Tenable Nessus Compliance Checks Reference Guide

Last Updated: May 14, 2025

Table of Contents

Compliance Checks Reference

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.

For the PDF version of this guide, see the PDF.

> **Tip:** Nessus supports SCADA system auditing; however, this functionality is outside of the scope of this document. Please reference the Tenable SCADA information page 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.

Compliance Standards

There are many different types of government and financial compliance requirements. It is important to understand that these compliance requirements are minimal baselines that can be interpreted differently depending on the business goals of the organization. Compliance requirements must be mapped with the business goals to ensure that risks are appropriately identified and mitigated.

For example, a business may have a policy that requires all servers with customer personally identifiable information (PII) on them to have logging enabled and minimum password lengths of 10 characters. This policy can help in an organization's efforts to maintain compliance with any number of different regulations.

Common compliance regulations and guides include, but are not limited to:

- BASEL II
- Center for Internet Security Benchmarks (CIS)
- Control Objectives for Information and related Technology (COBIT)
- Defense Information Systems Agency (DISA) STIGs

- Federal Information Security Management Act (FISMA)
- Federal Desktop Core Configuration (FDCC)
- Gramm-Leach-Bliley Act (GLBA)
- Health Insurance Portability and Accountability Act (HIPAA)
- ISO 27002/17799 Security Standards
- Information Technology Information Library (ITIL)
- National Institute of Standards (NIST) configuration guidelines
- National Security Agency (NSA) configuration guidelines
- Payment Card Industry Data Security Standards (PCI DSS)
- Sarbanes-Oxley (SOX)
- Site Data Protection (SDP)
- United States Government Configuration Baseline (USGCB)
- Various State Laws (e.g., California's Security Breach Notification Act - SB 1386)

These compliance checks also address real-time monitoring such as performing intrusion detection and access control. For a more in depth look at how Tenable's configuration auditing, vulnerability management, data leakage, log analysis, and network monitoring solutions can assist with the mentioned compliance regulations, please refer to the Tenable whitepaper Tenable Cyber Exposure Study: Host Audit Data.

Configuration Audits, Data Leakage, and Compliance

What is an audit?

You can use Tenable Nessus log into Unix and Windows servers, Cisco devices, SCADA systems, IBM iSeries servers, and databases to determine if they have been configured in accordance to the local site security policy. Tenable Nessus can also search the entire hard drive of Windows and Unix systems for unauthorized content.

It is important for your organization to establish a site security policy before performing an audit to ensure assets are appropriately protected. A vulnerability assessment determines if your systems

are vulnerable to known exploits but does not determine, for example, if personnel records are being stored on a public server.

There is no absolute standard on security; it is a question of managing risk and this varies between organizations.

For example, consider the password requirements such as minimum/maximum password ages and account lockout policies. There may be good reasons to change passwords frequently or infrequently. There may also be good reasons to lock an account out if there have been more than five login failures, but if this is a mission-critical system, setting something higher might be more prudent or even disabling lockouts altogether.

These configuration settings have much to do with system management and security policy, but not specifically system vulnerabilities or missing patches. Tenable Nessus can perform compliance checks for Unix and Windows servers. Policies can be either simple or complex depending on the requirements of each individual compliance scan.

Audit vs. Vulnerability Scan

Tenable Nessus can perform vulnerability scans of network services as well as log into servers to discover any missing patches. However, a lack of vulnerabilities does not mean the servers are configured correctly or are “compliant” with a particular standard.

The advantage of using Tenable Nessus to perform vulnerability scans and compliance audits is that all of this data can be obtained at one time. Knowing how a server is configured, how it is patched and what vulnerabilities are present can help determine measures to mitigate risk.

At a higher level, if this information is aggregated for an entire network or asset class (as with Tenable Security Center), security and risk can be analyzed globally. This allows auditors and network managers to spot trends in non-compliant systems and adjust controls to fix these on a larger scale.

Audit Reports

When an audit is performed, Tenable Nessus attempts to determine if the host is compliant, non-compliant or if the results are inconclusive.

Compliance results in Nessus are logged as **Pass**, **Fail**, and **Warning**. The Tenable Security Center log results as **Info** for passed, **High** for failed, and **Medium** for inconclusive (for example, a permissions check for a file that is not found on the system).

Unlike a vulnerability check, which only reports if the vulnerability is present, a compliance check always reports something. This way, the data can be used as the basis of an audit report to show that a host passed or failed a specific test, or if it could not be properly tested.

Credentialed Scanning and Privileged Account Use

Tenable provides authenticated vulnerability and configuration assessments of systems to validate the presence of vulnerabilities, patches, and secure configurations. To obtain accurate results when assessing a system, you must grant Nessus or Tenable Security Center privileged authentication and access levels to access the end system.

Performing a vulnerability scan or audit with an account lacking sufficient privileges may result in incomplete results. For example, Nessus may not find certain files and commands may return erroneous or incomplete information or lack output altogether.

Tenable recommends configuring administrator or root-equivalent accounts to avoid erroneous or inaccurate system assessments. You can create accounts with customized privileges for scanning and assessment, but this approach is fragile and not recommended. The methods used by Tenable products to assess systems may change to adapt to new technologies or vulnerabilities; therefore, the required granular privileges may also change.

Consider the following when reviewing strategies for authenticated assessment of systems in your environment:

1. Implement compensating controls for privileged accounts to limit risk, such as:
    a. Log monitoring for when the account is in use outside of standard change control hours, with alerts for activities outside of normal windows.
    b. Perform frequent password rotation for privileged accounts more often than the “normal” internal standard.
    c. Enable accounts only when the time window for scans is active; disable accounts at other times.

d. On non-Windows systems, do not allow remote root logins. Configure your scans to utilize escalation such as su, sudo, pbrun, .k5login, or dzdo.

e. Use key authentication instead of password authentication.

2. Use Nessus Agents where available.

3. If you do not grant an exception with compensating controls, perform a scan with an account having lower privileges than what Tenable recommends and observe any missing results. Modify the account privileges so that all expected results are shown. Changes to the audit file or plugins may impact results later.

For further information on credentialed checks, refer to the Nessus User Guide.

Tips on String Matching

As a general rule, where possible, it is most accurate (along with being easier to write and troubleshoot) to confine the matching to a single line of the message. Single quotes and double quotes are interchangeable when surrounding audit fields, except in the following cases:

- In Windows compliance checks where special fields such as CRLF 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'
  ```

- 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"

  - 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-".*'
    ```

    - Escape out any embedded quotes with a backslash (double quotes only). For example:

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

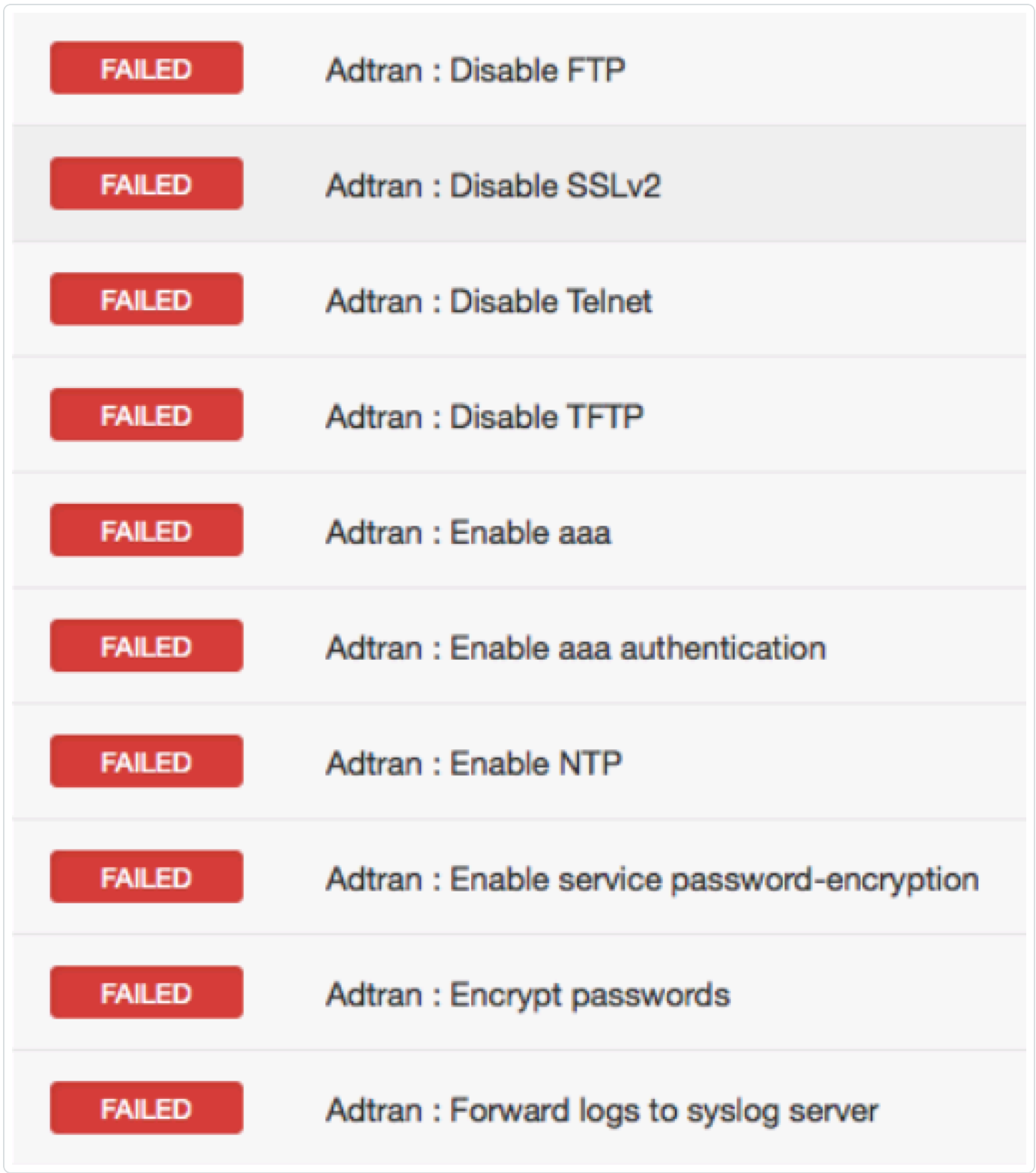
- Escaping a single character can be done so it matches the literal character rather than the

normal regex interpretation of any single character. For example:

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

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.



This section includes the following information:

- Adtran AOS Syntax

Adtran AOS 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>
```

Alcatel TiMOS Compliance Checks

Plugin ID: 102730

The Alcatel TiMOS audits include checks for various routing protocols, ICMP settings, DNS, ACLs, login settings, and much more. This plugin can be used against TiMOS and SR-OS devices.

Scan Requirements

Credentials

- SSH
- Escalation via `enable`

Permissions

An account that is able to retrieve the target configuration.

Configuration Gathering

This plugin gathers target information through the use of various commands, including:

- `show version`
- `admin display-config detail`

Offline auditing is supported by uploading related config files during the compliance policy scan setup process.

Checks

- check_type
- CONFIG_CHECK

check_type

All Alcatel compliance checks must be bracketed with the `check_type` encapsulation and the Alcatel designation. This is required to differentiate .audit files intended specifically for systems running an Alcatel system from other types of compliance audits.

Usage

```
\<check_type:"Alcatel">
 * audit contents
\</check_type>
```

CONFIG_CHECK

The CONFIG_CHECK check analyzes the configuration for regular expressions to identify if a configuration is set.

Usage

```
<custom_item>
 type                       : CONFIG_CHECK
 description                : ["description"]
 expect                     : ["lines to match against"]
 (optional) cmd             : ["command to run for additional content"]
 (optional) regex           : ["lines to filter"]
 (optional) context         : ["subcontext of config items to filter"]
 (optional) not_expect      : ["lines to match against"]
 (optional) required        : [YES|NO]
 (optional) match_all       : [YES|NO]
 (optional) match_case      : [YES|NO]
 (optional) min_occurrences : ["numerical value"]
 (optional) max_occurrences : ["numerical value"]
</custom_item>
```

Property	Description
type: CONFIG_CHECK	This setting is used to evaluate the positive response from the configuration file.
expect	The expected value being positively evaluated from the configuration file.
cmd	A command sent to the target device to gather additional information not displayed with the default configuration.
regex	Used for filtering specific configuration lines for `expect` evaluation. Useful

	for reducing output for review.
context	Used to return only specific indented contexts in the configuration, such as interfaces or line information.
not_expect	The expected value being negatively evaluated from the configuration file. Useful for checking that `telnet` is not found in any line.
required	A value of `NO` allows a check to pass if the item is not found. Defaults to `YES` if not specified.
match_all	Setting `match_all` to `YES` requires the expectation to match all lines of text, and not just a single line of text. If `match_all` is set to the default of `NO`, only one line must match for the check to pass.
match_case	Setting `match_case` to `YES` makes the comparison to be case sensitive. If `match_case` is set to the default of `NO`, the comparison is case insensitive.
min_ occurrences	Specify a number of minimum occurrences that must be met to obtain a passing result. Useful for matching against multiple NTP lines.
max_ occurrences	Specify a number of maximum occurrences that must be met to obtain a passing result. Useful for matching against a target having no more than one username line, or multiple SNMP strings being set.

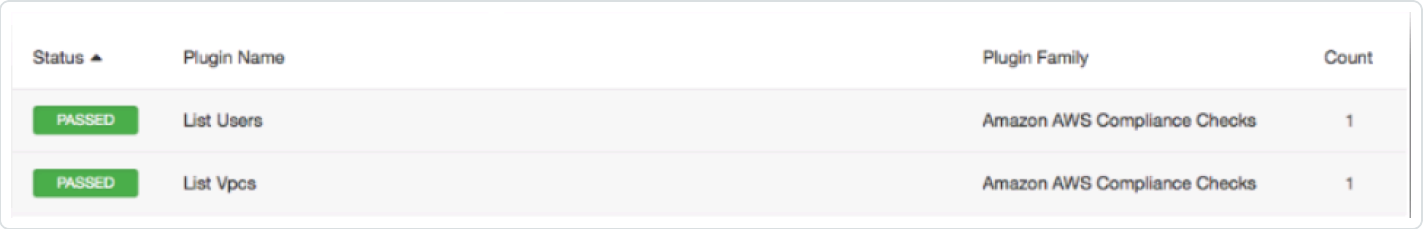
Example

```
<custom_item>
 description : "Login: ssh - v2 and later is enabled"
 cmd         : "show system security ssh"
 regex       : "SSH Protocol Version 2"
 expect      : "Enabled"
</custom_item>
<custom_item>
 description : "Login: Telnet is disabled (IPv4)"
 expect      : "no telnet-server"
</custom_item>
```

Amazon Web Services (AWS) Compliance File Reference

The Amazon Web Service (AWS) audit includes checks for running instances, network ACLs, firewall configurations, account attributes, user listing, and more. To audit your AWS account, you need a valid AWS access and secret key pair for an IAM user account that has been granted permissions within the AWS managed "ReadOnlyAccess" policy. For more information, see IAM Policy to Allow AWS Compliance Scanning.

Because AWS is a web-based service, the AWS audit does not have any designated targets, unlike a typical Nessus audit.



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

This section includes the following information:

- Audit File Syntax
- AWS Keywords
- AWS Debugging
- Known Good Auditing
- IAM Policy to Allow AWS Compliance Scanning

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**. With the exception of the description keyword, all other keywords are optional.

