
<b>acl_allow</b> <b>acl_deny</b>	<p>These settings are optional and are used to define the rights a user has on the object.</p> <p>Generic rights are:</p> <ul style="list-style-type: none"> <li>full control</li> <li>read</li> <li>start, stop and pause</li> <li>write</li> <li>delete</li> </ul> <p>Advanced rights are:</p> <ul style="list-style-type: none"> <li>full control</li> <li>delete</li> <li>query template</li> <li>change template</li> <li>query status</li> <li>enumerate dependents</li> <li>start</li> <li>stop</li> <li>pause and continue</li> <li>interrogate</li> <li>user-defined control</li> <li>read permissions</li> <li>change permissions</li> <li>take ownership</li> </ul>

An example service access control check is shown below:

```
<service_acl: "ALERT ACL">

  <user: "Administrators">
    acl_inheritance: "not inherited"
    acl_apply: "This object only"
    acl_allow: "query template" | "change template" | "query status" | "enumerate
      dependents" | "start" | "stop" | "pause and continue" | "interrogate" | "user-
        defined control" | "delete" | "read permissions" | "change permissions" | "take
          ownership"
  </user>

</acl>
```

## Launch Permission Control Checks

Usage

```
<launch_acl: ["name"]>

  <user: ["user_name"]>
```

```

acl_inheritance: ["value"]
acl_apply: ["value"]
(optional) acl_allow: ["rights value"]
(optional) acl_deny: ["rights value"]
</user>

</acl>

```

A launch ACL is identified by the keyword **launch\_acl**. The ACL name must be unique to be used with a DCOM launch permissions item. A launch ACL can contain one or multiple user entry.

Associated Types	Allowed Types
<b>acl_inheritance</b>	<ul style="list-style-type: none"> <li>not inherited</li> <li>inherited</li> </ul>
<b>acl_apply</b>	<ul style="list-style-type: none"> <li>this object only</li> </ul>
<b>acl_allow</b> <b>acl_deny</b>	<p>These settings are optional and are used to define the rights a user has on the object.</p> <p>Generic rights are:</p> <ul style="list-style-type: none"> <li>local launch</li> <li>remote launch</li> <li>local activation</li> <li>remote activation</li> </ul>



This ACL only works against Windows XP/2003/Vista (and partially against Windows 2000).

An example launch access control check is shown below:

```

<launch_acl: "2">

<user: "Administrators">
  acl_inheritance: "not inherited"
  acl_apply: "This object only"
  acl_allow: "Remote Activation"
</user>

<user: "INTERACTIVE">
  acl_inheritance: "not inherited"
  acl_apply: "This object only"
  acl_allow: "Local Activation" | "Local Launch"
</user>

<user: "SYSTEM">
  acl_inheritance: "not inherited"
  acl_apply: "This object only"
  acl_allow: "Local Activation" | "Local Launch"
</user>

```

```
</acl>
```

## Launch2 Permission Control Checks

### Usage

```
<launch2_acl: ["name"]>

  <user: ["user_name"]>
    acl_inheritance: ["value"]
    acl_apply: ["value"]
    (optional) acl_allow: ["rights value"]
    (optional) acl_deny: ["rights value"]
  </user>

</acl>
```

A launch2 ACL is identified by the keyword **launch2\_acl**. The ACL name must be unique to be used with a DCOM launch permissions item. A launch2 ACL can contain one or multiple user entry.

Associated Types	Allowed Types
<b>acl_inheritance</b>	<ul style="list-style-type: none"><li>not inherited</li><li>inherited</li></ul>
<b>acl_apply</b>	<ul style="list-style-type: none"><li>this object only</li></ul>
<b>acl_allow</b> <b>acl_deny</b>	<p>These settings are optional and are used to define the rights a user has on the object.</p> <p>Generic rights are:</p> <ul style="list-style-type: none"><li>launch</li></ul>



Only use the launch2 ACL against Windows 2000 and NT systems.

An example launch access control check is shown below:

```
<launch2_acl: "2">

  <user: "Administrators">
    acl_inheritance: "not inherited"
    acl_apply: "This object only"
    acl_allow: "Launch"
  </user>

  <user: "INTERACTIVE">
    acl_inheritance: "not inherited"
```

```

    acl_apply: "This object only"
    acl_allow: "Launch"
  </user>

  <user: "SYSTEM">
    acl_inheritance: "not inherited"
    acl_apply: "This object only"
    acl_allow: "Launch"
  </user>

</acl>

```

## Access Permission Control Checks

### Usage

```

<access_acl: ["name"]>

  <user: ["user_name"]>
    acl_inheritance: ["value"]
    acl_apply: ["value"]
    (optional) acl_allow: ["rights value"]
    (optional) acl_deny: ["rights value"]
  </user>

</acl>

```

An access ACL is identified by the keyword **access\_acl**. The ACL name must be unique to be used with a DCOM access permissions item. An access ACL can contain one or multiple user entry.

Associated Types	Allowed Types
<b>acl_inheritance</b>	<ul style="list-style-type: none"> <li>not inherited</li> <li>inherited</li> </ul>
<b>acl_apply</b>	<ul style="list-style-type: none"> <li>this object only</li> </ul>
<b>acl_allow</b> <b>acl_deny</b>	<p>These settings are optional and are used to define the rights a user has on the object.</p> <p>Generic rights are:</p> <ul style="list-style-type: none"> <li>local access</li> <li>remote access</li> </ul>

An example access control check is shown below:

```

<access_acl: "3">

  <user: "SELF">
    acl_inheritance: "not inherited"
    acl_apply: "This object only"
  </user>

</acl>

```

```

    acl_allow: "Local Access"
  </user>

  <user: "SYSTEM">
    acl_inheritance: "not inherited"
    acl_apply: "This object only"
    acl_allow: "Local Access"
  </user>

  <user: "Users">
    acl_inheritance: "not inherited"
    acl_apply: "This object only"
    acl_allow: "Local Access"
  </user>

</acl>

```

## Custom Items

A custom item is a complete check defined on the basis of the keywords defined above. The following is a list of available custom item types. 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.



Custom audit checks may use “`</custom_item>`” and “`</item>`” interchangeably for the closing tag.

## PASSWORD\_POLICY

### Usage

```

<custom_item>
  type: PASSWORD_POLICY
  description: ["description"]
  value_type: [VALUE_TYPE]
  value_data: [value]
  (optional) check_type: [value]
  password_policy: [PASSWORD_POLICY_TYPE]
</custom_item>

```

This policy item checks for the values defined in “Windows Settings -> Security Settings -> Account Policies -> Password Policy”.

The check is performed by calling the function `NetUserModalsGet` with the level 1.

These items use the `password_policy` field to describe which element of the password policy must be audited. The allowed types are:

- `ENFORCE_PASSWORD_HISTORY` (“Enforce password history”)
  - `value_type`: `POLICY_DWORD`
  - `value_data`: `DWORD` or `RANGE` [number of remembered passwords]

- **MAXIMUM\_PASSWORD\_AGE** ("Maximum password age")  
value\_type: TIME\_DAY  
value\_data: DWORD or RANGE [time in days]
- **MINIMUM\_PASSWORD\_AGE** ("Minimum password age")  
value\_type: TIME\_DAY  
value\_data: DWORD or RANGE [time in days]
- **MINIMUM\_PASSWORD\_LENGTH** ("Minimum password length")  
value\_type: POLICY\_DWORD  
value\_data: DWORD or RANGE [minimum number of characters in the password]
- **COMPLEXITY\_REQUIREMENTS** ("Password must meet complexity requirements")  
value\_type: POLICY\_SET  
value\_data: "Enabled" or "Disabled"
- **REVERSIBLE\_ENCRYPTION** ("Store passwords using reversible encryption for all users in the domain")  
value\_type: POLICY\_SET  
value\_data: "Enabled" or "Disabled"
- **FORCE\_LOGOFF** ("Network security: Force logoff when logon hours expire")  
value\_type: POLICY\_SET  
value\_data: "Enabled" or "Disabled"



There is currently no way to check for the policy "Store password using reversible encryption for all users in the domain".

The FORCE\_LOGOFF policy is located in "Security Settings -> Local Policies -> Security Options".

The following is an example password policy audit:

```
<custom_item>
  type: PASSWORD_POLICY
  description: "Minimum password length"
  value_type: POLICY_DWORD
  value_data: 7
  password_policy: MINIMUM_PASSWORD_LENGTH
</custom_item>
```

## LOCKOUT\_POLICY

### Usage

```
<custom_item>
  type: LOCKOUT_POLICY
  description: ["description"]
  value_type: [VALUE_TYPE]
  value_data: [value]
  (optional) check_type: [value]
  lockout_policy: [LOCKOUT_POLICY_TYPE]
</custom_item>
```

This policy item checks for the values defined in "Security Settings -> Account Policies -> Account Lockout Policy".



The check is performed by calling the function **NetUserModalsGet** with the level 3.

This item uses the **lockout\_policy** field to describe which element of the password policy must be audited. The allowed types are:

- **LOCKOUT\_DURATION** ("Account lockout duration")  
value\_type: TIME\_MINUTE  
value\_data: DWORD or RANGE [time in minutes]
- **LOCKOUT\_THRESHOLD** ("Account lockout threshold")  
value\_type: POLICY\_DWORD  
value\_data: DWORD or RANGE [time in days]
- **LOCKOUT\_RESET** ("Reset lockout account counter after")  
value\_type: TIME\_MINUTE  
value\_data: DWORD or RANGE [time in minutes]

Here is an example:

```
<custom_item>
  type: LOCKOUT_POLICY
  description: "Reset lockout account counter after"
  value_type: TIME_MINUTE
  value_data: 120
  lockout_policy: LOCKOUT_RESET
</custom_item>
```

## KERBEROS\_POLICY

### Usage

```
<custom_item>
  type: KERBEROS_POLICY
  description: ["description"]
  value_type: [VALUE_TYPE]
  value_data: [value]
  (optional) check_type: [value]
  kerberos_policy: [KERBEROS_POLICY_TYPE]
</custom_item>
```

This policy item checks for the values defined in "Security Settings -> Account Policies -> Kerberos Policy".

The check is performed by calling the function **NetUserModalsGet** with the level 1.

This item uses the **kerberos\_policy** field to describe which element of the password policy must be audited. The allowed types are:

- **USER\_LOGON\_RESTRICTIONS** ("Enforce user logon restrictions")  
value\_type: POLICY\_SET  
value\_data: "Enabled" or "Disabled"
- **SERVICE\_TICKET\_LIFETIME** ("Maximum lifetime for service ticket")  
value\_type: TIME\_MINUTE  
value\_data: DWORD or RANGE [time in minutes]

- **USER\_TICKET\_LIFETIME** ("Maximum lifetime for user ticket")  
value\_type: TIME\_HOUR  
value\_data: DWORD or RANGE [time in hours]
- **USER\_TICKET\_RENEWAL\_LIFETIME** ("Maximum lifetime for user renewal ticket")  
value\_type: TIME\_DAY  
value\_data: DWORD or RANGE [time in day]
- **CLOCK\_SYNCHRONIZATION\_TOLERANCE** ("Maximum tolerance for computer clock synchronization")  
value\_type: TIME\_MINUTE  
value\_data: DWORD or RANGE [time in minute]



The Kerberos policy can only be checked against a KDC (Key Distribution Center), which, under Windows, is usually a Domain Controller.

Example:

```
<custom_item>
  type: KERBEROS_POLICY
  description: "Maximum lifetime for user renewal ticket"
  value_type: TIME_DAY
  value_data: 12
  kerberos_policy: USER_TICKET_RENEWAL_LIFETIME
</custom_item>
```

## AUDIT\_POLICY

### Usage

```
<custom_item>
  type: AUDIT_POLICY
  description: ["description"]
  value_type: [VALUE_TYPE]
  value_data: [value]
  (optional) check_type: [value]
  audit_policy: [PASSWORD_POLICY_TYPE]
</custom_item>
```

This policy item checks for the values defined in "Security Settings -> Local Policies -> Audit Policy".

The check is performed by calling the function **LsaQueryInformationPolicy** with the level **PolicyAuditEventsInformation**.

This item uses the **audit\_policy** field to describe which element of the password policy must be audited. The allowed types are:

- **AUDIT\_ACCOUNT\_LOGON** ("Audit account logon events")
- **AUDIT\_ACCOUNT\_MANAGER** ("Audit account management")
- **AUDIT\_DIRECTORY\_SERVICE\_ACCESS** ("Audit directory service access")
- **AUDIT\_LOGON** ("Audit logon events")

- AUDIT\_OBJECT\_ACCESS ("Audit object access")
- AUDIT\_POLICY\_CHANGE ("Audit policy change")
- AUDIT\_PRIVILEGE\_USE ("Audit privilege use")
- AUDIT\_DETAILED\_TRACKING ("Audit process tracking")
- AUDIT\_SYSTEM ("Audit system events")

value\_type: AUDIT\_SET

value\_data: "No auditing", "Success", "Failure", "Success, Failure"



Note that there is a required space in "Success, Failure".

Example:

```
<custom_item>
  type: AUDIT_POLICY
  description: "Audit policy change"
  value_type: AUDIT_SET
  value_data: "Failure"
  audit_policy: AUDIT_POLICY_CHANGE
</custom_item>
```

## AUDIT\_POLICY\_SUBCATEGORY

### Usage

```
<custom_item>
  type: AUDIT_POLICY_SUBCATEGORY
  description: ["description"]
  value_type: [VALUE_TYPE]
  value_data: [value]
  (optional) check_type: [value]
  audit_policy_subcategory: [SUBCATEGORY_POLICY_TYPE]
</custom_item>
```

This policy item checks for the values listed in `auditpol /get /category:*`.

The check is performed by executing `cmd.exe auditpol /get /category:*` via WMI.

This item uses the `audit_policy_subcategory` field to determine which subcategory needs be audited. The allowed SUBCATEGORY\_POLICY\_TYPE (s) are:

- Security State Change
- Security System Extension
- System Integrity
- IPsec Driver
- Other System Events
- Logon
- Logoff

- 
- Account Lockout
  - IPsec Main Mode
  - IPsec Quick Mode
  - IPsec Extended Mode
  - Special Logon
  - Other Logon/Logoff Events
  - Network Policy Server
  - File System
  - Registry
  - Kernel Object
  - SAM
  - Certification Services
  - Application Generated
  - Handle Manipulation
  - File Share
  - Filtering Platform Packet Drop
  - Filtering Platform Connection
  - Other Object Access Events
  - Sensitive Privilege Use
  - Non Sensitive Privilege Use
  - Other Privilege Use Events
  - Process Creation
  - Process Termination
  - DPAPI Activity
  - RPC Events
  - Audit Policy Change
  - Authentication Policy Change
  - Authorization Policy Change
  - MPSSVC Rule-Level Policy Change
  - Filtering Platform Policy Change
  - Other Policy Change Events
  - User Account Management
  - Computer Account Management
  - Security Group Management
  - Distribution Group Management
  - Application Group Management
  - Other Account Management Events
  - Directory Service Access
  - Directory Service Changes
  - Directory Service Replication
  - Detailed Directory Service Replication
  - Credential Validation
  - Kerberos Service Ticket Operations
  - Other Account Logon Events

value\_type: AUDIT\_SET

value\_data: "No auditing", "Success", "Failure", "Success, Failure"



Note that there is a required space in "Success, Failure".

This check is only applicable for Windows Vista/2008 Server and later. If a firewall is enabled, then in addition to adding WMI as an exception in the firewall settings, "Windows Firewall : Allow inbound remote administration exception" must also

---

be enabled in the firewall settings using `gpedit.msc`. This check may not work on non-English Vista/2008 systems or systems that do not have auditpol installed.

Example:

```
<custom_item>
  type: AUDIT_POLICY_SUBCATEGORY
  description: "AUDIT Security State Change"
  value_type: AUDIT_SET
  value_data: "success, failure"
  audit_policy_subcategory: "Security State Change"
</custom_item>
```

## AUDIT\_POWERSHELL

### Usage

```
<custom_item>
  type: AUDIT_POWERSHELL
  description: "Powershell check"
  value_type: [value_type]
  value_data: [value]
  powershell_args: ["arguments for powershell.exe"]
  (optional) only_show_cmd_output: YES or NO
  (optional) check_type: [CHECK_TYPE]
  (optional) severity: ["HIGH" or "MEDIUM" or "LOW"]
  (optional) powershell_option: CAN_BE_NULL
  (optional) powershell_console_file: "C:\Program Files\Microsoft\Exchange
Server\ExShell.pscl"
</custom_item>
```

This check runs `powershell.exe` on the remote server along with the arguments supplied with "`powershell_args`" and returns the command output if "`only_show_cmd_output`" is set to YES or compares the result against "`value_data`" if `value_data` is specified.

Associated types:

This item uses the field "`powershell_args`" to specify the arguments that need to be supplied to `powershell.exe`. If the location of `powershell.exe` is not default, you must use the `powershell_console_file` keyword to specify the location. Currently the only "get-" cmdlets are supported. For example:

- `get-hotfix | where-object {$_.hotfixid -ne 'File 1'} | select Description,HotFixID,InstalledBy | format-list`
- `get-wmiobject win32_service | select caption,name, state| format-list`
- `(get-WmiObject -namespace root\MicrosoftIISv2 -Class IIsWebService).ListWebServiceExtensions().Extensions`
- `get-wmiobject -namespace root\cimv2 -class win32_product | select Vendor,Name,Version | format-list`
- `get-wmiobject -namespace root\cimv2\power -class Win32_powerplan | select description,isactive | format-list`

---

The item uses optional field “**only\_show\_cmd\_output**” if the entire command output needs to be reported:

- **only\_show\_cmd\_output**: YES or NO

Other considerations:

1. If you set “**only\_show\_cmd\_output**” and would like to set the severity of the output, then you could use the severity tag to change the severity. The default is INFO.
2. Powershell is not installed by default on some Windows operating systems (e.g., XP, 2003), and on such systems this check would not yield any result. Therefore make sure Powershell is installed on the remote target before using this check.
3. For this check to work correctly, WMI service needs to be enabled. Also configure the firewall to “Allow inbound remote administration exception”.
4. Cmdlet aliases (e.g., “gps” instead of “Get-Process”) are not allowed.

Example:

This example runs the “Get-Hotfix” powershell cmdlet, specifies a where-object to not select hotfixes with id “File 1”, and then reports Description, HotfixID, Installedby formatted as a list.

```
<custom_item>
type: AUDIT_POWERSHELL
description: "Show Installed Hotfix"
value_type: POLICY_TEXT
value_data: ""
powershell_args: "get-hotfix | where-object {$_.hotfixid -ne 'File 1'} | select
    Description,HotFixID,InstalledBy | format-list"
only_show_cmd_output: YES
</custom_item>
```

Example:

This example checks whether the windows service “WinRM” is running.

```
<custom_item>
type: AUDIT_POWERSHELL
description: "Check if WinRM service is running"
value_type: POLICY_TEXT
value_data: "Running"
powershell_args: "get-wmiobject win32_service | where-object {$_.name -eq 'WinRM' -
    and $_.state -eq 'Running'} | select state"
check_type: CHECK_REGEX
</custom_item>
```

Nessus also allows a user to pass a PowerShell script (.ps1) encoded as a base64 string to **PowerShell.exe** via the **EncodedCommand** switch. The following example script lists local user account information on the target:

```
$strComputer = "."

$scolItems = get-wmiobject -class "Win32_UserAccount" -namespace "root\CIMV2" -filter
    "LocalAccount = True" -computername $strComputer
```

```

foreach ($objItem in $colItems) {
    write-host "Account Type: " $objItem.AccountType
    write-host "Description: " $objItem.Description
    write-host "Disabled: " $objItem.Disabled
    write-host "Full Name: " $objItem.FullName
    write-host "Installation Date: " $objItem.InstallDate
    write-host "Lockout: " $objItem.Lockout
    write-host "Password Changeable: " $objItem.PasswordChangeable
    write-host "Password Expires: " $objItem.PasswordExpires
    write-host "Password Required: " $objItem.PasswordRequired
    write-host "SID: " $objItem.SID
    write-host "SID Type: " $objItem.SIDType
    write-host "Status: " $objItem.Status
    write-host
}

```

To pass this script to PowerShell, you must encode it and then pass it as a PowerShell command. Begin by assigning the contents of the file to a string. The basic syntax is as follows:

```

$foo = {
add your PowerShell code here....
}

```

A full example would look like the following:

```

$string = {

$strComputer = "."

$colItems = get-wmiobject -class "Win32_UserAccount" -namespace "root\CIMV2" -filter
    "LocalAccount = True" -computername $strComputer

foreach ($objItem in $colItems) {
    write-host "Account Type: " $objItem.AccountType
    write-host "Description: " $objItem.Description
    write-host "Disabled: " $objItem.Disabled
    write-host "Full Name: " $objItem.FullName
    write-host "Installation Date: " $objItem.InstallDate
    write-host "Lockout: " $objItem.Lockout
    write-host "Password Changeable: " $objItem.PasswordChangeable
    write-host "Password Expires: " $objItem.PasswordExpires
    write-host "Password Required: " $objItem.PasswordRequired
    write-host "SID: " $objItem.SID
    write-host "SID Type: " $objItem.SIDType
    write-host "Status: " $objItem.Status
    write-host
}
}

```

Next, Base64 encode it:

```

PS C:\Documents and Settings\Administrator>

```

```
[System.Convert]::ToBase64String([System.Text.Encoding]::UNICODE.GetBytes($string))
```

Use your resulting Base64 string in an .audit file. Be sure to set **ps\_encoded\_args** to **YES**:

This is an example only:

```
<custom_item>
  type: AUDIT_POWERSHELL
  description: "List local user account info"
  value_type: POLICY_TEXT
  value_data: ""
  powershell_args:
    'DQAKACIAMQAwAC4AMAAuADAAIgAgAHwAIABXAHIAaQB0AGUALQBPAHUAdABwAHUAdAA7AA0ACgA='
  ps_encoded_args: YES
  only_show_cmd_output: YES
</custom_item>
```

After the .audit is run, the information displayed will appear similar to the following example.

This is an example only:

```
"List local user account info": [INFO]

Account Type: 512
Description: Built-in account for administering the computer/domain
Disabled: False
Full Name:
Installation Date:
Lockout: False
Password Changeable: True
Password Expires: False
Password Required: True
SID: S-1-5-21-2137291905-473285123-5405471365-500
SID Type: 1
Status: OK

Account Type: 512
Description: Account used for running the ASP.NET worker process (aspnet_wp.exe)
Disabled: False
Full Name: ASP.NET Machine Account
Installation Date:
Lockout: False
Password Changeable: False
Password Expires: False
Password Required: False
SID: S-1-5-21-2137291905-473285123-5405471365-1006
SID Type: 1
Status: OK
```



## AUDIT\_FILEHASH\_POWERSHELL

### Usage

```
<custom_item>
  type: AUDIT_FILEHASH_POWERSHELL
  description: "Powershell FileHash Check"
  value_type: POLICY_TEXT
  file: "[FILE]"
  value_data: "[FILE HASH]"
</custom_item>
```

This check runs **powershell.exe** on the remote server along with the information supplied to compare an expected file hash with the hash of the file on the system.

Other considerations:

- By default, an MD5 hash of the file is compared, however users can compare hashes generated with SHA1, SHA256, SHA384, SHA512, or RIPEMD160 algorithm.
- For the check to work, PowerShell must be installed, and WMI be enabled on the target.

Example:

This example compares a supplied MD5 hash against the file hash of `C:\test\test2.zip`.

```
<custom_item>
  type: AUDIT_FILEHASH_POWERSHELL
  description: "Audit FILEHASH - MD5"
  value_type: POLICY_TEXT
  file: "C:\test\test2.zip"
  value_data: "8E653F7040AC4EA8E315E838CEA83A04"
</custom_item>
```

Example:

This example compares a supplied SHA1 hash against the the file hash of `C:\test\test3.zip`.

```
<custom_item>
  type: AUDIT_FILEHASH_POWERSHELL
  description: "Audit FILEHASH - SHA1"
  value_type: POLICY_TEXT
  file: "C:\test\test3.zip"
  value_data: "0C4B0AF91F62ECCED3B16D35DE50F66746D6F48F"
  hash_algorithm: SHA1
</custom_item>
```

## AUDIT\_IIS\_APPCMD

### Usage

```
<custom_item>
  type: AUDIT_IIS_APPCMD
  description: "Test appcmd output"
  value_type: [value_type]
  value_data: [value]
  appcmd_args: ["arguments for appcmd.exe"]
  (optional) only_show_cmd_output: YES or NO
  (optional) check_type: [CHECK_TYPE]
  (optional) severity: ["HIGH" or "MEDIUM" or "LOW"]
  (optional) appcmd_list: ["arguments for appcmd.exe to list multiple objects"]
  (optional) appcmd_filter: ["arguments for appcmd.exe to filter"]
  (optional) appcmd_filter_value: ["filter value"]
</custom_item>
```

This check is run **appcmd.exe** on a server running IIS, along with the arguments specified using “**appcmd\_args**”, and determines compliance by comparing the output with **value\_data**. In some cases (e.g., listing configuration) it may be desired to just report the command output. For such cases “**only\_show\_cmd\_output**” should be used.

This check is only applicable for Internet Information Services (IIS) version 7 and greater on Windows.

This item uses field “**appcmd\_args**” to specify the arguments that need to be supplied to **appcmd.exe**. Currently only “list” commands can be specified.

- list sites
- list AppPools /processModel.identityType:ApplicationPoolIdentity
- list config
- list config -section:system.web/authentication
- list app

The item uses optional field “**only\_show\_cmd\_output**” if the entire command output needs to be reported.

There are additional optional fields available to help check configurations on multiple objects in the web server configuration, and each one is a separate execution of **appcmd.exe**.

The “**appcmd\_list**” is an “**appcmd.exe**” execution that will generate a list of objects that the “**appcmd\_args**” will act upon. If “**appcmd\_list**” is used, then you will put a placeholder of “{}” in “**appcmd\_args**” where the object instance name will be inserted into.

An example of this to check the sslFlags for each site in the web server would be:

```
appcmd_list:
appcmd_list: "list sites"
appcmd_args: "list config {} /section:access /text:sslFlags"
```

Other optional fields with “**appcmd\_list**” are “**appcmd\_filter**” and “**appcmd\_filter\_value**”, which can be used to filter the list of objects to specific instances.

---

An example of the relation of the filter fields are would be to check sslFlags on web sites with https bindings only:

```
appcmd_filter: 'list sites {} /text:bindings'
appcmd_filter_value: 'https'
appcmd_list: 'list sites'
appcmd_args: 'list config {} /section:access /text:sslFlags'
```

Examples:

This check compares the result of “appcmd.exe list AppPools /processModel.identityType:ApplicationPoolIdentity” with **value\_data**, and passes only if the output contains “APPPool “DefaultAppPool””.

```
<custom_item>
  type: AUDIT_IIS_APPCMD
  description: "Set Default Application Pool Identity to Least Privilege Principal"
  value_type: POLICY_TEXT
  value_data: 'APPPool "DefaultAppPool"'
  appcmd_args: "list AppPools /processModel.identityType:ApplicationPoolIdentity"
  check_type: CHECK_REGEX
</custom_item>
```

This example checks all application pools to verify that the pool identity is set to ApplicationPoolIdentity.

```
<custom_item>
  type: AUDIT_IIS_APPCMD
  description: "All application pools have identity type of ApplicationPoolIdentity"
  value_type: POLICY_TEXT
  value_data: '^ApplicationPoolIdentity$'
  appcmd_list: 'list AppPools'
  appcmd_args: 'list AppPools {} /text:processModel.identityType'
  check_type: CHECK_REGEX
</custom_item>
```

This example checks the sslFlags of all sites with https bindings to check for SSL Required.

```
<custom_item>
  type: AUDIT_IIS_APPCMD
  description: "Ssl Flags that start with 'Ssl,'"
  value_type: POLICY_TEXT
  value_data: "^Ssl(,|$)"
  appcmd_filter: "list sites {} /text:bindings"
  appcmd_filter_value: "https"
  appcmd_list: "list sites"
  appcmd_args: "list config {} /section:access /text:sslFlags"
  check_type: CHECK_REGEX
</custom_item>
```

## AUDIT\_ALLOWED\_OPEN\_PORTS

### Usage

```
<custom_item>
  type: AUDIT_ALLOWED_OPEN_PORTS
  description: "Audit Open Ports"
  value_type: [value_type]
  value_data: [value]
  port_type: [port_type]
</item>
```

This check queries the list of open TCP/UDP ports on the target and compares them against an allowed list of ports. The check relies on output from either “netstat -ano” or “netstat -an” to get a list of open ports, and then verifies that the ports are indeed open by verifying the port state using (get\_port\_state()/get\_udp\_port\_state()).

Considerations:

- **value\_data** also accepts a regex as a port range, so something like 8[0-9]+ works as well.

Examples:

The following example compares “**value\_data**” against a list of TCP ports open on the target:

```
<custom_item>
  type: AUDIT_ALLOWED_OPEN_PORTS
  description: "Audit TCP OPEN PORTS"
  value_type: POLICY_PORTS
  value_data: "80,135,445,902,912,1024,1025,3389,5900,8[0-9]+,18208,32111,38311,47001,139"
  port_type: TCP
</custom_item>
```

The following example compares “**value\_data**” against a list of UDP ports open on the target:

```
<custom_item>
  type: AUDIT_ALLOWED_OPEN_PORTS
  description: "Audit UDP OPEN PORTS"
  value_type: POLICY_PORTS
  value_data: "161,445,500,1026,4501,123,137,138,5353"
  port_type: UDP
</custom_item>
```

## AUDIT\_DENIED\_OPEN\_PORTS

### Usage

```
<custom_item>
  type: AUDIT_DENIED_OPEN_PORTS
  description: "Audit Denied Open Ports"
  value_type: [value_type]
```

```
value_data: [value]
port_type: [port_type]
<item>
```

This check queries the list of open TCP/UDP ports on the target and compares them against a denied list of ports. The check relies on output from either “netstat -ano” or “netstat -an” to get a list of open ports, and then verifies that the ports are indeed open by verifying the port state using (get\_port\_state()/get\_udp\_port\_state()).

The allowed types are:

- value\_type: POLICY\_PORTS
- value\_data: "80,135,445,902,912,1024,1025,3389,5900,8[0-9]+,18208,32111,38311,47001,139"
- port\_type: TCP or UDP

Considerations:

- **value\_data** also accepts a regex as a port range, so something like 8[0-9]+ works as well.

Examples:

The following example compares “**value\_data**” against a list of TCP ports open on the target.

```
<custom_item>
type: AUDIT_DENIED_OPEN_PORTS
description: "Audit TCP OPEN PORTS"
value_type: POLICY_PORTS
value_data: "80,443"
port_type: TCP
</custom_item>
```

The following example compares “**value\_data**” against a list of UDP ports open on the target.

```
<custom_item>
type: AUDIT_DENIED_OPEN_PORTS
description: "Audit UDP OPEN PORTS"
value_type: POLICY_PORTS
value_data: "161,5353"
port_type: UDP
</custom_item>
```

## AUDIT\_PROCESS\_ON\_PORT

### Usage

```
<custom_item>
type: AUDIT_PROCESS_ON_PORT
description: "Audit Process on Port"
value_type: [value_type]
value_data: [value]
port_type: [port_type]
port_no: [port_no]
```

```
port_option: [port_option]
check_type: CHECK_TYPE
<item>
```

This check queries the process running on a given port. The check relies on output of “netstat -ano” and “tasklist /svc” to determine which process is running on which TCP/UDP port.

The allowed types are:

- value\_type: POLICY\_TEXT
- value\_data: Arbitrary string, e.g., "foo.exe"
- port\_type: TCP or UDP
- port\_no: port number, e.g., 80, 445
- port\_option: CAN\_BE\_CLOSED

Considerations:

- If **port\_option** is set to CAN\_BE\_CLOSED, then the check returns a PASS result if the port is not open on the remote system, otherwise it generates an error.
- Windows 2000 and earlier do not support “netstat -ano”, so this check only works against Windows XP and above.

Examples:

The following example checks whether the process running on tcp port 5900 is either “vss.exe” or “vssrvc.exe”.

```
<custom_item>
type: AUDIT_PROCESS_ON_PORT
description: "Audit OPEN PORT SERVICE"
value_type: POLICY_TEXT
value_data: "vssrvc.exe" || "vss.exe"
port_type: TCP
port_no: "5900"
port_option: CAN_BE_CLOSED
</custom_item>
```

The following example is similar to the first example, except that this example demonstrates use of check\_type.

```
<custom_item>
type: AUDIT_PROCESS_ON_PORT
description: "Audit Process on Port - check_regex"
value_type: POLICY_TEXT
value_data: "foo.exe" || "vss.+"
port_type: TCP
port_no: "5900"
check_type: CHECK_REGEX
</custom_item>
```

## AUDIT\_USER\_TIMESTAMPS

### Usage

```
<custom_item>
  type: AUDIT_USER_TIMESTAMPS
  description: "Users not logged in past 7 or more days."
  value_type: POLICY_DAY
  value_data: "7"
  timestamp: "LogonTime"
  ignore_users: "Admin*,foo"
  check_type: CHECK_GREATER_THAN_OR_EQUAL
</custom_item>
```

This check queries for inactive accounts by looking at the user timestamps.

The keyword timestamp allows following values:

- LogonTime
- LogoffTime
- KickoffTime
- PassLastSet
- PassCanChange
- PassMustChange
- ACB

### Considerations:

- By default, accounts that are disabled, or those for which passwords cannot change or never expire are excluded from the result. They can be included as follows: `include_users: "password never expires" || "cannot change password" || "disabled"`
- By default only those users with SID ranges within “SMB Use Host SID to Enumerate Local Users/SMB Use Domain SID to Enumerate Users” preference range.