AWS Keywords

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

Keyword	Example Use and Supported Settings
type	The keyword type specifies the API we are tapping into to pull back the information (in this case IAM).
description	The "**description**" keyword provides the ability to add a brief description of the check that is being performed. It is strongly recommended that the **description** field be unique and that no distinct checks have the same **description** field. Tenable uses this field to automatically generate a unique plugin ID number based on the **description** field.
info	The "**info**" 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 lines within a single info field is supported, as well as additional info fields on separate lines to format the text as a paragraph. There is no preset limit to the number of info fields that can be used. **Note:** 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 regular line breaks after each line (as with the enter key), use "**\n**" to create a new line, or add additional "**info**" tags as needed. Example: `info: "Review the  list of interfaces"` `info: "Disable unused interfaces"`
aws_action	This keyword specifies the Amazon API action we are running against the AWS setup.

Keyword	Example Use and Supported Settings
`xsl_stmt`	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.
`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. 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.
not_expect	This keyword allows searching the configuration items that should not be in the configuration. 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.

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

AWS 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.

ArubaOS Compliance File Reference

The ArubaOS plugin scans and audits the configuration of the ArubaOS target. Currently supported and tested models are "S" and "CX" switches.

The plugin supports the following connection.

- Connecting directly to the ArubaOS target and pulling the full running configuration.

This section includes the following information:

- ArubaOS Scan Requirements
- ArubaOS Compliance Checks

ArubaOS Scan Requirements

The following describes scan requirements for using the ArubaOS plugin compliance checks.

Credentials

The plugin requires SSH credentials for online scanning. It does not require or support any escalation method.

Permissions

You must have sufficient permissions needed to run a `show running-config` command.

Some audits may have requirements to run additional commands.

Offline Scanning

The plugin supports offline scanning of ArubaOS configurations. No permissions or credentials are required for offline scanning, but the results produced will not be associated directly with any asset.

Instead, the results display the name of the configuration filename in the *Hosts* field.

To run an offline scan, upload the ArubaOS configuration as a .txt file to the scan or policy.

To upload a file for offline scanning:

1. Log in to an existing ArubaOS target (for example, via SSH).
2. In the command line interface (CLI), run the following command:

```
show running-config
```

3. Copy the output to a .txt file.
4. (Optional) To analyze multiple configurations, place each file in a .zip file.
5. In the scan or policy with the ArubaOS audit, upload the .txt or .zip file to *ArubaOS config file (s)*.
6. Save and launch the scan or policy.

ArubaOS Compliance Checks

Check Type

All ArubaOS compliance checks must be bracketed with the `check_type` encapsulation with the `ArubaOS` designation:

```
<check_type:"ArubaOS">
...
</check_type>
```

This is required to differentiate ArubaOS .audit files from those intended for other platforms.

Checks

The following sections describe the checks you can add to a single audit file.

- BANNER_CHECK
- CONFIG_CHECK and CONFIG_CHECK_NOT
- CMD_EXEC

BANNER_CHECK

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

Usage

```
<custom item>
type: BANNER_CHECK
description: ["description"]
item: ["config item"]
content: ["banner content"]
(optional) is_substring: [YES|NO]
</custom item>
```

content

The content is what the expected banner should be. New lines in the banner are automatically processed through functions that determine delimiters and context.

is_substring

An optional flag that supports the possibility of location specific information being placed in a banner. If set to `YES`, the expected banner can be a substring of the file content, and not require a full match.

Examples

```
<custom item>
      type : BANNER_CHECK
      description : "banner motd is configured"
      item : "banner motd"
      content : "** No unauthorized access is allowed **"
      This device is monitored and all activity is logged
      Proceed with caution!"
</custom item>
<custom item>
      type : BANNER_CHECK
      description : "banner exec is configured"
      item : "banner exec"
      content : "No unauthorized access is allowed\nBy logging in you \"agree\" to the
terms of usage.\All activity is monitored and logged."
```

```
</custom item>
```

CONFIG_CHECK and CONFIG_CHECK_NOT

The CONFIG_CHECK check analyzes the configuration for regular expressions to identify if a configuration is set.

The CONFIG_CHECK_NOT check gives the opposite result as CONFIG_CHECK, and analyzes the configuration to identify if a regular expression is *not* present, which indicates the configuration is not set.

Usage

```
<custom_item>
      type : CONFIG_CHECK
      description : ["description"]
      (optional) context : ["regular expression to create contexts"]
      (optional) regex : ["regular expression to reduce config options"]
      item : ["regular expression of text that needs to be found"]
      (optional) match_all : [YES|NO]
      (optional) match_case : [YES|NO]
      (optional) min_occurrences : ["numerical value"]
      (optional) max_occurrences : ["numerical value"]
</custom_item>
```

context

(Optional) The `context` is a regular expression that returns one or more subsets of the configuration. When the context matches a line, it returns that line and any other lines directly below it that are indented more than the initial matching line. Multiple contexts can be used to narrow down then searchable configuration.

Contexts are evaluated independently. If one `context` fails, the entire check evaluation fails.

Contexts are defined as code in the following format:

```
context-1
      line item 1
      line item 2
```

```
context-2
      line item 1
      line item 2
```

regex

(Optional) The `regex` is used to filter the full configurations, or each of the context configurations, to a smaller set of lines of text based on the regular expression. Multiple regex can be used to narrow down the searchable configuration, and they are applied in the order that they are listed in the check.

item

The evaluation is based on `item`.

- For CONFIG_CHECK, if the regular expression in the item matches a line of text, the check results as PASSED. If there are no matches, the check results as FAILED.
- For CONFIG_CHECK_NOT, if the regular expression in the item matches a line of text, the check results as FAILED. If there are no matches, the check results as PASSED.

To indicate if all lines need to match or that lines are case-sensitive, use the modifiers `match_all` or `match_case`.

match_all

(Optional) Set `match_all` to `YES` to require all lines of text to match the expectation, and not just a single line of text. If you set `match_all` to the default of `NO`, only one line must match for the check to pass.

match_case

(Optional) Set `match_case` to `YES` to make the comparison case-sensitive. Set `match_case` to the default of `NO` to make the comparison case-insensitive.

min_occurrences

Specifies the minimum number of occurrences of the configuration item required to pass the audit.

This is useful in cases where a minimum number of servers (NTP, DNS, etc.) should be present.

Example:

```
min_occurrences: "3"
```

max_occurrences

Specifies the maximum number of occurrences of the configuration item allowed to pass the audit.

This is useful in cases when checking items such as a single local account should exist (account of last resort).

Example:

`max_occurrences: "1"`

Example

```
<custom_item>
      type : CONFIG_CHECK
      description : "Ensure telnet is disabled"
      item : "no telnet-server"
</custom_item>
<custom_item>
      type : CONFIG_CHECK_NOT
      description : "Verify common SNMP strings are not used"
      regex : "snmp-server community"
      item : "(public|readonly|write)"
      match_case: NO
</custom_item>
```

CMD_EXEC

The CMD_EXEC check runs a command and analyze the output with regular expressions to identify if a command matches the expected output.

If CMD_EXEC is used in an offline scan, a warning states that the command is not able to run in offline mode.

Usage

```
<custom_item>
      type : CMD_EXEC
      description : ["description"]
      cmd : ["command to run"]
```

```
      (optional) regex : ["regular expression to reduce config options"]
      expect : ["regular expression that passes if found"]
      not_expect : ["regular expression that passes if not found"]
      (optional) match_all : [YES|NO]
      (optional) match_case : [YES|NO]
</custom_item>
```

Keywords

cmd

The `cmd` is the command that should be run on the target. Only `show` commands are supported.

regex

(Optional) (Optional) The `regex` is used to filter the full configurations, or each of the context configurations, to a smaller set of lines of text based on the regular expression. Multiple regex can be used to narrow down the searchable configuration, and they are applied in the order that they are listed in the check.

expect or not_expect

The evaluation is based on `expect` or `not_expect`. Use only one of these fields in a check.

- For `expect`, if the regular expression matches a line of text, the check results as PASSED. If there are no matches, the check results as FAILED.
- For `not_expect`, if the regular expression matches a line of text, the check results as FAILED. If there are no matches, the check results as PASSED.

To indicate if all lines need to match or that lines are case-sensitive, use the modifiers `match_all` or `match_case`.

match_all

(Optional) (Optional) Set `match_all` to `YES` to require all lines of text to match the expectation, and not just a single line of text. If you set `match_all` to the default of `NO`, only one line must match for the check to pass.

match_case

(Optional) (Optional) Set `match_case` to `YES` to make the comparison case-sensitive. Set `match_case` to the default of `NO` to make the comparison case-insensitive.

Example

```
<custom_item>
      type : CMD_EXEC
      description : "Ensure '3ds-cbc' is disabled for SSH access"
      cmd : "show running-config all"
      regex : "ssh server encryption"
      expect : "ssh server encryption 3des-cbc disable"
</custom_item>
```

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'

This section includes the following information:

- Brocade Fabric OS Syntax

Brocade Fabric OS 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_v700.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>
```

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.

This section includes the following information:

- Check Type: CONFIG_CHECK
- Check Point GAiA Keywords
- CONFIG_CHECK Examples
- Conditions
- Reporting

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.

Check Point GAiA Keywords

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

Keyword	Example Use and Supported Settings
`type`	"CHECK_CONFIG" determines if the specified config item exists in the GAiA "show configuration" output.
`description`	The **`"description"`** keyword provides the ability to add a brief description of the check that is being performed. It is strongly recommended that the **`description`** field be unique and that no distinct checks have the same description field. Tenable uses this field to automatically generate a unique plugin ID number based on the **`description`** field.

Keyword	Example Use and Supported Settings
	Example: `description: "1.0 Require strong Password Controls - 'min-password-length >= 8'"`
`info`	The **"info"** 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 **info** fields can be added on separate lines to format the text as a paragraph. There is no preset limit to the number of info fields that can be used. **Note:** 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: "Enable palindrome-check on passwords"`
`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 snmp .+"` The following meta-characters require special treatment: + \ * ( ) ^ Escape these characters out twice with two backslashes "\\" or enclose

Keyword	Example Use and Supported Settings
	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. 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: "set password-controls complexity"` `expect: "set password-controls complexity [1-4]"` In the above case, the **“expect”** tag ensures that the complexity is set to a value between 1 and 4.
`not_expect`	This keyword allows searching the configuration items that should not be in the configuration. 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. Example: `regex: "set password-controls password-expiration"` `not_expect: "set password-controls password-expiration`

Keyword	Example Use and Supported Settings
	never" In the above case, the **“not_expect”** tag ensures that the password-controls are not set to “never”.

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".

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.

This section includes the following information:

- Check Type
- Cisco IOS Keywords
- Command Line Examples
- Conditions

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.

Cisco IOS Keywords

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

Keyword	Example Use and Supported Settings
`type`	CONFIG_CHECK, CONFIG_CHECK_NOT and RANDOMNESS_CHECK "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

Keyword	Example Use and Supported Settings
	case, at least a digit and at least one special character). **Note:** The randomness parameters are currently not configurable.
`description`	This keyword provides the ability to add a brief description of the check that is being performed. It is strongly recommended that the **`description`** field be unique and no distinct checks have the same description field. Tenable uses this field to automatically generate a unique plugin ID number based on the description field. Example: `description: "Forbid Remote Startup Configuration"`
`feature_set`	The "**`feature_set`**" keyword, similar to the "**`system`**" 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. Example: `<item>` `type: CONFIG_CHECK` `description: "Version Check"` `info: "SSH Access Control Check."` `feature_set: "K8" context:"line .*"` `item: "access-class [0-9]+ in"` `</item>` The check above will only run the "item" check if the Feature Set version matches the specified regex: (K8) In the event of a Feature Set version check failure, an error similar to the one below is displayed: `"Version Check" : [SKIPPED]`

Keyword	Example Use and Supported Settings
	`Test defined for 12.[5-9] whereas we are running 12.4 (15)T10`
`info`	The "**`info`**" 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 **`info`** fields can be added on separate lines to format the text as a paragraph. There is no preset limit to the number of **`info`** fields that can be used. **Note:** 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: "Verify at least one local user exists and ensure"` `info: "all locally defined user passwords are protected"` `info: "by encryption."`
`item`	The "**`item`**" keyword specifies the configuration item within the output of the "show config" output to be audited. Example: `item: "transport input ssh"` 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.
`regex`	The "**`regex`**" keyword enables searching the configuration item setting to match for a particular regular expression. Example: `regex: "snmp-server community ([^ ]*) .*"`

Keyword	Example Use and Supported Settings
	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.
`min_ occurrences`	The “**`min_occurrences`**” keyword specifies the minimum number of occurrences of the configuration item required to pass the audit. Example: `min_occurrences: "3"`
`max_ occurrences`	The “**`max_occurrences`**” keyword specifies the maximum number of occurrences of the configuration item allowed to pass the audit. Example: `max_occurrences: "1"`
`required`	The “**`required`**” keyword is used to specify if the audited item is required to be present or not on the remote system. For example, if **`required`** 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 **`required`** was set to “YES”, the above check would fail. Example: `required: NO`
`context`	The “**`context`**” keyword is useful where more than one instance of a particular configuration item exists. For example, consider the following configuration: `line con 0` `no modem enable`

Keyword	Example Use and Supported Settings
	`` line aux 0 access-class 42 in exec-timeout 10 0 no exec line vty 0 4 exec-timeout 2 0 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
```

where <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.

This section includes the following information:

- Search for a Defined SNMP ACL
- Disable "finger" Service
- Randomness Check to Verify SNMP Community Strings and Access Control are Sufficiently Random
- Context Check to Verify SSH Access Control

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.