Preference Type	SMB Use Host SID to Enumerate Local Users ▼
Start UID	1000
End UID	1200

Preference Type

SMB Use Domain SID to Enumerate Users ▼

Start UID

1000

End UID

1200

Examples:

The check also has the capability to exclude certain users from the result via the **ignore\_users** directive:

```
<custom_item>
  type: AUDIT_USER_TIMESTAMPS
  description: "Password not changed in last 90 days"
  value_type: POLICY_DAY
  value_data: "90"
  timestamp: "PassLastSet"
  ignore_users: "Admin*,foo"
  check_type: CHECK_GREATER_THAN_OR_EQUAL
</custom_item>
```

## CHECK\_ACCOUNT

### Usage

```
<custom_item>
  type: CHECK_ACCOUNT
  description: ["description"]
  value_type: [VALUE_TYPE]
  value_data: [value]
  account_type: [ACCOUNT_TYPE]
  (optional) check_type: [CHECK_TYPE]
</custom_item>
```

This policy item checks for the following values defined in “Security Settings -> Local Policies -> Security Options”:

- Accounts: Administrator account status
- Accounts: Guest account status
- Accounts: Rename administrator account
- Accounts: Rename guest account

The check is performed by calling the function **LsaQueryInformationPolicy** with the level **PolicyAccountDomainInformation** to obtain the domain/system SID, **LsaLookupSid** to obtain administrator and guest names and **NetUserGetInfo** to obtain account information.



This item uses the **account\_type** field to describe which account must be audited. The allowed types are:

- **ADMINISTRATOR\_ACCOUNT** ("Accounts: Administrator account status")  
value\_type: POLICY\_SET  
value\_data: "Enabled" or "Disabled"
- **GUEST\_ACCOUNT** ("Accounts: Guest account status")  
value\_type: POLICY\_SET  
value\_data: "Enabled" or "Disabled"
- **ADMINISTRATOR\_ACCOUNT** ("Accounts: Rename administrator account")  
value\_type: POLICY\_TEXT  
value\_data: "TEXT HERE" [administrator name]  
check\_type: [CHECK\_TYPE] (any one of the possible check\_type values)
- **GUEST\_ACCOUNT** ("Accounts: Rename guest account")  
value\_type: POLICY\_TEXT  
value\_data: "TEXT HERE" [guest name]  
check\_type: [CHECK\_TYPE] (any one of the possible check\_type values)



Depending on the Domain credential part, the local system accounts or the domain accounts may be checked.

Examples:

```
<custom_item>
  type: CHECK_ACCOUNT
  description: "Accounts: Guest account status"
  value_type: POLICY_SET
  value_data: "Disabled"
  account_type: GUEST_ACCOUNT
</custom_item>

<custom_item>
  type: CHECK_ACCOUNT
  description: "Accounts: Rename administrator account"
  value_type: POLICY_TEXT
  value_data: "Dom_adm"
  account_type: ADMINISTRATOR_ACCOUNT
</custom_item>

<custom_item>
  type: CHECK_ACCOUNT
  description: "Accounts: Rename administrator account"
  value_type: POLICY_TEXT
  value_data: "Administrator"
  account_type: ADMINISTRATOR_ACCOUNT
  check_type: CHECK_NOT_EQUAL
</custom_item>
```

## CHECK\_LOCAL\_GROUP

### Usage

```
<custom_item>
  type: CHECK_LOCAL_GROUP
  description: ["description"]
  value_type: [VALUE_TYPE]
  value_data: [value]
  group_type: [GROUP_TYPE]
  (optional) check_type: [CHECK_TYPE]
</custom_item>
```

This policy item checks group names and status of Groups listed in `lusmgr.msc`.

This item uses the **group\_type** field to describe which account must be audited. The allowed types are:

- ADMINISTRATORS\_GROUP
- USERS\_GROUP
- GUESTS\_GROUP
- POWER\_USERS\_GROUP
- ACCOUNT\_OPERATORS\_GROUP
- SERVER\_OPERATORS\_GROUP
- PRINT\_OPERATORS\_GROUP
- BACKUP\_OPERATORS\_GROUP
- REPLICATORS\_GROUP

The allowed types for the **value\_type** field are:

- POLICY\_SET (status of the group is checked)  
value\_type: POLICY\_SET  
value\_data: "Enabled" or "Disabled"
- POLICY\_TEXT (name of the group is checked)  
value\_type: POLICY\_TEXT  
value\_data: "Guests1" (In this case **value\_data** can be any text string)

Examples:

```
<custom_item>
  type: CHECK_LOCAL_GROUP
  description: "Local Guest group must be enabled"
  value_type: POLICY_SET
  value_data: "enabled"
  group_type: GUESTS_GROUP
  check_type: CHECK_EQUAL
</custom_item>
```

```
<custom_item>
```



```
type: CHECK_LOCAL_GROUP
description: "Guests group account name should be Guests"
value_type: POLICY_TEXT
value_data: "Guests"
group_type: GUESTS_GROUP
check_type: CHECK_EQUAL
</custom_item>
```

```
<custom_item>
type: CHECK_LOCAL_GROUP
description: "Guests group account name should not be Guests"
value_type: POLICY_TEXT
value_data: "Guests"
group_type: GUESTS_GROUP
check_type: CHECK_NOT_EQUAL
</custom_item>
```

## ANONYMOUS\_SID\_SETTING

### Usage

```
<custom_item>
type: ANONYMOUS_SID_SETTING
description: ["description"]
value_type: [VALUE_TYPE]
value_data: [value]
(optional) check_type: [value]
</custom_item>
```

This policy item checks for the following value defined in “Security Settings -> Local Policies -> Security Options -> Network access: Allow anonymous SID/Name translation”. The check is performed by calling the function **LsaQuerySecurityObject** on the LSA policy handle.

The allowed types are:

```
value_type: POLICY_SET
value_data: "Enabled" or "Disabled"
```

When using this audit, please note that this policy:

- is a permission check on the LSA service
- checks if the ANONYMOUS\_USER has the flag POLICY\_LOOKUP\_NAMES set
- is deprecated on Windows 2003 because an anonymous user cannot access the LSA pipe

Example:

```
<custom_item>
type: ANONYMOUS_SID_SETTING
description: "Network access: Allow anonymous SID/Name translation"
value_type: POLICY_SET
```

```
value_data: "Disabled"
</custom_item>
```

## SERVICE\_POLICY



This check requires remote registry access for the remote Windows system to function properly.

### Usage

```
<custom_item>
  type: SERVICE_POLICY
  description: ["description"]
  value_type: [VALUE_TYPE]
  value_data: [value]
  (optional) check_type: [value]
  service_name: ["service name"]
</custom_item>
```

This policy item checks for the startup values defined in “System Services”. The check is performed by calling the function **RegQueryValueEx** on the following keys:

- key: "SYSTEM\CurrentControlSet\Services\" + service\_name
- item: "Start"

The allowed types are:

```
value_type: SERVICE_SET
value_data: "Automatic", "Manual" or "Disabled"
svc_option: CAN_BE_NULL or CAN_NOT_BE_NULL
```

The **service\_name** field corresponds to the REAL name of the service. This name can be obtained by:

1. launching Services control panel (in Administrative tools)
2. selecting the desired service
3. opening properties dialog box (right click -> properties)
4. extracting the “Service name” part

The service permission setting can be checked with a SERVICE\_PERMISSIONS item.

Example:

```
<custom_item>
  type: SERVICE_POLICY
  description: "Background Intelligent Transfer Service"
  value_type: SERVICE_SET
  value_data: "Disabled"
  service_name: "BITS"
```



```
</custom_item>
```

## GROUP\_MEMBERS\_POLICY

### Usage

```
<custom_item>
  type: GROUP_MEMBERS_POLICY
  description: ["description"]
  value_type: [value type]
  value_data: [value]
  (optional) check_type: [value]
  group_name: ["group name"]
</custom_item>
```

This policy item checks that there is a specific list of users present in one or more groups.

The allowed type is:


```
value_type: POLICY_TEXT or POLICY_MULTI_TEXT
value_data: "user1" && "user2" && ... && "usern"
```

When using this audit, please note that a user name can be specified with the domain name like "MYDOMAIN\John Smith" and the **group\_name** field specifies a single group for auditing.

A single Nessus **.audit** file can specify multiple different customer items, so it is very easy to audit lists of users in multiple groups. Here is an example **.audit** policy that looks for the "Administrators" group to only contain the "Administrator" and "TENABLE\Domain admins" user:

```
<custom_item>
  type: GROUP_MEMBERS_POLICY
  description: "Checks Administrators members"
  value_type: POLICY_MULTI_TEXT
  value_data: "Administrator" && "TENABLE\Domain admins"
  group_name: "Administrators"
</custom_item>
```

Here is an example screen capture of running the above **.audit** file content against a Windows 2003 server:

Plugin ID : 21156		<a href="#">[Return to top]</a>
192.168.20.16 general/tcp	 "Checks Administrators members" : [FAILED] Remote value: [0: tenabled-9u86to\administrator] Policy value: "Administrator"   "TENABLE\Domain admins"	

## USER\_GROUPS\_POLICY

### Usage

```
<custom_item>
  type: USER_GROUPS_POLICY
  description: ["description"]
  value_type: [value type]
  value_data: [value]
  (optional) check_type: [value]
  user_name: ["user name"]
</custom_item>
```

This policy item checks that a Windows user belongs to the groups specified in **value\_data**. When using this audit, you can only test domain users against a domain controller. This check is not applicable to built-in users like “Local Service”.

Example:

```
<custom_item>
  type: USER_GROUPS_POLICY
  description: "3.72 DG0005: DBMS administration OS accounts"
  info: "Checking that the 'dba' account is a member of required groups only."
  info: "Modify the account/groups in this audit to match your environment."
  value_type: POLICY_MULTI_TEXT
  value_data: "Users" && "SQL Server DBA" && "SQL Server Users"
  user_name: "dba"
</custom_item>
```

## USER\_RIGHTS\_POLICY

### Usage

```
<custom_item>
  type: USER_RIGHTS_POLICY
  description: ["description"]
  value_type: [value type]
  value_data: [value]
  (optional) check_type: [value]
  right_type: [right]
</custom_item>
```

This policy item checks for the following value defined in “Security Settings -> Local Policies -> User Rights Assignment”. The check is performed by calling the function **LsaEnumerateAccountsWithUserRight** on the LSA policy handle.

The **right\_type** field corresponds to the right to test. Allowed values are:

**right\_type:** RIGHT

Where **RIGHT** can be:

SeAssignPrimaryTokenPrivilege  
SeAuditPrivilege  
SeBackupPrivilege

---

SeBatchLogonRight  
SeChangeNotifyPrivilege  
SeCreateGlobalPrivilege  
SeCreatePagefilePrivilege  
SeCreatePermanentPrivilege  
SeCreateTokenPrivilege  
SeDenyBatchLogonRight  
SeDenyInteractiveLogonRight  
SeDenyNetworkLogonRight  
SeDenyRemoteInteractiveLogonRight  
SeDenyServiceLogonRight  
SeDebugPrivilege  
SeEnableDelegationPrivilege  
SeImpersonatePrivilege  
SeIncreaseBasePriorityPrivilege  
SeIncreaseWorkingSetPrivilege  
SeIncreaseQuotaPrivilege  
SeInteractiveLogonRight  
SeLoadDriverPrivilege  
SeLockMemoryPrivilege  
SeMachineAccountPrivilege  
SeManageVolumePrivilege  
SeNetworkLogonRight  
SeProfileSingleProcessPrivilege  
SeRemoteShutdownPrivilege  
SeRemoteInteractiveLogonRight  
SeReLabelPrivilege  
SeRestorePrivilege  
SeSecurityPrivilege  
SeServiceLogonRight  
SeShutdownPrivilege  
SeSyncAgentPrivilege  
SeSystemEnvironmentPrivilege  
SeSystemProfilePrivilege  
SeSystemTimePrivilege  
SeTakeOwnershipPrivilege  
SeTcbPrivilege  
SeTimeZonePrivilege  
SeUndockPrivilege  
SeUnsolicitedInputPrivilege

The allowed type is:

```
value_type: USER_RIGHT  
value_data: "user1" && "user2" && "group1" && ... && "groupn"
```



User rights tests perform many requests against the domain controller. These tests must be included in a separate policy file and only launched against the Domain Controller and ONE system of the domain.



There must be no quotes around the **right** type as it is parsed as a token.

Example:



```
<custom_item>
  type: USER_RIGHTS_POLICY
  description: "Create a token object"
  value_type: USER_RIGHT
  value_data: "Administrators" && "Backup Operators"
  right_type: SeCreateTokenPrivilege
</custom_item>
```

## FILE\_CHECK



This check requires remote registry access for the remote Windows system to function properly.

### Usage

```
<custom_item>
  type: FILE_CHECK
  description: ["description"]
  value_type: [VALUE_TYPE]
  value_data: [value]
  (optional) check_type: [value]
  file_option: [OPTION_TYPE]
</custom_item>
```

This policy item checks whether the file (**value\_data**) exists or not (**file\_option**). The check is performed by calling the function **CreateFile**.

The allowed types are:

```
value_type: POLICY_TEXT
value_data: "file name"
file_option: MUST_EXIST or MUST_NOT_EXIST
```

Examples:

```
<custom_item>
  type: FILE_CHECK
  description: "Check that win.ini exists in the system root"
  value_type: POLICY_TEXT
  value_data: "%SystemRoot%\win.ini"
  file_option: MUST_EXIST
</custom_item>
```

```
<custom_item>
  type: FILE_CHECK
  description: "Check that bad.exe does not exist in the system root"
  value_type: POLICY_TEXT
  value_data: "%SystemRoot%\bad.exe"
  file_option: MUST_NOT_EXIST
</custom_item>
```



## FILE\_VERSION



This check requires remote registry access for the remote Windows system to function properly.

### Usage

```
<custom_item>
  type: FILE_VERSION
  description: ["description"]
  value_type: [VALUE_TYPE]
  value_data: [value]
  (optional) check_type: [value]
  file: PATH_TO_FILE
  file_option: [OPTION_TYPE]
  check_type: CHECK_TYPE
</custom_item>
```

This policy item checks if the version of the file specified by the **file** field is greater than or equal to the remote file version by default. The check can also be used to determine if the remote file version is lower by using the **check\_type** option.

The allowed types are:

```
value_type: POLICY_FILE_VERSION
value_data: "file version"
file_option: MUST_EXIST or MUST_NOT_EXIST
```

Examples:

```
<custom_item>
  type: FILE_VERSION
  description: "Audit for C:\WINDOWS\SYSTEM32\calc.exe"
  value_type: POLICY_FILE_VERSION
  value_data: "1.1.1.1"
  file: "C:\WINDOWS\SYSTEM32\calc.exe"
</custom_item>
```

```
<custom_item>
  type: FILE_VERSION
  description: "Audit for C:\WINDOWS\SYSTEM32\calc.exe"
  value_type: POLICY_FILE_VERSION
  value_data: "1.1.1.1"
  file: "C:\WINDOWS\SYSTEM32\calc.exe"
  check_type: CHECK_LESS_THAN
</custom_item>
```

## FILE\_PERMISSIONS



This check requires remote registry access for the remote Windows system to function properly.

### Usage

```
<custom_item>
  type: FILE_PERMISSIONS
  description: ["description"]
  value_type: [value_type]
  value_data: [value]
  (optional) check_type: [value]
  file: ["filename"]
  (optional) acl_option: [acl_option]
</custom_item>
```

This policy item checks if the FILE\_PERMISSIONS ACL is correct. The check is performed by calling the function **GetSecurityInfo** with level 7 on the file handle.

The allowed type is:

```
value_type: FILE_ACL
value_data: "ACLname"
file: "PATH\Filename"
```

The following predefined paths can be used in the file/folder name:

```
%allusersprofile%
%windir%
%systemroot%
%commonfiles%
%programfiles%
%systemdrive%
%systemdirectory%
```

When using this audit, please note the following:

- The **file** field must include the full path to the file or folder name (e.g., C:\WINDOWS\SYSTEM32) or make use of the above path keywords. If using path keywords, the remote registry must be enabled to allow Nessus to determine the path variable values.
- The **value\_data** field is the name of an ACL defined in the policy file.
- The **acl\_option** field can be set to CAN\_BE\_NULL or CAN\_NOT\_BE\_NULL to force a success/error if the file does not exist.

Examples:

```
<file_acl: "ACL1">

  <user: "Administrators">
    acl_inheritance: "not inherited"
```

```

    acl_apply: "This object only"
    acl_allow: "Full Control"
</user>

<user: "System">
    acl_inheritance: "not inherited"
    acl_apply: "This object only"
    acl_allow: "Full Control"
</user>

</acl>

<custom_item>
    type: FILE_PERMISSIONS
    description: "Permissions for C:\WINDOWS\SYSTEM32"
    value_type: FILE_ACL
    value_data: "ACL1"
    file: "C:\WINDOWS\SYSTEM32"
</custom_item>

```

```

<custom_item>
    type: FILE_PERMISSIONS
    description: "Permissions for C:\WINDOWS\SYSTEM32"
    value_type: FILE_ACL
    value_data: "ACL1"
    file: "%SystemRoot%\SYSTEM32"
</custom_item>

```

When the above check is executed, the compliance module will check if the permissions defined for "%SystemRoot%\SYSTEM32" match the ones described in file\_acl ACL1.

## FILE\_AUDIT



This check requires remote registry access for the remote Windows system to function properly.

### Usage

```

<custom_item>
    type: FILE_AUDIT
    description: ["description"]
    value_type: [value_type]
    value_data: [value]
    (optional) check_type: [value]
    file: ["filename"]
    (optional) acl_option: [acl_option]
</custom_item>

```

This policy item is used to check the audit properties (Properties -> Security -> Advanced -> Auditing) of a file or folder using the specified ACL. This check is performed by calling the function `GetSecurityInfo` with level `SACL_SECURITY_INFORMATION` on the file handle.

---

The allowed type is:

```
value_type: FILE_ACL
value_data: "ACLname"
file: "PATH\Filename"
```

The following predefined paths can be used in the file/folder name:

```
%allusersprofile%
%windir%
%systemroot%
%commonfiles%
%programfiles%
%systemdrive%
%systemdirectory%
```

When using this audit, please note the following:

- The **file** field must include the full path to the file or folder name (e.g., C:\WINDOWS\SYSTEM32) or make use of the above path keywords. If using path keywords, the remote registry must be enabled to allow Nessus to determine the path variable values.
- The **value\_data** field is the name of the ACL defined in the policy file.
- The **acl\_option** field can be set to CAN\_BE\_NULL or CAN\_NOT\_BE\_NULL to force a success/error if the file does not exist.
- The **acl\_allow** and **acl\_deny** fields correspond to "Successful" and "Failed" audit events.

Here is an example `.audit` file that implements the FILE\_AUDIT function, including an example access control list rule named "ACL1".

```
<check_type: "Windows" version:"2">
<group_policy: "Audits SYSTEM32 directory for correct auditing permissions">

<file_acl: "ACL1">
  <user: "Everyone">
    acl_inheritance: "not inherited"
    acl_apply: "This folder, subfolders and files"
    acl_deny: "full control"
    acl_allow: "full control"
  </user>
</acl>

<custom_item>
  type: FILE_AUDIT
  description: "Audit for C:\WINDOWS\SYSTEM32"
  value_type: FILE_ACL
  value_data: "ACL1"
  file: "%SystemRoot%\SYSTEM32"
</custom_item>

</group_policy>
</check_type>
```

## FILE\_CONTENT\_CHECK



This check requires remote registry access for the remote Windows system to function properly.

### Usage

```
<custom_item>
  type: FILE_CONTENT_CHECK
  description: ["description"]
  value_type: [value_type]
  value_data: ["filename"]
  (optional) check_type: [value]
  regex: ["regex"]
  expect: ["regex"]
  (optional) file_option: [file_option]
  (optional) avoid_floppy_access
</custom_item>
```

This policy item checks if the file contains the regular expression **regex** and that this expression matches **expect**.

The check is performed by calling the function **ReadFile** on the file handle.



The file is read over SMB into a memory buffer on the Nessus server, and then the buffer is processed to check for compliance/non-compliance. Files are not saved on the disk of the Nessus server, they are only copied to a memory buffer for analysis.

The allowed type is:

```
value_type: POLICY_TEXT
value_data: "PATH\Filename"
regex: "regex"
expect: "regex"
```

The following predefined paths can be used in the file/folder name:

```
%allusersprofile%
%windir%
%systemroot%
%commonfiles%
%programfiles%
%systemdrive%
```

When using this audit type, please note the following:

- The **value\_data** field must include the full path to the file or folder name (e.g., **C:\WINDOWS\SYSTEM32**) or make use of the above path keywords. If using path keywords, the remote registry must be enabled to allow Nessus to determine the path variable values.
- The **regex** field checks that an item is present in the file.
- The **expect** field checks that the item matches the regular expression.

- The **file\_option** field can be set to **CAN\_BE\_NULL** to force a success if the file does not exist.
- The **file\_option** field can be set to **CAN\_NOT\_BE\_NULL** to force an error if the file exists and is empty.
- The **avoid\_floppy\_access** field can be set to direct the audit not to perform a check that would result in accessing the floppy drive. This should be used if an audit is causing the floppy drive to be accessed when there is no disc in the drive.

Example:

```
<custom_item>
  avoid_floppy_access
  type: FILE_CONTENT_CHECK
  description: "File content for C:\WINDOWS\win.ini"
  value_type: POLICY_TEXT
  value_data: "C:\WINDOWS\win.ini"
  regex: "aif=.*"
  expect: "aif=MPEGVideo"
</custom_item>
```

## FILE\_CONTENT\_CHECK\_NOT



This check requires remote registry access for the remote Windows system to function properly.

### Usage

```
<custom_item>
  type: FILE_CONTENT_CHECK_NOT
  description: ["description"]
  value_type: [value_type]
  value_data: ["filename"]
  (optional) check_type: [value]
  regex: ["regex"]
  expect: ["regex"]
  (optional) file_option: [file_option]
</custom_item>
```

This policy item checks if the file contains the regular expression **regex** and that this expression does not match **expect**. The check is performed by calling the function **ReadFile** on the file handle.

The allowed type is:

```
value_type: POLICY_TEXT
value_data: "PATH\Filename"
regex: "regex"
expect: "regex"
```

The following predefined paths can be used in the file/folder name:

```
%allusersprofile%
%windir%
%systemroot%
```

---

```
%commonfiles%
%programfiles%
%systemdrive%
```

When using this audit type, please note the following:

- The **value\_data** field must include the full path to the file or folder name (e.g., **C:\WINDOWS\SYSTEM32**) or make use of the above path keywords. If using path keywords, the remote registry must be enabled to allow Nessus to determine the path variable values.
- The **regex** field checks that an item is present in the file.
- The **expect** field checks that the item matches the regular expression.
- The **file\_option** field can be set to **CAN\_BE\_NULL** to force a success if the file does not exist.
- The **file\_option** field can be set to **CAN\_NOT\_BE\_NULL** to force an error if the file exists and is empty.

Example:

```
<custom_item>
type: FILE_CONTENT_CHECK_NOT
description: "File content for C:\WINDOWS\win.ini"
value_type: POLICY_TEXT
value_data: "C:\WINDOWS\win.ini"
(optional) check_type: [value]
regex: "au=.*"
expect: "au=MPEGVideo2"
file_option: CAN_NOT_BE_NULL
</custom_item>
```

## REG\_CHECK



This check requires remote registry access for the remote Windows system to function properly.

### Usage

```
<custom_item>
type: REG_CHECK
description: ["description"]
value_type: [VALUE_TYPE]
value_data: [value]
reg_option: [OPTION_TYPE]
(optional) check_type: [value]
(optional) key_item: [item value]
</custom_item>
```

This policy item checks if the registry key (or item) exists or not. The check is performed by calling the functions **RegOpenKeyEx** and **RegQueryValueEx**.

The allowed types are:

```
value_type: POLICY_TEXT
value_data: "key path"
reg_option: MUST_EXIST or MUST_NOT_EXIST
key_item: "item name"
```

If the **key\_item** field is not specified, this item checks that the key path exists. Otherwise, it checks that the item exists.

Example:

```
<custom_item>
  type: REG_CHECK
  description: "Check the key HKLM\SOFTWARE\Adobe\Acrobat Reader\7.0\AdobeViewer"
  value_type: POLICY_TEXT
  value_data: "HKLM\SOFTWARE\Adobe\Acrobat Reader\7.0\AdobeViewer"
  reg_option: MUST_NOT_EXIST
  key_item: "EULA"
</custom_item>
```

## REGISTRY\_SETTING



This check requires remote registry access for the remote Windows system to function properly.

### Usage

```
<custom_item>
  type: REGISTRY_SETTING
  description: ["description"]
  value_type: [VALUE_TYPE]
  value_data: [value]
  reg_key: ["key name"]
  reg_item: ["key item"]
  (optional) check_type: [value]
  (optional) reg_option: [KEY_OPTIONS]
  (optional) reg_enum: ENUM_SUBKEYS
</custom_item>
```

This policy item is used to check the value of a registry key. Many policy checks in “Security Settings -> Local Policies -> Security Options” use this policy item. This check is performed by calling the function **RegQueryValueEx**.

The **reg\_key** field is the name of the registry key (e.g., “HKLM\SOFTWARE\Microsoft\Driver Signing”). The first part of the key (HKLM) is used to connect to the correct registry hive. The subsequent path is a static designation where the desired **reg\_item** is located.



The HKU (HKEY\_USERS) hive is a special case. It is not possible to specify a SID for HKU keys. What happens is the `nbin` internally iterates over each SID, and passes only if the value in each SID is valid.

For example:

```
<custom_item>
  type: REGISTRY_SETTING
```



```

description: "HKU\Control Panel\Desktop\ScreenSaveActive"
value_type: POLICY_DWORD
value_data: 1
reg_key: "HKU\Control Panel\Desktop"
reg_item: "ScreenSaveActive"
</item>

```

would loop over:

```

HKU\S-1-5-18\Control Panel\Desktop\ScreenSaveActive
HKU\S-1-5-19\Control Panel\Desktop\ScreenSaveActive
HKU\S-1-5-20\Control Panel\Desktop\ScreenSaveActive
...

```

and pass if item "ScreenSaveActive" is set to 1 for all SIDs.

The optional **reg\_option** field can be set to CAN\_BE\_NULL to force the check to succeed if the key does not exist or to the opposite CAN\_NOT\_BE\_NULL.

An additional option **reg\_enum** with the argument "ENUM\_SUBKEYS" can be used to enumerate a specified value for all subkeys of a registry key. For example, the key:

**HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall** has many software packages listed. If you wish to match the "CurrentVersion" value for all of the subkeys under "Uninstall", use **reg\_enum**.

Example:

```

<custom_item>
type: REGISTRY_SETTING
description: "DBMS network port, protocol, and services (PPS) usage"
info: "Checking whether TCPDynamicPorts key value is configured (should be blank)."
value_type: POLICY_TEXT
value_data: ""
reg_key: "HKLM\SOFTWARE\Microsoft\Microsoft SQL
        Server\MSSQL.1\MSSQLServer\SuperSocketNetLib\Tcp"
reg_item: "TCPDynamicPorts"
reg_enum: ENUM_SUBKEYS
reg_option: CAN_BE_NULL
</custom_item>

```

This audit of the HKU registry hive does not include the SID (security identifier) in the **reg\_key** registry path. This example will search every HKU SID for the specified **reg\_item**.

Example:

```

<custom_item>
type: REGISTRY_SETTING
description: "FakeAlert.BG trojan check"
value_type: POLICY_TEXT
reg_key: "HKU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run"
reg_item: "brastk"
value_data: "C:\WINDOWS\System32\brastk.exe"
reg_option: CAN_BE_NULL
check_type: CHECK_NOT_EQUAL

```



```
info: "A registry entry for FakeAlert.BG trojan/downloader was found."
info: "The contents of this audit can be edited as desired."
</custom_item>
```

The following main **value\_type** field types are available:

- **POLICY\_SET**  
value\_data: "Enabled" or "Disabled"
- **POLICY\_DWORD**  
value\_data: DWORD or RANGE [same dword as in registry or range]
- **POLICY\_TEXT**  
value\_data: "TEXT" [same text as in registry]
- **POLICY\_MULTI\_TEXT**  
value\_data: "TEXT1" && "TEXT2" && ... && "TEXTN" [same texts as in registry]
- **POLICY\_BINARY**  
value\_data: "0102ac0b...34fb" [same binary as in registry]
- **FILE\_ACL, REG\_ACL, SERVICE\_ACL, LAUNCH\_ACL, ACCESS\_ACL**  
value\_data: "acl\_name" [name of the acl to use]

The following optional **value\_type** field types are available and used in predefined items:

- **DRIVER\_SET**  
value\_data: "Silent Succeed", "Warn but allow installation", "Do not allow installation"
- **LDAP\_SET**  
value\_data: "None" or "Require Signing"
- **LOCKEDID\_SET**  
value\_data: "user display name, domain and user names", "user display name only", "do not display user information"
- **SMARTCARD\_SET**  
value\_data: "No action", "Lock workstation", "Force logoff", "Disconnect if a remote terminal services session"
- **LOCALACCOUNT\_SET**  
value\_data: "Classic - local users authenticate as themselves", "Guest only - local users authenticate as guest"
- **NTLMSSP\_SET**  
value\_data: "No minimum", "Require message integrity", "Require message confidentiality", "Require ntlmv2 session security", "Require 128-bit encryption"
- **CRYPTO\_SET**  
value\_data: "User input is not required when new keys are stored and used", "User is prompted when the key is first used" or "User must enter a password each time they use a key"
- **OBJECT\_SET**  
value\_data: "Administrators group", "Object creator"

- **DASD\_SET**  
value\_data: "Administrators", "administrators and power users", "Administrators and interactive users"
- **LANMAN\_SET**  
value\_data: "Send LM & NTLM responses", "send lm & ntlm - use ntlmv2 session security if negotiated", "send ntlm response only", "send ntlmv2 response only", "send ntlmv2 response only\refuse lm" or "send ntlmv2 response only\refuse lm & ntlm"
- **LDAPCLIENT\_SET**  
value\_data: "None", "Negotiate Signing" or "Require Signing"
- **EVENT\_METHOD**  
value\_data: "by days", "manually" or "as needed"
- **POLICY\_DAY**  
value\_data: DWORD or RANGE (time in days)
- **POLICY\_KBYTE**  
value\_data: DWORD or RANGE

For the **custom\_item** field, use the main **value\_type**. Optional types have been created for predefined items.

If the **value\_type** is an ACL, the registry item must be a security description in binary format.

Examples:

```
<custom_item>
type: REGISTRY_SETTING
description: "Network security: Do not store LAN Manager hash value on next password change"
value_type: POLICY_SET
value_data: "Enabled"
reg_key: "HKLM\SYSTEM\CurrentControlSet\Control\Lsa"
reg_item: "NoLMHash"
</custom_item>
```

```
<custom_item>
type: REGISTRY_SETTING
description: "Network access: Shares that can be accessed anonymously"
value_type: POLICY_MULTI_TEXT
value_data: "SHARE" && "EXAMPLE$"
reg_key: "HKLM\SYSTEM\CurrentControlSet\Services\LanManServer\Parameters"
reg_item: "NullSessionShares"
</custom_item>
```

```
<custom_item>
type: REGISTRY_SETTING
description: "DCOM: Network Provisioning Service - Launch permissions"
value_type: LAUNCH_ACL
value_data: "2"
reg_key: "HKLM\SOFTWARE\Classes\AppID\{39ce474e-59c1-4b84-9be2-2600c335b5c6}"
reg_item: "LaunchPermission"
```

```
</custom_item>
```

```
<custom_item>
  type: REGISTRY_SETTING
  description: "DCOM: Automatic Updates - Access permissions"
  value_type: ACCESS_ACL
  value_data: "3"
  reg_key: "HKLM\SOFTWARE\Classes\AppID\{653C5148-4DCE-4905-9CFD-1B23662D3D9E}"
  reg_item: "AccessPermission"
</custom_item>
```

## REGISTRY\_PERMISSIONS



This check requires remote registry access for the remote Windows system to function properly.

### Usage

```
<custom_item>
  type: REGISTRY_PERMISSIONS
  description: ["description"]
  value_type: [value_type]
  value_data: [value]
  (optional) check_type: [value]
  reg_key: ["regkeyname"]
  (optional) acl_option: [acl_option]
</custom_item>
```

This policy item checks if the registry key ACL is correct. The check is performed by calling the function **RegGetKeySecurity** on the registry key handle.

The allowed type is:

```
value_type: REG_ACL
value_data: "ACLname"
reg_key: "RegistryKeyName"
```

The following predefined paths can be used for the **reg\_key** field:

```
HKLM (HKEY_LOCAL_MACHINE)
HKU (HKEY_USERS)
HKCR (HKEY_CLASS_ROOT)
```

When using this audit, please note the following:

- The **reg\_key** field must include the full path to the file registry key.
- The **value\_data** field is the name of an ACL defined in the policy file.
- The **acl\_option** field can be set to **CAN\_BE\_NULL** or **CAN\_NOT\_BE\_NULL** to force a success/error if the key does not exist.

Example:

```
<registry_acl: "ACL2">

  <user: "Administrators">
    acl_inheritance: "not inherited"
    acl_apply: "This key and subkeys"
    acl_allow: "Full Control"
  </user>

  <user: "SYSTEM">
    acl_inheritance: "not inherited"
    acl_apply: "This key and subkeys"
    acl_allow: "Full Control"
  </user>

</acl>

<custom_item>
  type: REGISTRY_PERMISSIONS
  description: "Permissions for HKLM\SOFTWARE\Microsoft"
  value_type: REG_ACL
  value_data: "ACL2"
  reg_key: "HKLM\SOFTWARE\Microsoft"
</custom_item>
```

When the above check is executed, the compliance module will check if the permissions defined for **"HKLM\SOFTWARE\Microsoft"** match the ones described in registry\_acl ACL2.

## REGISTRY\_AUDIT



This check requires remote registry access for the remote Windows system to function properly.

### Usage

```
<custom_item>
  type: REGISTRY_AUDIT
  description: ["description"]
  value_type: [value_type]
  value_data: [value]
  reg_key: ["regkeyname"]
  (optional) acl_option: [acl_option]
</custom_item>
```

This policy item checks if the registry key ACL is correct. The check is performed by calling the function **RegGetKeySecurity** on the registry key handle.

The allowed type is:

```
value_type: REG_ACL
value_data: "ACLname"
reg_key: "RegistryKeyName"
```

The following predefined path can be used for the **reg\_key** field:

```
HKLM (HKEY_LOCAL_MACHINE)
HKU (HKEY_USERS)
HKCR (HKEY_CLASS_ROOT)
```

When using this audit, please note the following:

- The **reg\_key** field must include the full path to the file registry key.
- The **value\_data** field is the name of the ACL defined in the policy file.
- The **acl\_option** field can be set to CAN\_BE\_NULL or CAN\_NOT\_BE\_NULL to force a success/error if the key does not exist.
- The **acl\_allow** and **acl\_deny** fields correspond to “Successful” and “Failed” audit events.

Here is an example **.audit** file that audits the registry key of “HKLM\SOFTWARE\Microsoft” against an access control list named “ACL2” that is not shown:

```
<custom_item>
type: REGISTRY_AUDIT
description: "Audit for HKLM\SOFTWARE\Microsoft"
value_type: REG_ACL
value_data: "ACL2"
reg_key: "HKLM\SOFTWARE\Microsoft"
</custom_item>
```

## REGISTRY\_TYPE



This check requires remote registry access for the remote Windows system to function properly.

### Usage

```
<custom_item>
type: REGISTRY_TYPE
description: ["description"]
value_type: [VALUE_TYPE]
value_data: [value]
reg_key: ["key name"]
reg_item: ["key item"]
(optional) reg_option: [KEY_OPTIONS]
</item>
```

This policy item is used to check the value of a registry key type. The check is performed by calling the function **RegQueryValue**.

The **reg\_key** field is the name of the registry key (“HKLM\Software\Microsoft\Windows NT\CurrentVersion\Winlogon”). The first part of the key (HKLM, HKU, HKCU, ...) is used to connect to the correct registry hive. In most cases the **reg\_key** field requires a static registry entry with no wildcards, however, there is an exception allowed when searching for values within HKU (HKEY\_USERS). If a path is designated under HKU, the search iterates over all user values in HKU for the value under the designated path. For example, if **reg\_key: "HKU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run"** is specified

along with **reg\_item** "brastk", all users under HKU will be searched for the value of the "brastk" registry key under the relative path: "HKU\<user\_id>\SOFTWARE\Microsoft\Windows\CurrentVersion\Run". For example:

```
value_type: POLICY_TEXT
reg_key: "HKU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run"
reg_item: "brastk"
value_data: "C:\WINDOWS\System32\brastk.exe"
```

This check searches under:

```
HKU\S-1-5-18\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
HKU\S-1-5-19\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
```

The optional field **reg\_option** can be set to CAN\_BE\_NULL to force the check to succeed if the key does not exist or to the opposite CAN\_NOT\_BE\_NULL.

Only POLICY\_TEXT **value\_type** is available for this check.

Here is an example **.audit** file that audits the registry type of "HKLM\Software\Microsoft\Windows NT\CurrentVersion\Winlogon":

```
<custom_item>
type: REGISTRY_TYPE
description: "Check type - reg_sz"
value_type: POLICY_TEXT
value_data: "reg_sz"
reg_key: "HKLM\Software\Microsoft\Windows NT\CurrentVersion\Winlogon"
reg_item: "ScreenSaverGracePeriod"
</item>
```

Note that auditing HKCU may not work on many installations of Windows. To do so requires "Current user" keys, which typically do not exist when Nessus authenticates over SMB. To work around this, auditing HKU (all users) is possible. When the plugin detects a HKU key is being audited, it automatically loops over all the SIDs available except the **.DEFAULT** key. The disadvantage of this approach is that it will also audit system users (e.g., SYSTEM, NT Authority, etc.) To avoid these users, you can use the **reg\_ignore\_hku\_users**. For example:

```
reg_ignore_hku_users : "S-1-5-18,S-1-5-19,S-1-5-20"
```

This only works with **REGISTRY\_SETTING** check.

## SERVICE\_PERMISSIONS

### Usage

```
<custom_item>
type: SERVICE_PERMISSIONS
description: ["description"]
value_type: [value_type]
value_data: [value]
(optional) check_type: [value]
service: ["servicename"]
```

```
(optional) acl_option: [acl_option]
</custom_item>
```

This policy item checks if the service ACL is correct. The check is performed by calling the function **QueryServiceObjectSecurity** on the service handle.

The allowed type is:

```
value_type: SERVICE_ACL
value_data: "ACLname"
service: "ServiceName"
```

When using this audit, please note the following:

- The **value\_data** field is the name of an ACL defined in the policy file.
- The **acl\_option** field can be set to CAN\_BE\_NULL or CAN\_NOT\_BE\_NULL to force a success/error if the key does not exist.

Example:

```
<service_acl: "ACL3">

<user: "Administrators">
  acl_inheritance: "not inherited"
  acl_apply: "This object only"
  acl_allow: "query template" | "change template" | "query status" | "enumerate
    dependents" | "start" | "stop" | "pause and continue" | "interrogate" | "user-
    defined control" | "delete" | "read permissions" | "change permissions" | "take
    ownership"
</user>

<user: "SYSTEM">
  acl_inheritance: "not inherited"
  acl_apply: "This object only"
  acl_allow: "query template" | "change template" | "query status" | "enumerate
    dependents" | "start" | "stop" | "pause and continue" | "interrogate" | "user-
    defined control" | "delete" | "read permissions" | "change permissions" | "take
    ownership"
</user>

<user: "Interactive">
  acl_inheritance: "not inherited"
  acl_apply: "This object only"
  acl_allow: "query template" | "query status" | "enumerate dependents" |
    "interrogate" | "user-defined control" | "read permissions"
</user>

<user: "Everyone">
  acl_inheritance: "not inherited"
  acl_apply: "This object only"
  acl_allow: "query template" | "change template" | "query status" | "enumerate
    dependents" | "start" | "stop" | "pause and continue" | "interrogate" | "user-
    defined control" | "delete" | "read permissions" | "change permissions" | "take
    ownership"
```





```
</user>

</acl>

<custom_item>
  type: SERVICE_PERMISSIONS
  description: "Permissions for Alerter Service"
  value_type: SERVICE_ACL
  value_data: "ACL3"
  service: "Alerter"
</custom_item>
```

When the above check is executed, the compliance module will check if the permissions defined for alerter service match the ones described in service\_acl ACL3.

## SERVICE\_AUDIT

### Usage

```
<custom_item>
  type: SERVICE_AUDIT
  description: ["description"]
  value_type: [value_type]
  value_data: [value]
  (optional) check_type: [value]
  service: ["servicename"]
  (optional) acl_option: [acl_option]
</custom_item>
```

This policy item checks if the service ACL is correct. The check is performed by calling the function **QueryServiceObjectSecurity** on the service handle.

The allowed type is:

```
value_type: SERVICE_ACL
value_data: "ACLname"
service: "ServiceName"
```

When using this audit type, please note the following:

- The **value\_data** field is the name of the ACL defined in the policy file.
- The **acl\_option** field can be set to CAN\_BE\_NULL or CAN\_NOT\_BE\_NULL to force a success/error if the key does not exist.
- The **acl\_allow** and **acl\_deny** fields correspond to "Successful" and "Failed" audit events.

Here is an example `.audit` file for auditing the "Alerter" service:

```
<custom_item>
  type: SERVICE_AUDIT
  description: "Audit for Alerter Service"
  value_type: SERVICE_ACL
```

```
value_data: "ACL3"
service: "Alerter"
</custom_item>
```

## WMI\_POLICY

### Usage

```
<custom_item>
  type: WMI_POLICY
  description: "Test for WMI Value"
  value_type: [value_type]
  value_data: [value]
  (optional) check_type: [value]
  wmi_namespace: ["namespace"]
  wmi_request: ["request select statement"]
  wmi_attribute: ["attribute"]
  wmi_key: ["key"]
</custom_item>
```

This check queries the Windows WMI database for values specified within the namespace/class/attribute.

Either key values may be extracted or attribute names may be enumerated depending on the syntax used.

The allowed types are:

```
wmi_namespace: "namespace"
wmi_request: "WMI Query"
wmi_attribute: "Name"
wmi_key: "Name"
wmi_option: option
wmi_exclude_result: "result"
only_show_query_output: YES
check_type: CHECK_NOT_REGEX
```

If you choose from a service configuration with duplicate values on the system (e.g., "MSFTPSVC/83207416" and "MSFTPSVC/2") the request will extract the chosen attribute from both. If one of them does not match the policy value, the **wmi\_key** will be added to the report to indicate which one has failed. The **wmi\_enum** field allows you to enumerate configuration names within a namespace for comparison or policy value checking.

By default, if a WMI query returns no output, the check reports an error. This behavior can be changed and the check can be forced to report a PASS if **wmi\_option** is set to **CAN\_BE\_NULL**. By setting **only\_show\_query\_output** to YES, the output of the WMI query is now included in the Nessus report. Using the **check\_type** tag, you can have a PASS result as long as a certain string does not exist in the output. See the examples below.

Other Considerations:

- WMI attributes need to be explicitly specified. For example, `select * from foo` will not work.
- Attributes that have no value set will not be reported.
- The case of the attributes should be exactly as it appears in Microsoft documentation. For example, the attribute `HandleCount` cannot be `Handlecount` or `handlecount`.

- Values of array type are not included in the result.

Example 1:

```
<custom_item>
  type: WMI_POLICY
  description: "IIS test"
  value_type: POLICY_DWORD
  value_data: 0
  wmi_namespace: "root/MicrosoftIISv2"
  wmi_request: "SELECT Name, UserIsolationMode FROM IISFtpServerSetting"
  wmi_attribute: "UserIsolationMode"
  wmi_key: "Name"
</custom_item>
```

If there are two FTP service configurations on your system ("MSFTPSVC/83207416" and "MSFTPSVC/2") the request will extract the "UserIsolationMode" attribute from both. If one of them does not match the policy value (0) the **wmi\_key** (in this case) will be added to the report, indicating which one has failed.

Example 2:

```
<custom_item>
  type: WMI_POLICY
  description: "IIS test2"
  value_type: POLICY_MULTI_TEXT
  value_data: "MSFTPSVC/83207416" && "MSFTPSVC/2"
  wmi_namespace: "root/MicrosoftIISv2"
  wmi_request: "SELECT Name FROM IISFtpServerSetting"
  wmi_attribute: "Name"
  wmi_key: "Name"
  wmi_option: WMI_ENUM
</custom_item>
```

This example checks that there are two valid configuration names as specified in **value\_data**. If you wish to learn more about the WMI namespace and associated attributes, Microsoft's WMI CIM Studio is a valuable tool available at the following link:

<http://www.microsoft.com/downloads/details.aspx?FamilyID=6430f853-1120-48db-8cc5-f2abdc3ed314&displaylang=en>

Example 3:

```
<custom_item>
  type: WMI_POLICY
  description: "List All Windows Processes - except svchost.exe and iPodService.exe"
  value_type: POLICY_TEXT
  value_data: ""
  wmi_namespace: "root/cimv2"
  wmi_exclude_result: "svchost.exe,iPodService.exe"
  wmi_request: "select Caption,HandleCount,ThreadCount from Win32_Process"
  only_show_query_output: YES
</custom_item>
```

This example will list all Windows processes, but remove instances of **svchost.exe** and **iPodService.exe**.

## Items

“Items” are check types that are predefined in the Windows Compliance Checks Engine. They are used for commonly audited items and minimize the syntax required for audit check creation. An item has the following structure:

```
<item>
  name: ["predefined_entry"]
  value: [value]
</item>
```

The **name** field must have a name that is already defined (predefined names are listed in “Predefined policies” table below).

All predefined items correspond to the list available in the Domain Policy Editor on Windows 2003 SP1.

The following example checks if the minimum password length is between 8 and 14 characters:

```
<item>
  name: "Minimum password length"
  value: [8..14]
</item>
```

The corresponding custom item is:

```
<custom_item>
  type: PASSWORD_POLICY
  description: "Minimum password length"
  value_type: POLICY_DWORD
  value_data: [8..14]
  password_policy: MINIMUM PASSWORD LENGTH
</custom_item>
```

## Predefined Policies

Policy	Usage
<b>Password Policy</b>	<pre>name: "Enforce password history" value: POLICY_DWORD  name: "Maximum password age" value: TIME_DAY  name: "Minimum password age" value: TIME_DAY  name: "Minimum password length" value: POLICY_DWORD  name: "Password must meet complexity requirements" value: POLICY_SET</pre>
<b>Account Lockout Policy</b>	<pre>name: "Account lockout duration" value: TIME_MINUTE or</pre>



	<p>name: "Account lockout duration" value: TIME_SECOND</p> <p>name: "Account lockout threshold" value: POLICY_DWORD</p> <p>name: "Reset lockout account counter after" value: TIME_MINUTE</p> <p>name: "Enforce user logon restrictions" value: POLICY_SET</p>
<b>Kerberos Policy</b>	<p>name: "Maximum lifetime for service ticket" value: TIME_MINUTE</p> <p>name: "Maximum lifetime for user ticket" value: TIME_HOUR</p> <p>name: "Maximum lifetime for user renewal ticket" value: TIME_DAY</p> <p>name: "Maximum tolerance for computer clock synchronization" value: TIME_MINUTE</p>
<b>Audit Policy</b>	<p>name: "Audit account logon events" value: AUDIT_SET</p> <p>name: "Audit account management" value: AUDIT_SET</p> <p>name: "Audit directory service access" value: AUDIT_SET</p> <p>name: "Audit logon events" value: AUDIT_SET</p> <p>name: "Audit object access" value: AUDIT_SET</p> <p>name: "Audit policy change" value: AUDIT_SET</p> <p>name: "Audit privilege use" value: AUDIT_SET</p> <p>name: "Audit process tracking" value: AUDIT_SET</p> <p>name: "Audit system events" value: AUDIT_SET</p>
<b>Accounts</b>	<p>name: "Accounts: Administrator account status" value: POLICY_SET</p> <p>name: "Accounts: Guest account status" value: POLICY SET</p>



	<p>name: "Accounts: Limit local account use of blank password to console logon only" value: POLICY_SET</p> <p>name: "Accounts: Rename administrator account" value: POLICY_TEXT</p> <p>name: "Accounts: Rename guest account" value: POLICY_TEXT</p>
<b>Audit</b>	<p>name: "Audit: Audit the access of global system objects" value: POLICY_SET</p> <p>name: "Audit: Audit the use of Backup and Restore privilege" value: POLICY_SET</p> <p>name: "Audit: Shut down system immediately if unable to log security audits" value: POLICY_SET</p>
<b>DCOM</b>	<p>name: "DCOM: Machine Launch Restrictions in Security Descriptor Definition Language (SDDL) syntax" value: POLICY_TEXT</p> <p>name: "DCOM: Machine Access Restrictions in Security Descriptor Definition Language (SDDL) syntax" value: POLICY_TEXT</p>
<b>Devices</b>	<p>name: "Devices: Allow undock without having to log on" value: POLICY_SET</p> <p>name: "Devices: Allowed to format and eject removable media" value: DASD_SET</p> <p>name: "Devices: Prevent users from installing printer drivers" value: POLICY_SET</p> <p>name: "Devices: Restrict CD-ROM access to locally logged-on user only" value: POLICY_SET</p> <p>name: "Devices: Restrict floppy access to locally logged-on user only" value: POLICY_SET</p> <p>name: "Devices: Unsigned driver installation behavior" value: DRIVER_SET</p>
<b>Domain controller</b>	<p>name: "Domain controller: Allow server operators to schedule tasks" value: POLICY_SET</p> <p>name: "Domain controller: LDAP server signing requirements" value: LDAP_SET</p> <p>name: "Domain controller: Refuse machine account password changes" value: POLICY_SET</p>



<b>Domain member</b>	<p>name: "Domain member: Digitally encrypt or sign secure channel data (always)" value: POLICY_SET</p> <p>name: "Domain member: Digitally encrypt secure channel data (when possible)" value: POLICY_SET</p> <p>name: "Domain member: Digitally sign secure channel data (when possible)" value: POLICY_SET</p> <p>name: "Domain member: Disable machine account password changes" value: POLICY_SET</p> <p>name: "Domain member: Maximum machine account password age" value: POLICY_DAY</p> <p>name: "Domain member: Require strong (Windows 2000 or later) session key" value: POLICY_SET</p>
<b>Interactive logon</b>	<p>name: "Interactive logon: Display user information when the session is locked" value: LOCKEDID_SET</p> <p>name: "Interactive logon: Do not display last user name" value: POLICY_SET</p> <p>name: "Interactive logon: Do not require CTRL+ALT+DEL" value: POLICY_SET</p> <p>name: "Interactive logon: Message text for users attempting to log on" value: POLICY_TEXT</p> <p>name: "Interactive logon: Message title for users attempting to log on" value: POLICY_TEXT</p> <p>name: "Interactive logon: Number of previous logons to cache (in case domain controller is not available)" value: POLICY_DWORD</p> <p>name: "Interactive logon: Prompt user to change password before expiration" value: POLICY_DWORD</p> <p>name: "Interactive logon: Require Domain Controller authentication to unlock workstation" value: POLICY_SET</p> <p>name: "Interactive logon: Require smart card" value: POLICY_SET</p> <p>name: "Interactive logon: Smart card removal behavior" value: SMARTCARD_SET</p>

<b>Microsoft network client</b>	<p>name: "Microsoft network client: Digitally sign communications (always)" value: POLICY_SET</p> <p>name: "Microsoft network client: Digitally sign communications (if server agrees)" value: POLICY_SET</p> <p>name: "Microsoft network client: Send unencrypted password to third-party SMB servers" value: POLICY_SET</p>
<b>Microsoft network server</b>	<p>name: "Microsoft network server: Amount of idle time required before suspending session" value: POLICY_DWORD</p> <p>name: "Microsoft network server: Digitally sign communications (always)" value: POLICY_SET</p> <p>name: "Microsoft network server: Digitally sign communications (if client agrees)" value: POLICY_SET</p> <p>name: "Microsoft network server: Disconnect clients when logon hours expire" value: POLICY_SET</p>
<b>Network access</b>	<p>name: "Network access: Allow anonymous SID/Name translation" value: POLICY_SET</p> <p>name: "Network access: Do not allow anonymous enumeration of SAM accounts" value: POLICY_SET</p> <p>name: "Network access: Do not allow anonymous enumeration of SAM accounts and shares" value: POLICY_SET</p> <p>name: "Network access: Do not allow storage of credentials or .NET Passports for network authentication" value: POLICY_SET</p> <p>name: "Network access: Let Everyone permissions apply to anonymous users" value: POLICY_SET</p> <p>name: "Network access: Named Pipes that can be accessed anonymously" value: POLICY_MULTI_TEXT</p> <p>name: "Network access: Remotely accessible registry paths and sub-paths" value: POLICY_MULTI_TEXT</p> <p>name: "Network access: Remotely accessible registry paths" value: POLICY_MULTI_TEXT</p>





	<p>name: "Network access: Restrict anonymous access to Named Pipes and Shares" value: POLICY_SET</p> <p>name: "Network access: Shares that can be accessed anonymously" value: POLICY_MULTI_TEXT</p> <p>name: "Network access: Sharing and security model for local accounts" value: LOCALACCOUNT_SET</p>
<b>Network security</b>	<p>name: "Network security: Do not store LAN Manager hash value on next password change" value: POLICY_SET</p> <p>name: "Network security: Force logoff when logon hours expire" value: POLICY_SET</p> <p>name: "Network security: LAN Manager authentication level" value: LANMAN_SET</p> <p>name: "Network security: LDAP client signing requirements" value: LDAPCLIENT_SET</p> <p>name: "Network security: Minimum session security for NTLM SSP based (including secure RPC) clients" value: NTLMSSP_SET</p> <p>name: "Network security: Minimum session security for NTLM SSP based (including secure RPC) servers" value: NTLMSSP_SET</p>
<b>Recovery console</b>	<p>name: "Recovery console: Allow automatic administrative logon" value: POLICY_SET</p> <p>name: "Recovery console: Allow floppy copy and access to all drives and all folders" value: POLICY_SET</p>
<b>Shutdown</b>	<p>name: "Shutdown: Allow system to be shut down without having to log on" value: POLICY_SET</p> <p>name: "Shutdown: Clear virtual memory pagefile" value: POLICY_SET</p>
<b>System cryptography</b>	<p>name: "System cryptography: Force strong key protection for user keys stored on the computer" value: CRYPTO_SET</p> <p>name: "System cryptography: Use FIPS compliant algorithms for encryption, hashing, and signing" value: POLICY_SET</p>
<b>System objects</b>	<p>name: "System objects: Default owner for objects created by members of the Administrators group" value: OBJECT_SET</p>



	<p>name: "System objects: Require case insensitivity for non-Windows subsystems" value: POLICY_SET</p> <p>name: "System objects: Strengthen default permissions of internal system objects (e.g. Symbolic Links)" value: POLICY_SET</p>
<b>System settings</b>	<p>name: "System settings: Optional subsystems" value: POLICY_MULTI_TEXT</p> <p>name: "System settings: Use Certificate Rules on Windows Executables for Software Restriction Policies" value: POLICY_SET</p>
<b>Event Log</b>	<p>name: "Maximum application log size" value: POLICY_KBYTE</p> <p>name: "Maximum security log size" value: POLICY_KBYTE</p> <p>name: "Maximum system log size" value: POLICY_KBYTE</p> <p>name: "Prevent local guests group from accessing application log" value: POLICY_SET</p> <p>name: "Prevent local guests group from accessing security log" value: POLICY_SET</p> <p>name: "Prevent local guests group from accessing system log" value: POLICY_SET</p> <p>name: "Retain application log" value: POLICY_DAY</p> <p>name: "Retain security log" value: POLICY_DAY</p> <p>name: "Retain system log" value: POLICY_DAY</p> <p>name: "Retention method for application log" value: EVENT_METHOD</p> <p>name: "Retention method for security log" value: EVENT_METHOD</p> <p>name: "Retention method for system log" value: EVENT_METHOD</p>

---

## Forced Reporting

Audit policies can be forced to output a specific result by making use of the “**report**” keyword. Report types of PASSED, FAILED, and WARNING can be used. Below is an example policy:

```
<report type: "WARNING">
  description: "Audit 103-a requires a physical inspection of the pod bay doors Hal"
</report>
```

The text inside the “**description**” field would always be displayed in the report.

This type of reporting is useful if you wish to inform an auditor that an actual check being performed by Nessus cannot be accomplished. For example, perhaps there is a requirement to determine that a specific system has been physically secured and we wish to inform the auditor to perform the check or inspection manually. This type of report is also useful if the specific type of audit required to be performed by Nessus has not been determined with an OVAL check.

## Conditions

It is possible to define **if/then/else** logic in the Windows policy to only launch a check if preconditions are valid or to group multiple tests in one.

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>
```

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

The audit for the conditions above uses “**then**” and “**else**” statements, which can be a list of items (or custom items), or an “**if**” statement. The “**else**” and “**then**” statements can optionally make use of the “**report**” type to report a success or a failure depending on the condition return value:

```
<report type:"PASSED|FAILED">
  description: "the test passed (or failed)"
  (optional) severity: INFO|MEDIUM|HIGH
</report>
```

An “**if**” value returns SUCCESS or FAILURE and this value is used when the “**if**” statement is inside another “**if**” structure. For example, if the **<then>** structure is executed, the return value will be one of the following:

- audit contains only items: return SUCCESS if all items passed else return FAILURE
- audit contains only **<report>**: return the report type

- audit contains both items and **<report>**: return the report type

If the **<report>** statement is used and the type is “FAILED” then the reason why it failed will be displayed in the report along with a severity level if defined.

Following is an example that audits the password policy. Since the “**and**” type is used, for this policy to pass the audit both custom items would need to pass. This example tests for a very odd combination of valid password history policies to illustrate how sophisticated test logic can be implemented:

```
<if>
  <condition type:"and">
    <custom_item>
      type: PASSWORD_POLICY
      description: "2.2.2.5 Password History: 24 passwords remembered"
      value_type: POLICY_DWORD
      value_data: [22..MAX] || 20
      password_policy: ENFORCE_PASSWORD_HISTORY
    </custom_item>
    <custom_item>
      type: PASSWORD_POLICY
      description: "2.2.2.5 Password History: 24 passwords remembered"
      value_type: POLICY_DWORD
      value_data: 18 || [4..24]
      password_policy: ENFORCE_PASSWORD_HISTORY
    </custom_item>
  </condition>

  <then>
    <report type:"PASSED">
      description: "Password policy passed"
    </report>
  </then>

  <else>
    <report type:"FAILED">
      description: "Password policy failed"
    </report>
  </else>
</if>
```

In the above example, only the new “**report**” type was shown, but the **if/then/else** structure supports performing additional audits within the “**else**” clauses. Within a condition, nested **if/then/else** clauses can also be used. A more complex example is shown below:

```
<if>
  <condition type:"and">
    <custom_item>
      type: CHECK_ACCOUNT
      description: "Accounts: Rename Administrator account"
      value_type: POLICY_TEXT
      value_data: "Administrator"
      account_type: ADMINISTRATOR_ACCOUNT
      check_type: CHECK_NOT_EQUAL
    </custom_item>
  </condition>
```



```
<then>
  <report type:"PASSED">
    description: "Administrator account policy passed"
  </report>
</then>

<else>
  <if>
    <condition type:"or">
      <item>
        name: "Minimum password age"
        value: [1..30]
      </item>
      <custom_item>
        type: PASSWORD_POLICY
        description: "Password Policy setting"
        value_type: POLICY_SET
        value_data: "Enabled"
        password_policy: COMPLEXITY_REQUIREMENTS
      </custom_item>
    </condition>

    <then>
      <report type:"PASSED">
        description: "Administrator account policy passed"
      </report>
    </then>

    <else>
      <report type:"FAILED">
        description: "Administrator account policy failed"
      </report>
    </else>
  </if>

</else>
</if>
```

In this example, if the Administrator account has not been renamed, then audit that the minimum password age is 30 days or less. This audit policy would pass if the administrator account has been renamed regardless of the password policy and would only test the password age policy if the administrator account had not been renamed.

## Windows Content Audit Compliance File Reference

Windows Content `.audit` checks differ from Windows Configuration `.audit` checks in that they are designed to search a Windows 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 noncompliant data.

### Check Type

All Windows 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


### 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"]
</item>

```

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	Description
<b>type</b>	This must always be set to FILE_CONTENT_CHECK
<b>description</b>	This is the information that will be used as a title for unique compliance vulnerabilities in the SecurityCenter. It will also be the first set of data reported by Nessus.
<b>file_extension</b>	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.
<b>regex</b>	<div> <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>The <b>regex</b> keyword must be run with the <b>expect</b> keyword described below.</p> </div> </div>

	 <p>Unlike Windows Compliance Checks, Windows 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. Windows File Content checks simply require that both the <b>regex</b> and <b>expect</b> statements match data within the <code>&lt;max_size&gt;</code> bytes of the file searched.</p>
<b>expect</b>	<p>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.</p> <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>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> </div> <div>  <p>Unlike Windows Compliance Checks, Windows 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. Windows File Content checks simply require that both the <b>regex</b> and <b>expect</b> statements match data within the <code>&lt;max_size&gt;</code> bytes of the file searched.</p> </div>
<b>file_name</b>	<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>
<b>max_size</b>	<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>
<b>only_show</b>	<p>When matching sensitive data such as credit card numbers, your organization may require that only the last four digits be made visible in the report. This keyword supports revealing any number of bytes specified by policy.</p>
<b>regex_replace</b>	<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>
<b>include_paths</b>	<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 “<b>exclude_paths</b>” 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 <b>"include_paths"</b> keyword. File names cannot be included in the <b>"include_paths"</b> value string.</p>
<b>exclude_paths</b>	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 <b>"include_paths"</b> keyword. This is particularly helpful in cases where a particular drive, directory or file must be excluded from search results. Paths are double-quoted and separated by the pipe symbol where multiple paths are required.
<b>see_also</b>	<p>This keyword allows to include links to a reference.</p> <p>Example:  see_also:  "https://benchmarks.cisecurity.org/tools2/linux/CIS_Redhat_Linux_5_Benchmark_v2.0.0.pdf"</p>
<b>solution</b>	<p>This keyword provides a way to include "Solution" text if available.</p> <p>Example:  solution : "Remove this file, if its not required"</p>
<b>reference</b>	<p>This keyword provides a way to include cross-references in the <b>.audit</b>. The format is "ref ref-id1,ref ref-id2".</p> <p>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"</p>

## 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 Windows Content parameters.

We will also use the **nasl** command line binary. For each of the **.audit** files we are showing, you can easily drop these into your Nessus 6 or SecurityCenter 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/compliance_check_windows_file_content.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.

## Target Test File

The target file we will be using has the following content in it:

```
abcdefghijklmnopqrstuvwxyz
```



---

```
01234567890
Tenable Network Security
SecurityCenter
Nessus
Passive Vulnerability Scanner
Log Correlation Engine
AB12CD34EF56
Nessus
```

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

### Example 1: Search for .tns documents that contain the word "Nessus"

Following is a simple .audit file that looks for any .tns file that contains the word "Nessus" anywhere in the document.

```
<check_type:"FileContent">
<item>
  type: FILE_CONTENT_CHECK
  description: "TNS File that Contains the word Nessus"
  file_extension: "tns"
  expect: "Nessus"
</item>
</check_type>
```

When running this command, the following output is expected:

```
"TNS File that Contains the word Nessus" : [FAILED]
- error message:
The following files do not match your policy :
Share: C$, path: \share\new folder\tenable_content.tns
```

These results show that we found a match. The report says we "failed" because we found data we were not looking for. For example, if you are doing an audit for a Social Security number and had a positive match of the Social Security number on the public computer, although the match is positive, it is logged as a failure for compliance reasons.

### Example 2: Search for .tns documents that contain the word "France"

Following is a simple .audit file that looks for any .tns file that contains the word "France" anywhere in the document.

```
<check_type:"FileContent">
<item>
  type: FILE_CONTENT_CHECK
  description: "TNS File that Contains the word France"
  file_extension: "tns"
  expect: "France"
</item>
</check_type>
```

The output we get this time is as follows:

```
"TNS File that Contains the word France" : [PASSED]
```

We were able to “pass” the audit because none of the `.tns` files we audited had the word “France” in them.

### Example 3: Search for `.tns` and `.doc` documents that contain the word “Nessus”

Adding a second extension for file searches of Microsoft Word documents is very easy and shown below:

```
<check_type:"FileContent">
<item>
  type: FILE_CONTENT_CHECK
  description: "TNS or DOC File that Contains the word Nessus"
  file_extension: "tns" | "doc"
  expect: "Nessus"
</item>
</check_type>
```

The results (on our test computer) were as follows:

```
"TNS or DOC File that Contains the word Nessus" : [FAILED]
- error message:
The following files do not match your policy :
Share: C$, path: \share\new folder\tenable_content.tns
Share: C$, path: \documents and settings\jsmith\desktop\tns_roadmap.doc
```

We have the same “failure” as before with our test `.tns` file, but in this case, there was a second file that was a `.doc` that also had the word “Nessus” in it. If you are performing these tests on your own systems, you may or may not have a Word file that contains the word “Nessus” in it.

### Example 4: Search for `.tns` and `.doc` documents that contain the word “Nessus” and have an 11 digit number in them

Now we will add in our first regular expression to match an 11-digit number. We just need to add in the regular expression with the `regex` keyword to the same `.audit` file as before.

```
<check_type:"FileContent">
<item>
  type: FILE_CONTENT_CHECK
  description: "TNS or DOC File that Contains the word Nessus"
  file_extension: "tns" | "doc"
  regex: " ([0-9]{11}) "
  expect: "Nessus"
</item>
</check_type>
```

Running this produces the following output:

```
"TNS or DOC File that Contains the word Nessus" : [FAILED]
- error message:
The following files do not match your policy :
```

---

```
Share: C$, path: \share\new folder\tenable_content.tns      (01234567890)
```

The `.doc` file that matched in the last example is still being searched. Since it does not have the 11-digit number in it, it is not showing up anymore. Also, note that since we are using the **regex** keyword, we also get a match displayed in the data.

What if we needed to find a 10 digit number? The 11-digit number above has two 10-digit numbers in it (0123456789 and 1234567890). If we wanted to write a more exact match for just 11 digits, what we really want then is a regular expression that says:

*"Match any 11 digit number not preceded or followed by any other numbers".*

To do this in regular expressions we can add the "not" operator like this:

```
<check_type:"FileContent">
<item>
  type: FILE_CONTENT_CHECK
  description: "TNS or DOC File that Contains the word Nessus"
  file_extension: "tns" | "doc"
  regex: "([^\0-9]|^)([0-9]{11})([^\0-9]|$)"
  expect: "Nessus"
</item>
</check_type>
```

Reading from left to right, we also see the `^` character and the dollar sign character a few times. The `^` sometimes means the start of a line and other times it means to match the negative. The dollar sign means the end of a line. The above regular expression basically means to look for any patterns that do not start with a number but potentially start on a new line, contains 11 numbers and then are not followed by any more numbers or has a line end. Regular expressions treat the beginning and end of a line as special cases, hence requiring the use of the `^` or `$` characters.

### Example 5: Search for `.tns` and `.doc` documents that contain the word "Nessus" and have an 11 digit number in them, but only display last 4 bytes

Adding the keyword **only\_show** to our `.audit` file can limit the output. This can limit the auditors to only having access to the sensitive data they are looking for.

```
<check_type:"FileContent">
<item>
  type: FILE_CONTENT_CHECK
  description: "TNS or DOC File that Contains the word Nessus"
  file_extension: "tns" | "doc"
  regex: "([^\0-9]|^)([0-9]{11})([^\0-9]|$)"
  expect: "Nessus"
  only_show: "4"
</item>
</check_type>
```

When matched, the data is obscured with "X" characters as shown below:

```
"TNS or DOC File that Contains the word Nessus" : [FAILED]
- error message:
The following files do not match your policy :
Share: C$, path: \share\new folder\tenable_content.tns      (XXXXXXXXX7890)
```

---

## Example 6: Search for .tns documents that contain the word “Correlation” in the first 50 bytes

In this example, we will examine the use of the **max\_size** keyword. In our test file, the word “Correlation” is more than 50 bytes into the file.

```
<check_type:"FileContent">
<item>
  type: FILE_CONTENT_CHECK
  description: "TNS File that Contains the word Correlation"
  file_extension: ".tns"
  expect: "Correlation"
  max_size: "50"
</item>
</check_type>
```

When running this, we get a passing match:

```
"TNS File that Contains the word Correlation" : [PASSED]
```

Change the **max\_size** value from “50” to “50K” and rerun the scan. Now we get an error:

```
"TNS File that Contains the word Correlation" : [FAILED]
- error message:
The following files do not match your policy :
Share: C$, path: \share\new folder\tenable_content.tns
```

## Example 7: Controlling what is displayed in output

In this example, we will examine the use of the **regex\_replace** keyword. Consider the following **.audit** file:

```
<check_type:"FileContent">
<item>
  type: FILE_CONTENT_CHECK
  description: "Seventh Example"
  file_extension: ".tns"
  regex: "Passive Vulnerability Scanner"
  expect: "Nessus"
</item>
</check_type>
```

This check outputs as follows:

```
"Seventh Example" : [FAILED]
- error message:
The following files do not match your policy :
Share: C$, path: \share\new folder\tenable_content.tns      (Passive Vulnerability
Scanner)
```

---

However, consider what can occur if we really needed to have a regular expression that matched on the “Passive” and “Scanner” parts, but we were only interested in returning the “Vulnerability” part. A new regular expression would look like this:

```
<check_type:"FileContent">
<item>
  type: FILE_CONTENT_CHECK
  description: "Seventh Example"
  file_extension: "tns"
  regex: "(Passive) (Vulnerability) (Scanner)"
  expect: "Nessus"
</item>
</check_type>
```

The check still returns the entire match of “Passive Vulnerability Scanner” because the regular expression statement treats the entire string as the first match. To get only the second match, we need to add in the **regex\_replace** keyword.

```
<check_type:"FileContent">
<item>
  type: FILE_CONTENT_CHECK
  description: "Seventh Example"
  file_extension: "tns"
  regex: "(Passive) (Vulnerability) (Scanner)"
  regex_replace: "\3"
  expect: "Nessus"
</item>
</check_type>
```

The output from the scan is as follows:

```
"Seventh Example" : [FAILED]
- error message:
The following files do not match your policy :
Share: C$, path: \share\new folder\tenable_content.tns      (Vulnerability)
```

We use a “\3” to indicate the second item in our matching because the first (“\1”) is the entire string. If we had used “\2”, we would have returned “Passive” and a “\4” would have returned “Scanner”.