Disable "finger" Service

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
```

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.

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**”.

Database Configuration Audit Compliance File Reference

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

This section includes the following information:

- Database Configuration Check Type
- Database Configuration Keywords
- Database Configuration Command Line Examples
- Database Configuration Conditions

Database Configuration 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` field is set to "1".

Example:

```
<check_type: "Database" db_type:"SQLServer" version:"1">
```

Database Configuration Keywords

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

Keyword	Example Use and Supported Settings
`type`	`SQL_POLICY`
`description`	This keyword provides the ability to add a brief description of the check that is being performed. It is strongly recommended that the `description` field be unique and no distinct checks have the same description field. Tenable uses this field to automatically generate a unique plugin ID number based on the `description` field. Example: `description: "DBMS Password Complexity"`
`info`	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 `info` fields can be added on separate lines to format the text as a paragraph. There is no preset limit to the number of `info` fields that can be used. Example: `info: "Checking that \"password complexity\" requirements are enforced for systems using SQL Server authentication."`
`sql_request`	This keyword is used to determine the actual SQL request to be submitted to the database. Arrays of data may be requested and returned from a SQL request by using comma-delimited request/return values. Example: `sql_request: "select name from sys.sql_logins where type = 'S' and is_policy_checked <> '1'"`

Keyword	Example Use and Supported Settings
	Example: `sql_request: "select name, value_in_use from sys.configurations where name = 'clr enabled'"`
`sql_types`	This keyword has two available options: • `POLICY_INTEGER`: Numeric-based results. • `POLICY_VARCHAR`: Text-based results. Example 1: `sql_types: POLICY_VARCHAR` For multiple return items, configure `sql_types` in a comma-separated list to accept the data types of each SQL return result. The following example indicates that the first return value from the SQL query is text-based and the second return value is an integer. Example 2: `sql_types: POLICY_VARCHAR,POLICY_INTEGER`
`sql_expect`	A comma separated list of the values, or regular expression, to evaluate the results from the SQL query. The values for each of the columns must match the types that are defined in the `sql_types`. The number of `sql_expect` items must match the number of `sql_types`. Numbers do not need double quotes. For text values, surround the text in double quotes ("). If a returned text value can vary in what is returned, use the regular expression in the form of `regex:"<expression>"`. For cases where cases where no rows are returned, use `NO_ROWS_RETURNED`. This is more explicit than using `check_option`.

Keyword	Example Use and Supported Settings
	Example: `sql_expect: regex:"^.+(Failure|ALL)"` Example: `sql_expect: NULL` Example: `sql_expect: "clr enabled",0`
check_option	Options that are used to adjust how to handle special cases. The most notable option is how to handle what happens when no results are returned. • `CAN_BE_NULL`: Will pass if no data is returned from the query. • `CAN_NOT_BE_NULL`: (Default) Will not pass if no data is returned from the query.

Usage

```
<custom_item>
type: SQL_POLICY
description: ["description"]
sql_request: ["sql statement to run"]
sql_types: [POLICY_VARCHAR|POLICY_INTEGER][,....]
sql_expect: ["text"|number|regex:"expr"]
(optional) check_option: [CAN_BE_NULL|CAN_NOT_BE_NULL]
</custom_item>
```

Database Configuration 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 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.

Database Configuration 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"
```

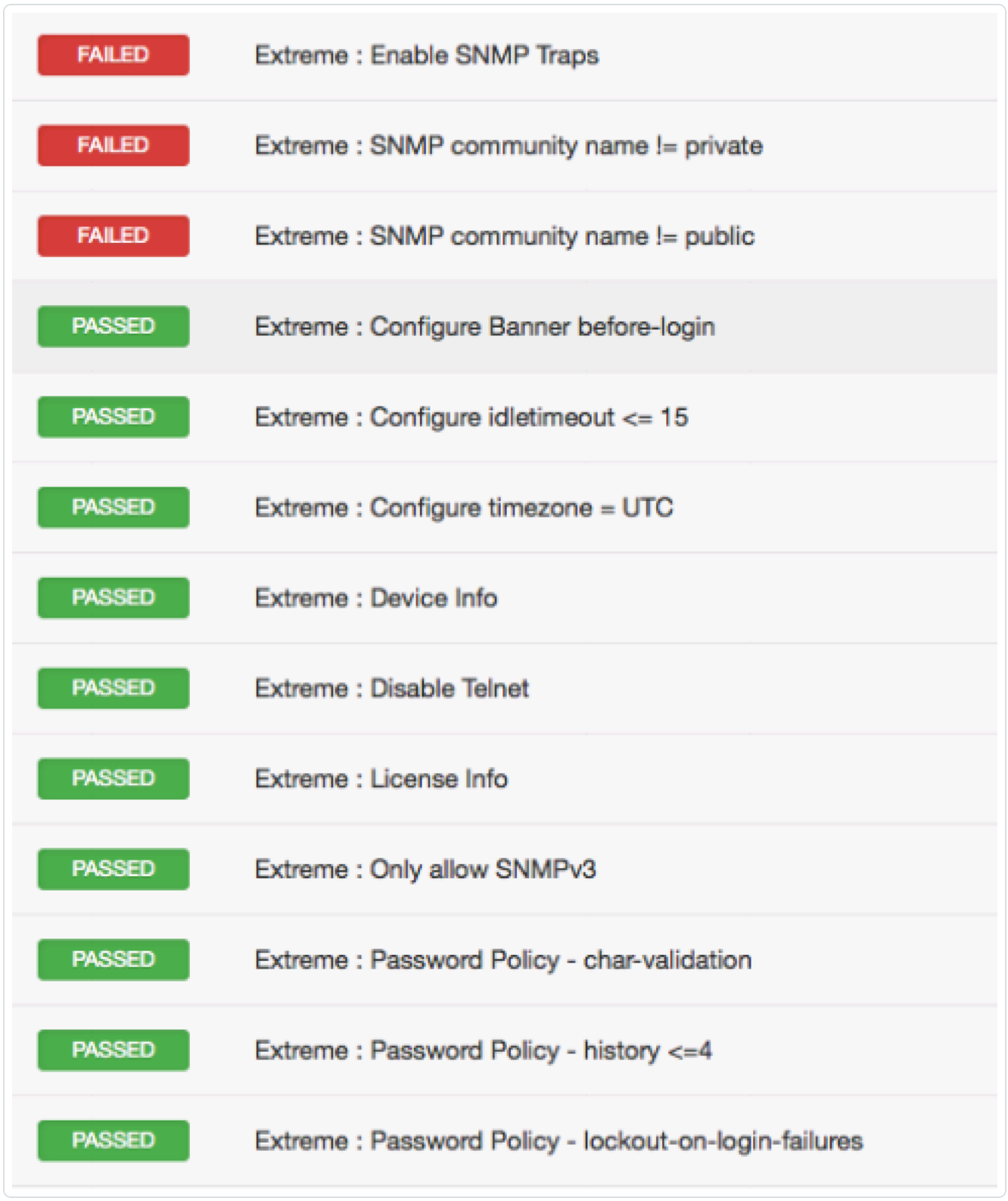
```
info: "Check system policy to confirm CLR requirements."
</report>
</else>
</if>
```

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**”.

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.



This section includes the following information:

- Extreme ExtremeXOS Syntax

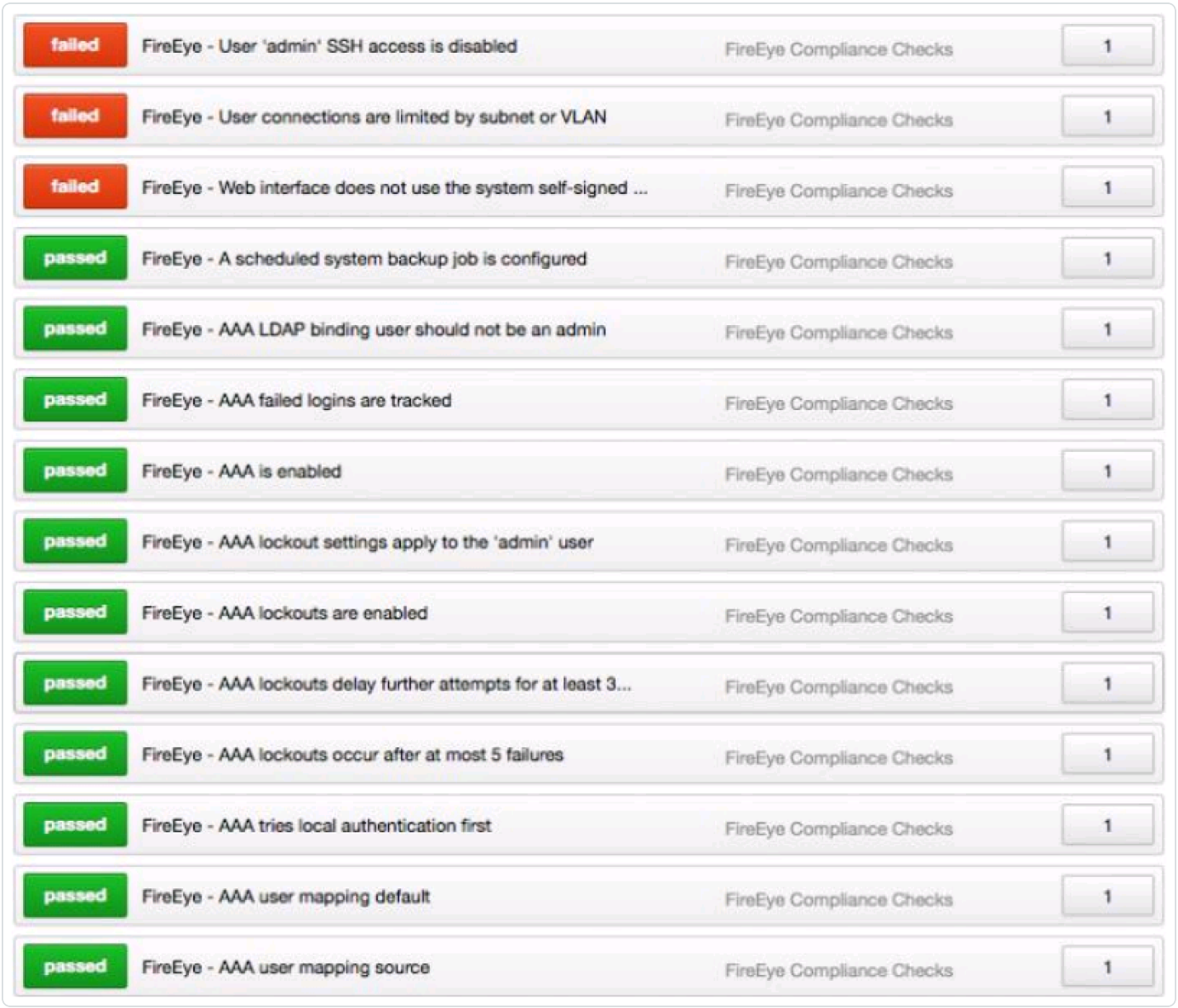
Extreme ExtremeXOS Syntax

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>
```

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.



This section includes the following information:

FireEye 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>
```

FireEye Keywords

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

Keyword	Example
`type`	`CONFIG_CHECK` `CONFIG_CHECK_NOT` `RANDOMNESS_CHECK`
`description`	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. Tenable uses this field to auto generate a plugin ID number based on the description field. Example: `description: " Verify login authentication"`
`info`	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.

Keyword	Example
	Example: `info: "Verifies login authentication configuration."`
`see_also`	This keyword allows users to include links that might provide helpful information about a check. Example: `see_also: "http://www.fireeye.com/support/"`
`reference`	This keyword allows including cross references for audit checks. Example: `reference: "PCI|2.2.3,SANS-CSC|1"`
`solution`	The keyword provides text to include solution text to fix a compliance failure. Example: `solution: "Modify the configuration to add missing line"`
`severity`	This keyword allows users to set the severity of the check. The severity can be set to HIGH, MEDIUM, or LOW. Example: `severity: MEDIUM`
`regex`	This keyword allows enumerating items that match a particular regex expression. If a check has "**`regex`**" keyword set, but no "**`expect`**" or "`not_expect`" keyword is set, then the check simply reports all items matching the regex. Example: `regex: "power-state.+"`
`expect`	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

Keyword	Example
	regex was provided, all lines will be checked but only one needs to be found. Example: `regex: "power"`
`not_expect`	Similar to **`expect`**, but if any matches are found, the check fails. If both **`expect`** and **`not_expect`** are omitted, all applicable lines will be reported as an info message.
`min_ occurrences`	Specifies the minimum number of occurrences of the configuration item required to pass the audit. Example: `min_occurrences: 3`
`max_ occurrences`	Specifies the maximum number of occurrences of the configuration item allowed to pass the audit.
`required`	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. Example: `required: YES`
`cmd`	This allows users to run a show command. Example: `cmd: "show version"` Only "`show`" commands are allowed. `<item>` `type: CONFIG_CHECK` `cmd: "show version"` `description: "Show Product version"` `regex: "Product model:"`

Keyword	Example
	expect: "1234" </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'

This section includes the following information:

- Fortinet FortiOS Syntax

Fortinet FortiOS 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 straight-forward. 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

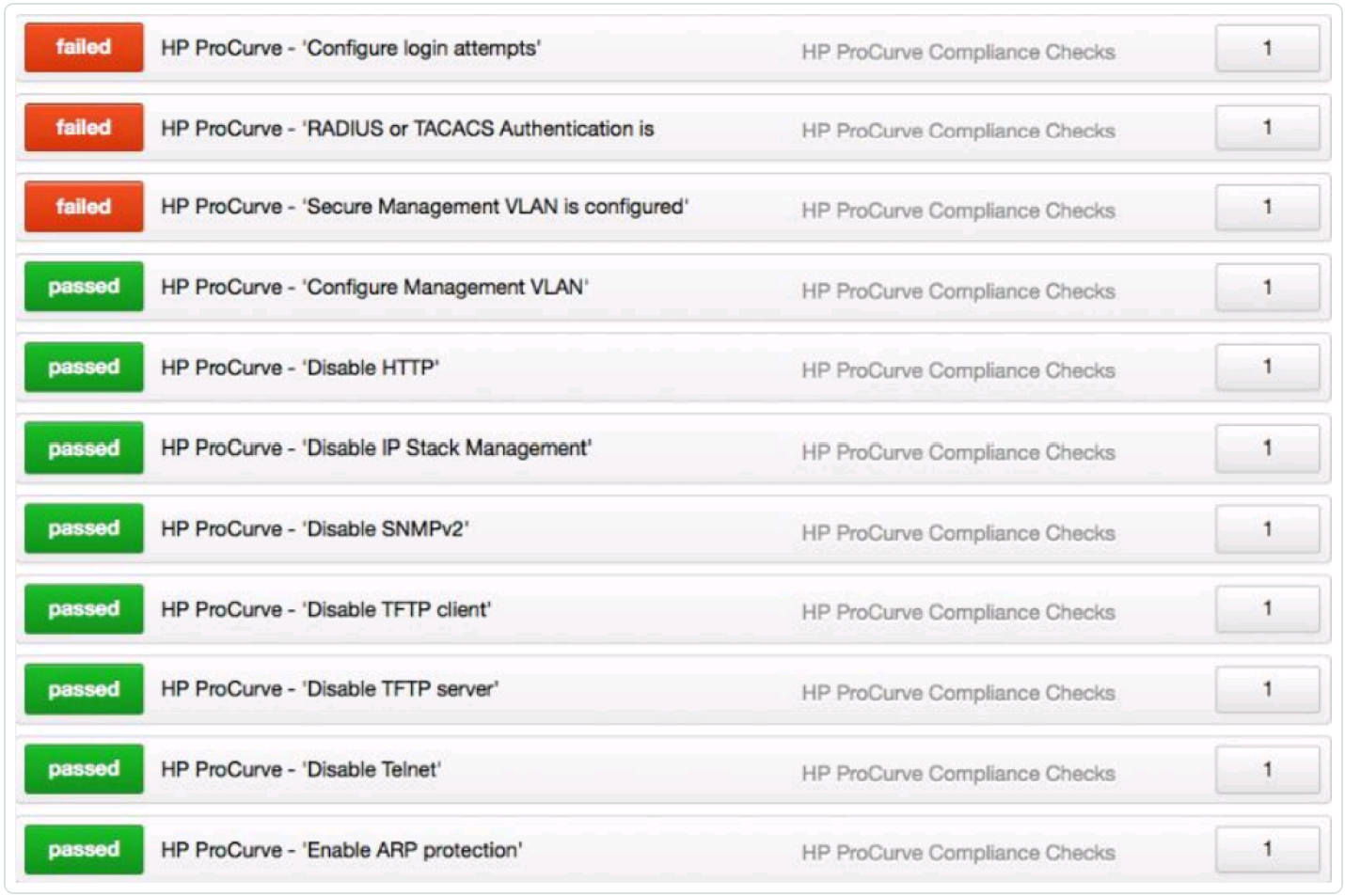
The plugin also supports the **cmd** keyword. This allows users to run any **get** or **show** command, and then include the resulting output in the report.

```
<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.

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 white paper 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.



This section includes the following information:

- HP ProCurve Check Types
- HP ProCurve Keywords

HP ProCurve 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>
```

HP ProCurve Keywords

Keyword	Example
`type`	`CONFIG_CHECK` `CONFIG_CHECK_NOT` `RANDOMNESS_CHECK`
`description`	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. Tenable uses this field to auto generate a plugin ID number based on the description field. Example: `description: " Verify login authentication"`
`info`	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. Example: `info: "Verifies login authentication configuration."`
`see_also`	This keyword allows users to include links that might provide helpful information about a check.

Keyword	Example
`reference`	This keyword allows including cross references for audit checks. Example: `reference: "PCI|2.2.3,SANS-CSC|1"`
`solution`	The keyword provides text to include solution text to fix a compliance failure. Example: `solution: "Modify the configuration to add missing line"`
`severity`	This keyword allows users to set the severity of the check. The severity can be set to HIGH, MEDIUM, or LOW. Example: `severity: MEDIUM`
`regex`	This keyword allows enumerating items that match a particular regex expression. If a check has "**`regex`**" keyword set, but no "**`expect`**" or "**`not_expect`**" keyword is set, then the check simply reports all items matching the regex. Example: `regex: "power-state.+"`
`item`	This keyword allows searching within the lines found by regex. If no regex was provided, all lines will be checked. Example: `regex: "power"`
`context`	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. Example:

Keyword	Example
	`context: "line .*"` The following is a sample config item, that could be audited by leveraging context: `vlan 1` `name "DEFAULT_VLAN"` `untagged 2-24` `ip address dhcp-bootp` `no untagged 1` `exit` `<item>` `type: CONFIG_CHECK` `description: "HP ProCurve - 'dhcp-bootp'"` `context: "vlan 1"` `item: "ip address dhcp-bootp"` `</item>` The check above will ensure "`ip address dhcp-bootp`" is set for context "`vlan 1`".
min_occurrences	This keyword allows setting a minimum number of occurrences of the check. Example: `min_occurrences: 3`
max_ occurrences	Like `min_occurrences`, but a maximum value instead of a minimum.
required	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. Example: `required: YES`

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

This section includes the following information:

Huawei VRP Syntax

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

```
<custom_item>
description: "Huawei: Set super password"
info: "Set super password for management 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>
```

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.

This section includes the following information:

- Required User Privileges
- Check Type
- Keywords
- Custom Items
- Conditions

Required User Privileges

To perform a successful compliance scan against an iSeries system, authenticated users must have privileges as defined below:

- A user with (*ALLOBJ) or audit (*AUDIT) authority can audit all system values. Such a user typically belongs to class (*SECOFR).
- 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
`type`	`AUDIT_SYSTEMVAL` `SHOW_SYSTEMVAL`
`systemvalue`	This keyword is used to specify a specific value to be checked within the IBM iSeries system. Example: `systemvalue: "QALWUSRDMN"`
`description`	This keyword provides the ability to add a brief description of the check that is being performed. It is strongly recommended that the **`description`** field be unique and no distinct checks have the same description field. Tenable uses this field to automatically generate a unique plugin ID number based on the description field. Example: `description: "Allow User Domain Objects (QALWUSRDMN) - '*all'"`
`value_type`	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"`
`value_data`	This keyword defines that data value that is expected for a system value. Example: `value_type: "^([6-9]|[1-9][0-9]+)$"`
`check_type`	This keyword defines the type of check being used against a data value.

Keyword	Example Use and Supported Settings
	Examples: `check_type: "CHECK_EQUAL"` `check_type: "CHECK_NOT_EQUAL"` `check_type: "CHECK_GREATER_THAN"` `check_type: "CHECK_GREATER_THAN_OR_EQUAL"` `check_type: "CHECK_LESS_THAN"` `check_type: "CHECK_LESS_THAN_OR_EQUAL"` `check_type: "CHECK_REGEX"` `<custom_item>` `type: AUDIT_SYSTEMVAL` `systemvalue: "QUSEADPAUT"` `description: "Use Adopted Authority (QUSEADPAUT) - '!= *none'"` `value_type: POLICY_TEXT` `value_data: "*none"` `check_type: CHECK_NOT_EQUAL` `</custom_item>`
`info`	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 **`info`** fields can be added on separate lines to format the text as a paragraph. There is no preset limit to the number of **`info`** fields that can be used.

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.

Tip: 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/iseries/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/iseries/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/iseries/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**.

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.

This section includes the following information:

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 "$1$hSGSlnwfdsdfdfsdfsdf43534"
```

Juniper CONFIG_CHECK Keywords

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

Keyword	Example Use and Supported Settings
`type`	CHECK_CONFIG and SHOW_CHECK_CONFIG “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.
`description`	This keyword provides the ability to add a brief description of the check that is being performed. It is strongly recommended that the **`description`** field be unique and no distinct checks have the same description field. Tenable uses this field to automatically generate a unique plugin ID number based on the description field. Example: `description: " 3.1 Disable Unused Interfaces"`
`info`	The “**`info`**” 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 **`info`** fields can be added on separate lines to format the text as a paragraph. There is no preset limit to the number of **`info`** fields that can be used. **Note:** 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 list of interfaces"` `info: "Disable unused interfaces"`
`severity`	The “`severity`” keyword specifies the severity of the check being

Keyword	Example Use and Supported Settings
	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:

Keyword	Example Use and Supported Settings
	`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. Example: `not_expect: "syslog host 1.1.1.1"` 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. Example: `regex: "syslog host [0-9\.]+"` `not_expect: "syslog host 1.1.1.1"` In the above case, the "**`not_expect`**" tag ensures that the syslog host is not set to 1.1.1.1.
`number_of_ lines`	This keyword allows testing compliance of an audit check based on the number of matching lines returned by the config. `<custom_item>` `type: CONFIG_CHECK` `description: "Syslog"` `regex: "syslog host [0-9\.]+"` `number_of_lines: "^1$"` `</custom_item>` In the above case the check will pass as long as only one line is returned that matches the "**`regex`**".

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.

Juniper SHOW_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
hierarchy	This keyword allows users to navigate to a specific hierarchy in the Junos configuration. Example: `hierarchy: "interfaces"` Internally the hierarchy keyword gets appended to the “show configuration”

Keyword	Example Use and Supported Settings
	command in a SHOW_CONFIG_CHECK. For example: `<custom_item>` `type: SHOW_CONFIG_CHECK` `description: "3.6 Forbid Multiple Loopback Addresses"` `hierarchy: "interfaces"` `</custom_item>` The check above is the equivalent of running: `show configuration interfaces`
property	This keyword allows users to audit a specific **"property"** on the Junos device. By default the SHOW_CONFIG_CHECK audits the "show configuration" command followed by one or more keywords such as **match**, **except**, and **find**. In the case where **"property"** keyword is set, it audits the specific property. Example: `property: "ospf"` `<custom_item>` `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"` `</custom_item>` The check above is the equivalent of running: `show ospf interface detail`

Keyword	Example Use and Supported Settings
	Note that the above example did not run “show configuration”, as was the case in other examples.
`find`	This keyword finds the appropriate config hierarchy in a SHOW_CONFIG_CHECK **`.audit`** check. `find: "chap"` The find keyword gets appended to the “show configuration” request. `<custom_item>` `type: SHOW_CONFIG_CHECK` `description: "3.8.2 Require CHAP Authentication if Incoming Map is Used"` `hierarchy: "interfaces"` `find: "chap"` `match: "access-profile"` `</custom_item>` The check above is the equivalent of running: `show configuration interfaces | find "chap" | match "access-profile"`
`match`	This keyword looks for matching lines in a SHOW_CONFIG_CHECK .audit check. `match: "multihop"` The **`match`** keyword gets appended to the “show configuration” request. `<custom_item>` `type: SHOW_CONFIG_CHECK` `description: "3.6 Forbid Multiple Loopback Addresses"` `hierarchy: "interfaces"`

Keyword	Example Use and Supported Settings
	`match: "lo[0-9]"` `</custom_item>` The check above is the equivalent of running: `show configuration interfaces | match "lo[0-9]"`
except	This keyword excludes certain lines from the config in a SHOW_CONFIG_CHECK .audit check. `except: "multihop"` The **except** keyword gets appended to the “show configuration” request. `<custom_item>` `type: SHOW_CONFIG_CHECK` `description: "6.8.1 Require External Time Sources"` `hierarchy: "system ntp"` `match: "server"` `except: "boot-server"` `</custom_item>` The check above is the equivalent of running: `show configuration system ntp | match "server" | except "boot-server"`
expect	This keyword allows auditing the config item matched by the **“regex”** tag or if the **“regex”** tag is not used it looks for the **“expect”** string in the entire config. 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. `regex: "syslog host [0-9\.]+"`

Keyword	Example Use and Supported Settings
	`expect: "syslog host 1.2.4.5"` In the above case, the **"expect"** tag ensures that the complexity is set to a value between 1 and 4. `expect: "syslog host"` In the case above, the **"expect"** tag ensures that the complexity is set to 4.
not_expect	This keyword allows searching the configuration items that should not be in the configuration. 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. `regex: "syslog host [0-9\.]+"` `not_expect: "syslog host 1.2.3.4"` `not_expect: "syslog host"`
number_of_lines	This keyword allows testing for compliance of a .audit check based on the number of matching lines returned by the config. `<custom_item>` `type: CONFIG_CHECK` `description: "Syslog"` `regex: "syslog host [0-9\.]+"` `number_of_lines: "^1$"` `</custom_item>` In the above case the check will pass as long as only one line is returned that matches the **"regex"**.

SHOW_CONFIG_CHECK Examples

The following are examples of using SHOW_CONFIG_CHECK against a Juniper device:

```
<custom_item>
type: SHOW_CONFIG_CHECK
description: "6.1.2 Require Accounting of Logins & Configuration Changes"
hierarchy: "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>
```

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

Microsoft Azure Audit Compliance Reference

Azure refers to a series of Microsoft cloud services including virtual machine hosting, data storage, and hosted versions of IIS, MS SQL, and Active Directory. The Active Directory service is also used for Windows InTune and Office 365.

The Azure plugin utilizes the Azure REST API in order to obtain configuration information for your cloud environment. The REST API accepts and returns JSON.

The Microsoft Azure plugin provides debug information when the **Plugin Debugging** scan policy preference is set. The debug log is attached to scan results.

The plugin supports evaluation of output by `regex`, `expect`, `not_expect`, `known_good`, and `json_transform` keywords.

This section includes the following information:

- Scan Requirements
- Microsoft Azure Keywords
- Request Types
- Microsoft Azure Syntax

Scan Requirements

To run a scan that audits Azure, you must set up your Azure environment and configure a scan in Tenable Vulnerability Management or Tenable Nessus using the appropriate credentials.

Azure Environment

Configure the Azure environment as described in Configure Microsoft Azure for Auditing in the *Tenable for Microsoft Azure Guide*

Scan Configuration

Configure a scan in Tenable Vulnerability Management, as described in Audit Microsoft Azure in Tenable Vulnerability Management in the *Tenable for Microsoft Azure Guide*.

Configure a scan in Tenable Nessus, as described in Audit Microsoft Azure in Nessus in the *Tenable for Microsoft Azure Guide*.

The plugin requires one of two supported Microsoft Azure credential sets.

Key:

Option	Description	Required
Tenant ID	The Tenant ID or Directory ID for your Azure environment.	Yes
Application ID	The application ID (also known as client ID) for your registered application.	Yes
Client Secret	The secret key for your registered application.	Yes
Subscription IDs	List of subscription IDs to scan, separated by a comma. If this field is blank, all subscriptions are audited.	No

Password:

Option	Description	Required
Username	The username required to log in to Microsoft Azure.	Yes
Password	The password associated with the username.	Yes
Client ID	The application ID (also known as client ID) for your registered application.	Yes
Subscription IDs	List of subscription IDs to scan, separated by a comma. If this field is blank, all subscriptions are audited.	No

Microsoft Azure Syntax

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

Example 1

```
<custom_item>
description     : "Virtual Machines List"
```

```
info            : "A list of all virtual machines"
request         : "getresourcesubs"
json_transform  : '.[] | .subscriptionId as $subID | .resourceGroups[].virtualMachines
[] |
"Subscription: " + $subID + " - Virtual Machine: " +
([.properties.instanceView.fullyQualifiedDomainName] | join (", "))'
</custom_item>
```

Example 2

```
<custom_item>
description     : "Stopped Virtual Machines List"
info            : "A list of all virtual machines that are stopped"
request         : "getresourcesubs"
json_transform  : '.[] | .subscriptionId as $subID | .resourceGroups[].virtualMachines
[].properties.instanceView | select (.powerState == "Stopped") |
"Subscription: " + $subID + " - Virtual Machine: " +
([.properties.instanceView.fullyQualifiedDomainName] | join (", "))'
regex           : ".+"
expect          : "Subscription:.+"
</custom_item>
```

Microsoft Azure Keywords

The following keywords are supported in Microsoft Azure audits:

Keyword	Description
`json_transform`	`json_transform` uses JQ to transform the aggregate JSON file from Azure into a format that is easier to understand and evaluate. For an example, see JQ Example.
`subscriptions`	When combined with the login credentials in the scan wizard, this keyword displays a comma-separated list of subscription IDs to be scanned. By default, all accessible subscriptions will be scanned.
`request`	This keyword specifies the plugin should return a data set.

Keyword	Description
`regex`	The regex is used to filter the JQ outputs to a smaller set of lines of text based on the regular expression. It is an optional transformation.
`expect` and `not_expect`	The evaluation is based on `expect` or `not_expect`. Use only one of these fields in a check. For `expect`, if the regular expression matches a line of text, the check results as PASSED. If there are no matches, the check results as FAILED. For `not_expect`, if the regular expression matches a line of text, the check results as FAILED. If there are no matches, the check results as PASSED.
`match_all`	Setting `match_all` to `YES` requires the item to match all lines of text, and not just a single line of text. If `match_all` is set to the default `NO`, only one line must match for the check to pass.

The Azure plugin utilizes the Azure REST API in order to obtain configuration information for your cloud environment. At the Tenable .audit and check level, action types are used in the `request` field. These action types correlate to documented API endpoints with some modifications. If there are prerequisites for a given API call, for example, the subscription ID or resource group name, that information is queried for and prepopulated into an aggregate JSON document before attaching the specified action type's information. This aggregate JSON document is then filtered using JQ in order to format the configuration data for evaluation and review.

Note: When writing your own checks for Azure, you can list the aggregate JSON document by using a request type with no `json_transform, regex` or `expect fields`. For more information, see Request Types.

Aggregate JSON Example

The following is an example of the aggregate JSON document with subscription IDs as a prerequisite:

```
[ { "id" : "/subscriptions/12345", "subscriptionId" : "12345", "displayName" :
"Microsoft Azure Enterprise", "state" : "Enabled", "subscriptionPolicies" : {
"locationPlacementId" : "Public_2014-09-01", "quotaId" : "EnterpriseAgreement_2014-09-
01", "spendingLimit" : "Off" }, "value" : [] }, { "id" : "/subscriptions/123456",-9
"subscriptionId" : "123456", "displayName" : "Microsoft Azure Enterprise", "state" :
"Enabled", "subscriptionPolicies" : { "locationPlacementId" : "Public_2014-09-01",
"quotaId" : "EnterpriseAgreement_2014-09-01", "spendingLimit" : "Off" }, "value" : []
}, { "id" : "/subscriptions/1234567", "subscriptionId" : "1234567", "displayName" :
"Microsoft Azure Enterprise", "state" : "Enabled", "subscriptionPolicies" : {
"locationPlacementId" : "Public_2014-09-01", "quotaId" : "EnterpriseAgreement_2014-09-
01", "spendingLimit" : "Off" }, "value" : [ { "id" :
"/subscriptions/1234567providers/microsoft.insights/logprofiles/default", "type" :
null, "name" : "default", "location" : null, "kind" : null, "tags" : null, "properties"
: { "storageAccountId" :
"/subscriptions/1234567/resourceGroups/testservice1/providers/Microsoft.Storage/stor-
ageAccounts/testservice1diag830", "serviceBusRuleId" : null, "locations" : [ "eastus",
"eastus2", "global" ], "categories" : [ "Write", "Delete", "Action" ],
"retentionPolicy" : { "enabled" : true, "days" : 90 } }, "identity" : null } ] } ]
```

JQ Example

The following is an example of how an aggregate JSON document gets transformed into JQ:

```
JQ example: .[]| if ((.value | length) != 0) then "Sub ID: (.subscriptionId) has a Log
Profile" else "Sub ID: (.subscriptionId) does not have a Log Profiles" end
Plugin Output example: Remote value:
Sub ID: 12345 does not have a Log Profile Sub ID: 2123456 does not have a Log Profile
Sub ID: 1234567 has a Log Profile
Policy value:
request: 'listLogProfiles'
```

Example

The following is an example check that uses the previously listed JSON document and JQ:

```
<custom_item>
description : "Ensure that a Log Profile exists"
```

```
request : "listLogProfiles"
json_transform:'.[]| if ((.value | length) != 0) then "Sub ID: (.subscriptionId) has a
Log Profile" else "Sub ID: (.subscriptionId) does not have a Log Profile" end'
regex: "Sub ID:" not_expect:'does not have a Log Profile' </custom_item>
```

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'

Note: MongoDB is a NoSQL database, which means it does not use the SQL query language for accessing the data.

This section includes the following information:

MongoDB 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>
```

MongoDB Keywords

Keyword	Example Use and Supported Settings
`description`	This keyword provides the ability to add a brief description of the check that is being performed. It is strongly recommended that the **`description`** field be unique and no distinct checks have the same description field. Tenable uses this field to automatically generate a unique plugin ID number based on the description field. Example: `description: "Require Authentication – DB users –'User authenticated by MongoDB'"`
`collection`	The name of the MongoDB that the plugin connects to get information. Example: info: "admin.system.users."
`query`	The MongoDB query. Example: `query: '{"credentials.MONGODB-CR": {"$exists": 1}}'"`
`fieldsSelector`	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.

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.

This section includes the following information:

Required User Privileges

To perform a successful compliance scan against a NetApp Data ONTAP system, authenticated users must have `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.

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
`type`	"CHECK_CONFIG" determines if the specified config item exists in the NetApp Data ONTAP "show configuration" output.
`description`	This keyword provides the ability to add a brief description of the check that is being performed. It is strongly recommended that the **`description`** field be unique and no distinct checks have the same description field. Tenable uses this field to automatically generate a unique plugin ID number based on the description field. Example: `description: "1.0 Require strong Password Controls - 'min-password-length >= 8'"`
`info`	The **`info`** 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 **`info`** fields can be added on separate lines to format the text as a paragraph. There is no preset limit to the number of **`info`** fields that can be used. **Note:** 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: "Enable palindrome-check on passwords"`
`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.

Keyword	Example Use and Supported Settings
`regex`	The **`regex`** keyword enables searching the configuration item setting to match for a particular regular expression. Example: `regex: "set snmp .+"` 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. 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: "set password-controls complexity"` `expect: "set password-controls complexity [1-4]"` In the above case, the **`expect`** tag ensures that the complexity is set to a value between 1 and 4.
`not_expect`	This keyword allows searching the configuration items that should not be in the configuration.

Keyword	Example Use and Supported Settings
	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. Example: `regex: "set password-controls password-expiration"` `not_expect: "set password-controls password-expiration never"` In the above case, the **not_expect** tag ensures that the password-controls are not set to “never”.

Example

The following is an example 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"
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".

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.

This section includes the following information:

- OpenStack Syntax
- OpenStack Keywords

OpenStack 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>
```

OpenStack Keywords

Keyword	Example
`description`	This keyword provides the ability to add a brief description of the check that is being performed. It is strongly recommended that the **`description`** field be unique and no distinct checks have the same description field. Tenable uses this field to automatically generate a unique plugin ID number based on the description field.
`info`	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 should be enclosed in double quotes.

Keyword	Example
`see_also`	This keyword allows users to include links that might provide helpful information about a check, e.g., “http://docs.openstack.org/”.
`request`	This keyword describes the type of REST API request for OpenStack.
`regex`	This keyword allows searching items that match a particular regex expression.
`expect`	This keyword provides matching text from the query output.
`service`	This keyword indicates the service (compute, identity, network) which will be queried by the plugin.
`json_transform`	The keyword provides the json_transform that will be performed on the output of the check.

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. 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, 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.

This section includes the following information:

AUDIT_XML

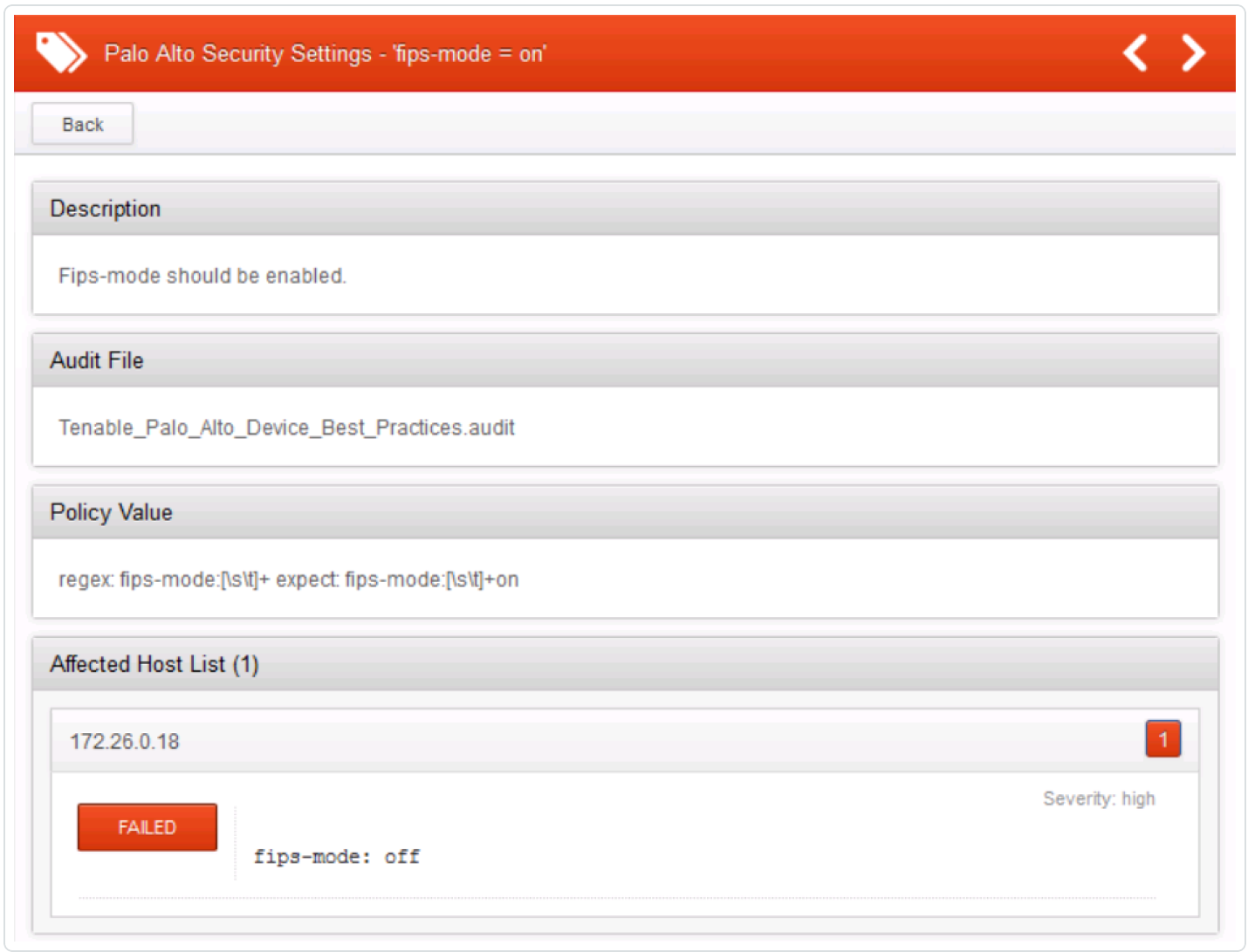
The following is an example of a Palo Alto AUDIT_XML check:

```
<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:

- 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.
- 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.
- 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.
- 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.

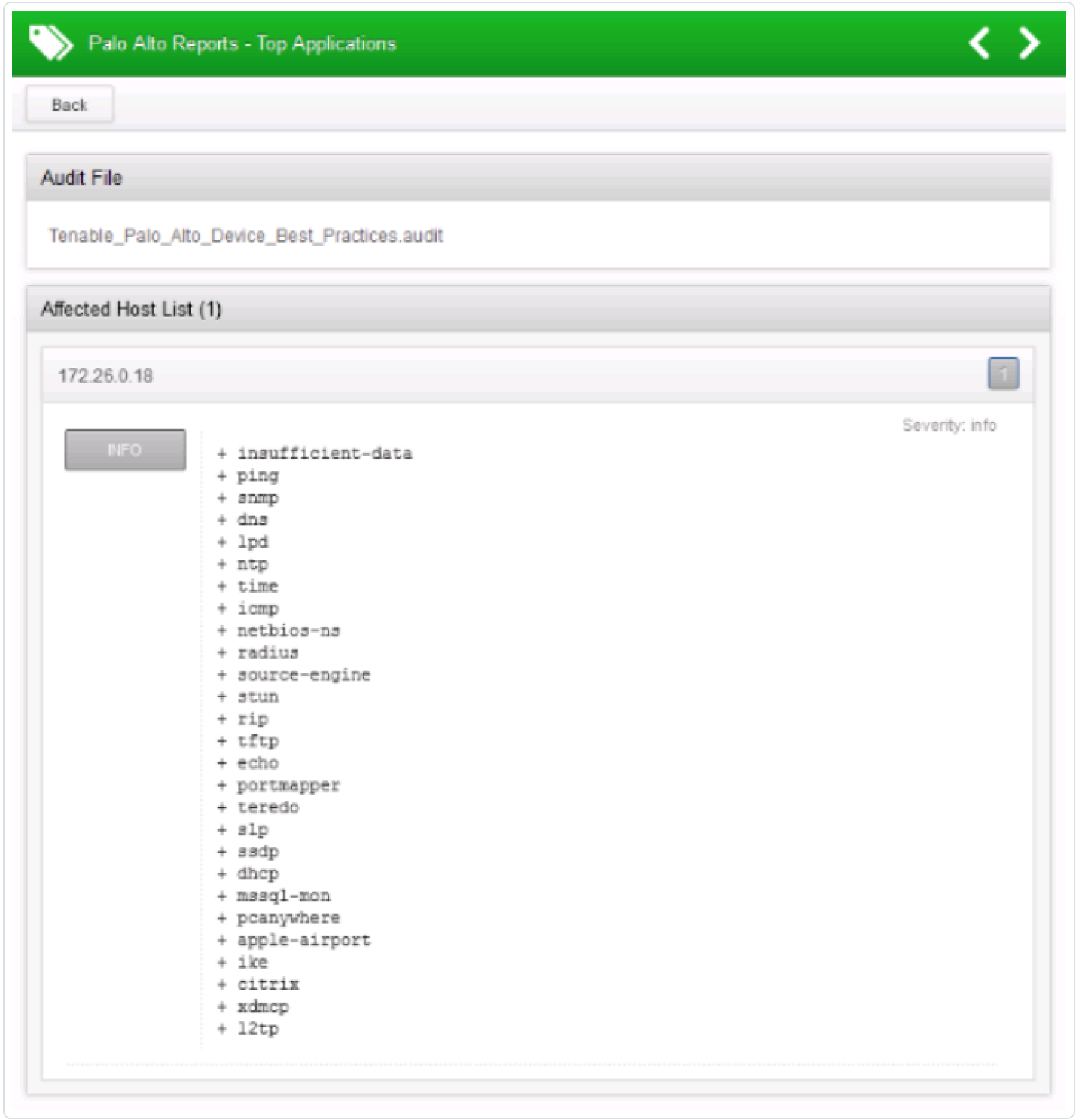
This feature has two benefits. First, users do not have to traverse different interfaces to get the same data. Second, this gives us the ability to audit the report. For example, if you do not want Facebook to be an application used within the network, then administrators can generate a failed report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