Why does this feature exist? 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 in capturing the desired data with greater accuracy.

### Example 8: Using the file name as a filter

If you consider the **.audit** file from the third example, it returned a result for both a **.tns** file and a **.doc** file.

```
<check_type:"FileContent">
<item>
  type: FILE_CONTENT_CHECK
  description: "TNS or DOC File that Contains the word Nessus"
  file_extension: "tns" | "doc"
  expect: "Nessus"
```

```
</item>
</check_type>
```

The results (on our test computer) were as follows:

```
"TNS or DOC File that Contains the word Nessus" : [FAILED]
- error message:
The following files do not match your policy :
Share: C$, path: \share\new folder\tenable_content.tns
Share: C$, path: \documents and settings\jsmith\desktop\tns_roadmap.doc
```

The **file\_name** keyword can also be used to filter out files we want or do not want. Adding it to the **.audit** file and asking it to only consider files with “tenable” in their name looks like this:

```
<check_type:"FileContent">
<item>
  type: FILE_CONTENT_CHECK
  description: "TNS or DOC File that Contains the word Nessus"
  file_extension: "tns" | "doc"
  file_name: "tenable"
  expect: "Nessus"
</item>
</check_type>
```

The output is as follows:

```
"TNS or DOC File that Contains the word Nessus" : [FAILED]
- error message:
The following files do not match your policy :
Share: C$, path: \share\new folder\tenable_content.tns
```

The matching **.doc** file is not present because it did not have the word “tenable” in its path.

The matching string is a regular expression, so it can be very flexible to match a wide variety of files we want and do not want. For example, we could have used the string “[Tt]enable” to match the word “Tenable” or “tenable”. Similarly, if we want to match an extension or a partial extension, we need to escape the dot with a slash such as “\.” to look for any extensions that start with “t”.

## Example 9: Using the inclusion/exclusion keywords

The **“include\_paths”** and **“exclude\_paths”** keywords may be used to filter searches based on drive letter, directory and even file name exclusion.

```
<item>
  type: FILE_CONTENT_CHECK
  description: "Does the file contain a valid VISA Credit Card Number"
  file_extension: "xls" | "pdf" | "txt"
  regex: "([^\^0-9-]|^)(4[0-9]{3}(|-|)([0-9]{4})(|-|)([0-9]{4})(|-|)([0-9]{4}))([^\^0-9-]|$)"
  regex_replace: "\3"
  expect: "."
```



```
max_size: "50K"  
only_show: "4"  
include_paths: "c:\" | "g:\" | "h:\"  
exclude_paths: "g:\dontscan"  
</item>
```

The output is as follows:

```
Windows File Contents Compliance Checks  
"Determine if a file contains a valid VISA Credit Card Number" : [FAILED]  
- error message:  
The following files do not match your policy :  
Share: C$, path: \documents and settings\administrator\desktop\ccn.txt  
      (XXXXXXXXXXXX0552)  
  
Nessus ID : 24760
```

Note that the output does not differ from a standard Windows file content search result, but, excludes the excluded path. If a single path is included using “**include\_paths**” (e.g., “**c:\**”), all other paths are excluded automatically. Also, if a drive letter is excluded (e.g., “**d:\**”), but, a folder under that drive is included (e.g., “**d:\users**”), the “**exclude\_paths**” keyword takes precedence and the drive will not be searched. However, you can include a drive **C:\** and then exclude a subfolder within the drive (e.g., **C:\users**).

## 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 **.nbin** file will string out all “NULL” bytes that are encountered.

Additionally, all versions of Microsoft Office documents are supported. This includes the newer encoded versions added with Office 2007 such as **.xlsx** and **.docx**.

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 a SecurityCenter, the data only needs to travel from the scanned Windows host to the scanner performing the vulnerability audit.

## Cisco IOS Configuration Audit Compliance File Reference

This section describes the format and functions of the Cisco IOS compliance checks and the rationale behind each setting.

### Check Type

All Cisco IOS compliance checks must be bracketed with the **check\_type** encapsulation and the “Cisco” designation. This is required to differentiate **.audit** files intended specifically for systems running the Cisco IOS operating system from other types of compliance audits.


Example:

```
<check_type:"Cisco">
```

Unlike other compliance audit types, no additional type or version keywords are available.


### Keywords

The following table indicates how each keyword in the Cisco compliance checks can be used:

Keyword	Example Use and Supported Settings
<b>type</b>	<p>CONFIG_CHECK, CONFIG_CHECK_NOT and RANDOMNESS_CHECK</p> <p>“CONFIG_CHECK” determines if the specified item exists in the CISCO IOS “show config” output. In the same manner, “CONFIG_CHECK_NOT” determines if the specified item does not exist. “RANDOMNESS_CHECK” is used to perform string complexity checks (e.g., password checks). If you specify an item to look for (via a regex), it will tell you if the string is “random” enough (at least eight characters long, with upper case, lower case, at least a digit and at least one special character).</p> <div> The randomness parameters are currently not configurable.</div>
<b>description</b>	<p>The “<b>description</b>” 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 that no distinct checks have the same <b>description</b> field. Tenable’s SecurityCenter uses this field to automatically generate a unique plugin ID number based on the <b>description</b> field.</p> <p>Example: description: "Forbid Remote Startup Configuration"</p>
<b>feature_set</b>	<p>The “<b>feature_set</b>” keyword, similar to the “<b>system</b>” keyword in Unix compliance checks, checks the Feature Set version of the Cisco IOS and either runs the resulting check or skips the check because of a failed regex. This is useful for cases where a check is only applicable to systems with a particular Feature Set.</p> <p>Example: &lt;item&gt; type: CONFIG_CHECK description: "Version Check"</p>





	<pre>info: "SSH Access Control Check." feature_set: "K8" context:"line .*" item: "access-class [0-9]+ in" &lt;/item&gt;</pre> <p>The check above will only run the “item” check if the Feature Set version matches the specified regex: (K8)</p> <p>In the event of a Feature Set version check failure, an error similar to the one below is displayed:</p> <pre>"Version Check" : [SKIPPED] Test defined for the feature set 'K8' whereas we are running C850-ADVSECURITYK9-M</pre>
<b>ios_version</b>	<p>The “<b>ios_version</b>” keyword, similar to the “<b>system</b>” keyword in Unix compliance checks, checks the version of the Cisco IOS and either runs the resulting check or skips the check because of a failed regex. This is useful for cases where a check is only applicable to systems with a particular IOS version.</p> <p>Example:</p> <pre>&lt;item&gt;   type: CONFIG_CHECK   description: "Version Check"   info: "SSH Access Control Check."   ios_version: "12\.[5-9]"   context: "line .*"   item: "access-class [0-9]+ in" &lt;/item&gt;</pre> <p>The check above will only run the “item” check if the IOS version matches the specified regex: (12\.[5-9]).</p> <p>In the event of a IOS version check failure, an error similar to the one below is displayed:</p> <pre>"Version Check" : [SKIPPED] Test defined for 12.[5-9] whereas we are running 12.4(15)T10</pre>
<b>info</b>	<p>The “<b>info</b>” keyword is used to add a more detailed description to the check that is being performed. Rationale for the check could be a regulation, URL with more information, corporate policy and more. Multiple <b>info</b> fields can be added on separate lines to format the text as a paragraph. There is no preset limit to the number of <b>info</b> fields that can be used.</p> <div><p>Each “<b>info</b>” tag must be written on a separate line with no line breaks. If more than one line is required (e.g., formatting reasons), add additional “<b>info</b>” tags.</p></div> <p>Example:</p> <pre>info: "Verify at least one local user exists and ensure" info: "all locally defined user passwords are protected" info: "by encryption."</pre>

<b>item</b>	<p>The “<b>item</b>” keyword specifies the configuration item within the output of the “show config” output to be audited.</p> <p>Example:  <code>item: "transport input ssh"</code></p> <p>Regular expressions can be used within this keyword to filter the results of the match. Please see the <b>regex</b> keyword description for more details of the <b>regex</b> functionality.</p>
<b>regex</b>	<p>The “<b>regex</b>” keyword enables searching the configuration item setting to match for a particular regular expression.</p> <p>Example:  <code>regex: "snmp-server community ([^ ]*) .*"</code></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>
<b>min_occurrences</b>	<p>The “<b>min_occurrences</b>” keyword specifies the minimum number of occurrences of the configuration item required to pass the audit.</p> <p>Example:  <code>min_occurrences: "3"</code></p>
<b>max_occurrences</b>	<p>The “<b>max_occurrences</b>” keyword specifies the maximum number of occurrences of the configuration item allowed to pass the audit.</p> <p>Example:  <code>max_occurrences: "1"</code></p>
<b>required</b>	<p>The “<b>required</b>” 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 type is “CONFIG_CHECK”, then the check will pass if the configuration item exists or if the configuration item does not exist. On the other hand, if <b>required</b> was set to “YES”, the above check would fail.</p> <p>Example:  <code>required: NO</code></p>
<b>context</b>	<p>The “<b>context</b>” keyword is useful where more than one instance of a particular configuration item exists. For example, consider the following configuration:</p> <pre> line con 0   no modem enable line aux 0   access-class 42 in   exec-timeout 10 0   no exec line vty 0 4   exec-timeout 2 0 </pre>

```
password 7 15010X1C142222362G
transport input ssh
```

If you want to test a value from a particular serial line, using the **item** keyword with “line” will not be sufficient as there is more than one “line” option. If you use “**context**”, you will only focus on the item you are interested in. For example:

```
context: "con 0"
```

You will only grep on the following configuration item:

```
line con 0
  no modem enable
```

Regular expressions can be used within this keyword to filter the results of the match. Please see the **regex** keyword description for more details of the **regex** functionality.

## Command Line Examples

This section provides some examples of common audits used for Cisco IOS compliance checks. The **nasl** command line binary is used as a quick means of testing audits on the fly. Each of the **.audit** files demonstrated below can easily be dropped into your Nessus scan policies. For quick audits of one system, however, command-line tests are more efficient. The command will be executed each time from the **/opt/nessus/bin** directory as follows:

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

The <IP> is the IP address of the system to be audited.

The “enable” password is requested:

```
Which file contains your security policy ? cisco_test.audit
SSH login to connect with : admin
How do you want to authenticate ? (key or password) [password]
SSH password :
Enter the 'enable' password to use :
```

Consult your Cisco administrator for the correct “enable” login parameters.

### Example 1: Search for a Defined SNMP ACL

Following is a simple **.audit** file that looks for a defined “deny” SNMP ACL. If none are found, the audit will display a failure message. This check will only run if the router IOS version matches the specified regex. Otherwise the check will be skipped.

```
<check_type: "Cisco">

<item>
  type: CONFIG_CHECK
  description: "Require a Defined SNMP ACL"
  info: "Verify a defined simple network management protocol (SNMP) access control list
        (ACL) exists with rules for restricting SNMP access to the device."
  ios_version: "12\[4-9]"
```



```
    item: "deny ip any any"
  </item>

</check_type>
```

When running this command, the following output is expected from a compliant system:

```
"Require a Defined SNMP ACL" : [PASSED]

Verify a defined simple network management protocol (SNMP) access control list (ACL)
exists with rules for restricting SNMP access to the device.
```

A failed audit would return the following output:

```
"Require a Defined SNMP ACL" : [FAILED]

Verify a defined simple network management protocol (SNMP) access control list (ACL)
exists with rules for restricting SNMP access to the device.

- error message: deny ip any any not found in the configuration file
```

In this case, the check failed because we were looking for a “deny ip” rule, and none was found.

## Example 2: Make Sure the “finger” Service is Disabled

The following is a simple `.audit` file that looks for the insecure “**finger**” service on the remote router. This check will only run if the router IOS version matches the specified regex. Otherwise the check will be skipped. If the service is found, the audit will display a failure message.

```
<check_type: "Cisco">

<item>
  type: CONFIG_CHECK_NOT
  description: "Forbid Finger Service"
  ios_version: "12\[4-9]"
  info: "Disable finger server."
  item: "(ip|service) finger"
</item>

</check_type>
```

When running this command, the following output is expected from a compliant system:

```
"Forbid Finger Service" : [PASSED]

Disable finger server.
```

A failed audit would return the following output:

```
"Forbid Finger Service" : [FAILED]
```

---

```
Disable finger server.
- error message:
The following configuration line is set:
ip finger <----

Policy value:
(ip|service) finger
```

### Example 3: Randomness Check to Verify SNMP Community Strings and Access Control are Sufficiently Random

The following is a simple `.audit` file that looks for SNMP community strings that are insufficiently random. If a community string is found that is not determined to be sufficiently random, the audit will display a failure message. Because the “required” option is set to “NO”, the check will still pass if no `snmp-server` community strings exist. This check will only run if the router is using Feature Set: “K9”. Otherwise the check will be skipped.

```
<check_type: "Cisco">

<item>
  type: RANDOMNESS_CHECK
  description: "Require Authorized Read SNMP Community Strings and Access Control"
  info: "Verify an authorized community string and access control is configured to
        restrict read access to the device."
  feature_set: "K9"
  regex: "snmp-server community ([^ ]*) .*"
  required: NO
</item>

</check_type>
```

When running this command, the following output is expected from a compliant system:

```
"Require Authorized Read SNMP Community Strings and Access Control" : [PASSED]

Verify an authorized community string and access control is configured to restrict
read access to the device.
```

A failed audit would return the following output:

```
"Require Authorized Read SNMP Community Strings and Access Control" : [FAILED]

Verify an authorized community string and access control is configured to restrict
read access to the device.
- error message:

The following configuration line does not contain a token deemed random enough:
snmp-server community foobar RO

The following configuration line does not contain a token deemed random enough:
snmp-server community public RO
```

---

In the case above, there were two strings: “foobar” and “public” that did not have a sufficiently random token and thus failed the check.

#### Example 4: Context Check to Verify SSH Access Control

The following is a simple `.audit` file that looks at all “line” configuration items using the “**context**” keyword and performs a **regex** to see if SSH access control is set.

```
<check_type: "Cisco">

<item>
  type: CONFIG_CHECK
  description: "Require SSH Access Control"
  info: "Verify that management access to the device is restricted on all VTY lines."
  context: "line .*"
  item: "access-class [0-9]+ in"</item>
</item>

</check_type>
```

When running this command, the following output is expected from a compliant system:

```
"Require SSH Access Control" : [PASSED]

Verify that management access to the device is restricted on all VTY lines.
```

A failed audit would return the following output:

```
"Require SSH Access Control" : [FAILED]

Verify that management access to the device is restricted on all VTY lines.

- error message:
The following configuration is set:
line con 0
  exec-timeout 5 0
  no modem enable

Missing configuration: access-class [0-9]+ in

The following configuration is set:
line vty 0 4
  exec-timeout 5 0
  password 7 15010A1C142222362D
  transport input ssh

Missing configuration: access-class [0-9]+ in
```

In the case above, there were two strings that matched the “**context**” keyword regex of “**line .\***”. Since neither line contained the “**item**” regex, the audit returned a “**FAILED**” message.

---

## Conditions

It is possible to define **if/then/else** logic in the Cisco audit policy. This allows the end-user to return a warning message rather than pass/fail in case an audit passes.

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: "AND">
  <item>
    type: CONFIG_CHECK
    description: "Forbid Auxiliary Port"
    info: "Verify the EXEC process is disabled on the auxiliary (aux) port."
    context: "line aux "
    item: "no exec"
  </item>
  <item>
    type: CONFIG_CHECK_NOT
    description: "Forbid Auxiliary Port"
    info: "Verify the EXEC process is disabled on the auxiliary (aux) port."
    context: "line aux "
    item: "transport input [^n][^o]?[^n]?[^e]?$"
  </item>
</condition>
<then>
  <report type: "PASSED">
    description: "Forbid Auxiliary Port"
    info: "Verify the EXEC process is disabled on the auxiliary (aux) port."
  </report>
</then>
<else>
  <report type: "FAILED">
    description: "Forbid Auxiliary Port"
    info: "Verify the EXEC process is disabled on the auxiliary (aux) port."
  </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**”.

## Juniper Junos Configuration Audit Compliance File Reference

This section describes the format and functions of the Juniper Junos compliance checks and the rationale behind each setting.

### Check Type: CONFIG\_CHECK

Juniper operating system (Junos) compliance checks are bracketed in **custom\_item** encapsulation and either CONFIG\_CHECK or SHOW\_CONFIG\_CHECK. These are treated like any other **.audit** files and work for systems running Junos. The CONFIG\_CHECK check consists of two or more keywords. Keywords **type** and **description** are mandatory, which are followed by one or more keywords. The check works by auditing the config in the “set” format.

The config in “set” format can be obtained by appending “display set” to the “show configuration” request. For example:

```
show configuration | display set
```

```
admin> show configuration | display set
set version 10.2R3.10
set system time-zone GMT
set system no-ping-record-route
set system root-authentication encrypted-password "$1$hSGSlnwfdsdffsdffsdff43534"
```

### Keywords

The following table indicates how each keyword in the Juniper compliance checks can be used:

Keyword	Example Use and Supported Settings
<b>type</b>	<p>CHECK_CONFIG and SHOW_CHECK_CONFIG</p> <p>“CHECK_CONFIG” determines if the specified config item exists in the Juniper “show configuration” output in “set” format. In the same manner, “SHOW_CONFIG_CHECK” audits if the config item exists in the “show configuration” output in default format.</p>
<b>description</b>	<p>The “<b>description</b>” 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 that no distinct checks have the same <b>description</b> field. Tenable’s SecurityCenter uses this field to automatically generate a unique plugin ID number based on the <b>description</b> field.</p> <p>Example: description: "3.1 Disable Unused Interfaces"</p>
<b>info</b>	<p>The “<b>info</b>” keyword is used to add a more detailed description to the check that is being performed. Rationale for the check could be a regulation, URL with more information, corporate policy, and more. Multiple <b>info</b> fields can be added on separate lines to format the text as a paragraph. There is no preset limit to the number of <b>info</b> fields that can be used.</p>





Each “**info**” tag must be written on a separate line with no line breaks. If more than one line is required (e.g., formatting reasons), add additional “**info**” tags.

Example:

```
info: "Review the the list of interfaces"
info: "Disable unused interfaces"
```

## severity

The “**severity**” keyword specifies the severity of the check being performed.

Example:

```
severity: MEDIUM
```

The severity can be set to HIGH, MEDIUM, or LOW.

## regex

The “**regex**” keyword enables searching the configuration item setting to match for a particular regular expression.

Example:

```
regex: "set system syslog .+"
```

The following meta-characters require special treatment: + \ \* ( ) ^

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: . ? " '

This has to do with the way that the compiler treats these characters.

If a check has “**regex**” tag set, but no “**expect**” or “**not\_expect**” or “**number\_of\_lines**” tag is set, then the check simply reports all lines matching the regex.

## expect

This keyword allows auditing the configuration item matched by the “**regex**” tag or if the “**regex**” tag is not used it looks for the “**expect**” string in the entire config.

Example:

```
expect: "syslog host 1.1.1.1"
```

The check passes as long as the config line found by “**regex**” matches the “**expect**” tag or in the case where “**regex**” is not set, it passes if the “**expect**” string is found in the config.

Example:

```
regex: "syslog host [0-9\\.]+\"
expect: "syslog host 1.1.1.1"
```

In the above case, the “**expect**” tag ensures that the syslog host is set to 1.1.1.1.

## not\_expect

This keyword allows searching the configuration items that should not be in the configuration.



	<p>Example: not_expect: "syslog host 1.1.1.1"</p> <p>It acts as the opposite of “<b>expect</b>”. The check passes as the config line found by “<b>regex</b>” does not match the “<b>not_expect</b>” tag or if the “<b>regex</b>” tag is not set, it passes as long as “<b>not_expect</b>” string is not found in the config.</p> <p>Example: regex: "syslog host [0-9\\.]+" not_expect: "syslog host 1.1.1.1"</p> <p>In the above case, the “<b>not_expect</b>” tag ensures that the syslog host is not set to 1.1.1.1.</p>
<b>number_of_lines</b>	<p>This keyword allows testing compliance of an audit check based on the number of matching lines returned by the config.</p> <pre>&lt;custom_item&gt;   type: CONFIG_CHECK   description: "Syslog"   regex: "syslog host [0-9\\.]+"   number_of_lines: "^1\$" &lt;/custom_item&gt;</pre> <p>In the above case the check will pass as long as only one line is returned that matches the “<b>regex</b>”.</p>

## CONFIG\_CHECK Examples

The following are examples of using CONFIG\_CHECK against a Juniper device:

```
<custom_item>
  type: CONFIG_CHECK
  description: "Audit Syslog host message severity"
  regex: "syslog host [0-9\\.]+"
  expect: "syslog host [0-9\\.]+ 6 .+"
</custom_item>
```

```
<custom_item>
  type: CONFIG_CHECK
  description: "Audit Syslog host"
  regex: "syslog host [0-9\\.]+"
  number_of_lines: "^1$"
</custom_item>
```

```
<custom_item>
  type: CONFIG_CHECK
  description: "Audit Syslog host"
  regex: "syslog host [0-9\\.]+"
  not_expect: "syslog host 1.2.3.4"
</custom_item>
```



```
<custom_item>
  type: CONFIG_CHECK
  description: "Audit Syslog settings"
  regex: "syslog .+"
</custom_item>
```

## Check Type: SHOW\_CONFIG\_CHECK

This check in many ways audits the same settings audited by the CONFIG\_CHECK .audit check. However, the format of the configuration audited is different. SHOW\_CONFIG\_CHECK audits the configuration in its default format.

For example, here is the configuration in the default format:

```
admin> show configuration system syslog
user * {
    any emergency;
}
host 1.1.1.1 {
    any none;
}
file messages {
    any any;
    authorization info;
}
file interactive-commands {
    interactive-commands any;
}
```

This check is not recommended unless you need greater flexibility over CONFIG\_CHECK. As each SHOW\_CONFIG\_CHECK .audit check results in a separate command being executed on the Juniper device, the process can result in more CPU overhead and take longer to complete. This check exists to provide flexibility to the auditor, and support a future use case that may not be efficiently audited using a CONFIG\_CHECK.

## Keywords

The following table indicates how each keyword in the Junos compliance checks can be used. Note that the compliance of a check can be determined by comparing the output of the check to either “**expect**”, “**not\_expect**”, or “**number\_of\_lines**” tag. There cannot be more than one compliance testing tags (i.e., either “**expect**”, “**not\_expect**”, or “**number\_of\_lines**” can exist but not “**expect**” and “**not\_expect**”).

Keyword	Example Use and Supported Settings
<b>hierarchy</b>	<p>This keyword allows users to navigate to a specific hierarchy in the Junos configuration.</p> <p>Example: hierarchy: "interfaces"</p> <p>Internally the hierarchy keyword gets appended to the “show configuration” command in a SHOW_CONFIG_CHECK. For example:</p>



	<pre>&lt;custom_item&gt;   type: SHOW_CONFIG_CHECK   description: "3.6 Forbid Multiple Loopback Addresses"   hierarchy: "interfaces" &lt;/custom_item&gt;</pre> <p>The check above is the equivalent of running:</p> <pre>show configuration interfaces</pre>
<b>property</b>	<p>This keyword allows users to audit a specific “<b>property</b>” on the Junos device. By default the SHOW_CONFIG_CHECK audits the “show configuration” command followed by one or more keywords such as <b>match</b>, <b>except</b>, and <b>find</b>. In the case where “<b>property</b>” keyword is set, it audits the specific property.</p> <p>Example: property: "ospf"</p> <pre>&lt;custom_item&gt;   type: SHOW_CONFIG_CHECK   description: "4.3.1 Require MD5 Neighbor Authentication               (where OSPF is used)"   info: "Level 2, Scorable"   property: "ospf"   hierarchy: "interface detail"   match: "Auth type MD5" &lt;/custom_item&gt;</pre> <p>The check above is the equivalent of running:</p> <pre>show ospf interface detail</pre> <p>Note that the above example did not run “show configuration”, as was the case in other examples.</p>
<b>find</b>	<p>This keyword finds the appropriate config hierarchy in a SHOW_CONFIG_CHECK .<b>audit</b> check.</p> <pre>find: "chap"</pre> <p>The <b>find</b> keyword gets appended to the “show configuration” request.</p> <pre>&lt;custom_item&gt;   type: SHOW_CONFIG_CHECK   description: "3.8.2 Require CHAP Authentication if Incoming               Map is Used"   hierarchy: "interfaces"   find: "chap"</pre>



	<pre>match: "access-profile" &lt;/custom_item&gt;</pre> <p>The check above is the equivalent of running:</p> <pre>show configuration interfaces   find "chap"   match "access-profile"</pre>
<b>match</b>	<p>This keyword looks for matching lines in a SHOW_CONFIG_CHECK .audit check.</p> <pre>match: "multihop"</pre> <p>The <b>match</b> keyword gets appended to the “show configuration” request.</p> <pre>&lt;custom_item&gt; type: SHOW_CONFIG_CHECK description: "3.6 Forbid Multiple Loopback Addresses" hierarchy: "interfaces" match: "lo[0-9]" &lt;/custom_item&gt;</pre> <p>The check above is the equivalent of running:</p> <pre>show configuration interfaces   match "lo[0-9]"</pre>
<b>except</b>	<p>This keyword excludes certain lines from the config in a SHOW_CONFIG_CHECK .audit check.</p> <pre>except: "multihop"</pre> <p>The <b>except</b> keyword gets appended to the “show configuration” request.</p> <pre>&lt;custom_item&gt; type: SHOW_CONFIG_CHECK description: "6.8.1 Require External Time Sources" hierarchy: "system ntp" match: "server" except: "boot-server" &lt;/custom_item&gt;</pre> <p>The check above is the equivalent of running:</p> <pre>show configuration system ntp   match "server"   except "boot-server"</pre>
<b>expect</b>	<p>This keyword allows auditing the config item matched by the “<b>regex</b>” tag or if the</p>



	<p>"<b>regex</b>" tag is not used it looks for the "<b>expect</b>" string in the entire config. The check passes as long as the config line found by "<b>regex</b>" matches the "<b>expect</b>" tag or in the case where "<b>regex</b>" is not set, it passes if the "<b>expect</b>" string is found in the config.</p> <pre>regex: "syslog host [0-9\.]+"\nexpect: "syslog host 1.2.4.5"</pre> <p>In the above case, the "<b>expect</b>" tag ensures that the complexity is set to a value between 1 and 4.</p> <pre>expect: "syslog host"</pre> <p>In the case above, the "<b>expect</b>" tag ensures that the complexity is set to 4.</p>
<b>not_expect</b>	<p>This keyword allows searching the configuration items that should not be in the configuration.</p> <p>It acts as the opposite of "<b>expect</b>". The check passes as the config line found by "<b>regex</b>" does not match the "<b>not_expect</b>" tag or if the "<b>regex</b>" tag is not set, it passes as long as "<b>not_expect</b>" string is not found in the config.</p> <pre>regex: "syslog host [0-9\.]+"\nnot_expect: "syslog host 1.2.3.4"</pre> <pre>not_expect: "syslog host"</pre>
<b>number_of_lines</b>	<p>This keyword allows testing for compliance of a .audit check based on the number of matching lines returned by the config.</p> <pre>&lt;custom_item&gt;\ntype: CONFIG_CHECK\ndescription: "Syslog"\nregex: "syslog host [0-9\.]+"\nnumber_of_lines: "^1\$"\n&lt;/custom_item&gt;</pre> <p>In the above case the check will pass as long as only one line is returned that matches the "<b>regex</b>".</p>

## SHOW\_CONFIG\_CHECK Examples

The following are examples of using SHOW\_CONFIG\_CHECK against a Juniper device:

```
<custom_item>\ntype: SHOW_CONFIG_CHECK\ndescription: "6.1.2 Require Accounting of Logins & Configuration Changes"\nhierarchy: "system accounting"
```

```
find: "accounting"
expect: "events [change-log login];"
</custom_item>
```

```
<custom_item>
type: SHOW_CONFIG_CHECK
description: "6.2.2 Require Archive Site"
hierarchy: "system archival configuration archive-sites"
match: "scp://"
number_of_lines: "^[1-9]|[0-9][0-9]+)$"
</custom_item>
```

```
<custom_item>
type: SHOW_CONFIG_CHECK
description: "4.7.1 Require BFD Authentication (where BFD is used)"
hierarchy: "protocols"
match: "authentication"
except: "loose"
number_of_lines: "^2$"
check_option: CAN_BE_NULL
</custom_item>
```

```
<custom_item>
type: SHOW_CONFIG_CHECK
description: "4.3.1 Require MD5 Neighbor Authentication (where OSPF is used)"
property: "ospf"
hierarchy: "interface detail"
match: "Auth type MD5"
number_of_lines: "^[1-9]|[0-9][0-9]+)$"
check_option: CAN_BE_NULL
</custom_item>
```

## Conditions

It is possible to define **if/then/else** logic in the Juniper audit policy. This allows the end-user to use a single file that is able to handle multiple configurations.

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: CONFIG_CHECK
    description: "Configure Syslog Host"
    regex: "syslog host [0-9\.]+"
    not_expect: "syslog host 1.2.3.4"
  </custom_item>

  </condition>
  <then>
    <report type: "PASSED">
      description: "Configure Syslog Host."
    </report>
  </then>
  <else>
  <custom_item>
    type: CONFIG_CHECK
    description: "Configure Syslog Host"
    regex: "syslog host [0-9\.]+"
    not_expect: "syslog host 1.2.3.4"
  </custom_item>

  </else>
</if>
```

The condition never shows up in the report - that is, whether it fails or passes it won't show up (it's a "silent" check).

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

## Reporting

Can be performed in a <then> or <else> to achieve a desired PASSED/FAILED condition.

```
<if>
  <condition type: "OR">
  <custom_item>
    type: CONFIG_CHECK
    description: "Configure Syslog Host"
    regex: "syslog host [0-9\.]+"
    not_expect: "syslog host 1.2.3.4"
  </custom_item>
  </condition>
  <then>
    <report type: "PASSED">
      description: "Configure Syslog host"
    </report>
  </then>
  <else>
    <report type: "FAILED">
      description: "Configure Syslog host"
    </report>
  </else>
</if>
```





```
</else>
</if>
```

PASSED, WARNING, and FAILED are acceptable values for “report type”.

## Check Point GAIa Configuration Audit Compliance File Reference


This section describes the format and functions of the [Check Point GAIa](#) compliance checks and the rationale behind each setting.

### Check Type: CONFIG\_CHECK

Check Point compliance checks are bracketed in `custom_item` encapsulation and CONFIG\_CHECK. This is treated like any other `.audit` files and work for systems running the Check Point GAIa operating system. The CONFIG\_CHECK check consists of two or more keywords. Keywords `type` and `description` are mandatory, which are followed by one or more keywords. The check works by auditing the “`show config`” command output, which is in the “set” format by default.

### Keywords

The following table indicates how each keyword in the GAIa compliance checks can be used:

Keyword	Example Use and Supported Settings
<code>type</code>	“CHECK_CONFIG” determines if the specified config item exists in the GAIa “show configuration” output.
<code>description</code>	<p>The “<code>description</code>” keyword provides the ability to add a brief description of the check that is being performed. It is strongly recommended that the <code>description</code> field be unique and that no distinct checks have the same <code>description</code> field. Tenable’s SecurityCenter uses this field to automatically generate a unique plugin ID number based on the <code>description</code> field.</p> <p>Example: description: "1.0 Require strong Password Controls - 'min-password-length &gt;= 8'"</p>
<code>info</code>	<p>The “<code>info</code>” keyword is used to add a more detailed description to the check that is being performed. Rationale for the check could be a regulation, URL with more information, corporate policy, and more. Multiple <code>info</code> fields can be added on separate lines to format the text as a paragraph. There is no preset limit to the number of <code>info</code> fields that can be used.</p> <div> Each “<code>info</code>” tag must be written on a separate line with no line breaks. If more than one line is required (e.g., formatting reasons), add additional “<code>info</code>” tags.</div> <p>Example: info: "Enable palindrome-check on passwords"</p>



---

## CONFIG\_CHECK Examples

The following are examples of using CONFIG\_CHECK against a Check Point device:

```
<custom_item>
type: CONFIG_CHECK
description: "1.0 Require strong Password Controls - 'min-password-length >= 8'"
regex: "set password-controls min-password-length"
expect: "set password-controls min-password-length ([8-9]|[0-9][0-9]+)"
info: "Require Password Lengths greater than or equal to 8."
</custom_item>
```

```
<custom_item>
type: CONFIG_CHECK
description: "1.0 Require strong Password Controls - 'password-expiration != never'"
regex: "set password-controls password-expiration"
not_expect: "set password-controls password-expiration never"
info: "Allow passwords to expire"
</custom_item>
```

```
<custom_item>
type: CONFIG_CHECK
description: "2.13 Secure SNMP"
regex: "set snmp .+"
severity: MEDIUM
info: "Manually review SNMP settings."
</custom_item>
```

## Conditions

It is possible to define **if/then/else** logic in the Check Point audit policy. This allows the end-user to use a single file that is able to handle multiple configurations.

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: CONFIG_CHECK
  description: "2.6 Install and configure Encrypted Connections to devices - 'telnet'"
  regex: "set net-access telnet"
  expect: "set net-access telnet off"
  info: "Do not use plain-text protocols."
</custom_item>
</condition>
<then>
  <report type: "PASSED">
    description: "Telnet is disabled"
  </report>
</then>
<else>
  <custom_item>
    type: CONFIG_CHECK
    description: "2.6 Install and configure Encrypted Connections to devices - 'telnet'"
    regex: "set net-access telnet"
    expect: "set net-access telnet off"
    info: "Do not use plain-text protocols."
  </custom_item>
</else>
</if>
```

The condition never shows up in the report - that is, whether it fails or passes it won't show up (it's a "silent" check).