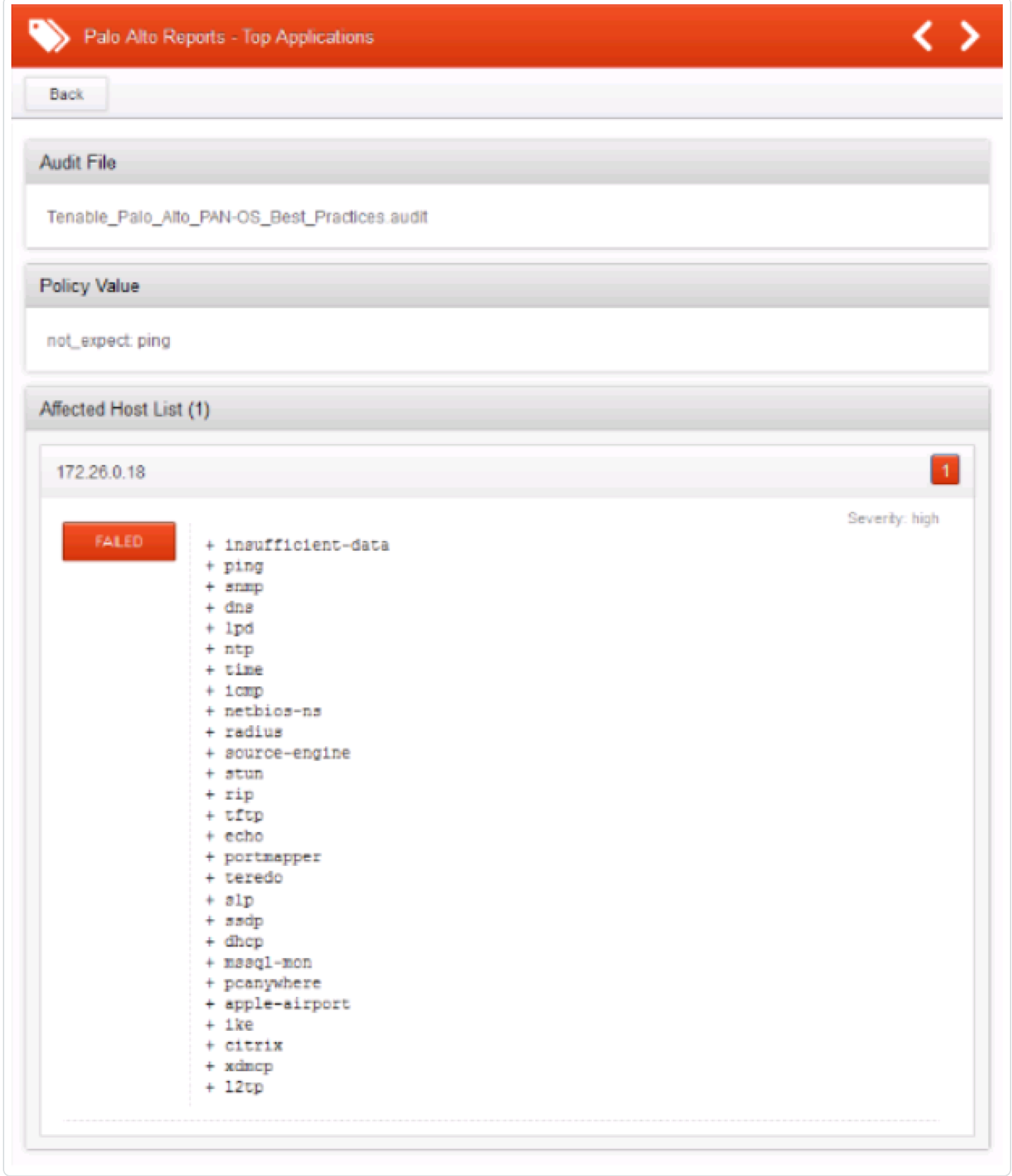
This report can be modified to use a **`not_expect`** keyword:

```
<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:



Palo Alto Firewall Keywords

The following keywords are supported in Palo Alto audits:

Keyword	Description
`type`	This must always be set to AUDIT_XML or AUDIT_REPORTS.
`description`	This keyword provides the ability to add a brief description of the check that is being performed. It is strongly recommended that the **`description`** field be unique and no distinct checks have the same description field. Tenable uses this field to automatically generate a unique plugin ID number based on the description field.
`info`	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 should be enclosed in double-quotes.
`api_request_type`	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 op is exposed.
`request`	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 AUDIT_REPORTS 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 `type=report`. For example: `/api/?type=report&reporttype=predefined&reportname=hruser-top-url-categories`
`regex`	This keyword allows searching items that match a particular regex expression. If a check has **`regex`** keyword set, but no **`expect`** or **`not_expect`** keyword is set, then the check simply reports all lines matching the regex.

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

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 data center 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.

This section includes the following information:

Red Hat Enterprise Virtualization 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>
```

Red Hat Enterprise Virtualization 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 sub-directory called **/compliance_debug**. On Red Hat, the full path would be **/opt/nessus/var/nessus/tmp/compliance_debug/**.

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

This section includes the following information:

- SalesForce Setup Requirements
- SalesForce Syntax

SalesForce Setup Requirements

One of these two methods are required to allow Tenable Nessus access:

- Add the scanner IP to the Trusted IP Ranges in Salesforce.
- Use a security token.

Adding a trusted IP range

- In Salesforce, go to Setup > Security Controls > Network Access.
- 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.
- When you enter the credentials in Salesforce plugin preferences in Tenable Nessus:
    - Enter the username.
    - Enter the user password.

Using a security token

- 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.
- When you enter the credentials in Salesforce plugin preferences in Tenable Nessus:
    - Enter the username
    - Append the security token to the user password (e.g., If the security password is"MyPassword" and the security token is "MyToken", enter "MyPasswordMyToken")

User Permissions

The login user must have a profile set with the following permissions enabled:

- **API Enabled**

  Salesforce location: Profiles > *Profile Name* > Administrative Permissions > API Enabled
- **Modify All Permissions**

  Salesforce location: Profiles > *Profile Name* > Administrative Permissions > Modify All Data
- **Modify Metadata**

Salesforce location: Profiles > *Profile Name* > Administrative Permissions > Modify Metadata

- **View All Users**

  Salesforce location: Profiles > *Profile Name* > Administrative Permissions > View All Users

- **View Roles and Role Hierarchy**

  Salesforce location: Profiles > *Profile Name* > Administrative Permissions > View Roles and Role Hierarchy

- **View Setup and Configuration**

  Salesforce location: Profiles > *Profile Name* > Administrative Permissions > View Setup and Configuration

SalesForce 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>
```

Look up 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>
```

Snowflake Compliance Checks

The Snowflake plugin is meant to connect to Snowflake REST API endpoints that can be found in the Snowflake products. The plugin connects to Snowflake targets, fetches data from REST API endpoints, and evaluates the output for specific expressions.

Scan Requirements

Credentials

The plugin requires the cloud services scanning credentials of Snowflake API that include the following:

- **Username**—The required username for an account on the Snowflake target with a 2048-bit RSA key pair assigned.
- **Account Identifier**—The required Snowflake account identifier (for more information, see https://docs.snowflake.com/en/user-guide/admin-account-identifier).
- **Role**—The optional Snowflake role. If you provide the roll, the audit runs SQL statements using the specified role. The default value is `ACCOUNTADMIN`. If you omit the role, the SQL statement runs using the user `DEFAULT_ROLE` (for more information, see https://docs.snowflake.com/en/sql-reference/sql/alter-user).
- **Warehouse**—The optional Snowflake warehouse. If you provide the warehouse, the audit runs SQL statements using the specified warehouse. The default value is **COMPUTE_WH**. If you omit the warehouse, the SQL statement runs using your `DEFAULT_WAREHOUSE` (for more information, see https://docs.snowflake.com/en/user-guide/warehouses).
- **Private Key**—The required PEM formatted 2048-bit RSA private key to use when connecting to the target. For instructions on how to generate the key and assign it to a user, see https://docs.snowflake.com/en/user-guide/key-pair-auth.
- **Passphrase**—The optional passphrase for the private key.

Permissions

A user with the `ACCOUNTADMIN` role is required.

Checks

All Snowflake REST API compliance checks must be bracketed with the `check_type` encapsulation and the `Snowflake` designation. This is required to differentiate .audit files intended specifically for Snowflake REST API from other types of compliance audits.

```
<check_type:"Snowflake">
* audit content
</check_type>
```

Review the following topics on specific elements in the audit language.

- Snowflake Audit Containers
- Snowflake Audit Items
- Snowflake Comments
- Snowflake SQL_POLICY
- Snowflake KB_VALUE

Snowflake Audit Containers

An audit file contains one or more containers that can control the flow of the audit that is executed. The containers are methods to encapsulate Audit Items or other containers.

- `check_type`—The top level container that must exist in every audit.
- `if`—The container to define conditional auditing.
    - `condition`—Contains audit items to define the conditional requirements.
    - `then`—Contains the checks that are evaluated if the conditional audit items are PASSED.
    - `else`—Contains the checks that are evaluated if the conditional audit items are FAILED or WARNING.

check_type

You must bracket all compliance checks with the `check_type` encapsulation. The value of the `check_type` is used to identify what plugins are used to evaluate the audit.

The audit content that is supported inside the `check_type` are Audit Items and `if` containers.

```
<check_type:"[Plugin_Designation]">
...audit content...
</check_type>
```

if

The `if` is a wrapper around the conditional containers. Based on the result of the `condition`, if the `condition` passes, the audit content in the `then` is evaluated. If the `condition` fails, the audit content in the `else` is evaluated.

The audit content that is supported inside the `if` are `condition`, `then`, and `else` containers.

```
<if>
 [condition]
 [then]
 [else]
</if>
```

condition

The `condition` defines the audit items to evaluate and if one or all must pass.

- `AND`–All audit items must pass to evaluate the `then`.
- `OR`–One audit items must pass to evaluate the `then`.

The audit content that is supported inside the `condition` are Audit Items.

```
<condition type:"[AND|OR]">
...audit content...
</condition>
```

then/else

The `then` and `else` are generic containers of other audit content, and are only differentiated in the context of an `if`.

The audit content that is supported inside the `then` and `else` are Audit Items and `if` containers.

```
<then>
...audit content...
</then>
```

```
<else>
...audit content...
</else>
```

Snowflake Audit Items

Each check in an audit area is defined using a couple of foundational audit items: `custom_item` and `report`.

A `custom_item` is the base of all functional checks inside an audit. It is the wrapper that manages the definition of each audit item.

A `report` is a method in the audit file to report a static result that does not change regardless of how a target is configured. It is commonly used in reporting of conditional checks, reporting audit items that are not technically possible to retrieve data, or high level information on the audit that is being evaluated.

Usage

```
<custom_item>
  type: [TYPE_OF_CHECK]
  description: ["description"]
  (optional) info: ["information regarding the audit item"]
  (optional) solution: ["information on how to remediate the audit item"]
  (optional) see_also: ["url reference for audit"]
  (optional) reference: ["standard|control,standard|control,..."]
  (optional) severity: [HIGH|MEDIUM|LOW]
</custom_item>
```

```
<report type:"[PASSED|WARNING|FAILED]">
  description: ["description"]
  (optional) info: ["information regarding the audit item"]
  (optional) solution: ["information on how to remediate the audit item"]
  (optional) see_also: ["url reference for audit"]
  (optional) reference: ["standard|control,standard|control,..."]
  (optional) output : ["custom output other than report type"]
</report>
```

type

The `type` field in a `custom_item` is used to identify what other fields are required and how to gather, transform, and evaluate data from the target.

The `type` attribute in a `report` is used to provide the result for the audit item.

description

A `description` is required as it is the most common identifier of the audit items.

info

The `info` is general information about the audit item. It is commonly used to communicate what is being evaluated and why it is important.

solution

The `solution` is text that relays how an audit item can be remediated if it has FAILED.

see_also

The `see_also` is a URL that is used as a reference to the configuration guide or benchmark that is being audited. It is commonly used as a method to report on audit items that refer to the same benchmark.

reference

severity (custom_item only)

The severity is a method to "soften" a FAILED result that is posted by an audit item. The example of a softening is a result of FAILED would be reported as a WARNING when a `severity` of MEDIUM is used.

The following severities are defined:

- HIGH has no change
- MEDIUM to WARNING
- LOW to PASSED

If there is a scenario in an audit file that a result should be moved from PASSED to a lower result, the method is to adjust the evaluation of the audit item to always fail, and then apply the desired severity.

output (report only)

The output field is a method to provide static content in the "output" of the result and attempts to keep all other informational fields the same between different reports for the same control. The best example of this is the use of a report in a "then" or "else" should maintain the same informational fields, but may need a differentiator for why the result changes.

Examples

Custom item examples are available in the documentation for each specific type.

```
<report type:"WARNING">
  description : "Audit file for Unix"
  output : "NOTE: This audit file does not support the OS that is identified on your target."
</custom_item>
```

Snowflake Comments

Comments in the audit file are identified by lines that begin with a hash symbol (#). Any commented lines are ignored by the plugin.

The one exception for commented lines is the `ui_metadata` block at the beginning of Tenable published audit files. This block is an XML structure of metadata that was originally only read by Tenable Nessus to construct elements in its user interface, but is now used by the plugins to configure audit file metadata and variables.

Snowflake SQL_POLICY

This policy item executes a SQL statement on the DBMS being audited and evaluates the resulting columns returned from the SQL result.

Usage

```
<custom_item>
  type: SQL_POLICY
  description: ["description"]
  sql_request: ["sql statement to run"]
  sql_types: [STRING|REGEX|INTEGER][,....]
  sql_expect: ["text"|"regex"|number]
  (optional) match_all : [YES|NO]
  (optional) match_case : [YES|NO]
  (optional) num_rows : [number|range]
</custom_item>
```

sql_request

The SQL statement that is queried against the DBMS.

sql_types

A comma separate list of the types for the columns that are returned from the SQL query. The values that are allowed are:

- `STRING`—Text based results.
- `REGEX`—Text based results, evaluated by a regular expression.
- `INTEGER`—Numeric based results, including the use of a range as designated by `[MIN..MAX]`.
- `NULL`—NULL value return.
- `STRING_OR_NULL`—Text based results or NULL value.
- `REGEX_OR_NULL`—Text based results, evaluated by a regular expression or NULL value.
- `INTEGER_OR_NULL`—Numeric based results or NULL value.

sql_expect

A comma separated list of the values, or regular expression, to evaluate the results from the SQL query.

The values for each of the columns must match the types that are defined in the `sql_types`. With this, text values need to be wrapped in double quotes (") and numbers do not need the double quotes.

A special value of `NO_ROWS_RETURNED` can be used as the only expected value to match on cases that no rows are returned. This is more explicit than using a `check_option` as described below.

> **Note:** The number of `sql_expect` must match the number of `sql_types`, with the exception of `NO_ROWS_RETURNED` expect.

match_all

(Optional) Setting `match_all` to `YES` requires the expectation to match all rows of query result, and not just a row. If `match_all` is set to the default of `NO`, only one row must match for the check to pass.

match_case

(Optional) Setting `match_case` to `YES` makes all the column comparisons to be case sensitive. If `match_case` is set to the default of `NO`, the comparison is case insensitive.

num_rows

(Optional) Setting `num_rows` to an integer value allows the check to pass if that number of rows is returned, and fail if more or less rows are returned. Not setting `num_rows` should not affect the result based on the number of rows returned. A variable number of rows can be evaluated by specifying a range in the form of `[MIN..MAX]`.

Example

```
<custom_item>
  type        : SQL_POLICY
  description : "SQL4-00-037700 - SQL generates Trace/Audit records for all activities or other sys-
level access - DATABASE_CHANGE_GROUP"
  sql_request : "SELECT audit_action_name, audited_result FROM sys.server_audit_specification_details
WHERE server_specification_id =(SELECT server_specification_id FROM sys.server_audit_specifications)
AND audit_action_name = 'DATABASE_CHANGE_GROUP';"
  sql_types   : STRING, REGEX
  sql_expect  : "DATABASE_CHANGE_GROUP", "(SUCCESS|SUCCESS AND FAILURE)"
</custom_item>
```

Snowflake KB_VALUE

The `KB_VALUE` check fetches data from the Knowldegebase (KB) and analyzes the output with regular expressions to identify if the data associated with the provided path matches the expected output. A KB is created for each target during a Tenable Nessus scan and is the collected information that is shared with other plugins.

Usage

```
<custom_item>
  type : KB_VALUE
  description : ["description"]
  kb_path : ["kb path to check"]
  (optional) regex : ["regular expression to reduce options"]
  expect : ["regular expression that passes if found"]
  (optional) kb_path_required : [YES|NO]
  (optional) match_all : [YES|NO]
  (optional) match_case : [YES|NO]
</custom_item>
```

kb_path

The `kb_path` is the path to the kb value(s) to be evaluated.

regex

(Optional) The `regex` is used to filter the full configurations to a smaller set of lines of text based on the regular expression. You can use multiple `regex` to narrow down the searchable configuration.

expect

For `expect`, if the regular expression matches a line of text, the check results as PASSED. If there are no matches, the check results as FAILED.

To indicate if all lines need to match or that lines are case-sensitive, use the modifiers `match_all` or `match_case`.

kb_path_required

(Optional) The `kb_path_required` field can be set to specify if the audited `kb_path` is required to be present or not. If this option is not set, it is assumed it is required.

match_all

(Optional) Setting `match_all` to `YES` requires the expectation to match all lines of text, and not just a single line of text. If `match_all` is set to the default of `NO`, only one line must match for the check to pass.

match_case

(Optional) Setting `match_case` to `YES` makes the comparison to be case sensitive. If `match_case` is set to the default of `NO`, the comparison is case insensitive.

Example

```
<custom_item>
  type        : KB_VALUE
  description : "Check port"
  kb_path     : "public/port*"
  regex       : "[0-9]+"
  expect      : "443"
</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.

Tip: 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

This section includes the following information:

- SonicWALL SonicOS Syntax

SonicWALL SonicOS 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>
```

Unix Configuration Audit Compliance File Reference

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

This section includes the following information:

- Unix Configuration Check Type
- Unix Configuration Keywords
- Unix Configuration Custom Items
- Built-In Checks
- Conditions
- Global Settings

Unix Configuration Check Type

All Unix compliance checks must be bracketed with the "**check_type**" encapsulation and the "Unix" designation. All Compliance and Audit Files contains an example Unix compliance check starting with the **check_type** setting for "Unix" and is finished by the "**</check_type>**" tag.

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

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

Unix Configuration Keywords

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

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

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

Keyword	Example Usage and Supported Settings
	`except: "guest" | "guest1" | "guest2"`
`expect`	This keyword is used in combination with `regex`. It provides the ability to look for specific values within files. Example: `<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>`
`file`	This keyword is used to describe the absolute or relative path of a file to be checked for permissions and ownership settings. Examples: `file: "/etc/inet/inetd.conf"` `file: "~/inetd.conf"` The `file` value can also be a glob. Example: `file: "/var/log/*"` 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.
`file_required`	This keyword is used with FILE_CHECK, FILE_CHECK_NOT, FILE_CONTENT_CHECK, and FILE_CONTENT_CHECK NOT. The file_

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

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

Keyword	Example Usage and Supported Settings
`json_transform`	This keyword is used with FILE_CONTENT_CHECK and FILE_CONTENT_CHECK_NOT to evaluate JSON formatted data.
`mask`	This keyword is the opposite of mode where one can specify permissions that should not be available for a particular user, group or other member. Unlike **`mode`** that checks for an exact permission value, **`mask`** 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 **`mask`**. (Where mode may fail a file with a permission of 640 as not matching an audit expecting a value of 644, **`mask`** will see that 640 is "more secure" and will pass the audit as successful.) Example: `mask: 022` This would specify any permission is OK for owner and no write permissions for group and other member. A **`mask`** value of "7" would mean no permissions for that particular owner, group or other member.
`md5`	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. Example: `<custom_item>` `type: FILE_CHECK` `description: "/etc/passwd has the proper md5 set"` `required: YES` `file: "/etc/passwd"` `md5: "ce35dc081fd848763cab2cfd442f8c22"` `</custom_item>`
`min_occurrences`	This keyword specifies the minimum number of specific values in FILE_CONTENT_CHECK files. Example:

Keyword	Example Usage and Supported Settings
	`min_occurrences: "3"`
`mode`	This keyword describes the set of permissions for a file/folder under consideration. The mode keyword can be represented in string or octal format. Examples: `mode: "-rw-r--r--"` `mode: "644"` `mode: "7644"`
`name`	This keyword is used to identify process name in PROCESS_CHECK. Example: `name: "syslogd"`
`not_expect`	This keyword is used in combination with regex. It provides the ability to look for specific failing values in FILE_CONTENT_CHECK.
`not_regex`	This keyword is used with MACOSX_DEFAULTS_READ to evaluate all items found do not match the regex specified.
`operator`	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: • `lt` (less than) • `lte` (less than or equal) • `gte` (greater than equal) • `gt` (greater than) • `eq` (equal) Example:

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

Keyword	Example Usage and Supported Settings
	`regex: ".*LogLevel=9$"` 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.
`required`	This keyword is used to specify if the audited item is required to be present or not on the remote system. For example, if **`required`** is set to “NO” and the check **`type`** is “FILE_CHECK”, then the check will pass if the file exists and permissions are as specified in the `.audit` file or if the file does not exist. On the other hand, if **`required`** was set to “YES”, the above check would fail.
`rpm`	This keyword is used to specify the RPM to look for when used in conjunction with RPM_CHECK. Example: `<custom_item>` `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` `</custom_item>`
`search_ locations`	This keyword can be used to specify searchable locations within a file system. Example: `search_locations: "/bin"`

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

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

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

Unix Configuration Custom Items

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

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

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

This section includes the following information:

- AUDIT_XML
- AUDIT_ALLOWED_OPEN_PORTS
- AUDIT_DENIED_OPEN_PORTS
- AUDIT_PROCESS_ON_PORT
- BANNER_CHECK
- CHKCONFIG
- CMD_EXEC
- FILE_CHECK
- FILE_CHECK_NOT
- FILE_CONTENT_CHECK
- FILE_CONTENT_CHECK_NOT
- FIND_CMD
- GRAMMAR_CHECK
- MACOSX_DEFAULTS_READ
- MACOSX_OSASCRIPT
- PKG_CHECK
- PROCESS_CHECK
- RPM_CHECK
- SVC_PROP
- XINETD_SVC

AUDIT_XML

The “AUDIT_XML” audit check allows you to examine and audit the contents of an XML file by first applying XSL transforms, extracting relevant data, and then determine compliance based on the `regex`, `expect`, and `not_expect` keywords. The check consists of four or more keywords, keywords type, description file, and xsl_stmt directives (mandatory), which are followed by `regex`, `expect`, or `not_expect` keywords to audit the content.

Tip: For information about the parameters commonly found in Unix custom items, see Unix Configuration Keywords.

Example

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

Tip: For information about the parameters commonly found in Unix custom items, see Unix Configuration Keywords.

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

AUDIT_DENIED_OPEN_PORTS

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

Tip: For information about the parameters commonly found in Unix custom items, see Unix Configuration Keywords.

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

```
</custom_item>
```

AUDIT_PROCESS_ON_PORT

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

Tip: For information about the parameters commonly found in Unix custom items, see Unix Configuration Keywords.

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

BANNER_CHECK

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

Usage

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

The following are descriptions of the keywords:

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

Tip: For information about the parameters commonly found in Unix custom items, see Unix Configuration Keywords.

Example

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

CHKCONFIG

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

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

Tip: For information about the parameters commonly found in Unix custom items, see Unix Configuration Keywords.

Example

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

CMD_EXEC

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

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

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

> **Tip:** For information about the parameters commonly found in Unix custom items, see Unix Configuration Keywords.

Example

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

FILE_CHECK

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

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

The allowed keywords are:

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

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

Tip: For information about the parameters commonly found in Unix custom items, see Unix Configuration Keywords.

Examples:

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

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

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

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

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

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

FILE_CHECK_NOT

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

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

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

The allowed keywords are:

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

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

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

Tip: For information about the parameters commonly found in Unix custom items, see Unix Configuration Keywords.

Examples

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

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

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

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

FILE_CONTENT_CHECK

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

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

Tip: For information about the parameters commonly found in Unix custom items, see Unix Configuration Keywords.

Examples

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

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

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

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

Add a "~" to a file parameter to configure FILE_CONTENT_CHECK to scan a user's home directories for non-compliant content.

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

FILE_CONTENT_CHECK_NOT

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

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

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

> **Tip:** For information about the parameters commonly found in Unix custom items, see Unix Configuration Keywords.

Example

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

GRAMMAR_CHECK

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

> **Tip:** For information about the parameters commonly found in Unix custom items, see Unix Configuration Keywords.

Example

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

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

MACOSX_DEFAULTS_READ

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

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

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

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

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

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

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

The following properties are supported:

Property	Description	Accepted Value
`plist_name`	The plist that you want to query.	Example: `com.apple.digihub`

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

Tip: For information about the parameters commonly found in Unix custom items, see Unix Configuration Keywords.

Examples

Example 1:

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

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

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

Example 2:

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

MACOSX_OSASCRIPT

The **MACOSX_OSASCRIPT** audit check uses the `osascript` command to return configured payload data.

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

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

Usage

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

The following properties are supported:

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

Examples

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

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

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

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

PKG_CHECK

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

> **Tip:** For information about the parameters commonly found in Unix custom items, see Unix Configuration Keywords.

Examples

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

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

PROCESS_CHECK

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

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

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

RPM_CHECK

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

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

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

Examples

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

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

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

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

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

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

SVC_PROP

The “SVC_PROP” audit check lets one interact with the **`svcprop -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 **`svcprop_option`** field can be set to specify if the audited string being searched for is required to be present or not. This field access CAN_BE_NULL or CANNOT_BE_NULL as arguments.

> **Tip:** For information about the parameters commonly found in Unix custom items, see Unix Configuration Keywords.

Examples

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

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

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

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

XINETD_SVC

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

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

Tip: For information about the parameters commonly found in Unix custom items, see Unix Configuration Keywords.

Example

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

Built-In Checks

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

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

Note: All built-in and `custom_item` checks support the following optional keywords:

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

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

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

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

- `see_also` –Allows you to include links that might provide helpful information about a check.

  Example: *see_also: "http://www.fireeye.com/support/"*

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

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

This section includes the following information:

Password Management

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

This section includes the following information:

- min_password_length
- max_password_age
- min_password_age

min_password_length

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

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

Usage

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

Example

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

max_password_age

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

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

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

Usage

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

Example

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

min_password_age

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

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

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

Usage

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

Example

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

Root Access

root_login_from_console

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

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

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

Usage

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

```
</item>
```

Permissions Management

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

- accounts_bad_home_permissions
- accounts_bad_home_group_permissions
- accounts_without_home_dir
- active_accounts_without_home_dir
- invalid_login_shells
- login_shells_with_suid
- login_shells_writeable
- login_shells_bad_owner

accounts_bad_home_permissions

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

This check can be modified using the following tags:

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

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

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

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

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

Usage

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

accounts_bad_home_group_permissions

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

Usage

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

accounts_without_home_dir

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

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

Usage

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

active_accounts_without_home_dir

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

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

Usage

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

invalid_login_shells

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

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

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

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

Usage

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

login_shells_with_suid

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

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

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

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

Usage

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

login_shells_writeable

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

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

Ensure the permissions of each shell are set appropriately.

Usage

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

login_shells_bad_owner

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

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

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

Usage

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

Password File Management

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

- passwd_file_consistency
- passwd_zero_uid
- passwd_duplicate_uid
- passwd_duplicate_gid
- passwd_duplicate_username
- passwd_duplicate_home
- passwd_shadowed
- passwd_invalid_gid

passwd_file_consistency

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

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

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

Usage

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

passwd_zero_uid

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

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

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

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

Usage

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

passwd_duplicate_uid

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

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

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

Give each user on the system a unique ID.

Usage

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

passwd_duplicate_gid

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

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

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

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

Usage

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

passwd_duplicate_username

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

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

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

Delete duplicate usernames or modify them to be different.

Usage

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

passwd_duplicate_home

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

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

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

Usage

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

passwd_shadowed

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

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

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

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

Usage

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

passwd_invalid_gid

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

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

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

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

Usage

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

Group File Management

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

- group_file_consistency
- group_zero_gid
- group_duplicate_name
- group_duplicate_gid
- group_duplicate_members
- group_nonexistent_users

group_file_consistency

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

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

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

Usage

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

group_zero_gid

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

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

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

Usage

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

group_duplicate_name

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

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

Delete or rename duplicate group names.

Usage

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

group_duplicate_gid

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

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

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

Usage

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

group_duplicate_members

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

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

Ensure that each member is listed only once.

Usage

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

group_nonexistent_users

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

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

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

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

Usage

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

Root Environment

dot_in_root_path_variable

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

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

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

Usage

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

writeable_dirs_in_root_path_variable

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

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

Usage

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

File Permissions

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

- find_orphan_files
- find_world_writeable_files
- find_world_writeable_directories
- find_world_readable_files
- find_suid_sgid_files
- home_dir_localization_files_user_check
- home_dir_localization_files_group_check

find_orphan_files

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

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

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

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

Usage

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

Examples

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

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

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

```
</item>
```

find_world_writeable_files

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

`# chmod o-w world_writeable_file`

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

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

Usage

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

Example

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

find_world_writeable_directories

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

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

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

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

Usage

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

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

Example

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

find_world_readable_files

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

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

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

Usage

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

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

Example

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

find_suid_sgid_files

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

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

`find_option: [suid | sgid | both]`

The **`both`** setting is default if no **`find_option`** tag is specified.

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

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

Usage

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

Example

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

home_dir_localization_files_user_check

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

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

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

- **.bash_profile**
- **.bashrc**
- **.bash_logout**
- **.env**
- **.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

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

Usage

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

Unnecessary Files

find_pre-CIS_files

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

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

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

Usage

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

Docker Containers

list_docker_container_packages

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

Usage

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

Conditions

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

The syntax to perform conditions is the following:

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

Example

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

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

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

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

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

Caveats

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

Unix Content Audit Compliance File Reference

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

This section includes the following information:

Check Type

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

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

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

Item Format

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

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

Keyword	Required	Description
		Multiple patterns are listed in quotes and separated with pipe characters. Simple pattern matching is also supported in this keyword with the period. When matching the string “C.T”, the **`expect`** statement would match “CAT”, “CaT”, “COT”, “C T” and so on. **Note:** The **`expect`** keyword may be run standalone for single pattern matching, however, if the **`regex`** keyword is used, **`expect`** is required. Unlike Compliance Checks, File Content Compliance Check **`regex`** and expect do not have to match the same data string(s) within the searched file. File Content checks simply require that both the **`regex`** and **`expect`** statements match data within the `<max_size>` bytes of the file searched.
`file_name`	no	Whereas the **`file_extension`** 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. 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".
`max_size`	no	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.
`only_show`	no	This keyword supports revealing a specific number of characters specified by policy. When matching sensitive data such as credit card numbers, your organization may

Keyword	Required	Description
		require that only a limited number of digits be made visible in the report. The default is 4 or half of the matched string, whichever is smaller. For example, if a matched string is 10 characters long and `only_show` is set to 4, only the last 4 characters are shown. If the matched string is 6 characters long, only 3 characters will be shown. **Note:** When you match against US Social Security numbers (SSNs), the specified number of digits are revealed *in front* of the string (for example, `123-XX-XXXX`).
`regex_replace`	no	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.
`include_paths`	no	This keyword allows for directory or drive inclusion within the search results. This keyword may be used in conjunction with, or independently of the **`exclude_paths`** 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. Only drive letters or folder names can be specified with the **`include_paths`** keyword. File names cannot be included in the **`include_paths`** value string.
`exclude_paths`	no	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 **`include_paths`** keyword. This is particularly helpful in cases where a particular drive, directory, or file must be excluded from

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

Usage

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

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

Unix Content Command Line Examples

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

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

./nasl -t <IP> /opt/nessus/lib/nessus/plugins/ unix_file_content_compliance_check.nbin

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

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

This section includes the following information:

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

Target Test File

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

```
abcdefghijklmnopqrstuvwxyz
01234567890
```

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

Please take this data and copy it to any Unix system you have credentialed access to. Name the file “Tenable_Content.tns”.