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

## Reporting

Can be performed in a <then> or <else> to achieve a desired PASSED/FAILED condition.

```
<if>
<condition type: "OR">
<custom_item>
  type: CONFIG_CHECK
  description: "2.6 Install and configure Encrypted Connections to devices - 'telnet'"
  regex: "set net-access telnet"
  expect: "set net-access telnet off"
  info: "Do not use plain-text protocols."
</custom_item>
</condition>
<then>
  <report type: "PASSED">
    description: "Telnet is disabled"
  </report>
</then>
<else>
  <report type: "FAILED">
    description: "Telnet is disabled"
  </report>
</else>
</if>
```

---

PASSED, WARNING, and FAILED are acceptable values for "report type".

## Palo Alto Firewall Configuration Audit Compliance File Reference

The compliance checks for Palo Alto are different than other compliance audits. One major difference in these audits is the heavy use of XSL Transforms (XSLT) to extract the relevant pieces of information (see [Appendix C](#) for more information). Palo Alto Firewall responses are in XML format for most of the API requests, making XSLT the most efficient method for auditing. If you are not familiar with XSLT, you can think of it as a way to query an XML file to extract the data that you want, in a format that you want. In simple terms, XSLT is what SQL is to databases.

The Palo Alto Audit supports two types of checks: AUDIT\_XML and AUDIT\_REPORTS.

### AUDIT\_XML

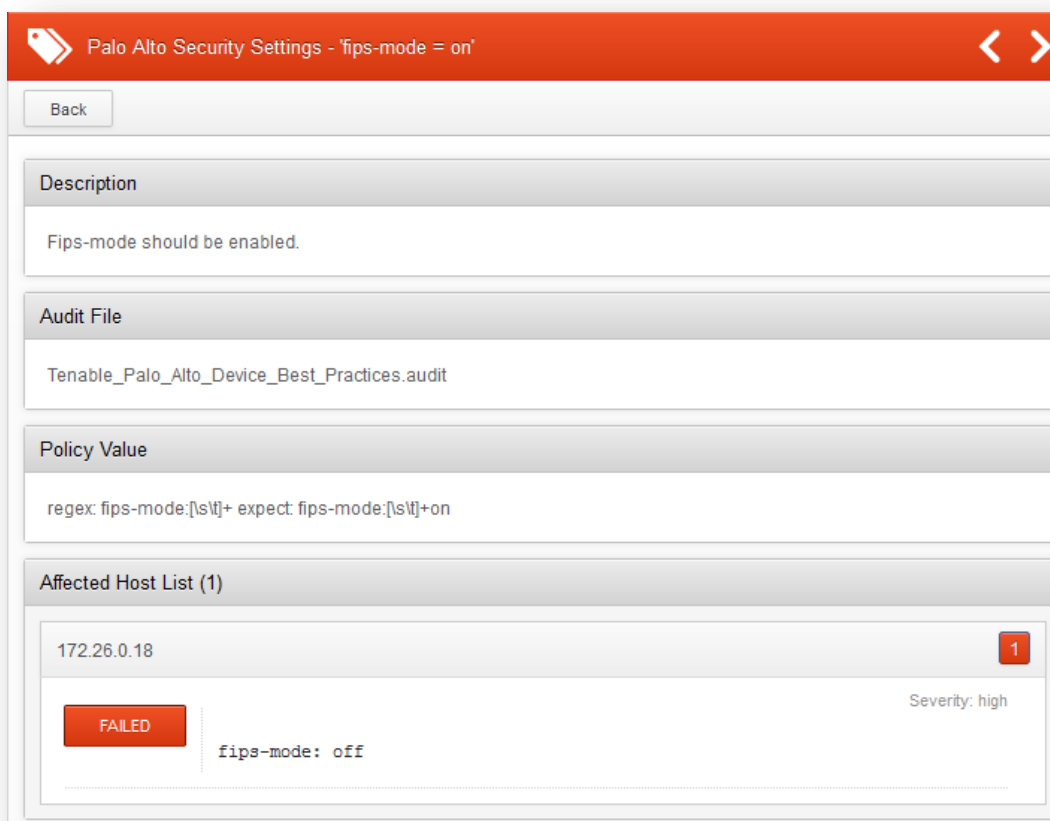
The following is an example of a Palo Alto AUDIT\_XML check:

```
<custom_item>
  type: AUDIT_XML
  description: "Palo Alto Security Settings - 'fips-mode = on'"
  info: "Fips-mode should be enabled."
  api_request_type: "op"
  request: "<show><fips-mode></fips-mode></show>"
  xsl_stmt: "<xsl:template match=\"/\">"
  xsl_stmt: "  <xsl:apply-templates select=\"//result\"/>"
  xsl_stmt: "</xsl:template>"
  xsl_stmt: "<xsl:template match=\"//result\">"
  xsl_stmt: "  fips-mode: <xsl:value-of select=\"text()\"/>"
  regex: "fips-mode:[\\s\\t]+"
  expect : "fips-mode:[\\s\\t]+on"
</custom_item>
```

There are four basic parts to this audit:

1. The **type** describes the type of audit (in this case it audits the XML) and a **description** of the audit. The **info** keyword provides a way to include relevant text in the report.
2. The **api\_request\_type** describes the type of request (op == operational config), and the request is the actual request we end up running. Currently, this is the only type of request supported.
3. The **xsl\_stmt** keyword gives us a way to define the XSL Transform we are going to apply on the XML returned after running the API request.
4. Finally, the **regex** and **expect** keywords allow us to do compliance/configuration auditing.

The example check above will generate the following report in Nessus:



## AUDIT\_REPORTS

One of the nice features of a Palo Alto Firewall is that it continuously profiles its network, generating over 40 predefined reports on a daily basis. Reports such as Top Applications, Top Attackers, and Spyware Infected Hosts. Administrators can also generate dynamic reports at their discretion (e.g., the last-hour). Nessus can now directly query these reports, and include them in a Nessus report.

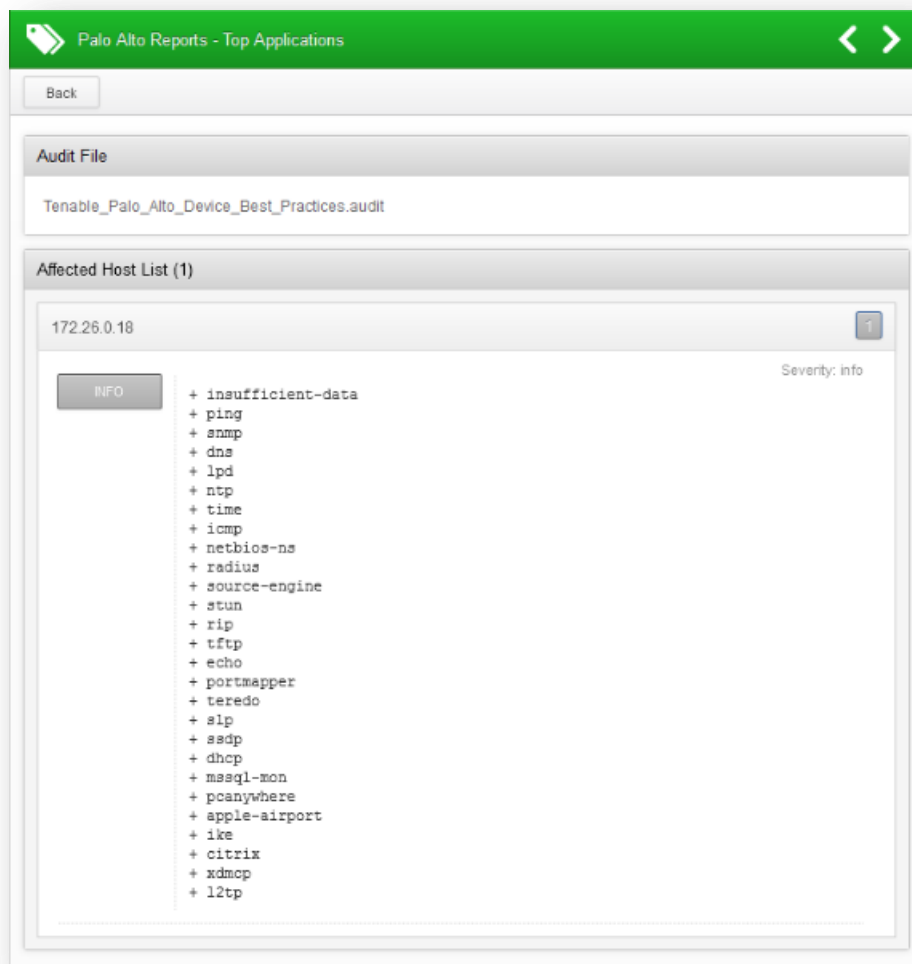
This feature has two benefits. First, users do not have to traverse different interfaces to get the same data. Second, this gives us the ability to audit the report. For example, if you do not want Facebook to be an application used within the network, then administrators can generate a failed report if Facebook shows up on the Top Applications report. For example:

```
<custom_item>
  type: AUDIT_REPORTS
  description: "Palo Alto Reports - Top Applications"
  request: "&reporttype=predefined&reportname=top-applications"
  xsl_stmt: "<xsl:template match=\"result\">"
  xsl_stmt: "<xsl:for-each select=\"entry\">"
  xsl_stmt: "+ <xsl:value-of select=\"name\"/>"
  xsl_stmt: "</xsl:for-each>"
  check_option: CAN_BE_NULL
</custom_item>
```

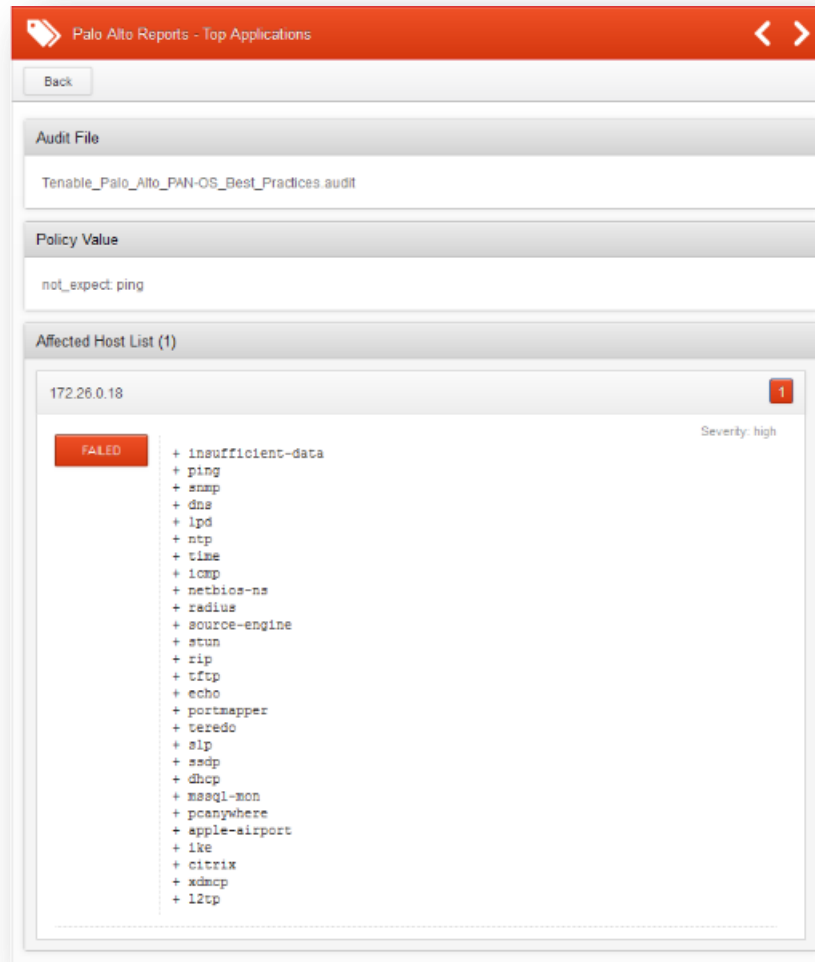
This report can be modified to use a **not\_expect** keyword:

```
<custom_item>
  type: AUDIT_REPORTS
  description: "Palo Alto Reports - Top Applications"
  request: "&reporttype=predefined&reportname=top-applications"
  xsl_stmt: "<xsl:template match=\"result\">"
  xsl_stmt: "<xsl:for-each select=\"entry\">"
  xsl_stmt: "+ <xsl:value-of select=\"name\"/>"
  xsl_stmt: "</xsl:for-each>"
  not_expect: "ping"
  check_option: CAN_BE_NULL
</custom_item>
```

The first example will return a report like this:



The second example will return a report that fails:



## Keywords

The following keywords are supported in Palo Alto audits:

Keyword	Description
<b>type</b>	This must always be set to AUDIT_XML or AUDIT_REPORTS.
<b>description</b>	This is the information used as a title for unique compliance vulnerabilities in the SecurityCenter. It will also be the first set of data reported by Nessus.
<b>info</b>	This keyword allows users to add a more detailed description to the check that is being performed. Multiple <b>info</b> fields are allowed with no preset limit. The <b>info</b> content should be enclosed in double-quotes.
<b>api_request_type</b>	This keyword describes the type of request. The Palo Alto API supports six types of requests: keygen, op, commit, reports, export, and config. For the purposes of this plugin, only request type <b>op</b> is exposed.





<b>request</b>	<p>This keyword specifies the request to run on the firewall. The result of each request is cached, so subsequent requests do not result in another request. In addition, for <b>AUDIT_REPORTS</b> check, the default Tenable audit only includes 9 checks. To include more reports, users are encouraged to create new checks, and replace request keyword with the REST API URL after <code>type=report</code>. For example:</p> <pre>/api/?type=report&amp;reporttype=predefined&amp;reportname=hruser-top-url-categories</pre>
<b>regex</b>	<p>This keyword allows searching items that match a particular regex expression. If a check has <b>regex</b> keyword set, but no <b>expect</b> or <b>not_expect</b> keyword is set, then the check simply reports all lines matching the regex.</p>

The compliance of a check can be determined by comparing the output of the check to either **expect** or **not\_expect** keyword. There cannot be more than one compliance testing tag (i.e., either **expect** or **not\_expect** can exist but not **expect** and **not\_expect**).

Keyword	Description
<b>expect</b>	This keyword allows auditing the config item matched by the <b>regex</b> keyword or if the <b>regex</b> keyword is not used it looks for the <b>expect</b> string in the entire config. The check passes as long as the config line found by <b>regex</b> matches the <b>expect</b> string or in the case where <b>regex</b> is not set, it passes if the <b>expect</b> string is found in the config.
<b>not_expect</b>	This keyword allows searching the configuration items that should <b>not</b> be in the configuration. It acts as the opposite of <b>expect</b> . The check passes as long as the config line found by <b>regex</b> does not match the <b>not_expect</b> string or if the <b>regex</b> keyword is not set, it passes as long as <b>not_expect</b> string is not found in the config.

## Citrix XenServer Audit Compliance File Reference

The compliance checks for Citrix XenServer are heavily based on the [Unix Configuration Audit Compliance File Reference](#) section below, with one exception. An additional audit titled **AUDIT\_XE** is available to perform patch auditing. The following check types are available for XenServer audits:

- **FILE\_CHECK\_NOT**
- **PROCESS\_CHECK**
- **FILE\_CONTENT\_CHECK**
- **FILE\_CONTENT\_CHECK\_NOT**
- **CMD\_EXEC**
- **GRAMMAR\_CHECK**
- **RPM\_CHECK**
- **CHKCONFIG**
- **XINETD\_SVC**
- **AUDIT\_XE**

failed	XenServer - The hosts.deny file blocks access by default	Citrix XenServer Compliance	2
failed	XenServer - Use a static IP on the storage network interface...	Citrix XenServer Compliance	2
warning	XenServer - List security roles	Citrix XenServer Compliance	2
warning	XenServer - Review accounts used to mount remote storage	Citrix XenServer Compliance	2
passed	XenServer - Administrative actions are logged	Citrix XenServer Compliance	2
passed	XenServer - All network interfaces are operating in full-dup...	Citrix XenServer Compliance	2
passed	XenServer - Auto-start is not enabled	Citrix XenServer Compliance	2
passed	XenServer - Enable only necessary and secure services, proto...	Citrix XenServer Compliance	2
passed	XenServer - Enable port locking by default on the VM guest n...	Citrix XenServer Compliance	2
passed	XenServer - External authentication is disabled	Citrix XenServer Compliance	2
passed	XenServer - Host is enabled	Citrix XenServer Compliance	2

## Check Type: AUDIT\_XE

The following is an example of a XenServer AUDIT\_XE check:

```
<custom_item>
  type: AUDIT_XE
  description: "List halted VMs"
  info: "Current guest VM status."
  reference: "PCI|2.2.3,SANS-CSC|1"
  cmd: "/usr/bin/xm vm-list power-state=halted params=uuid,name-label,power-state"
  # You can ignore VMs expected to be halted by entering their UUID here
  # Example ignore
  # ignore: "669e1681-2968-7435-c88e-663501f7d8f3"
</custom_item>
```

## Keywords

The following table indicates how each keyword in the Citrix XenServer compliance checks can be used:

Keyword	Example Use and Supported Settings
<b>type</b>	AUDIT_XE
<b>description</b>	This keyword gives a brief description of the check that is being performed. It is required that description field be unique and no two checks should have the same description field. This is required because SecurityCenter uses this field to auto generate a plugin ID number based on the description field.



	<p>Example: description: "List running VMs"</p>
<b>info</b>	<p>This keyword allows users 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.</p> <p>Example: info: "The allocated virtual CPUs (VCPU) should be reviewed. Desired settings depend on workload and operating system type."</p>
<b>see_also</b>	<p>This keyword allows users to include links that might provide helpful information about a check.</p> <p>Example: see_also: "http://support.citrix.com/article/CTX137828"</p>
<b>reference</b>	<p>This keyword allows including cross references for audit checks.</p> <p>Example: reference: "PCI 2.2.3,SANS-CSC 1"</p>
<b>solution</b>	<p>The keyword provides text to include solution text to fix a compliance failure.</p>
<b>severity</b>	<p>This keyword allows users to set the severity of the check. The severity can be set to HIGH, MEDIUM, or LOW.</p> <p>Example: severity: MEDIUM</p>
<b>cmd</b>	<p>This keyword specified the <b>xe</b> command being run on the target.</p> <p>Example: cmd: "/usr/bin/xe subject-list params=all"</p>
<b>regex</b>	<p>This keyword allows enumerating items that match a particular regex expression. If a check has "<b>regex</b>" keyword set, but no "<b>expect</b>" or "<b>not_expect</b>" keyword is set, then the check simply reports all items matching the regex.</p> <p>Example: regex: "power-state.+"</p>
<b>expect</b>	<p>If expect keyword is specified, then the check passes only if all results match the "<b>expect</b>" keyword. If a result does not match the <b>expect</b> keyword, then the check will fail with all the results that do not match the <b>expect</b>.</p> <p>Example:</p> <pre>&lt;custom_item&gt; type: AUDIT_XE description: "List Running VMs - Any non running vms." cmd: "/usr/bin/xe vm-list params=uuid,name-label,is-a-       template,power-state,allowed-operations" regex: "power-state .+" expect: "running"</pre>



	<pre>&lt;/custom_item&gt;</pre>
<b>not_expect</b>	<p>If not_expect keyword is set, then the check the passes as long as none of the results match the not_expect regex.</p> <p>Example:</p> <pre>&lt;custom_item&gt; type: AUDIT_XE description: "List Running VMs" cmd: "/usr/bin/xe vm-list params=uuid,name-label,is-a-       template,power-state,allowed-operations" regex: "power-state .+" not_expect: "halted" &lt;/custom_item&gt;</pre>
<b>ignore</b>	<p>This keyword allows ignoring/skipping certain items from the result.</p> <p>Example:</p> <pre>&lt;custom_item&gt; type: AUDIT_XE description: "List halted VMs" info: "Current guest VM status." cmd: "/usr/bin/xe vm-list power-state=halted       params=uuid,name-label,power-state" # You can ignore VMs expected to be halted by entering their   UUID here # Example ignore ignore: "669e1681-2968-7435-c88e-663501f7d8f3" &lt;/custom_item&gt;</pre>

## HP ProCurve Audit Compliance File Reference

The HP ProCurve audit is in many respects an extension of the Cisco compliance plugin. The Tenable HP ProCurve audit file is based on an HP whitepaper on hardening ProCurve switches. The audit includes checks for disabling insecure services, and enabling access control (e.g., TACACS, RADIUS). Valid SSH credentials for root or an administrator with full privileges are required.

failed	HP ProCurve - 'Configure login attempts'	HP ProCurve Compliance Checks	1
failed	HP ProCurve - 'RADIUS or TACACS Authentication is	HP ProCurve Compliance Checks	1
failed	HP ProCurve - 'Secure Management VLAN is configured'	HP ProCurve Compliance Checks	1
passed	HP ProCurve - 'Configure Management VLAN'	HP ProCurve Compliance Checks	1
passed	HP ProCurve - 'Disable HTTP'	HP ProCurve Compliance Checks	1
passed	HP ProCurve - 'Disable IP Stack Management'	HP ProCurve Compliance Checks	1
passed	HP ProCurve - 'Disable SNMPv2'	HP ProCurve Compliance Checks	1
passed	HP ProCurve - 'Disable TFTP client'	HP ProCurve Compliance Checks	1
passed	HP ProCurve - 'Disable TFTP server'	HP ProCurve Compliance Checks	1
passed	HP ProCurve - 'Disable Telnet'	HP ProCurve Compliance Checks	1
passed	HP ProCurve - 'Enable ARP protection'	HP ProCurve Compliance Checks	1

## Check Types

HP ProCurve compliance checks use one of three check types. The following is the general syntax for an audit:

```
<custom_item>
type: CONFIG_CHECK
description: "Verify login authentication"
info: "Verifies login authentication configuration"
reference: "PCI|2.2.3,SANS-CSC|1"
context: "line .*"
item: "login authentication"
</custom_item>
```

## Keywords

The following table indicates how each keyword in the HP ProCurve compliance checks can be used:

Keyword	Example Use and Supported Settings
<b>type</b>	<code>CONFIG_CHECK</code> <code>CONFIG_CHECK_NOT</code> <code>RANDOMNESS_CHECK</code>
<b>description</b>	<p>This keyword gives a brief description of the check that is being performed. It is required that description field be unique and no two checks should have the same description field. This is required because SecurityCenter uses this field to auto generate a plugin ID number based on the description field.</p> <p>Example: <code>description: "Verify login authentication"</code></p>
<b>info</b>	<p>This keyword allows users 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.</p> <p>Example: <code>info: "Verifies login authentication configuration."</code></p>
<b>see_also</b>	<p>This keyword allows users to include links that might provide helpful information about a check.</p> <p>Example: <code>see_also: "http://www.hp.com/rnd/support/faqs/1800.htm"</code></p>
<b>reference</b>	<p>This keyword allows including cross references for audit checks.</p> <p>Example: <code>reference: "PCI 2.2.3, SANS-CSC 1"</code></p>
<b>solution</b>	<p>The keyword provides text to include solution text to fix a compliance failure.</p> <p>Example: <code>solution: "Modify the configuration to add missing line"</code></p>
<b>severity</b>	<p>This keyword allows users to set the severity of the check. The severity can be set to HIGH, MEDIUM, or LOW.</p> <p>Example: <code>severity: MEDIUM</code></p>
<b>regex</b>	<p>This keyword allows enumerating items that match a particular regex expression. If a check has "<b>regex</b>" keyword set, but no "<b>expect</b>" or "<b>not_expect</b>" keyword is set, then the check simply reports all items matching the regex.</p> <p>Example: <code>regex: "power-state.+"</code></p>
<b>item</b>	<p>This keyword allows searching within the lines found by regex. If no regex was provided, all lines will be checked.</p>



	<p>Example:</p> <pre>regex: "power"</pre>
<b>context</b>	<p>This keyword allows searching through a specific context. A context is defined by a left justified line followed by any lines that are prefixed by white space.</p> <p>Example:</p> <pre>context: "line .*"</pre> <p>The following is a sample config item, that could be audited by leveraging context:</p> <pre>vlan 1   name "DEFAULT VLAN"   untagged 2-24   ip address dhcp-bootp   no untagged 1   exit</pre> <pre>&lt;item&gt;   type: CONFIG_CHECK   description: "HP ProCurve - 'dhcp-bootp'"   context: "vlan 1"   item: "ip address dhcp-bootp" &lt;/item&gt;</pre> <p>The check above will ensure "ip address dhcp-bootp" is set for context "vlan 1".</p>
<b>min_occurrences</b>	<p>This keyword allows setting a minimum number of occurrences of the check.</p> <p>Example:</p> <pre>min_occurrences: 3</pre>
<b>max_occurrences</b>	<p>Like <b>min_occurrences</b>, but a maximum value instead of a minimum.</p>
<b>required</b>	<p>This keyword allows specifying if a check match is required or not. The value of the required field can be YES, NO, ENABLED, or DISABLED.</p> <p>Example:</p> <pre>required: YES</pre>

## FireEye Audit Compliance File Reference

The FireEye audit is based off of product documentation from FireEye, and common criteria guidelines. The audit includes checks for auditing, identification and authentication, appliance management, intelligent platform management interface (IPMI), enabled services, encryption, and malware detection system configuration. Valid SSH credentials for root or an administrator with full privileges are required.

failed	FireEye - User 'admin' SSH access is disabled	FireEye Compliance Checks	1
failed	FireEye - User connections are limited by subnet or VLAN	FireEye Compliance Checks	1
failed	FireEye - Web interface does not use the system self-signed ...	FireEye Compliance Checks	1
passed	FireEye - A scheduled system backup job is configured	FireEye Compliance Checks	1
passed	FireEye - AAA LDAP binding user should not be an admin	FireEye Compliance Checks	1
passed	FireEye - AAA failed logins are tracked	FireEye Compliance Checks	1
passed	FireEye - AAA is enabled	FireEye Compliance Checks	1
passed	FireEye - AAA lockout settings apply to the 'admin' user	FireEye Compliance Checks	1
passed	FireEye - AAA lockouts are enabled	FireEye Compliance Checks	1
passed	FireEye - AAA lockouts delay further attempts for at least 3...	FireEye Compliance Checks	1
passed	FireEye - AAA lockouts occur after at most 5 failures	FireEye Compliance Checks	1
passed	FireEye - AAA tries local authentication first	FireEye Compliance Checks	1
passed	FireEye - AAA user mapping default	FireEye Compliance Checks	1
passed	FireEye - AAA user mapping source	FireEye Compliance Checks	1

## Check Types

FireEye compliance checks use one of three check types. The following is the general syntax for an audit:

```
<item>
  type: CONFIG_CHECK
  description: "Specific user privs"
  info: "Expect to fail on running config since not all username lines match"
  regex: "username .+"
  expect: "username egossell capability admin"
</item>
```

## Keywords

The following table indicates how each keyword in the HP ProCurve compliance checks can be used:

Keyword	Example Use and Supported Settings
type	CONFIG_CHECK CONFIG_CHECK_NOT





	RANDOMNESS_CHECK
<b>description</b>	<p>This keyword gives a brief description of the check that is being performed. It is required that description field be unique and no two checks should have the same description field. This is required because SecurityCenter uses this field to auto generate a plugin ID number based on the description field.</p> <p>Example: description: "Verify login authentication"</p>
<b>info</b>	<p>This keyword allows users 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.</p> <p>Example: info: "Verifies login authentication configuration."</p>
<b>see_also</b>	<p>This keyword allows users to include links that might provide helpful information about a check.</p> <p>Example: see_also: "http://www.fireeye.com/support/"</p>
<b>reference</b>	<p>This keyword allows including cross references for audit checks.</p> <p>Example: reference: "PCI 2.2.3,SANS-CSC 1"</p>
<b>solution</b>	<p>The keyword provides text to include solution text to fix a compliance failure.</p> <p>Example: solution: "Modify the configuration to add missing line"</p>
<b>severity</b>	<p>This keyword allows users to set the severity of the check. The severity can be set to HIGH, MEDIUM, or LOW.</p> <p>Example: severity: MEDIUM</p>
<b>regex</b>	<p>This keyword allows enumerating items that match a particular regex expression. If a check has "<b>regex</b>" keyword set, but no "<b>expect</b>" or "<b>not_expect</b>" keyword is set, then the check simply reports all items matching the regex.</p> <p>Example: regex: "power-state.+"</p>
<b>expect</b>	<p>This keyword allows searching within the lines found by regex. All lines found by regex must match the expect setting for the check to pass. If no regex was provided, all lines will be checked but only one needs to be found.</p> <p>Example: regex: "power"</p>
<b>not_expect</b>	<p>Similar to <b>expect</b>, but if any matches are found, the check fails. If both <b>expect</b> and <b>not_expect</b> are omitted, all applicable lines will be reported as an info message.</p>



<b>min_occurrences</b>	<p>This keyword allows setting a minimum number of occurrences of the check.</p> <p>Example: min_occurrences: 3</p>
<b>max_occurrences</b>	<p>Like <b>min_occurrences</b>, but a maximum value instead of a minimum.</p>
<b>required</b>	<p>This keyword allows specifying if a check match is required or not. The value of the required field can be YES, NO, ENABLED, or DISABLED.</p> <p>Example: required: YES</p>
<b>cmd</b>	<p>This allows users to run a show command.</p> <p>Example: cmd: "show version"</p> <p>Only "show" commands are allowed.</p> <pre>&lt;item&gt;   type: CONFIG_CHECK   cmd: "show version"   description: "Show Product version"   regex: "Product model:"   expect: "1234" &lt;/item&gt;</pre>

## Brocade Fabric OS (FOS) Compliance File Reference

The Brocade Fabric OS (FOS) runs on the Brocade family of Fibre Channel and FICON switches. This audit includes checks for password policy, enabled services, lockout policy, insecure service configurations, authentication related settings, as well as logging and audit settings. Valid SSH credentials for root or an administrator with full privileges are required.

FAILED	Brocade : 'Disable HTTP'
FAILED	Brocade : 'Enable HTTPS ssl log'
FAILED	Brocade : 'Enable HTTPS'
FAILED	Brocade : 'Ensure a SSL certificate file is established'
FAILED	Brocade : 'FIPS Mode is enabled'
FAILED	Brocade : 'Forward all error logs to syslog daemon'
PASSED	Brocade : 'administrator account is enabled with admin role assigned'
PASSED	Brocade : 'All audit severity level must be audited'
PASSED	Brocade : 'Authentication policy must be rejected'
PASSED	Brocade : 'Bottleneck alerts must be enabled'
PASSED	Brocade : 'Bottleneck detection must be enabled'
PASSED	Brocade : 'Configures filters for a specified audit class'
PASSED	Brocade : 'Device Connection Control policy must be rejected'
PASSED	Brocade : 'Disable HTTP IPv4'
PASSED	Brocade : 'Disable HTTP IPv6'

## Syntax

The syntax for this plugin and an audit are as follows:

```
<custom_item>
description: "Brocade : 'Enable SSH IPv4'"
info: "SSH uses asymmetric authentication to exchange keys and create a secure
      encrypted session."
info: "It is recommended that you use Secure Shell (SSH) instead of Telnet."
see_also:
  "http://www.brocade.com/downloads/documents/product_manuals/B_SAN/FOS_CmdRef_v7
  00.pdf"
solution: "The command to enable SSH is as follows\n
switch:admin> ipfilter --addrule policy_name -rule rule_number -sip any -dp 22 -
proto\n
tcp -act permit\n"
```



```
reference: "SANS-CSC|11,SANS-CSC|10,PCI|2.2.3,800-53|CM-7,800-53|AC-1,800-53|SC-7"
cmd: "ipfilter --show"
context: "ipv4.+active"
regex: "tcp\\s+22"
expect: "permit"
</custom_item>
```

## Huawei VRP Compliance File Reference

The Versatile Routing Platform (VRP) software runs on a wide variety of routing and switching devices produced by [Huawei](#). This audit includes checks for password policy, banner configuration, inactivity timeout, logging and auditing settings, insecure services, device and license information, and SNMP settings. Valid SSH credentials for root or an administrator with full privileges are required.

FAILED	Huawei: User Interfaces Configured Inbound SSH
FAILED	Huawei: User Interfaces Idle Timeout Less Than 5 Minutes
PASSED	Huawei: Command Levels Not Changed
PASSED	Huawei: Device clock = UTC
PASSED	Huawei: Device clock disable DST adjustment
PASSED	Huawei: Disable FTP IPV4
PASSED	Huawei: Disable FTP IPV6
PASSED	Huawei: Enable AAA authentication
PASSED	Huawei: Enable AAA authorization
PASSED	Huawei: HTTPS Server is not configured
PASSED	Huawei: Information Center is not disabled.
PASSED	Huawei: Insecure HTTP is not configured.
PASSED	Huawei: NTP is enabled

## Syntax

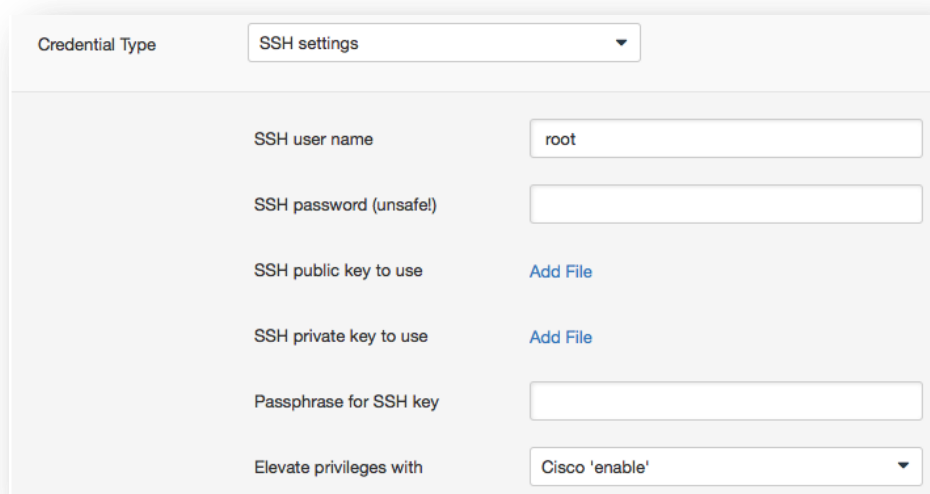
The syntax for this plugin and an audit are as follows:

```
<custom_item>
description: "Huawei: Set super password"
info: "Set super password for managment levels of 3-15."
solution: "In system view, run the following command to configure super
          password super password level <level> encryption-type cipher
          <password>"
reference: "SANS-CSC|10,PCI|2.2.4,COBIT5|BAI10.01,800-53|CM-2"
```

```
expect: "^super password level ([3-9]|1[0-5]) cipher"
</custom_item>
```

## Dell Force10 Compliance File Reference

The Dell **Force10** (FTOS) devices comprise a wide range of high-capacity switches. This audit includes checks for password policy, enabled services, lockout policy, insecure service configurations, authentication related settings, SNMP & NTP configuration, as well as logging and audit settings. Valid SSH credentials for root or an administrator with full privileges are required. The device configuration is only accessible via the “enable” mode.



The screenshot shows a web form titled "SSH settings". At the top, there is a dropdown menu labeled "Credential Type" with "SSH settings" selected. Below this, the form contains several fields and links:

- SSH user name:** A text input field containing the value "root".
- SSH password (unsafe!):** An empty text input field.
- SSH public key to use:** A link labeled "Add File".
- SSH private key to use:** A link labeled "Add File".
- Passphrase for SSH key:** An empty text input field.
- Elevate privileges with:** A dropdown menu with "Cisco 'enable'" selected.

In the preferences there is only one “enable” option, which is tied to “cisco enable”. This plugin essentially piggy backs on that preference to set the enable password. As such, users should use the “cisco enable” option to save the enable password.

FAILED	Dell Force 10 : Device clock disable DST adjustment
FAILED	Dell Force 10 : Disable FTP
FAILED	Dell Force 10 : Disable SNMP write access
FAILED	Dell Force 10 : Enable aaa accounting exec
FAILED	Dell Force 10 : Enable aaa accounting system
FAILED	Dell Force 10 : Enable aaa authentication
FAILED	Dell Force 10 : Enable aaa authentication login
FAILED	Dell Force 10 : Set 'exec' Banner
FAILED	Dell Force 10 : Set 'login' Banner
FAILED	Dell Force 10 : Set 'motd' Banner
FAILED	Dell Force 10 : SNMP Community string != private
FAILED	Dell Force 10 : SNMP Community string != public
PASSED	Dell Force 10 : Allow SSH version 2
PASSED	Dell Force 10 : Configure External Syslog server
PASSED	Dell Force 10 : Configure NTP server
PASSED	Dell Force 10 : Device clock = UTC

## Syntax

The syntax for this plugin and an audit are as follows:

```
<custom_item>
description: "Dell Force 10 : Min Password Length >= 8"
info: "Passwords should be at least 8 characters in length"
expect: "password-attributes.+min-length ([8-9]|[1-9][0-9]+)"
solution: "To configure password length run the following command :\n

password-attributes min-length 8"
reference: "SANS-CSC|10,HIPAA|164.308(a)(5)(ii)(D),PCI|2.2.4,PCI|8.2.3"
</custom_item>
```

## Fortinet FortiOS Audit Compliance File Reference

The Fortinet FortiOS audit includes checks for password policy, malware detection configuration, enabled services, license information and status, log threshold configuration, NTP configuration, SNMP configuration, administrator user enumeration, patch update method, audit and log configuration, as well as authentication. Valid SSH credentials for root or an administrator with full privileges are required.

FAILED	Fortigate - Use non default admin access ports - 'HTTPS'
FAILED	Fortigate - Use non default admin access ports - 'SSH'
FAILED	Fortigate - Virus database - 'extreme'
FAILED	Fortigate - VPN SSL cipher suite > than 128 bits
FAILED	Fortigate - Webfilter License - Not Expired
FAILED	The device does not appear to support or is not configured for administrative password policy ...
PASSED	Fortigate - AAA - TACACS+ server is trusted
PASSED	Fortigate - Admin access - trusted hosts
PASSED	Fortigate - Admin password lockout >= 300 seconds
PASSED	Fortigate - AV Grayware - 'Adware'
PASSED	Fortigate - AV Grayware - 'BHO'
PASSED	Fortigate - AV Heuristic - 'block'
PASSED	Fortigate - DNS - primary server
PASSED	Fortigate - DNS - secondary server
PASSED	Fortigate - External Logging - 'syslogd'

## Syntax

The syntax for this plugin and an audit are as follows:

```
<custom_item>
description: "Fortigate - SSH login grace time <= 30 seconds"
info: "SSH login grace time <= 30 seconds."
reference: "HIPAA|HIPAA 164.308(a)(5)(ii)(D), SANS-CSC|16, PCI|2.2.3, 800-53|AC-2(5)"
solution: "Issue the following command to configure SSH login grace time.

config system global
set admin-ssh-grace-time <time_int>
end"
context: "config system global"
regex: "set[\\s]+admin-ssh-grace-time"
expect: "set[\\s]+admin-ssh-grace-time[\\s]+([1-2][0-9]|30)$"
</custom_item>
```

---

The **description**, **info**, **reference**, and **solution** keywords can contain arbitrary text, and their purpose is straightforward. These keywords allow a user to include metadata related to a check within an `.audit` file. Note that the **description** keyword is required, but any of the others are optional.

This audit detects whether a setting is compliant or not based on the **regex**, **expect**, and **not\_expect** keywords. As of the release of the Fortigate plugin (January 21, 2014), Tenable will support six variations of these keywords to perform a compliance audit moving forward.

### no regex, expect, or not\_expect

If no **regex**, **expect**, or **not\_expect** keywords are set, then the check will either report the entire config (or if **cmd** is specified the entire command output).

```
<custom_item>
  description: "Fortigate - HTTPS/SSH admin access strong ciphers"
  context: "config system global"
</custom_item>
```

The above check will report the entire "config system global" context.

### regex only

If only **regex** is specified then all lines matching the regex will be reported.

```
<custom_item>
  description: "Fortigate - Review Admin Settings"
  context: "config system global"
  regex: "set[\\s]+admin-."
</custom_item>
```

This option is primarily for informational purposes. For example, the check above will list all the admin settings under the global context. If no matching lines are found, the check will issue a WARNING result, unless **required** is set to YES, in which case the check will issue a FAIL.

### expect only

If only **expect** is specified, then the check will PASS as long as a matching line/config item has been found.

```
<custom_item>
  description: "Fortigate - Admin password lockout = 300 seconds"
  context: "config system global"
  expect: "set[\\s]+admin-lockout-duration[\\s]+300$"
</custom_item>
```

The check above will pass as long as the admin password lockout is set to 300 seconds.

### not\_expect only

If only the **not\_expect** keyword is specified, then the check will PASS as long as a matching line/config item does not exist.

```
<custom_item>
```





```
description: "Fortigate - Use non default admin access ports - 'HTTPS'"
context: "config system global"
not_expect: "set[\\s]+admin-sport[\\s]+443$"
</custom_item>
```

The check above will FAIL if admin port is set to 443.

## regex and expect

If both the **regex** and **expect** keywords are specified, then the **regex** extracts all the relevant lines from the config, and **expect** performs the config audit. If any line matching the **regex** does not match the **expect**, the check will FAIL.

```
<custom_item>
description: "Fortigate - DNS - primary server"
context: "config system dns"
regex: "set[\\s]+primary"
expect: "set[\\s]+primary[\\s]+1.1.1.1"
</custom_item>
```

## regex and not\_expect

If both the **regex** and **not\_expect** keywords are specified, then the **regex** extracts are the relevant lines from the config, and **not\_expect** performs the config audit. If any line matching the **regex** matches the **not\_expect**, the check will FAIL.

```
<custom_item>
description: "Fortigate - Disable insecure services - TELNET"
context: "config system interface"
regex: "set[\\s]+allowaccess"
not_expect: "set[\\s]+allowaccess[\\s]+.*?(telnet[\\s]|telnet$)"
</custom_item>
```

The check above will fail if telnet is enabled in the config.

## context

The concept of context is not applicable to all compliance plugins. When the config of a device is structured in such a way that one or more lines are applicable to a single section of the config, then we use the **context** keyword to audit that specific section of the **.audit**. For example, in the following, the example admin settings are configured/mapped to the global config:

```
config system global
  set access-banner disable
  set admin-https-pki-required disable
  set admin-lockout-duration 60
  set admin-lockout-threshold 3
  set admin-maintainer enable
  set admin-port 80
.
```

## cmd

The plugin also supports the **cmd** keyword. This allows users to run any **get** or **show** command, and then include the resulting output in the report.

```
<custom_item>
  description: "Fortigate - Review users with admin privileges"
  cmd: "get system admin"
  expect: ".+"
  severity: MEDIUM
</custom_item>
```

The check above lists admin users found on the target.

## Amazon AWS Compliance File Reference

The Amazon AWS audit includes checks for running instances, network acls, firewall config, account attributes, user listing, and more. A valid set of Amazon AWS access keys and access to an IAM account assigned to a **ReadOnly** access group are required to audit the remote instance. The Amazon AWS scan differs from a typical Nessus audit in one major way; it doesn't have any designated targets since AWS is a web-based service. Compliance results will output like any others:

Status ▲	Plugin Name	Plugin Family	Count
PASSED	List Users	Amazon AWS Compliance Checks	1
PASSED	List Vpcs	Amazon AWS Compliance Checks	1

## Audit File Syntax

Here is an example of an Amazon AWS configuration check:

```
<custom_item>
  type: CONFIG_CHECK
  description: "Verify login authentication"
  info: "Verifies login authentication configuration"
  reference: "PCI|2.2.3,SANS-CSC|1"
  context: "line .*"
  item: "login authentication"
</custom_item>
```

The keywords **description**, **info**, **reference**, and **solution** keywords can contain any text. It allows users to include metadata related to a check within an **.audit** file. With the exception of the **description** keyword, all other keywords are optional.

## Keywords

The following table indicates how each keyword in the Amazon AWS compliance checks can be used:

Keyword	Example Use and Supported Settings
<b>Type</b>	The keyword type specifies the API we are tapping into to pull back the information (in this case IAM).
<b>description</b>	The “ <b>description</b> ” 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 that no distinct checks have the same <b>description</b> field. Tenable’s SecurityCenter uses this field to automatically generate a unique plugin ID number based on the <b>description</b> field.
<b>info</b>	<p>The “<b>info</b>” keyword is used to add a more detailed description to the check that is being performed. Rationale for the check could be a regulation, URL with more information, corporate policy, and more. Multiple <b>info</b> fields can be added on separate lines to format the text as a paragraph. There is no preset limit to the number of <b>info</b> fields that can be used.</p> <div>  <p>Each “<b>info</b>” tag must be written on a separate line with no line breaks. If more than one line is required (e.g., formatting reasons), add additional “<b>info</b>” tags.</p> </div> <p>Example:  info: "Review the the list of interfaces"  info: "Disable unused interfaces"</p>
<b>aws_action</b>	This keyword specifies the Amazon API action we are running against the AWS setup.
<b>xsl_stmt</b>	This keyword gives you a way to define the XSL Transform that will be applied on the XML file you get back after running the API request.
<b>regex</b>	<p>The “<b>regex</b>” keyword enables searching the configuration item setting to match for a particular regular expression.</p> <p>Example:  regex: "set system syslog .+"</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> <p>If a check has “<b>regex</b>” tag set, but no “<b>expect</b>” or “<b>not_expect</b>” or “<b>number_of_lines</b>” tag is set, then the check simply reports all lines matching the regex.</p>
<b>expect</b>	<p>This keyword allows auditing the configuration item matched by the “<b>regex</b>” tag or if the “<b>regex</b>” tag is not used it looks for the “<b>expect</b>” string in the entire config.</p> <p>The check passes as long as the config line found by “<b>regex</b>” matches the “<b>expect</b>” tag</p>



	or in the case where “ <b>regex</b> ” is not set, it passes if the “ <b>expect</b> ” string is found in the config.
<b>not_expect</b>	<p>This keyword allows searching the configuration items that should not be in the configuration.</p> <p>It acts as the opposite of “<b>expect</b>”. The check passes as the config line found by “<b>regex</b>” does not match the “<b>not_expect</b>” tag or if the “<b>regex</b>” tag is not set, it passes as long as “<b>not_expect</b>” string is not found in the config.</p>

If **regex**, **expect**, and **not\_expect** are not specified, it will report the entire output from the API query.

## Debugging

If there are any problems that caused the scan not to work, there is a new debug flag in the audit that triggers the plugin to run in debug mode. Add `<debug/>` anywhere in the audit, and the plugin will log verbose information that will help you troubleshoot the plugin issues.

## Known Good Auditing

Compliance auditing is all about consistency and conformance to a known good standard, and being able to demonstrate a system matches it repeatedly. If a system deviates from a known good value it is critical to know about it, so that you can isolate what happened and any impact that may result from the deviation. This is typically done with a combination of **regex**, **expect**, **not\_expect**, and other similar types of compliance directives. This method is versatile and functional, but eventually hits a limitation when comparing two blobs of text. No matter how well-formed your regex syntax is, there simply isn't a way around comparing a large blob of text against a known good value. With this in mind, you can utilize a feature that is designed to do this allowing for the comparison of a blob of text against a “known good” value.

For the feature to work, the user must copy the acceptable value to a **known\_good** keyword. More than one good values are allowed but are separated by a comma. For example:

```
<custom_item>
  Description: "EC2: DescribeRegions - 'Regions that are currently available'"
  type: EC2
  aws_action: "DescribeRegions"
  xsl_stmt: "<xsl:template match=\"/\">"
  xsl_stmt: "<xsl:for-each select=\"/ec2:item\">"
  xsl_stmt: "Region: <xsl:value-of select=\"/ec2:regionName\"/> End-Point: <xsl:value-of
    select=\"/ec2:regionEndpoint\"/><xsl:text>#10;</xsl:text>"
  xsl_stmt: "</xsl:for-each>"
  xsl_stmt: "</xsl:template>"
  known_good: 'us-east-1:
Region: eu-west-1 End-Point: ec2.eu-west-1.amazonaws.com
Region: sa-east-1 End-Point: ec2.sa-east-1.amazonaws.com
Region: us-east-1 End-Point: ec2.us-east-1.amazonaws.com
Region: ap-northeast-1 End-Point: ec2.ap-northeast-1.amazonaws.com
Region: ap-northeast-2 End-Point: ec2.ap-northeast-1.amazonaws.com
Region: us-west-2 End-Point: ec2.us-west-2.amazonaws.com
Region: us-west-1 End-Point: ec2.us-west-1.amazonaws.com
Region: ap-southeast-2 End-Point: ec2.ap-southeast-2.amazonaws.com'
</custom_item>
```

```
Output

us-east-1:
Region: eu-west-1 End-Point: ec2.eu-west-1.amazonaws.com
Region: sa-east-1 End-Point: ec2.sa-east-1.amazonaws.com
Region: us-east-1 End-Point: ec2.us-east-1.amazonaws.com
Region: ap-northeast-1 End-Point: ec2.ap-northeast-1.amazonaws.com
Region: us-west-2 End-Point: ec2.us-west-2.amazonaws.com
Region: us-west-1 End-Point: ec2.us-west-1.amazonaws.com
Region: ap-southeast-1 End-Point: ec2.ap-southeast-1.amazonaws.com
Region: ap-southeast-2 End-Point: ec2.ap-southeast-2.amazonaws.com

No known good matches found.

--- actual
+++ known_good_1
@@ -1,8 +1,9 @@
+us-east-1:
Region: eu-west-1 End-Point: ec2.eu-west-1.amazonaws.com
Region: sa-east-1 End-Point: ec2.sa-east-1.amazonaws.com
Region: us-east-1 End-Point: ec2.us-east-1.amazonaws.com
Region: ap-northeast-1 End-Point: ec2.ap-northeast-1.amazonaws.com
+Region: ap-northeast-2 End-Point: ec2.ap-northeast-1.amazonaws.com
+Region: us-west-2 End-Point: ec2.us-west-2.amazonaws.com
Region: us-west-1 End-Point: ec2.us-west-1.amazonaws.com
-Region: ap-southeast-1 End-Point: ec2.ap-southeast-1.amazonaws.com
Region: ap-southeast-2 End-Point: ec2.ap-southeast-2.amazonaws.com

Status ▼ Hosts
FAILED Amazon AWS
```

Notice in the output that a diff is included for ease in auditing.

## Use Cases

One of the most useful use cases of this feature is to create a “Gold Standard” audit with all known good values. For example, users would be able to run a scan against a target configured to meet the requirements, grab “known\_good” values from the `.nessus` file, update the audit file, and run the scan again to receive an “all pass” result.

## Miscellaneous

- `known_good` overrides `expect` and `not_expect` but does take into account `regex`. So if a `regex` is specified, the output will be compared against the regex-filtered data.
- More than one `known_good` can be specified in a rule but must be separated by a comma.
- The feature is implemented as a standalone feature in an `.inc` file, and can be easily used in any Nessus plugin as well.

---

## Adtran AOS Compliance File Reference

The Adtran AOS audit includes checks for password policy, enabled services, insecure service configuration, authentication, logging & audit settings, and SNMP & NTP configuration settings. Valid SSH credentials for root or an administrator with full privileges are required.

FAILED	Adtran : Disable FTP
FAILED	Adtran : Disable SSLv2
FAILED	Adtran : Disable Telnet
FAILED	Adtran : Disable TFTP
FAILED	Adtran : Enable aaa
FAILED	Adtran : Enable aaa authentication
FAILED	Adtran : Enable NTP
FAILED	Adtran : Enable service password-encryption
FAILED	Adtran : Encrypt passwords
FAILED	Adtran : Forward logs to syslog server

## Syntax

The syntax for this plugin and an audit are as follows:

```
<custom_item>
description: "Adtran : Disable FTP"
info: "Disable ftp server, if not required."
not_expect: "^ip ftp server"
solution: "Do disable FTP Server, run the following command :\n
no ip ftp server"
reference: "PCI|2.2.3,SANS-CSC|10,CSF|PR.DS-2,800-53|AC-17,800-53|SC-9"
</custom_item>
```

## SonicWALL SonicOS Compliance File Reference

The SonicWALL SonicOS audit includes checks the SSL configuration, password policy, banner configuration, administrative access ports, inactivity timeout setting, flood protection setting, client AV enforcement policy, logging & audit settings, enabled security services, gateway anti-virus configuration, authorization & authentication settings, and intrusion prevention service configuration.



The SSH implementation on SonicWall may be unreliable at times based on extensive testing. If the SSH API fails during an audit, Tenable recommends that you use the offline config audit method.

FAILED	SonicWALL - User Inactivity Timeout - 5 minutes or less
FAILED	SonicWALL - Web Interface - Does not use self-signed cert
PASSED	SonicWALL - AAA - LDAP server is trusted
PASSED	SonicWALL - AAA - RADIUS server is trusted
PASSED	SonicWALL - AutoDownload Firmware - Enabled
PASSED	SonicWALL - AutoUpdate - Enabled
PASSED	SonicWALL - AV License - Not Expired
PASSED	SonicWALL - Client AV Enforcement On - DMZ
PASSED	SonicWALL - Client AV Enforcement On - LAN

## Syntax

The syntax for this plugin and an audit are as follows:

```
<custom_item>
description: "SonicWALL - Disable insecure services - HTTP"
info: "HTTP is insecure by nature as it sends all traffic across the wire in clear
      text."
solution: "Navigate to network->interfaces. Configure each interface by unchecking
          the http management box."
reference: "800-53|CM-7,SANS-CSC|11,SANS-CSC|10,PCI|2.2.3,CSF|PR.PT-3,800-53|CM-6"
cmd: "show interface all"
regex: "http[\\s]mgmt"
not_expect: "http[\\s]mgmt[\\s]+on"
</custom_item>
```

## Extreme ExtremeXOS Compliance File Reference

The Extreme ExtremeXOS audit includes checks for the password policy, banner configuration, inactivity timeout setting, logging & audit settings, insecure services, device license information, and SNMP settings.

FAILED	Extreme : Enable SNMP Traps
FAILED	Extreme : SNMP community name != private
FAILED	Extreme : SNMP community name != public
PASSED	Extreme : Configure Banner before-login
PASSED	Extreme : Configure idletimeout <= 15
PASSED	Extreme : Configure timezone = UTC
PASSED	Extreme : Device Info
PASSED	Extreme : Disable Telnet
PASSED	Extreme : License Info
PASSED	Extreme : Only allow SNMPv3
PASSED	Extreme : Password Policy - char-validation
PASSED	Extreme : Password Policy - history <=4
PASSED	Extreme : Password Policy - lockout-on-login-failures

## Syntax

The syntax for this plugin and an audit are as follows:

```
<custom_item>
description: "Extreme : Password Policy - min-length >= 8"
info: "Do not allow password lengths less than 8 characters"
expect: "configure account all password-policy min-length ([8-9]|[1-9][0-9]+) "
solution: "Run the following command to enforce min password length :\n
    configure account all password-policy min-length 8"
reference: "SANS-
    CSC|10,HIPAA|164.308(a)(5)(ii)(D),PCI|2.2.4,PCI|8.2.3,COBIT5|BAI10.01,800-
    53|CM-2"
</custom_item>
```



## Red Hat Enterprise Virtualization (RHEV) Compliance File Reference

The Red Hat Enterprise Virtualization (RHEV) audit includes checks for the currently running or stopped VMs, product version, users, roles and group configuration, as well as datacenter and cluster information. To audit a device, admin SSH credentials for the Red Hat Enterprise Manager Admin portal are required.

FAILED	RHEV: Review Events with severity >= Error	1
PASSED	RHEV: Administrative Roles	1
PASSED	RHEV: Clusters	1
PASSED	RHEV: Datacenters	1
PASSED	RHEV: Disks	1
PASSED	RHEV: Domains	1
PASSED	RHEV: Groups	1
PASSED	RHEV: Hosts	1
PASSED	RHEV: Product Info	1
PASSED	RHEV: Roles	1

The plugin supports evaluation of output by `regex`, `expect`, `not_expect`, and `known_good` keywords.

### Syntax

The syntax for this plugin and an audit are as follows:

```
<custom_item>
  description: "RHEV: Authorized Users"
  info: "Make sure only authorized users allowed to log in to the target."
  request: "/api/users"
  xsl_stmt: '<xsl:template match="users">
<xsl:for-each select="user">
  UserName: <xsl:value-of select="user_name"/>
  Name: <xsl:value-of select="name"/>
-</xsl:for-each>
</xsl:template>'
  solution: "Review the list of users, and disable any unauthorized users"
</custom_item>
```

This plugin also allows you to include API requests with the search feature. The following example runs a search for events that have a severity of greater than or equal to "error".

```
<custom_item>
```



```
description: "RHEV: Review Events with severity >= Error"
request: "/api/events?search=severity>=error"
xsl_stmt: '<xsl:template match="events">
<xsl:for-each select="event">
  description: <xsl:value-of select="description"/>
  time: <xsl:value-of select="time"/>
-
</xsl:for-each>
</xsl:template>'
not_expect : "Description"
</custom_item>
```

## Debugging

Adding a `<debug/>` string anywhere in the audit will force the plugin to run in debug mode. This may be helpful figuring out any issues with an audit, and will assist Tenable support should you need it. The debug log will be saved to the Nessus `tmp` directory in a subdirectory called `/compliance_debug`. On Red Hat, the full path would be `/opt/nessus/var/nessus/tmp/compliance_debug/`.

## MongoDB Compliance File Reference

The MongoDB audit includes checks for authentication, user listing, RBAC configuration, version Info, server status, host information, audit and logging info, SSL configuration, service configuration, IP and port configuration, and general MongoDB settings.

FAILED	Run MongoDB with Secure Configuration Options - version - 'mongoDB version is 2.4.10 or hi.
PASSED	Configure RBAC - DB Roles - 'Role List'
PASSED	Configure RBAC - Role Privileges - 'Roles with authSchemaUpgrade privilege'
PASSED	Configure RBAC - Role Privileges - 'Roles with changeCustomData privilege'
PASSED	Configure RBAC - Role Privileges - 'Roles with changePassword privilege'
PASSED	Configure RBAC - Role Privileges - 'Roles with closeAllDatabases privilege'
PASSED	Configure RBAC - Role Privileges - 'Roles with createRole privilege'
PASSED	Configure RBAC - Role Privileges - 'Roles with createUser privilege'
PASSED	Configure RBAC - Role Privileges - 'Roles with dropDatabase privilege'
PASSED	Configure RBAC - Role Privileges - 'Roles with dropRole privilege'
PASSED	Configure RBAC - Role Privileges - 'Roles with dropUser privilege'
PASSED	Configure RBAC - Role Privileges - 'Roles with getCmdLineOpts privilege'
PASSED	Configure RBAC - Role Privileges - 'Roles with getParameter privilege'



MongoDB is a NoSQL database, which means it does not use the SQL query language for accessing the data.

## Syntax

The syntax for this plugin and an audit are as follows:

```
<custom_item>
  description: "MongoDB - single_user_in_any_database"
  mongo_function: "single_user_in_any_database"
  known_good: "no single-user databases"
</custom_item>

<custom_item>
  description: "MongoDB - matching_hashes"
  mongo_function: "matching_hashes"
  known_good: "no matching hashes"
</custom_item>

<custom_item>
  description: "MongoDB - user_can_eval"
  mongo_function: "user_can_eval"
  known_good: "no user can run eval commands"
</custom_item>
```

MongoDB audit can also support custom checks:

```
<custom_item>
  description: "Require Authentication - DB Users - 'User authenticated by MONGODB-CR'"
  collection: "admin.system.users"
  query: '{"credentials.MONGODB-CR": {"$exists": 1}}'
  fieldsSelector: '{"_id": 0, "user" : 1}'
  regex: "user"
</custom_item>
```

Keyword	Example Use and Supported Settings
<b>description</b>	<p>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's SecurityCenter uses this field to automatically generate a unique plugin ID number based on the <b>description</b> field.</p> <p>Example:</p> <pre>description: "Require Authentication - DB users - 'User authenticated by MongoDB'"</pre>
<b>collection</b>	<p>The name of the MongoDB that the plugin connects to to get information.</p> <p>Example:</p> <pre>info: 'admin.system.users'.</pre>

<b>query</b>	The MongoDB query.  Example: query: '{"credentials.MONGODB-CR": {"\$exists": 1}}'
<b>fieldsSelector</b>	This is an optional field that allows selecting specific attributes from a result. This field the equivalent of “select attribute from database” from a traditional database.  Example: fieldsSelector: '{"_id": 0, "user" : 1}'

The MongoDB audit also supports **regex**, **expect**, **not\_expect**, and **known\_good** keywords in its syntax.

## OpenStack

This plugin queries an OpenStack deployment through the REST API and provides a snapshot of the complete deployment (e.g., active/inactive servers, users, networks, subnets). When used in combination with the OpenStack audits for Unix compliance plugin, this plugin/audit can be used to harden a typical OpenStack deployment.

PASSED	OpenStack Active Servers
PASSED	OpenStack Deployment Snapshot
PASSED	OpenStack Inactive Servers
PASSED	OpenStack Networks and their attached subnets
PASSED	OpenStack Server Flavors

## Syntax

The syntax for this plugin and an audit are as follows:

```
<custom_item>
description: "Arbitrary text"
info: "Arbitrary text"
solution: "Arbitrary text"
reference: "REF|ID1,REF|ID2"
service: 'service to audit'           # compute,network or identity
request: 'rest query'
json_transform: '' (optional) # json transform to perform on the query output
expect: ""                          # expected value
severity: LOW MEDIUM OR HIGH
</custom_item>
```

## Example Queries

```
<custom_item>
description: "OpenStack Servers and their details"
info: "The Servers and their current state will determine what services are
      available."
solution: "Review the list of Servers. If any are unknown or not in the expected
          state they should be investigated."
reference: "CCM-3|IVS-07,HIPAA|164.308(a)(2)(D),800-53|CM-2,800-53|CM-6,800-53|CM-
          8,800-53|PM-7,PCI-DSS|2.2"
service: 'compute'
request: 'servers/detail'
json_transform: '.servers[]|
                "\n\nName: " + .name
                + "\nID: " + .id
                + "\nStatus: " + .status
                + "\nUser_ID: " + .user_id
                + "\nCreated: " + .created
                + "\nUpdated: " + .updated
                + "\nHost_ID: " + .hostId
                + "\nTenant_ID: " + .tenant_id
                + "\n- addresses: - " + ([.addresses.[]|.[]|.addr] | join("\n - "))
                '
expect: ""
severity: LOW
</custom_item>
```

```
<custom_item>
description: "OpenStack Deployment Snapshot"
info: "The OpenStack resources and their current state will determine what services
      are available."
solution: "Review the list of OpenStack resources. If any are unknown they should be
          investigated."
reference: "CCM-3|IVS-07,HIPAA|164.308(a)(2)(D),800-53|CM-2,800-53|CM-6,800-53|CM-
          8,800-53|PM-7,PCI-DSS|2.2"
see_also: "http://docs.openstack.org/"
service: 'compute'
request: 'limits'
json_transform: 'openstack_data
                " Users: \\.users | length)\n"
                + ([.users[] | " \\.id) - \\.username)\n"] | sort | join(""))
                + " Servers: \\.servers | length)\n"
                + ([.servers[] | " \\.id) - \\.name)\n"] | sort | join(""))
                + " Networks: \\.networks | length)\n"
                + ([.networks|.networks[] | " \\.id) - \\.name)\n"] | sort | join(""))
                + " Ports: \\.networks|.ports | length)\n"
                + ([.networks|.ports[] | " \\.id)\n"] | sort | join(""))
                + " Subnets: \\.networks|.subnets | length)\n"
                + ([.networks|.subnets[] | " \\.id) - \\.name)\n"] | sort | join(""))
                + " Images: \\.images | length)\n"
                + ([.images[] | " \\.id) - \\.name)\n"] | sort | join(""))
                '
expect: ""
severity: LOW
</custom_item>
```



Keyword	Example Use and Supported Settings
<b>description</b>	This is the information used as a title for unique compliance mis-configuration in SecurityCenter. It will also be the first set of data reported by Nessus.
<b>info</b>	This keyword allows users to add a more detailed description to the check that is being performed. Multiple info fields are allowed with no preset limit. The <b>info</b> content should be enclosed in double quotes.
<b>see_also</b>	This keyword allows users to include links that might provide helpful information about a check, e.g., "http://docs.openstack.org/".
<b>request</b>	This keyword describes the type of REST API request for OpenStack.
<b>regex</b>	This keyword allows searching items that match a particular regex expression.
<b>expect</b>	This keyword provides matching text from the query output.
<b>service</b>	This keyword indicates the service (compute, identity, network) which will be queried by the plugin.
<b>json_transform</b>	The keyword provides the json_transform that will be performed on the output of the check.

## Salesforce Compliance File Reference

The Salesforce audit includes checks for the network-based security settings, secure data access, user access options, object permissions, session security, password policies, federated authentication settings, single sign-on configuration, login history, cron jobs, and email services.



FAILED	Setting Session Security - '...	Salesforce.com Compliance Checks	1
FAILED	Setting Session Security - '...	Salesforce.com Compliance Checks	1
FAILED	Setting Session Security - '...	Salesforce.com Compliance Checks	1
FAILED	Setting Session Security - '...	Salesforce.com Compliance Checks	1
FAILED	Setting Session Security - '...	Salesforce.com Compliance Checks	1
PASSED	Monitoring Login History - ...	Salesforce.com Compliance Checks	1
PASSED	Monitoring Login History - ...	Salesforce.com Compliance Checks	1
PASSED	Monitoring Login History - ...	Salesforce.com Compliance Checks	1
PASSED	Network-Based Security - '...	Salesforce.com Compliance Checks	1
PASSED	Securing Data Access - 'Ch...	Salesforce.com Compliance Checks	1

## Setup Requirements

One of these two methods are required to allow Nessus access:

1. Add the scanner IP to the Trusted IP Ranges in Salesforce
2. Use a security token.

### Adding a trusted IP range

1. In Salesforce, go to Setup -> Security Controls -> Network Access
2. Add the public IP the scanner will use to connect to Salesforce, or a range of IP addresses. This is the IP address as it will appear to Salesforce, not an internal IP behind NAT.
3. When you enter the credentials in Salesforce plugin preferences in Nessus:
  - a. Enter the username
  - b. Enter the user password

### Using a security token

1. Log in as the user you will use and reset their security token if you do not already have it. The security token is sent via email to the user.
2. When you enter the credentials in Salesforce plugin preferences in Nessus:
  - a. Enter the username

- 
- b. Append the security token to the user password (e.g., If the security password is "MyPassword" and the security token is "MyToken", enter "MyPasswordMyToken")

## Syntax

The syntax for this plugin and an audit are as follows:

```
<custom_item>
  description: "List SecuritySettings details"
  settings_name: "SecuritySettings"
</custom_item>
```

The following values for **settings\_name** are allowed:

- AccountSettings
- ActivitiesSettings
- AddressSettings
- CaseSettings
- ChatterAnswersSettings
- CompanySettings
- ContractSettings
- EntitlementSettings
- ForecastingSettings
- IdeasSettings
- KnowledgeSettings
- MobileSettings
- SecuritySettings

The plugin supports evaluation of output by:

- xsl\_stmt
- regex/expect/not\_expect
- known\_good

## Example Queries

Simple example query:

```
<custom_item>
  description: "List user names"
  query: "SELECT Name FROM User"
</custom_item>
```

Lookup example query that returns the Name of the user who created each user, instead of listing a GUID:

```
<custom_item>
  description: "List user names and who added them"
  query: "SELECT Name, CreatedBy.Name FROM User"
</custom_item>
```

Join example query that returns information from the PermissionSet assigned to the user, crossing two tables/object types:



```
<custom_item>
  description: "List user names and whether the permission set assigned to them
    prevents password expiration"
  query: "SELECT Name, (SELECT PermissionSet.PermissionsPasswordNeverExpires FROM
    PermissionSetAssignments) FROM User"
</custom_item>
```

## BlueCoat ProxySG Compliance File Reference

The BlueCoat ProxySG audit includes checks for the syslog configuration, SNMP settings, intercepted protocols, general settings, password settings, authentication methods, and more. To audit a device, admin SSH credentials and enable credentials via the “cisco enable” option are required.

Note that a full configuration dump suitable for backups is available on these devices via the **show configuration expanded noprompts with-keyrings unencrypted** command. However, this is not used to avoid the plaintext passwords being included in the Nessus KB.

### Syntax

Any command beginning with show is allowed. The syntax for this plugin and an audit are as follows:

```
<custom_item>
  description: "BlueCoat:SSL Mode"
  info: "Make sure SSL mode is enabled"
  solution: "Turn on SSL"
  see_also: "https://bto.bluecoat.com/documentation/pubs/ProxySG"
  reference: "PCI|2.2.3"
  expect: "ssl.;mode"
</custom_item>
```

### Context

Configuration blocks may be indicated in two ways:

1. Begin with a line ending in ;mode, end with exit
2. Begin with a line starting with edit, and with exit

These options may be nested by including multiple context tags. For example:

```
!- BEGIN networking
interface 0:0 ;mode
ip-address 172.1.2.34 255.255.252.0
exit
ip-default-gateway 172.1.0.1 1 100
dns-forwarding ;mode
edit primary
clear server
add server 172.200.1.23
exit
edit alternate
```

```

clear server
exit
exit
!- END networking
!- BEGIN ssl
ssl ;mode
edit primary
certificate disable
exit
exit
!- END ssl

```

## Database Configuration Audit Compliance File Reference

This section describes the format and functions of the database compliance checks and the rationale behind each setting.

### Check Type

All database compliance checks must be bracketed with the **check\_type** encapsulation and the “Database” designation. This is required to differentiate **.audit** files intended specifically for databases from other types of compliance audits. The **check\_type** field requires two additional parameters:

- **db\_type**
- **version**

Available database types for audits include:

- SQLServer
- Oracle
- MySQL
- PostgreSQL
- DB2
- Informix

The **version** is currently always set to “1”.

Example:

```
<check_type: "Database" db_type:"SQLServer" version:"1">
```

### Keywords

The following table indicates how each keyword in the database compliance checks can be used:

Keyword	Example Use and Supported Settings
<b>type</b>	SQL_POLICY
<b>description</b>	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’s SecurityCenter uses this field to



	<p>automatically generate a unique plugin ID number based on the <b>description</b> field.</p> <p>Example: description: "DBMS Password Complexity"</p>
<b>info</b>	<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 other reason why the setting is required. Multiple <b>info</b> fields can be added on separate lines to format the text as a paragraph. There is no preset limit to the number of <b>info</b> fields that can be used.</p> <p>Example: info: "Checking that \"password complexity\" requirements are enforced for systems using SQL Server authentication."</p>
<b>sql_request</b>	<p>This keyword is used to determine the actual <b>SQL</b> request to be submitted to the database. Arrays of data may be requested and returned from a <b>SQL</b> request by using comma-delimited request/return values.</p> <p>Example: sql_request: "select name from sys.sql_logins where type = 'S' and is_policy_checked &lt;&gt; '1'"</p> <p>Example: sql_request: "select name, value_in_use from sys.configurations where name = 'clr enabled'"</p>
<b>sql_types</b>	<p>This keyword has two available options: <b>POLICY_VARCHAR</b> and <b>POLICY_INTEGER</b>. Use <b>POLICY_INTEGER</b> for numeric values from 0 to 2147483647 and <b>POLICY_VARCHAR</b> for any other return value type.</p> <p>Example: sql_types: POLICY_VARCHAR</p> <p>Example: sql_types: POLICY_VARCHAR, POLICY_INTEGER</p> <p>For multiple return items, configure <b>sql_types</b> in a comma-separated list to accept the data types of each <b>SQL</b> return result. The example above indicates that the first return value from the <b>SQL</b> query is <b>varchar</b> and the second return value is an integer.</p>
<b>sql_expect</b>	<p>This keyword is used to determine the return value expected from the <b>SQL</b> request. An exact value including <b>NULL</b> or "0" may be required. Additionally, regular expressions may be required for <b>POLICY_VARCHAR sql_types</b>.</p> <p>Example: sql_expect: regex:"^.+Failure"    regex:"^.+ALL"</p> <p>Example: sql_expect: NULL</p> <p>Example: sql_expect: 0    "0"</p> <p>Double-quotes are optional for integer return values.</p>



#### Example:

```
sql_expect: "clr enabled",0
```

An array of data may be returned from a **SQL** request and included in a comma-separated format in the **sql\_expect** field.

## Command Line Examples

This section provides some examples of common audits used for database compliance checks. The **nasl** command line binary is used as a quick means of testing audits on the fly. Each of the **.audit** files demonstrated below can easily be dropped into your Nessus 6 or SecurityCenter scan policies. For quick audits of one system, however, command-line tests are more efficient. The command will be executed each time from the **/opt/nessus/bin** directory as follows:

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

The **<IP>** is the IP address of the system to be audited.

Depending on the type of database being audited you may be prompted for other parameters beyond the audit file to be used. For example, Oracle audits will prompt for the database SID and the Oracle login type:

```
Which file contains your security policy : oracle.audit
login : admin
Password :
Database type: ORACLE(0), SQL Server(1), MySQL(2), DB2(3), Informix/DRDA(4),
              PostgreSQL(5)
type : 0
sid: oracle
Oracle login type: NORMAL (0), SYSOPER (1), SYSDBA (2)
type: 2
```

Consult with your database administrator for the correct database login parameters.

### Example 1: Search for logins with no expiration date

Following is a simple **.audit** file that looks for any SQL Server logins with no expiration date. If any are found, the audit will display a failure message along with the offending login(s).

```
<check_type: "Database" db_type:"SQLServer" version:"1">
<group_policy: "Login expiration check">
<custom_item>
  type: SQL_POLICY
  description: "Login expiration check"
  info: "Database logins with no expiration date pose a security threat. "
  sql_request: "select name from sys.sql_logins where type = 'S' and
               is_expiration_checked = 0"
  sql_types: POLICY_VARCHAR
  sql_expect: NULL
</custom_item>
</group_policy>
</check_type>
```

When running this command, the following output is expected from a compliant system:

```
"Login expiration check": [PASSED]
```

Compliance requirements usually mandate that database logins have an expiration date.

A failed audit would return the following output:

```
"Login expiration check": [FAILED]

Database logins with no expiration date pose a security threat.

Remote value:

"distributor_admin"

Policy value:

NULL
```

This output indicates that the “distributor\_admin” account has no configured expiration date and needs to be checked against the system security policy.

## Example 2: Check enabled state of unauthorized stored procedure

This audit checks if the stored procedure “SQL Mail XPs” is enabled. External stored procedures can constitute a security threat for some systems and are often required to be disabled.

```
<check_type: "Database" db_type:"SQLServer" version:"1">
<group_policy: "Unauthorized stored procedure check">
<custom_item>
  type: SQL_POLICY
  description: "SQL Mail XPs external stored procedure check"
  info: "Checking whether SQL Mail XPs is disabled."
  sql_request: "select value_in_use from sys.configurations where name = 'SQL Mail
               XPs'"
  sql_types: POLICY_INTEGER
  sql_expect: 0
</custom_item>
</group_policy>
</check_type>
```

The check above will return a “passed” result if the “SQL Mail XPs” stored procedure is disabled (value\_in\_use = 0). Otherwise, it will return a “failed” result.

## Example 3: Check database state with mixed result sql\_types

In some cases, compliance database queries require multiple data requests with multiple data type results. The example audit below mixes data types and demonstrates how the output can be parsed.

```
<check_type: "Database" db_type:"SQLServer" version:"1">
```

```

<group_policy: "Mixed result type check">
<custom_item>
  type: SQL_POLICY
  description: "Mixed result type check"
  info: "Checking values for the master database."
  sql_request: " select database_id,user_access_desc,is_read_only from sys.databases
               where is_trustworthy_on=0 and name = 'master'"
  sql_types: POLICY_INTEGER,POLICY_VARCHAR,POLICY_INTEGER
  sql_expect: 1,MULTI_USER,0
</custom_item>
</group_policy>
</check_type>

```

Note that the `sql_request`, `sql_types`, and `sql_expect` values all contain comma-separated values.

## Conditions

It is possible to define `if/then/else` logic in the database policy. This allows the end-user to return a warning message rather than pass/fail in case an audit passes.

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: SQL_POLICY
      description: "clr enabled option"
      info: "Is CLR enabled?"
      sql_request: "select value_in_use from sys.configurations where name = 'clr
                    enabled'"
      sql_types: POLICY_INTEGER
      sql_expect: "0"
    </custom_item>
  </condition>

  <then>
    <custom_item>
      type: SQL_POLICY
      description: "clr enabled option"

```

```

info: "CLR is disabled?"
sql_request: "select value_in_use from sys.configurations where name = 'clr
enabled'"
sql_types: POLICY_INTEGER
sql_expect: "0"
</custom_item>
</then>

<else>
<report type: "WARNING">
description: "clr enabled option"
info: "CLR(Command Language Runtime objects) is enabled"
info: "Check system policy to confirm CLR requirements."
</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”.

## Unix Configuration Audit Compliance File Reference

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

### Check Type

All Unix compliance checks must be bracketed with the “**check\_type**” encapsulation and the “Unix” designation. [Appendix A](#) 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.



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.

### Keywords

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

Keyword	Example Usage and Supported Settings
<b>attr</b>	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 chatter(1) man page for details on configuring the file attributes of a file.
<b>comment</b>	<p>This field is used to add any additional information that does not fit into the description field.</p> <p>Example: comment: (CWD - Current working directory)</p>

<b>description</b>	<p>This keyword provides a brief description of the check that is being performed. It is required that the <b>description</b> field is unique and no two checks should have the same description field. Tenable's SecurityCenter uses this field to automatically generate a unique plugin ID number based on the <b>description</b> field.</p> <p>Example: description: "Permission and ownership check for /etc/at.allow"</p>
<b>dont_echo_cmd</b>	<p>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.</p> <p>Example: dont_echo_cmd: YES</p>
<b>except</b>	<p>This keyword is used to exclude certain users, services and files from the check.</p> <p>Example: except: "guest"</p> <p>Multiple user accounts can be piped together.</p> <p>Example: except: "guest"   "guest1"   "guest2"</p>
<b>expect</b>	<p>This keyword is used in combination with <b>regex</b>. It provides the ability to look for specific values within files.</p> <p>Example: &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;</p>
<b>file</b>	<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: file: "/etc/inet/inetd.conf" file: "~/inetd.conf"</p> <p>The <b>file</b> value can also be a glob.</p> <p>Example: file: "/var/log/*"</p> <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>



<b>file_type</b>	<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: file_type: "f"</p> <p>One or more types of file types can be piped together in the same string.</p> <p>Example: file_type: "c b"</p>
<b>group</b>	<p>This keyword is used to specify the group of a file; it is always used in conjunction with <b>file</b> keyword. The <b>group</b> keyword can have a value of "none" that helps with searching for files with no owner.</p> <p>Example: group: "root"</p> <p>Group can also be specified with a logical "OR" condition using the following syntax:</p> <p>group: "root"    "bin"    "sys"</p>
<b>ignore</b>	<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>
<b>info</b>	<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 <b>info</b> fields can be added on separate lines to format the text as a paragraph. There is no preset limit to the number of <b>info</b> fields that can be used.</p> <p>Example: info: "ref. CIS_AIX_Benchmark_v1.0.1.pdf ch 1, pg 28-29."</p>
<b>levels</b>	<p>This keyword is used in conjunction with CHKCONFIG and is used to specify the runlevels for which a service is required to be running. All the runlevels 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 <b>levels</b> value in the CHKCONFIG check would be:</p>



	<code>levels: "123"</code>
<b>mask</b>	<p>This keyword is the opposite of <b>mode</b> where one can specify permissions that should <b>not</b> 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 <b>mode</b> 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: <code>mask: 022</code></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>
<b>md5</b>	<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: <code>&lt;custom_item&gt;</code> <code>  type: FILE_CHECK</code> <code>  description: "/etc/passwd has the proper md5 set"</code> <code>  required: YES</code> <code>  file: "/etc/passwd"</code> <code>  md5: "ce35dc081fd848763cab2cfd442f8c22"</code> <code>&lt;/custom_item&gt;</code></p>
<b>mode</b>	<p>This keyword describes the set of permissions for a file/folder under consideration. The <b>mode</b> keyword can be represented in string or octal format.</p> <p>Examples: <code>mode: "-rw-r--r--"</code> <code>mode: "644"</code> <code>mode: "7644"</code></p>
<b>name</b>	<p>This keyword is used to identify process name in PROCESS_CHECK.</p> <p>Example: <code>name: "syslogd"</code></p>
<b>operator</b>	<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>• <b>lt</b> (less than)</li><li>• <b>lte</b> (less than or equal)</li><li>• <b>gte</b> (greater than equal)</li><li>• <b>gt</b> (greater than)</li><li>• <b>eq</b> (equal)</li></ul> <p>Example: <code>operator: "lt"</code></p>

<b>owner</b>	<p>This keyword is used to specify the owner of a file; it is always used in conjunction with <b>file</b> keyword. The <b>owner</b> keyword can have a value of “none” that helps with searching for files with no owner.</p> <p>Example: owner: "root"</p> <p>Ownership can also be specified with a logical “OR” condition using the following syntax: owner: "root"    "bin"    "adm"</p>
<b>reference</b>	<p>This keyword provides a way to include cross-references in the <b>.audit</b>. The format is “ref ref-id1,ref ref-id2”.</p> <p>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"</p>
<b>regex</b>	<p>This keyword enables searching a file to match for a particular regex expression.</p> <p>Example: 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>
<b>required</b>	<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 <b>.audit</b> 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>
<b>rpm</b>	<p>This keyword is used to specify the RPM to look for when used in conjunction with RPM_CHECK.</p> <p>Example: &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;</p>
<b>search_locations</b>	<p>This keyword can be used to specify searchable locations within a file system.</p> <p>Example: search_locations: "/bin"</p> <p>Multiple search locations can be piped together.</p>

	<p>Example:</p> <pre>search_locations: "/bin"   "/etc/init.d"   "/etc/rc0.d"</pre>
<b>see_also</b>	<p>This keyword allows to include links to a reference.</p> <p>Example:</p> <pre>see_also: "https://benchmarks.cisecurity.org/tools2/linux/CIS_Redhat_Linux_5_Benchmark_v2.0.0.pdf"</pre>
<b>service</b>	<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>
<b>severity</b>	<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>
<b>solution</b>	<p>This keyword provides a way to include “Solution” text if available.</p> <p>Example:</p> <pre>solution: "Remove this file, if its not required"</pre>
<b>status</b>	<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 <b>status</b> keyword can take 2 values: “ON” or “OFF”.</p> <p>Example:</p> <pre>status: ON status: OFF</pre>
<b>system</b>	<p>This keyword specifies the type of system the check is to be performed on.</p> <div data-bbox="511 1528 581 1598"> </div> <div data-bbox="617 1528 1511 1614"> <p>The “<b>system</b>” keyword is only applicable to “<b>custom_item</b>” checks, not built-in “<b>item</b>” checks.</p> </div> <p>The available values are the ones returned by the “<b>uname</b>” command on the target OS. For example, on Solaris the value is “SunOS”, on Mac OS X it is “Darwin”, on FreeBSD it is “FreeBSD”, etc.</p> <p>Example:</p> <pre>system: "SunOS"</pre>
<b>timeout</b>	<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.</p>

	<p>This keyword is useful in cases where a particular command, such as the Unix “<b>find</b>” 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: timeout: "600"</p>
<b>type</b>	<p>CHKCONFIG CMD_EXEC FILE_CHECK FILE_CHECK_NOT FILE_CONTENT_CHECK FILE_CONTENT_CHECK_NOT GRAMMAR_CHECK PKG_CHECK PROCESS_CHECK RPM_CHECK SVC_PROP XINETD_SVC</p>
<b>value</b>	<p>The <b>value</b> keyword is useful to check if a setting on the system confirms to the policy value.</p> <p>Example: value: "90..max"</p> <p>The <b>value</b> keyword can be specified as a range [number..max]. If the value lies between the specified number and “max”, the check will pass.</p>

## Custom Items

A custom item is a complete check defined on the basis of the keywords defined above. The following is 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.



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

## 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 (see [Appendix C](#) for more information). 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.

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.txtimer.DatabasePersistencePolicy']/
  attribute[@name='DatabasePersistencePlugin']/text()\"/>"
xsl_stmt: "</xsl:template>"
regex: "DatabasePersistencePlugin = .+"
not_expect: "org.jboss.ejb.txtimer.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.txtimer.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.

```
<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.

```
<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.

```
<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>
```

## 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**.



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

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.

Example:

```
<custom_item>
  type: CMD_EXEC
  description: "Make sure that we are running FreeBSD 4.9 or higher"
  cmd: "uname -a"
  timeout: 7200
```



```
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)  
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”.

Here are some 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.

Here are some 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(s).

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.

Here are some 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>
```

By adding a “~” to a file parameter, it is possible to have FILE\_CONTENT\_CHECK scan 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**.

The allowed type is:

```
value_type: POLICY_TEXT
value_data: "PATH\\Filename"
regex: "regex"
expect: "regex"
```

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

Here is an 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.

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 MAC OS X. This check behaves differently if certain properties are set.

If **plist\_user** is set to “all”, all user settings are audited, otherwise the specified user setting is audited.

If the **byhost** property is set to YES in addition to **plist\_user** property being set, the following query is run:

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

If the **byhost** property is not set (and **plist\_user** property is set), then the following query is run:

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

If the **byhost** property is not set (and **plist\_user** property is not set), the following query is run:

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

The following properties are supported:

**plist\_name** : the plist we want to query. for e.g. com.apple.digihub  
**plist\_item** : The plist item to be audited. for e.g. com.apple.digihub.blank.cd.appeared  
**plist\_option** : CANNOT\_BE\_NULL. If this is set to CANNOT\_BE\_NULL, the check fails if the setting being audited is not set.  
**byhost** : YES. Setting byhost to YES, results in a slightly different query.

## Examples:

```
<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>
```

## 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.

## 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.

Examples:

```
<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**, **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.



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

Here are some examples, assuming **iproute-2.4.7-10** is installed:

```
<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.

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**.



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

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.



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

## Password Management



In the examples below **<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.



## *min\_password\_length*

### 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."
  except: "user1" | "user2" (list of users to be excluded)
  value: "<min>..<max>"
</item>
```

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 PASS_MIN_LEN in <code>/etc/login.defs</code> .
Solaris	The minimum password length is defined as PASSLENGTH in <code>/etc/default/passwd</code> . Note that this also controls the password maximum length.
HP-UX	The minimum password length is defined as MIN_PASSWORD_LENGTH in <code>/etc/default/security</code> .
Mac OS X	The minimum password length is defined as “minChar” in the local policy, defined using the command <code>pwpolicy</code> .

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*

### 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."
  except: "user1" | "user2" (list of users to be excluded)
  value: "<min>..<max>"
</item>
```

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> .
Mac OS X	The option "maxMinutesUntilChangePassword" of the password policy (as set through the <code>pwpolicy</code> tool) can be used to set this value.

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*

### 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."
  except: "user1" | "user2" (list of users to be excluded)
  value: "<min>..<max>"
</item>
```

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 <code>PASS_MIN_DAYS</code> is defined in <code>/etc/login.defs</code> .

<b>Solaris</b>	The variable MINWEEKS in <code>/etc/default/passwd</code> defines the maximum number of weeks a password can be used.
<b>HP-UX</b>	This value is controlled by the variable PASSWORD_MINDAYS in <code>/etc/default/security</code> .
<b>Mac OS X</b>	This option is not supported.

Example:

```
<item>7
  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*

#### Usage

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

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

---

## Permissions Management

### *accounts\_bad\_home\_permissions*

#### Usage

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

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 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.

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

#### 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>
```

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.

### *accounts\_without\_home\_dir*

#### Usage

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

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 **/**.

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

#### Usage

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

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 **/**.

### *invalid\_login\_shells*

#### 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>
```

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.

---

## *login\_shells\_with\_suid*

### Usage

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

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”.

## *login\_shells\_writeable*

### Usage

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

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.

## *login\_shells\_bad\_owner*

### 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>
```

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.

---

## Password File Management

### *passwd\_file\_consistency*

#### Usage

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

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**.

### *passwd\_zero\_uid*

#### Usage

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

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".

## *passwd\_duplicate\_uid*

### Usage

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

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 lookup 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.

## *passwd\_duplicate\_gid*

### Usage

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

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.

## *passwd\_duplicate\_username*

### Usage

```
<item>
  name: "passwd_duplicate_username"
```



---

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

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.

### *passwd\_duplicate\_home*

#### Usage

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

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.

### *passwd\_shadowed*

#### Usage

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

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.

---

## *passwd\_invalid\_gid*

### Usage

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

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.

## Group File Management

### *group\_file\_consistency*

### Usage

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

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.

### *group\_zero\_gid*

### Usage

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

---

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.

### *group\_duplicate\_name*

#### Usage

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

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.

### *group\_duplicate\_gid*

#### Usage

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

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.

### *group\_duplicate\_members*

#### Usage

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

---

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.

### *group\_nonexistant\_users*

#### Usage

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

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**.

## Root Environment

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

#### 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>
```

This check ensures that the current working directory (".") is not included in the executable path of the root user. 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.

## *writeable\_dirs\_in\_root\_path\_variable*

### 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>
```

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
```

## File Permissions

### *find\_orphan\_files*

### 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) dir: "<directory>"
</item>
```

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**. When searching file systems, it will, by default, ignore any directories mounted over NFS unless they have been specified with the optional keyword **dir**.

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. A default timeout value, which is the time after which Nessus will stop processing results for this check, has been set at five hours and this value cannot be changed.

Example:

```
<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>
```

## *find\_world\_writeable\_files*

### 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) dir: "<directory>"
</item>
```

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**. When searching file systems, it will, by default, ignore any directories mounted over NFS unless they have been specified with the optional keyword **dir**.

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. A default timeout value, which is the time after which Nessus will stop processing results for this check, has been set at five hours and this value cannot be changed.

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*

### 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) dir: "<directory>"
</item>
```

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**. When searching file systems, it will, by default, ignore any directories mounted over NFS unless they have been specified with the optional keyword **dir**.

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. A default timeout value, which is the time after which Nessus will stop processing results for this check, has been set at five hours and this value cannot be changed.

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*

### 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) dir: "<directory>"
```

---



---

```
</item>
```

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**. When searching file systems, it will, by default, ignore any directories mounted over NFS unless they have been specified with the optional keyword **dir**.

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. A default timeout value, which is the time after which Nessus will stop processing results for this check, has been set at five hours and this value cannot be changed.

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"
  dir: "/home/extended"
</item>
```

## *find\_suid\_sgid\_files*

### 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) dir: "<directory>"
</item>
```

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.

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**. When searching file systems, it will, by default, ignore any directories mounted over NFS unless they have been specified with the optional keyword **dir**.



---

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. A default timeout value, which is the time after which Nessus will stop processing results for this check, has been set at five hours and this value cannot be changed.

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`
- `.dtprofile`
- `.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`
- `.dtprofile`
- `.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*

#### 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>
```

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`.

## Unnecessary Files

### *find\_pre-CIS\_files*

#### Usage

```
<item>
  name: "find_preCIS_files"
  description: "Find and list all files created by CIS backup script."
  # Globbs allowed (? and *)
  (optional) basedir: "<directory>"
  (optional) ignore: "<directory>"
</item>
```

---

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. A default timeout value, which is the time after which Nessus will stop processing results for this check, has been set at five hours and this value cannot be changed.

## 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: "Make sure /etc/passwd contains root"
      file: "/etc/passwd"
      owner: "root"
    </custom_item>
  </condition>

  <then>
    <custom_item>
      type: FILE_CONTENT_CHECK
      description: "Make sure /etc/passwd contains root (then)"
      file: "/etc/passwd"
```

```

    regex: "^root"
    expect: "^root"
  </custom_item>
</then>

<else>
  <custom_item>
    type: FILE_CONTENT_CHECK
    description: "Make sure /etc/passwd contains root (else)"
    file: "/etc/passwd"
    regex: "^root"
    expect: "^root"
  </custom_item>
</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**”.

## 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 noncompliant data.

### 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

#### 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>
```

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	Description
<b>type</b>	This must always be set to FILE_CONTENT_CHECK
<b>description</b>	This is the information that will be used as a title for unique compliance vulnerabilities in the SecurityCenter. It will also be the first set of data reported by Nessus.
<b>File_extension</b>	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.
<b>Regex</b>	<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>  The <b>regex</b> keyword must be run with the <b>expect</b> keyword described below.         </div> <div>  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.         </div>
<b>Expect</b>	<p>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.</p> <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>  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.         </div>

	 <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 <code>&lt;max_size&gt;</code> bytes of the file searched.</p>
<b>File_name</b>	<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>
<b>Max_size</b>	<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>
<b>Only_show</b>	<p>When matching sensitive data such as credit card numbers, your organization may require that only the last four digits be made visible in the report. This keyword supports revealing any number of bytes specified by policy.</p>
<b>Regex_replace</b>	<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>
<b>Include_paths</b>	<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 “<b>exclude_paths</b>” 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> <div>  <p>Only drive letters or folder names can be specified with the “<b>include_paths</b>” keyword. File names cannot be included in the “<b>include_paths</b>” value string.</p> </div>
<b>Exclude_paths</b>	<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 “<b>include_paths</b>” keyword. This is particularly helpful in cases where a particular drive, directory, or file must be excluded from search results. Paths are double-quoted and separated by the pipe symbol where multiple paths are required.</p>
<b>See_also</b>	<p>This keyword allows to include links to a reference.</p> <p>Example:  see_also:  "https://benchmarks.cisecurity.org/tools2/linux/CIS_Redhat_Linux_5_Benchmark_v2.0.0.pdf"</p>
<b>solution</b>	<p>This keyword provides a way to include “Solution” text if available.</p> <p>Example:  solution: "Remove this file, if its not required"</p>

<b>reference</b>	<p>This keyword provides a way to include cross-references in the <code>.audit</code>. The format is "ref ref-id1,ref ref-id2".</p> <p>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"</p>
<b>luhn</b>	<p>Setting <code>luhn</code> to YES forces the plugin to only report credit card numbers that are Luhn algorithm verified.</p>

## 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 we are showing, you can easily drop these into your Nessus 6 or SecurityCenter 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.

## Target Test File

The target file we will be using has the following content in it:

```
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".

## Example 1: 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" | "xltx" |
  "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.

## Example 2: Search files for a properly formatted AMEX credit card number

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 had an AMEX credit card number in them.



---

## 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 `.nbin` 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 a SecurityCenter, the data only needs to travel from the scanned Unix host to the scanner performing the vulnerability audit.

## NetApp Data ONTAP

This section describes the format and functions of the storage systems running NetApp Data ONTAP compliance checks and the rationale behind each setting.

Compliance Summary

Sort Options

Filter compliance checks

failed	1.2 Secure Storage Design - 'cifs.signing.enable = on'	NetApp Data ONTAP Compliance	1
failed	1.2 Secure Storage Design - 'cifs.signing.enable = on'	NetApp Data ONTAP Compliance	1
failed	1.2 Secure Storage Design - 'cifs.smb2.signing.required = on...	NetApp Data ONTAP Compliance	1
failed	1.2 Secure Storage Design - 'ldap.ssl.enable = on'	NetApp Data ONTAP Compliance	1
failed	1.2 Secure Storage Design - 'nfs.v3.enable = off'	NetApp Data ONTAP Compliance	1
failed	1.2 Secure Storage Design - 'nfs.v4.enable = on'	NetApp Data ONTAP Compliance	1
failed	1.2 Secure Storage Design - Enable Kerberos with CIFS - 'nfs...	NetApp Data ONTAP Compliance	1
failed	1.2 Secure Storage Design, Enable Kerberos with NFS - 'nfs.k...	NetApp Data ONTAP Compliance	1
failed	2.0 Install & Config - 'Disable SNMPv2'	NetApp Data ONTAP Compliance	1
failed	2.0 Install & Config - 'Disable SSHv1'	NetApp Data ONTAP Compliance	1
failed	2.0 Install & Config - 'Disable WebDAV'	NetApp Data ONTAP Compliance	1
failed	2.0 Install & Config - 'Enable TLSv1'	NetApp Data ONTAP Compliance	1
failed	2.0 Install & Config - 'Secure Sockets Layer v2 (SSLv2'	NetApp Data ONTAP Compliance	1

## Required User Privileges

To perform a successful compliance scan against a NetApp Data ONTAP system, authenticated users must have privileges as defined below:

`root` credentials for NetApp Data ONTAP filer

In addition to the privileges above, an audit policy for NetApp Data ONTAP Compliance Checks and Nessus Plugin ID #66934 (NetApp Data ONTAP Compliance Checks) are required.

To run a scan against the device, start by creating the audit policy. Next, use the “**SSH settings**” menu under the “**Credentials**” tab of the policy to supply `root` credentials. Under the “**Plugins**” tab of the policy, select the “Policy Compliance” plugin family, and enable plugin ID #66934 titled “**NetApp Data ONTAP Compliance Checks**”. Next, under the “**Preferences**” tab, select the “NetApp Data ONTAP Compliance Checks” drop-down and add the NetApp `.audit` file from the Tenable Support Portal. Last, save the policy and execute the scan.

In the case where providing `root` credentials is not an option, a lesser privileged account can be created to facilitate the audit:

1. Create a new role (e.g., `nessus_audit`):  

```
# role add nessus_audit -a login-ssh,cli-version,cli-options,cli-uptime
```
2. Assign the role to a group (e.g., `nessus_admins`):  

```
# group add nessus_admins -r nessus_audit
```

3. Assign the group to a user:


```
# useradmin user add nessus -g nessus_admins
```

## Check Type: CONFIG\_CHECK

NetApp compliance checks are bracketed in `custom_item` encapsulation and CONFIG\_CHECK. This is treated like any other `.audit` files and work for systems running the NetApp Data ONTAP system. The CONFIG\_CHECK check consists of two or more keywords. Keywords `type` and `description` are mandatory, which are followed by one or more keywords. The check works by auditing the “options” command output.

## Keywords

The following table indicates how each keyword in the NetApp Data ONTAP compliance checks can be used:

Keyword	Example Use and Supported Settings
<code>type</code>	“CHECK_CONFIG” determines if the specified config item exists in the NetApp Data ONTAP “show configuration” output.
<code>description</code>	<p>The “<code>description</code>” keyword provides the ability to add a brief description of the check that is being performed. It is strongly recommended that the <code>description</code> field be unique and that no distinct checks have the same <code>description</code> field. Tenable’s SecurityCenter uses this field to automatically generate a unique plugin ID number based on the <code>description</code> field.</p> <p>Example: description: "1.0 Require strong Password Controls - 'min-password-length &gt;= 8'"</p>
<code>info</code>	<p>The “<code>info</code>” keyword is used to add a more detailed description to the check that is being performed. Rationale for the check could be a regulation, URL with more information, corporate policy, and more. Multiple <code>info</code> fields can be added on separate lines to format the text as a paragraph. There is no preset limit to the number of <code>info</code> fields that can be used.</p> <div><p>Each “<code>info</code>” tag must be written on a separate line with no line breaks. If more than one line is required (e.g., formatting reasons), add additional “<code>info</code>” tags.</p></div> <p>Example: info: "Enable palindrome-check on passwords"</p>
<code>severity</code>	<p>The “<code>severity</code>” keyword specifies the severity of the check being performed.</p> <p>Example: severity: MEDIUM</p> <p>The severity can be set to HIGH, MEDIUM, or LOW.</p>
<code>regex</code>	The “ <code>regex</code> ” keyword enables searching the configuration item setting to match for a particular regular expression.



	<p>Example: regex: "set snmp .+"</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> <p>If a check has "regex" tag set, but no "expect" or "not_expect" or "number_of_lines" tag is set, then the check simply reports all lines matching the regex.</p>
<b>expect</b>	<p>This keyword allows auditing the configuration item matched by the "regex" tag or if the "regex" tag is not used it looks for the "expect" string in the entire config.</p> <p>The check passes as long as the config line found by "regex" matches the "expect" tag or in the case where "regex" is not set, it passes if the "expect" string is found in the config.</p> <p>Example: regex: "set password-controls complexity" expect: "set password-controls complexity [1-4]"</p> <p>In the above case, the "expect" tag ensures that the complexity is set to a value between 1 and 4.</p>
<b>not_expect</b>	<p>This keyword allows searching the configuration items that should not be in the configuration.</p> <p>It acts as the opposite of "expect". The check passes as the config line found by "regex" does not match the "not_expect" tag or if the "regex" tag is not set, it passes as long as "not_expect" string is not found in the config.</p> <p>Example: regex: "set password-controls password-expiration" not_expect: "set password-controls password-expiration never"</p> <p>In the above case, the "not_expect" tag ensures that the password-controls are not set to "never".</p>

## CONFIG\_CHECK Examples

The following are examples of using CONFIG\_CHECK against a NetApp Data ONTAP device:

```
<custom_item>
type: CONFIG_CHECK
description: "1.2 Secure Storage Design, Enable Kerberos with NFS -
'nfs.kerberos.enable = on'"
info: "NetApp recommends the use of security features in IP storage protocols to
secure client access"
```



```
solution: "Enable Kerberos with NFS"
reference: "PCI|2.2.3"
see_also: "http://media.netapp.com/documents/tr-3649.pdf"
regex: "nfs.kerberos.enable[\\s\\t]+"
expect: "nfs.kerberos.enable[\\s\\t]+on"
</custom_item>
```

## Conditions

It is possible to define **if/then/else** logic in the NetApp Data ONTAP audit policy. This allows the end-user to use a single file that is able to handle multiple configurations.

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: CONFIG_CHECK
  description: "2.6 Install and configure Encrypted Connections to devices - 'telnet'"
  regex: "set net-access telnet"
  expect: "set net-access telnet off"
  info: "Do not use plain-text protocols."
</custom_item>
</condition>
<then>
  <report type: "PASSED">
    description: "Telnet is disabled"
  </report>
</then>
<else>
  <custom_item>
  type: CONFIG_CHECK
  description: "2.6 Install and configure Encrypted Connections to devices - 'telnet'"
  regex: "set net-access telnet"
  expect: "set net-access telnet off"
  info: "Do not use plain-text protocols."
</custom_item>
  </else>
</if>
```

---

The condition never shows up in the report - that is, whether it fails or passes it won't show up (it's a "silent" check).

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

## Reporting

Can be performed in a <then> or <else> to achieve a desired PASSED/FAILED condition.

```
<if>
  <condition type: "OR">
    <custom_item>
      type: CONFIG_CHECK
      description: "2.6 Install and configure Encrypted Connections to devices - 'telnet'"
      regex: "set net-access telnet"
      expect: "set net-access telnet off"
      info: "Do not use plain-text protocols."
    </custom_item>
  </condition>
  <then>
    <report type: "PASSED">
      description: "Telnet is disabled"
    </report>
  </then>
  <else>
    <report type: "FAILED">
      description: "Telnet is disabled"
    </report>
  </else>
</if>
```

PASSED, WARNING, and FAILED are acceptable values for "report type".

## IBM iSeries Configuration Audit Compliance File Reference

This section describes the format and functions of the IBM iSeries compliance checks and the rationale behind each setting.

### Required User Privileges

To perform a successful compliance scan against an iSeries system, authenticated users must have privileges as defined below:

1. A user with (\*ALLOBJ) or audit (\*AUDIT) authority can audit all system values. Such a user typically belongs to class (\*SECOFR).
2. Users of class (\*USER) or (\*SYSOPR) can audit most values, except QAUDCTL, QAUDENDACN, QAUDFRCLVL, QAUDLVL, QAUDLVL2, and QCRTOBJAUD.

If a user does not have privileges to access a value, then the value returned will be \*NOTAVL.

## Check Type

All IBM iSeries compliance checks must be bracketed with the **check\_type** encapsulation and the “AS/400” designation. This is required to differentiate .**audit** files intended specifically for systems running an IBM iSeries system from other types of compliance audits.

Example:

```
<check_type:"AS/400">
```

Unlike other compliance audit types, no additional type or version keywords are available.

## Keywords

The following table indicates how each keyword in the IBM iSeries compliance checks can be used:

Keyword	Example Use and Supported Settings
<b>type</b>	AUDIT_SYSTEMVAL SHOW_SYSTEMVAL
<b>systemvalue</b>	This keyword is used to specify a specific value to be checked within the IBM iSeries system.  Example: systemvalue: "QALWUSRDMN"
<b>description</b>	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's SecurityCenter uses this field to automatically generate a unique plugin ID number based on the <b>description</b> field.  Example: description: "Allow User Domain Objects (QALWUSRDMN) - '*all'"
<b>value_type</b>	This keyword is used to define the type of value (either “POLICY_DWORD” or “POLICY_TEXT”) being checked on the IBM iSeries system.  Example: value_type: "POLICY_DWORD"  Example: value_type: "POLICY_TEXT"
<b>value_data</b>	This keyword defines that data value that is expected for a system value.  Example: value_type: "^[6-9] [1-9][0-9]+)\$"
<b>check_type</b>	This keyword defines the type of check being used against a data value.  Examples: check_type: "CHECK_EQUAL" check_type: "CHECK_NOT_EQUAL" check_type: "CHECK_GREATER_THAN"



	<pre>check_type: "CHECK_GREATER_THAN_OR_EQUAL" check_type: "CHECK_LESS_THAN" check_type: "CHECK_LESS_THAN_OR_EQUAL" check_type: "CHECK_REGEX"  Example: &lt;custom_item&gt;   type: AUDIT_SYSTEMVAL   systemvalue: "QUSEADPAUT"   description: "Use Adopted Authority (QUSEADPAUT) - '!= *none'"   value_type: POLICY_TEXT   value_data: "*none"   check_type: CHECK_NOT_EQUAL &lt;/custom_item&gt;</pre>
<b>info</b>	<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 other reason why the setting is required. Multiple <b>info</b> fields can be added on separate lines to format the text as a paragraph. There is no preset limit to the number of <b>info</b> fields that can be used.</p> <p>Example:</p> <pre>info: "\nref : http://publib.boulder.ibm.com/infocenter/ iseries/v5r4/topic/books/sc415302.pdf pg. 21"</pre>

## Custom Items

A custom item is a complete check defined on the basis of the keywords defined above. The following is a list of available custom item types. 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.



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

### AUDIT\_SYSTEMVAL

“AUDIT\_SYSTEMVALUE” audits the value of the configuration setting identified by “**systemvalue**” keyword. The type of comparison against the value being audited is specified by the “**check\_type**” keyword.

```
<custom_item>
  type: AUDIT_SYSTEMVAL
  systemvalue: "QALWUSRDMN"
  description: "Allow User Domain Objects (QALWUSRDMN) - '*all'"
  value_type: POLICY_TEXT
  value_data: "*all"
  info: "\nref :
    http://publib.boulder.ibm.com/infocenter/iseries/v5r4/topic/books/sc415302.pdf
    pg. 21"
</custom_item>
```



---

## SHOW\_SYSTEMVAL

The “SHOW\_SYSTEMVAL” audit only reports the value of the configuration setting identified by the “**systemvalue**” keyword.

```
<custom_item>
  type: SHOW_SYSTEMVAL
  systemvalue: "QAUDCTL"
  description: "show QAUDCTL value"
  severity: MEDIUM
</custom_item>
```

## Conditions

It is possible to define **if/then/else** logic in the IBM iSeries policy. This allows the end-user to return a warning message rather than pass/fail in case an audit passes.

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: AUDIT_SYSTEMVAL
      systemvalue: "QDSPSGNINF"
      description: "Sign-on information is displayed (QDSPSGNINF)"
      info: "\nref :
        http://publib.boulder.ibm.com/infocenter/iseriess/v5r4/topic/books/sc415302.pdf
        pg. 23"
      value_type: POLICY_DWORD
      value_data: "1"
    </custom_item>
  </condition>

  <then>
    <custom_item>
      type: AUDIT_SYSTEMVAL
      systemvalue: "QDSPSGNINF"
      description: "Sign-on information is not displayed (QDSPSGNINF)"
      info: "\nref :
```

```

        http://publib.boulder.ibm.com/infocenter/iserics/v5r4/topic/books/sc415302.pdf
        pg. 23"
    value_type: POLICY_DWORD
    value_data: "1"
</custom_item>
</then>

<else>
<report type: "WARNING">
    description: "Sign-on information is displayed (QDSPSGNINF) "
    info: ""\nref :
        http://publib.boulder.ibm.com/infocenter/iserics/v5r4/topic/books/sc415302.pdf
        pg. 23"
    info: "Check system policy to confirm requirements."
</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**”.

## VMware vCenter/ESXi Configuration Audit Compliance File Reference

This section describes the format and functions of the VMware vCenter and ESXi compliance checks and the rationale behind each setting.

Nessus has the ability to audit VMware via the native APIs by extracting the configuration, and then performing the audit based on the checks listed in the associated `.audit` file.

### Requirements

To perform a successful compliance scan against VMware systems, users must have the following:

1. Administrative credentials for VMware vCenter or ESXi. (Tenable has developed APIs for both ESXi (the interface available for free to manage VMs on ESX/ESXi), and vCenter (an add-on product available from VMware at some cost to manage one or more ESX/ESXi servers). This plugin can leverage either ESXi or vCenter credentials to do its job.)
2. Audit policy for VMware vCenter/ESXi Compliance Checks.
3. Plugin ID #64455 (VMware vCenter/ESXi Compliance Checks)

### Supported Versions

Currently, Nessus can audit ESXi 4.x and 5.x, as well as vCenter 4.x and 5.

---

## Check Type

The syntax for the VMware `.audit` capability relies heavily on XPATH and XSL Transforms to perform the functionality.

The VMware audit supports three types of checks:

### AUDIT\_VM

This check type allows you to audit virtual machine settings (see [Appendix C](#) for more information):

```
<custom_item>
  type: AUDIT_VM
  description: "VM Setting - 'vmsafe.enable = False'"
  xsl_stmt: "<xsl:template match=\"audit:returnval\">"
  xsl_stmt: "<xsl:value-of
    select=\"audit:propSet/audit:val[@xsi:type='VirtualMachineConfigInfo']/audit:na
    me\"/> : vmsafe.enable : <xsl:value-of
    select=\"audit:propSet/audit:val[@xsi:type='VirtualMachineConfigInfo']/audit:ex
    traConfig[audit:key[text()='vmsafe.enable']]/audit:value\"/>."
  xsl_stmt: "</xsl:template>"
  expect: "vmsafe.enable : 0"
</custom_item>
```

### AUDIT\_ESX

This check type allows you to audit ESX/ESXi server settings:

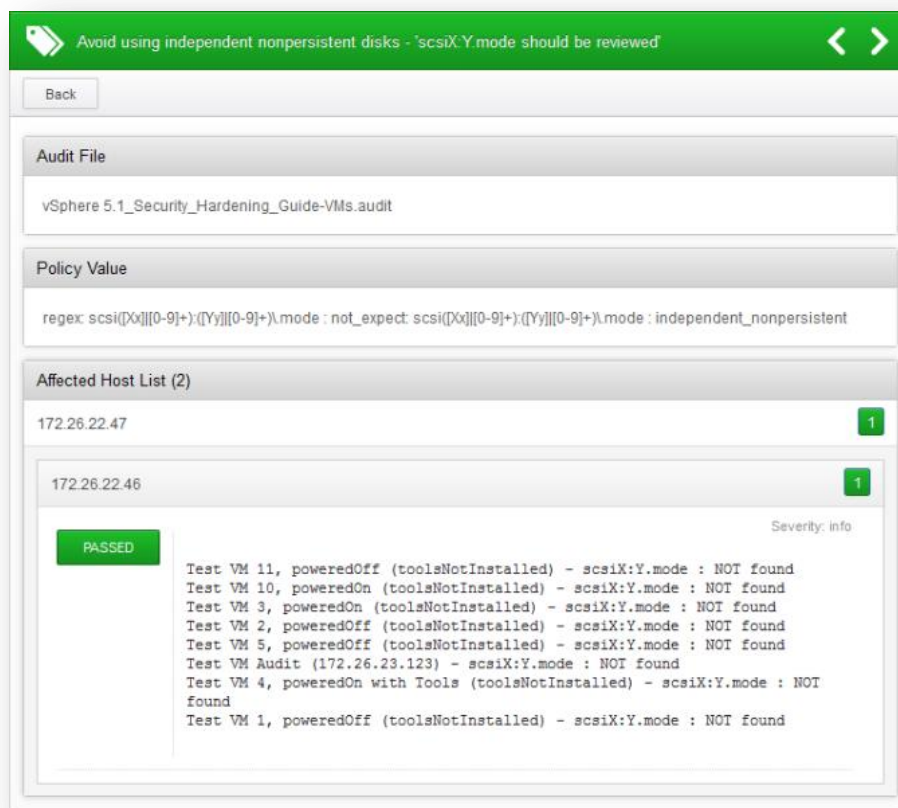
```
<custom_item>
  type: AUDIT_ESX
  description : "ESX/ESXi Setting - Syslog.global.logDir"
  xsl_stmt: "<xsl:template match=\"audit:returnval\">"
  xsl_stmt: "Syslog.global.logDir = <xsl:value-of
    select=\"audit:propSet/audit:val[@xsi:type='HostConfigInfo']/audit:option[audit
    :key[text()='Syslog.global.logDir']]/audit:value\"/>"
  xsl_stmt: "</xsl:template>"
  expect: "Syslog.global.logDir : /foo/bar"
</custom_item>
```

### AUDIT\_VCENTER

This check type allows you to audit vCenter settings:

```
<custom_item>
  type: AUDIT_VCENTER
  description: "VMware vCenter Setting - config.vpxd.hostPasswordLength"
  xsl_stmt: "<xsl:template match=\"audit:returnval\">"
  xsl_stmt: "config.vpxd.hostPasswordLength = <xsl:value-of
    select=\"audit:propSet/audit:val[@xsi:type='ArrayOfOptionValue']/audit:OptionVal
    ue[audit:key[text()='config.vpxd.hostPasswordLength']]/audit:value\"/>"
  xsl_stmt: "</xsl:template>"
  expect: "config.vpxd.hostPasswordLength : 30"
</custom_item>
```

An example of a vSphere audit that passed:



## Keywords

The following table indicates how each keyword in the VMware compliance checks can be used:

Keyword	Example Use and Supported Settings
<b>type</b>	<p>This keyword describes the type of check that is being performed by a given item in an audit file. VMware audits can be performed with the following three types of audit checks:</p> <ul style="list-style-type: none"> <li>AUDIT_VM</li> <li>AUDIT_ESX</li> <li>AUDIT_VCENTER</li> </ul>
<b>description</b>	<p>This keyword gives a brief description of the check that is being performed. It is required that <b>description</b> field be unique and no two checks should have the same <b>description</b> field. This is required because SecurityCenter uses this field to auto generate a plugin ID number based on the <b>description</b> field.</p> <p>Example:  description: "Disconnect unauthorized devices - 'floppyX.present = false'" </p>



<b>info</b>	<p>This keyword allows users to add a more detailed description to the check that is being performed. Multiple <b>info</b> fields are allowed with no preset limit. The info content must be enclosed in double-quotes.</p> <p>Example: info: "Make sure floppy drive is not attached"</p>
<b>regex</b>	<p>This keyword allows searching items that match a particular regex expression.</p> <p>Example: regex: "floppy([Xx] [0-9]+)\\.present :"</p>

The compliance of a check can be determined by comparing the output of the check to either the **expect** or **not\_expect** keyword. You cannot use more than one compliance testing tag in a given check.

Keyword	Example Use and Supported Settings
<b>expect</b>	<p>This keyword allows auditing the config item matched by the <b>regex</b> keyword or if the <b>regex</b> keyword is not used it looks for the <b>expect</b> string in the entire config.</p> <p>The check passes as long as the config line found by <b>regex</b> matches the <b>expect</b> string or in the case where <b>regex</b> is not set, it passes if the <b>expect</b> string is found in the config.</p> <p>Example:</p> <pre>regex: "floppy([Xx] [0-9]+)\\.present :"</pre> <pre>expect: floppy([Xx] [0-9]+)\\.present : false"</pre> <p>Or:</p> <pre>expect: floppy([Xx] [0-9]+)\\.present : false"</pre> <p>In the above cases, the <b>expect</b> keyword ensures that the floppy drive is not present.</p>
<b>not_expect</b>	<p>This keyword allows searching the configuration items that should not be in the configuration.</p> <p>It acts as the opposite of <b>expect</b>. The check passes as long as the config line found by <b>regex</b> does not match the <b>not_expect</b> string or if the <b>regex</b> keyword is not set, it passes as long as <b>not_expect</b> string is not found in the config.</p> <p>Example:</p> <pre>regex: floppy([Xx] [0-9]+)\\.present : " not_expect: floppy([Xx] [0-9]+)\\.present : false"</pre> <p>Or:</p>

```
not_expect: floppy([Xx]|[0-9]+)\\.present : false"
```

In the above cases, the **expect** keyword ensures that the floppy drive is not present.

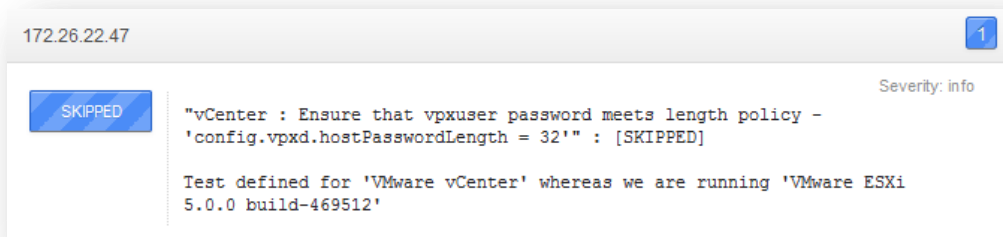
## Additional Notes

If a check passes, this plugin reports all the VMs that matched the policy. The audit supplied by Tenable will report both the VM name and IP of the target. However, note that the IP address for a VM is not available unless VMware tools is installed.

Here's how the reports will show up:

```
Test VM 2, poweredOff (toolsNotInstalled) - vmsafe.enable : NOT found
Test VM Audit (172.26.23.123) - vmsafe.enable : NOT found
```

Both ESX/ESXi and vCenter can be scanned with the same policy. Although note that vCenter checks run against ESX/ESXi hosts will be skipped.



## For Further Information

Tenable has produced a variety of other documents detailing Nessus' installation, deployment, configuration, user operation, and overall testing:

- [Nessus User Guide](#) – includes information to prepare you for installing, configuring, and using Nessus Manager, Nessus Professional, and Nessus Agents
- [Nessus Cloud User Guide](#) – includes information to prepare you for configuring and using Nessus Cloud
- [Nessus v6 SCAP Assessments](#) – describes how to use Tenable's Nessus to generate SCAP content audits as well as view and export the scan results
- [Nessus Compliance Checks](#) – high-level guide to understanding and running compliance checks using Nessus and SecurityCenter
- [Nessus v2 File Format](#) – describes the structure for the **.nessus** file format, which was introduced with Nessus 3.2 and NessusClient 3.2

- 
- [Nessus and Antivirus](#) – outlines how several popular security software packages interact with Nessus, and provides tips or workarounds to allow the software to better co-exist without compromising your security or hindering your vulnerability scanning efforts
  - [Comprehensive Malware Detection with SecurityCenter Continuous View and Nessus](#) – describes how Tenable's SecurityCenter CV can detect a variety of malicious software and identify and determine the extent of malware infections
  - [Real-Time Compliance Monitoring](#) – outlines how Tenable's solutions can be used to assist in meeting many different types of government and financial regulations
  - [Tenable Products Plugin Families](#) – provides a description and summary of the plugin families for Nessus, Log Correlation Engine, and the Passive Vulnerability Scanner

Other online resources are listed below:

- Nessus Discussions Forum: <https://discussions.tenable.com/>
- Tenable Blog: <http://www.tenable.com/blog>
- Tenable Podcast: <http://www.tenable.com/podcast>
- Example Use Videos: <http://www.youtube.com/user/tenablesecurity>
- Tenable Twitter Feed: <http://twitter.com/tenablesecurity>

Please feel free to contact Tenable at [support@tenable.com](mailto:support@tenable.com), [sales@tenable.com](mailto:sales@tenable.com), or visit our website at <http://www.tenable.com/>.

## About Tenable Network Security

Tenable Network Security provides continuous network monitoring to identify vulnerabilities, reduce risk, and ensure compliance. Our family of products includes SecurityCenter Continuous View™, which provides the most comprehensive and integrated view of network health, and Nessus®, the global standard in detecting and assessing network data. Tenable is relied upon by many of the world's largest corporations, not-for-profit organizations and public sector agencies, including the entire U.S. Department of Defense. For more information, visit [tenable.com](http://tenable.com).

---

## Appendix A: Example Unix Compliance File

**Note:** The following file, `tenable_unix_compliance_template.audit`, is available from the Tenable Support Portal located at <https://support.tenable.com/>. This file lists the different types of Unix compliance checks that can be performed using Tenable's Unix compliance module. The actual file may have updates that are not reflected here.

```
#
# (C) 2008-2010 Tenable Network Security, Inc.
#
# This script is released under the Tenable Subscription License and
# may not be used from within scripts released under another license
# without authorization from Tenable Network Security, Inc.
#
# See the following licenses for details:
#
# http://cgi.tenablesecurity.com/Nessus_3_SLA_and_Subscription_Agreement.pdf
# http://cgi.tenablesecurity.com/Subscription_Agreement.pdf
#
# @PROFESSIONALFEED@
#
# $Revision: 1.11 $
# $Date: 2010/11/04 15:54:36 $
#
# NAME                : Cert UNIX Security Checklist v2.0
#
#
# Description         : This file is used to demonstrate the wide range of
#                       checks that can be performed using Tenable's Unix
#                       compliance module. It consists of all the currently
#                       implemented built-in checks along with examples of all
#                       the other Customizable checks. See:
#                       https://plugins-customers.nessus.org/support-
# center/nessus_compliance_checks.pdf
#                       For more information.
#
#
#####
#                               #
# File permission related checks #
#                               #
#####

<check_type:"Unix">

# Example 1.
# File check example with owner and group
# fields set and mode field set in Numeric
# format

<custom_item>
#system          : "Linux"
#type            : FILE_CHECK
#description     : "Permission and ownership check /etc/inetd.conf"
#info           : "Checking that /etc/inetd.conf has owner/group of root and is mode
'600'"
```



```

        file          : "/etc/inetd.conf"
        owner         : "root"
        group         : "root"
        mode          : "600"
</custom_item>

```

```

# Example 2.
# File check example with just owner field set
# and mode set.

```

```

<custom_item>
    #system          : "Linux"
    type             : FILE_CHECK
    description      : "Permission and ownership check /etc/hosts.equiv"
    info             : "Checking that /etc/hosts.equiv is owned by root and mode '500'"
    file             : "/etc/hosts.equiv"
    owner            : "root"
    mode             : "-r-x-----"
</custom_item>

```

```

# Example 3.
# File check example with just file field set
# starting with "~". This check will search
# and audit the file ".rhosts" in home directories
# of all accounts listed in /etc/passwd.

```

```

<custom_item>
    #system          : "Linux"
    type             : FILE_CHECK
    description      : "Permission and ownership check ~/.rhosts"
    info             : "Checking that .rhosts in home directories have the specified
ownership/mode"
    file             : "~/.rhosts"
    owner            : "root"
    mode             : "600"
</custom_item>

```

```

# Example 4.
# File check example with mode field having
# sticky bit set. Notice the first integer in
# the mode field 1 indicates that sticky bit is
# set. The first integer can be modified to check
# for SUID and SGUID fields. Use the table below
# to determine the first integer field.
#
# 0   000   setuid, setgid, sticky bits are cleared
# 1   001   sticky bit is set
# 2   010   setgid bit is set
# 3   011   setgid and sticky bits are set
# 4   100   setuid bit is set
# 5   101   setuid and sticky bits are set
# 6   110   setuid and setgid bits are set
# 7   111   setuid, setgid, sticky bits are set

```

```

<custom_item>

```

```

#system          : "Linux"
type             : FILE_CHECK
description      : "Permission and ownership check /var/tmp"
info            : "Checking that /var/tmp is owned by root and mode '1777'"
file            : "/var/tmp"
owner           : "root"
mode            : "1777"
</custom_item>

```

```

# Example 5.
# File check example with mode field having
# sticky bit set in textual form and is owned by root.

```

```

<custom_item>
#system          : "Linux"
type             : FILE_CHECK
description      : "Permission and ownership check /tmp"
info            : "Checking that the /tmp mode has the sticky bit set in textual form
and is owned by root"
file            : "/tmp"
owner           : "root"
mode            : "-rwxrwxrwt"
</custom_item>

```

```

#####
#                               #
# Service/Process related checks #
#                               #
#####

```

```

# Example 6.
# Process check to audit if fingerd is turned
# OFF on a given host.

```

```

<custom_item>
#system          : "Linux"
type             : PROCESS_CHECK
description      : "Check fingerd process status"
info            : "This check looks for the finger daemon to be 'OFF'"
name            : "fingerd"
status          : OFF
</custom_item>

```

```

# Example 7.
# Process check to audit if sshd is turned
# ON on a given host.

```

```

<custom_item>
#system          : "Linux"
type             : PROCESS_CHECK
description      : "Check sshd process status"
info            : "This check looks for the ssh daemon to be 'ON'"
name            : "sshd"
status          : ON
</custom_item>

```

```

#####

```

```

#                                     #
# File Content related checks #
#                                     #
#####

# Example 8
# File content check to audit if file /etc/host.conf
# contains the string described in the regex field.
#

<custom_item>
    #System                : "Linux"
    type                    : FILE_CONTENT_CHECK
    description             : "This check reports a problem if the order is not 'order hosts,bind'
in /etc/host.conf"
    file                    : "/etc/host.conf"
    search_locations        : "/etc"
    regex                   : "order hosts,bind"
    expect                  : "order hosts,bind"
</custom_item>

# Example 9
# This is a better example of a file content check. It first looks
# for the string ".*LogLevel=.*" and if it matches it checks whether
# it matches .*LogLevel=9. For example, if the file was to have LogLevel=8
# this check will fail since the expected value is set to 9.
#

<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"
    search_locations        : "/etc:/etc/mail:/usr/local/etc/mail"
    regex                   : ".*LogLevel=.*"
    expect                  : ".*LogLevel=9"
</custom_item>

# Example 10
# With compliance checks you can cause the shell to execute a command
# and parse the result to determine compliance. The check below determines
# whether the version of FreeBSD on the remote system is compliant with
# corporate standards. Note that since we determine the system type using
# the "system" tag, the check will skip if the remote OS doesn't match
# the one specified.

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

#####
#                                     #

```

---

# Builtin Checks #  
#  
#####

# Checks that are not customizable are built  
# into the Unix compliance check module. Given below  
# are the list of all the checks are the performed  
# using the builtin functions. Please refer to the  
# the Unix compliance checks documentation for more  
# details about each check.  
#

<item>  
name: "minimum\_password\_length"  
description : "Minimum password length"  
value : "14..MAX"  
</item>

<item>  
name: "max\_password\_age"  
description : "Maximum password age"  
value: "1..90"  
</item>

<item>  
name: "min\_password\_age"  
description : "Minimum password age"  
value: "6..21"  
</item>

<item>  
name: "accounts\_bad\_home\_permissions"  
description : "Account with bad home permissions"  
</item>

<item>  
name: "accounts\_without\_home\_dir"  
description : "Accounts without home directory"  
</item>

<item>  
name: "invalid\_login\_shells"  
description: "Accounts with invalid login shells"  
</item>

<item>  
name: "login\_shells\_with\_suid"  
description : "Accounts with suid login shells"  
</item>

<item>  
name: "login\_shells\_writeable"  
description : "Accounts with writeable shells"  
</item>

<item>

---

```
name: "login_shells_bad_owner"
description : "Shells with bad owner"
</item>

<item>
name: "passwd_file_consistency"
description : "Check passwd file consistency"
</item>

<item>
name: "passwd_zero_uid"
description : "Check zero UID account in /etc/passwd"
</item>

<item>
name : "passwd_duplicate_uid"
description : "Check duplicate accounts in /etc/passwd"
</item>

<item>
name : "passwd_duplicate_gid"
description : "Check duplicate gid in /etc/passwd"
</item>

<item>
name : "passwd_duplicate_username"
description : "Check duplicate username in /etc/passwd"
</item>

<item>
name : "passwd_duplicate_home"
description : "Check duplicate home in /etc/passwd"
</item>

<item>
name : "passwd_shadowed"
description : "Check every passwd is shadowed in /etc/passwd"
</item>

<item>
name: "passwd_invalid_gid"
description : "Check every GID in /etc/passwd resides in /etc/group"
</item>

<item>
name : "group_file_consistency"
description : "Check /etc/group file consistency"
</item>

<item>
name: "group_zero_gid"
description : "Check zero GUID in /etc/group"
</item>

<item>
name: "group_duplicate_name"
description : "Check duplicate group names in /etc/group"
```

---

</item>

<item>  
name: "group\_duplicate\_gid"  
description : "Check duplicate gid in /etc/group"  
</item>

<item>  
name : "group\_duplicate\_members"  
description : "Check duplicate members in /etc/group"  
</item>

<item>  
name: "group\_nonexistant\_users"  
description : "Check for nonexistent users in /etc/group"  
</item>

</check\_type>

---

## Appendix B: Example Windows Compliance File

**Note:** The following file is available from the Tenable Support Portal located at <https://support.tenable.com/>. The actual file may have updates that are not reflected here. This particular script name is called **financial\_microsoft\_windows\_user\_audit\_guideline\_v2.audit** and is based on common hardening guides for user administration. This policy looks for a reasonable password policy, account lockout policy and ensures that login events are logged to the Windows event log.

```
# (C) 2008 Tenable Network Security
#
# This script is released under the Tenable Subscription License and
# may not be used from within scripts released under another license
# without authorization from Tenable Network Security Inc.
#
# See the following licenses for details:
#
# http://cgi.tenablesecurity.com/Nessus_3_SLA_and_Subscription_Agreement.pdf
# http://cgi.tenablesecurity.com/Subscription_Agreement.pdf
#
# @PROFESSIONALFEED@
#
# $Revision: 1.2 $
# $Date: 2008/10/07 15:48:17 $
#
# Synopsis: This file will be read by compliance_check.nbin
#           to check compliance of a Windows host to
#           typical financial institution audit policy
#
<check_type:"Windows" version:"2">
<group_policy:"User audit guideline">

    <item>
    name: "Enforce password history"
    value: 24
    </item>

    <item>
    name: "Maximum password age"
    value: 90
    </item>

    <item>
    name: "Minimum password age"
    value: 1
    </item>

    <item>
    name: "Minimum password length"
    value: [12..14]
    </item>

    <item>
    name: "Account lockout duration"
    value: [15..30]
    </item>
```



```
<item>
name: "Account lockout threshold"
value: [3..5]
</item>

<item>
name: "Reset lockout account counter after"
value: [15..30]
</item>

<item>
  name: "Audit account logon events"
  value: "Success, Failure"
</item>

  <item>
    name: "Audit logon events"
    value: "Success, Failure"
  </item>
</group_policy>
</check_type>
```



---

## Appendix C: XSL Transform to .audit Conversion

Several compliance check plugins rely on auditing XML content, such as Palo Alto, VMware, and Unix compliance checks. To better take advantage of these capabilities, it is beneficial to become familiar with creating XSL Transforms. In some cases, building an XSL Transform will require a bit of trial-and-error. Once you become familiar with that process, converting into an `.audit` is the next step and may not be intuitive. This appendix provides users proper guidance on how to build and utilize custom XSL Transforms, and convert them into `.audit` files.

Several audit checks (e.g., AUDIT\_XML, AUDIT\_VCENTER, AUDIT\_ESX) are separate and distinct, but use the same underlying logic. Understanding the fundamentals of working with XML allow you to translate them directly to other platforms that utilize XML.

By using the `xsltproc` utility, users can follow 7 steps to generate custom `.audit` files for XML content.

### Step 1: Install xsltproc

Verify `xsltproc` is installed on your system, or install it if required. You can verify it is installed and works by typing the command:

```
[tater@pearl ~]# xsltproc
Usage: xsltproc [options] stylesheet file [file ...]
Options:
  --version or -V: show the version of libxml and libxslt used
  --verbose or -v: show logs of what's happening
[...]
```

### Step 2: Identify the XML File to Use

Determine the XML file you are going to use. Verify the location of the file, and that it is XML content. For example:

```
[tater@pearl ~]# ls top-applications.xml
-rw-r--r-- 1 tater gpigs 3857 2011-09-08 21:20 top-applications.xml
[tater@pearl ~]# head top-applications.xml
<?xml version="1.0"?>
<report reportname="top-applications" logtype="appstat">
  <result name="Top applications" logtype="appstat" start="2013/01/29 00:00:00" start-
    epoch="1359446400" end="2013/01/29 23:59:59" end-epoch="1359532799" generated-
    at="2013/01/30 02:02:09" generated-at-epoch="1359540129" range="Tuesday,
    January 29, 2013">
    <entry>
[...]
```

### Step 3: Become Familiar with XSL Transforms and XPath

This process requires a basic understanding of XSL Transforms and XPath concepts. For additional information:

- [w3schools.com](http://w3schools.com) - XSLT – Transformation
- [w3schools.com](http://w3schools.com) - XPath Introduction

---

## Step 4: Create the XSLT Transform

For the next step, the goal is to extract relevant data from an XML file using XSL Transforms. Start by creating an XSL Transform, which is required to extract relevant data from the file. As an example, assume we need to extract the “name” element from an XML. The following XSLT will extract the information required:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="text"/>

<xsl:template match="result">
<xsl:for-each select="entry">
+ <xsl:value-of select="name"/>
</xsl:for-each>

</xsl:template>
</xsl:stylesheet>
```

Once the XSLT is created, save it in a convenient place for testing in the next step. This example can be saved as **pa.xsl**.

When using a custom XSLT in an .audit, the first 3 three lines and the last 2 lines should be ignored. Those standard lines are added by the Nessus plugin **nbins** during processing. In this example, lines 5-8 are the ones of interest that will need to be used in the **AUDIT\_XML** or **AUDIT\_REPORTS** item.

The testing process in Step 5 can also be used while building the XSLT to validate assumptions and/or new techniques. This process is especially useful if you are new to XSLT or working on more complex transforms.

## Step 5: Verify the XSLT Transform Works

Verify your XSL Transform works with **xsltproc**. The general format for testing is:

```
/usr/bin/xsltproc {XSLT file} {Source XML}
```

Plugging in the sample file names from the steps above will return the following. This lets you know that the XSL Transform is correct and properly formatted, and that the data you expect is being returned.

```
[tater@pearl ~]# xsltproc pa.xsl top-applications.xml

+ insufficient-data
+ ping
+ snmp
+ dns
+ lpd
+ ntp
+ time
+ icmp
+ netbios-ns
+ radius
+ source-engine
+ stun
+ rip
+ tftp
```

```
+ echo
+ portmapper
+ teredo
+ slp
+ ssdp
+ dhcp
+ mssql-mon
+ pcanywhere
+ apple-airport
+ ike
+ citrix
+ xdmcp
+ l2tp
```

## Step 6: Copy the XSLT to the .audit

Once the XSL Transform works as intended, copy the XSLT lines of interest (lines 5-8 in this example) to the `.audit` check.

```
xsl_stmt: "<xsl:template match=\"result\">"
xsl_stmt: "<xsl:for-each select=\"entry\">"
xsl_stmt: "+ <xsl:value-of select=\"name\"/>"
xsl_stmt: "</xsl:for-each>"
```

Each line of the custom XSL transform must be placed into its own `xsl_stmt` element enclosed in double quotes. Since the `xsl_stmt` element uses double quotes to encapsulate the `<xsl>` statements, any double quotes used must be escaped. **Escaping the double quotes is important and not doing so risks errors in check execution.**

```
/usr/bin/xsltproc {XSLT file} {Source XML}
```

In the next step you can see several examples of properly escaped double quotes.

## Step 7: Final Audit

Once the first six steps are complete, you will have everything required to construct an audit:

```
<custom_item>
  type: AUDIT_REPORTS
  description: "Palo Alto Reports - Top Applications"
  request: "&reporttype=predefined&reportname=top-applications"
  xsl_stmt: "<xsl:template match=\"result\">"
  xsl_stmt: "<xsl:for-each select=\"entry\">"
  xsl_stmt: "+ <xsl:value-of select=\"name\"/>"
  xsl_stmt: "</xsl:for-each>"
</custom_item>
```