Search Files for Properly Formatted VISA Credit Card Numbers

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

```
<item>
type: FILE_CONTENT_CHECK
description: "Determine if a file contains a properly formatted VISA credit card
number."
file_extension: "pdf" | "doc" | "xls" | "xlsx" | "xlsm" | "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.

Search for AMEX Credit Card Numbers

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

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

The output we get this time is as follows:

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

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

Auditing Different Types of File Formats

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

For documents that store data in Unicode format, the parsing routines of the **.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 Tenable Security Center, the data only needs to travel from the scanned Unix host to the scanner performing the vulnerability audit.

Performance Considerations

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

Start small and build up

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

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

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

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

Which extensions should we search for?

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

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

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

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

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

How much data should be scanned?

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

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

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

How many policies are being evaluated?

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

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

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

Can an agent be used?

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

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.

This section includes the following information:

Requirements

To perform a successful compliance scan against VMware systems, users must have the following:

- Administrative credentials for VMware vCenter or ESXi

  vCenter compliance auditing is intended to be used with vCenter SOAP API

  > **Note:** 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). Use ESXi compliance auditing with the ESXi SOAP API, and use vCenter compliance auditing with the vCenter SOAP API.

- Audit policy for VMware vCenter/ESXi Compliance Checks
- Plugin ID #64455 (VMware vCenter/ESXi Compliance Checks)

Supported Versions

Tenable Nessus can audit ESXi and vCenter versions 5.x, 6.x, 7.x, and 8.x.

Check Types

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:

```
<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:name\"/> : vmsafe.enable : <xsl:value-of
select=\"audit:propSet/audit:val
[@xsi:type='VirtualMachineConfigInfo']/audit:extraConfig[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:OptionValue
[audit:key[text()='config.vpxd.hostPasswordLength']]/audit:value\"/>"
xsl_stmt: "</xsl:template>"
expect: "config.vpxd.hostPasswordLength : 30"
</custom_item>
```

Keywords

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

Keyword	Example Use and Supported Settings
`type`	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: • AUDIT_VM • AUDIT_ESX • AUDIT_VCENTER
`description`	This keyword provides the ability to add a brief description of the check that is being performed. It is strongly recommended that the **`description`** field be unique and no distinct checks have the same description field. Tenable uses this field to automatically generate a unique plugin ID number based on the description field. Example: `description: "Disconnect unauthorized devices - 'floppyX.present = false'"`
`info`	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

Keyword	Example Use and Supported Settings
	limit. The info content must be enclosed in double-quotes. Example: `info: "Make sure floppy drive is not attached"`
`regex`	This keyword allows searching items that match a particular regex expression. Example: `regex: "floppy([Xx]|[0-9]+)\\.present :"`

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
`expect`	This keyword allows auditing the config item matched by the **`regex`** keyword, or if the `regex` keyword is not used, looks for the **`expect`** string in the entire config. The check passes as long as the config line found by **`regex`** matches the **`expect`** string or in the case where **`regex`** is not set, it passes if the **`expect`** string is found in the config. Example: `regex: "floppy([Xx]|[0-9]+)\\.present :"` `expect: "floppy([Xx]|[0-9]+)\\.present : false"` Or: `expect: "floppy([Xx]|[0-9]+)\\.present : false"` In the above cases, the **`expect`** keyword ensures that the floppy drive is not present.
`not_expect`	This keyword allows searching the configuration items that should not be in the

Keyword	Example Use and Supported Settings
	configuration. It acts as the opposite of **expect**. The check passes as long as the config line found by **regex** does not match the **not_expect** string or if the **regex** keyword is not set, it passes as long as **not_expect** string is not found in the config. Example: `regex: "floppy([Xx]|[0-9]+)\\.present :"` `not_expect: "floppy([Xx]|[0-9]+)\\.present : false"` Or: `not_expect: "floppy([Xx]|[0-9]+)\\.present : false"` In the above cases, the **not_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.

The report will appear as follows:

```
Test VM 2, poweredOff (toolsNotInstalled) - vmsafe.enable : NOT found
Test VM Audit (192.0.2..123) - vmsafe.enable : NOT found
```

Both ESX/ESXi and vCenter can be scanned with the same policy.

Note: vCenter checks run against ESX/ESXi hosts will be skipped.

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.

This section includes the following information:

- Value Data
- ACL Format
- Custom Items
- Items
- Forced Reporting
- Conditions

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">
```

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.

This section includes the following information:

- Complex Expressions
- The "check_type" Field
- The "group_policy" Field

- The "info" Field
- The "debug" Field

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 in the Windows Configuration topic 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 that the remote value is different than the policy value
- CHECK_NOT_REGEX: checks that the remote value does not match the regex in the policy value (only works with POLICY_TEXT and POLICY_MULTI_TEXT)
- CHECK_GREATER_THAN: checks that the remote value is greater than the policy value
- CHECK_GREATER_THAN_OR_EQUAL: checks that the remote value is greater or equal than the policy value
- CHECK_LESS_THAN: checks that the remote value is less than the policy value
- CHECK_LESS_THAN_OR_EQUAL: checks that 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 the Tenable Security Center 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 regular line breaks, or the **\n** separator to specify multiple strings. For example, consider the following audit with regular line breaks:

```
<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         : "CCE-60
Tenable Best Practices Policy 1005-a
This items tests for the presence of the administrator account"
</custom_item>
```

Or using \n separator:

```
<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         : "CCE-60\nTenable Best Practices Policy 1005-a\nThis items tests for the
presence of the administrator account"
```

```
</custom_item>
```

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 settings:

- File Access Control Checks
- Registry Access Control Checks
- Service Access Control Checks
- Launch Permission Control Checks
- Launch2 Permission Control Checks
- Access Permission Control Checks

File Access Control Checks

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.

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>
```

Syntax

Associated Types	Allowed Types
`acl_inheritance`	not inherited inherited not used
`acl_apply`	this folder only this object only this folder and files this folder and subfolders this folder, subfolders and files files only subfolders only subfolders and files only
`acl_allow` `acl_deny`	These settings are optional. Generic rights: • full control • modify

Associated Types	Allowed Types
	• read & execute • read • write • list folder contents Advanced rights: • full control • traverse folder / execute file • list folder / read data • read attributes • read extended attributes • create files / write data • create folders / append data • write attributes • write extended attributes • delete subfolder and files • delete • read permissions • change permissions • take ownership

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

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.

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>
```

Syntax

Associated Types	Allowed Types
`acl_ inheritance`	not inherited inherited not used
`acl_apply`	this key only this key and subkeys subkeys only
`acl_allow` `acl_deny`	These settings are optional and are used to define the rights a user has on the object. Generic rights: • full control • read Advanced rights: • full control • query value • set value • create subkey • enumerate subkeys • notify • create link • delete • write dac • write owner • read control

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

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.

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>
```

Syntax

Associated Types	Allowed Types
acl_inheritance	not inherited inherited not used
acl_apply	this object only
acl_allow acl_deny	These settings are optional and are used to define the rights a user has on the object. Generic rights: • full control • read • start, stop and pause • write • delete Advanced rights: • full control • delete

Associated Types	Allowed Types
	• query template • change template • query status • enumerate dependents • start • stop • pause and continue • interrogate • user-defined control • read permissions • change permissions • take ownership

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" | "userdefined
control" | "delete" | "read permissions" | "change permissions" | "take
ownership"
</user>

</acl>
```

Launch Permission Control Checks

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.

Usage

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

<user: ["user_name"]>
Copyright © 2016. Tenable Network Security, Inc. All rights reserved. Tenable Network
Security and Nessus are registered trademarks of Tenable Network Security, Inc. 20
acl_inheritance: ["value"]
acl_apply: ["value"]
(optional) acl_allow: ["rights value"]
(optional) acl_deny: ["rights value"]
</user>

</acl>
```

Syntax

Associated Types	Allowed Types
acl_inheritance	not inherited inherited
acl_apply	this object only
acl_allow acl_deny	These settings are optional and are used to define the rights a user has on the object. Generic rights: • local launch • remote launch • local activation • remote activation

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

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.

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>
```

Syntax

Associated Types	Allowed Types
`acl_inheritance`	not inherited inherited
`acl_apply`	this object only
`acl_allow` `acl_deny`	These settings are optional and are used to define the rights a user has on the object. Generic rights: • launch

> 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

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.

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>
```

Syntax

Associated Types	Allowed Types
`acl_inheritance`	not inherited inherited
`acl_apply`	this object only
`acl_allow`	These settings are optional and are used to define the rights a user has on

Associated Types	Allowed Types
acl_deny	the object. Generic rights: • local access • remote access

An example access control check is shown below:

```
<access_acl: "3">

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

Tip: Custom audit checks may use **</custom_item>** and **</item>** interchangeably for the closing tag.

Note: All built-in and `custom_item` checks support the following optional keywords:

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

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

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

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

- `see_also` –Allows you to include links that might provide helpful information about a check.

  Example: *see_also: "http://www.fireeye.com/support/"*

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

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

This section includes the following information:

- AUDIT_USER_TIMESTAMPS
- BANNER_CHECK
- CHECK_ACCOUNT
- CHECK_LOCAL_GROUP
- FILE_AUDIT
- FILE_CHECK
- FILE_CONTENT_CHECK
- FILE_CONTENT_CHECK_NOT
- FILE_PERMISSIONS
- FILE_VERSION
- GROUP_MEMBERS_POLICY
- GUID_REGISTRY_SETTING
- KERBEROS_POLICY
- LOCKOUT_POLICY
- PASSWORD_POLICY
- REG_CHECK
- REGISTRY_AUDIT
- REGISTRY_PERMISSIONS
- REGISTRY_SETTING
- REGISTRY_TYPE
- SERVICE_AUDIT
- SERVICE_PERMISSIONS
- SERVICE_POLICY
- USER_GROUPS_POLICY

- USER_RIGHTS_POLICY
- WMI_POLICY

PASSWORD_POLICY

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.

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>
```

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 log off when log on hours expire”)

  `value_type: POLICY_SET`

  `value_data: "Enabled" or "Disabled"`

- LOCKOUT_ADMINS ("Allow Administrator account lockout")

  `value_type: POLICY_SET`

  `value_data: "Enabled" or "Disabled"`

> **Note:** 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”.

Example

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

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.

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 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]`

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

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.

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 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]`

Note: 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

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`**.

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 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:** 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

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.

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 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:** 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

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.

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.psc1"
</custom_item>
```

Required Fields

Field	Description
`type: AUDIT_POWERSHELL`	This specifies that the audit check will be performed using PowerShell. PowerShell is often used in audits to extract system configurations, registry values, security policies, and other settings.
`description: "Powershell check"`	Usually a title that comes from the recommendation found in the authority guidance - Benchmark or STIG. Could also be a brief explanation of what the check does.
`value_type: [value_type]`	Defines the type of data being evaluated. For AUDIT_POWERSHELL this will always be POLICY_TEXT.
`value_data: [value]`	The expected value for the check. This evaluates the output generated by the `powershell_args` command.
`powershell_args: ["arguments for powershell.exe"]`	Specifies the command-line arguments passed to `powershell.exe`.

Optional Parameters

Parameter	Description
`only_show_cmd_output: YES or NO`	Determines whether only command output is shown. `YES` the entire output is reported.
`check_type: [CHECK_TYPE]`	Defines how the check is performed (for example, `CHECK_REGEX`). Example: `CHECK_REGEX` ensures the retrieved value matches the regex used in the `value_data`.

`severity: ["MEDIUM"]`	Assigns a severity level to the finding. If set to `MEDIUM`, the check will return a warning for failing results.
`powershell_option: CAN_BE_NULL`	Specifies if the PowerShell evaluates as a pass if no output is returned from the `powershell_args`.
`powershell_console_file: "C:\Program Files\Microsoft\Exchange Server\ExShell.psc1"`	Loads a PowerShell console file before executing commands.

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 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 the optional field **`only_show_cmd_output`** if the entire command output needs to be reported:

`only_show_cmd_output: YES or NO`

Other Considerations

- PowerShell scripts included in audits have a 8,192 character limit.
- 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.
- Powershell is not installed by default on some Windows operating systems (for example, 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.
- For this check to work correctly, WMI service needs to be enabled. Also configure the firewall to “Allow inbound remote administration exception."
- Cmdlet aliases (for example, “gps” instead of “Get-Process”) are not allowed.

Examples

This example runs the **Get-Hotfix** PowerShell cmdlet, specifies a where-object not to 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>
```

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 = "."

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

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

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-output "Account Type: " $objItem.AccountType
write-output "Description: " $objItem.Description
write-output "Disabled: " $objItem.Disabled
write-output "Full Name: " $objItem.FullName
write-output "Installation Date: " $objItem.InstallDate
write-output "Lockout: " $objItem.Lockout
write-output "Password Changeable: " $objItem.PasswordChangeable
write-output "Password Expires: " $objItem.PasswordExpires
write-output "Password Required: " $objItem.PasswordRequired
write-output "SID: " $objItem.SID
write-output "SID Type: " $objItem.SIDType
write-output "Status: " $objItem.Status
write-output ""
}
}
```

Next, Base64 encodes 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**, per the following example:

```
<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 appears similar to the following example:

```
"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

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.

Usage

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

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.

Examples

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>
```

This example compares a supplied SHA1 hash against 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

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.

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 item uses the 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.

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

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()).

Usage

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

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

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()).

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>
```

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_EXCHANGE

This policy item runs exchange cmdlets on the target Exchange server and returns the results. Results are evaluated with use of string or regular expression matching of the PowerShell output, similar to `AUDIT_POWERSHELL`.

Usage

```
<custom_item>
type: AUDIT_EXCHANGE
description: ["description"]
value_type: POLICY_TEXT
value_data: ["banner content"]
powershell_args: ["exchange powershell cmdlets"]
(optional) powershell_option: [CAN_BE_NULL]
(optional) secure_string: ["encrypted secure string"]
(optional) check_type: [CHECK_EQUAL|CHECK_REGEX|...]
(optional) only_show_cmd_output: [YES|NO]
</custom_item>
```

The following are descriptions of the keywords:

- `value_type:` The value is `POLICY_TEXT`. If you use `POLICY_MULTI_TEXT`, the evaluation will work, but NULL will appear as the *Remote* value.
- `value_data:` The content of the expected PowerShell output.
- `powershell_args:` The value is the Exchange cmdlet with additional PowerShell formatting options. The output of this command will be returned, and should resemble what would be returned from the Exchange shell.

- `powershell_option:` The value is `CAN_BE_NULL`, which allows the check to pass if there is no data returned from PowerShell.
- `secure_string:` You can use this field to specify a secured string to run with the check. To create a secure string for this field, as the scanning user on the target being scanned, run the following commands and copy the output into the `secure_string` field.

```
$secstr = 'clear text password' | ConvertTo-SecureString -AsPlainText -Force;
$secstr | ConvertFrom-SecureString;
```

- `check_type:` This field changes how the string is evaluated. By default, the evaluation checks that the output exactly matches the contents of `value_data`. You can change the evaluation with `CHECK_NOT_EQUAL`, `CHECK_REGEX`, or `CHECK_NOT_REGEX`.
- `only_show_cmd_output:` If you set this field to YES, the check result will be INFO/LOW and will report the value that was returned from the PowerShell. If you set this field to NO, the evaluation will be defined by other fields in the check.

Note: The comparison that the check performs is not case sensitive.

Example

```
<custom_item>
type: AUDIT_EXCHANGE
description: "Exchange - Check Type Example"
value_type: POLICY_TEXT
value_data: ".*"
powershell_args: "get-exchangeserver | fl -Property ExchangeVersion"
secure_string: "bad_value"
check_type: CHECK_REGEX
</custom_item>
```

AUDIT_PROCESS_ON_PORT

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

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

This check queries for inactive accounts by looking at the 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>
```

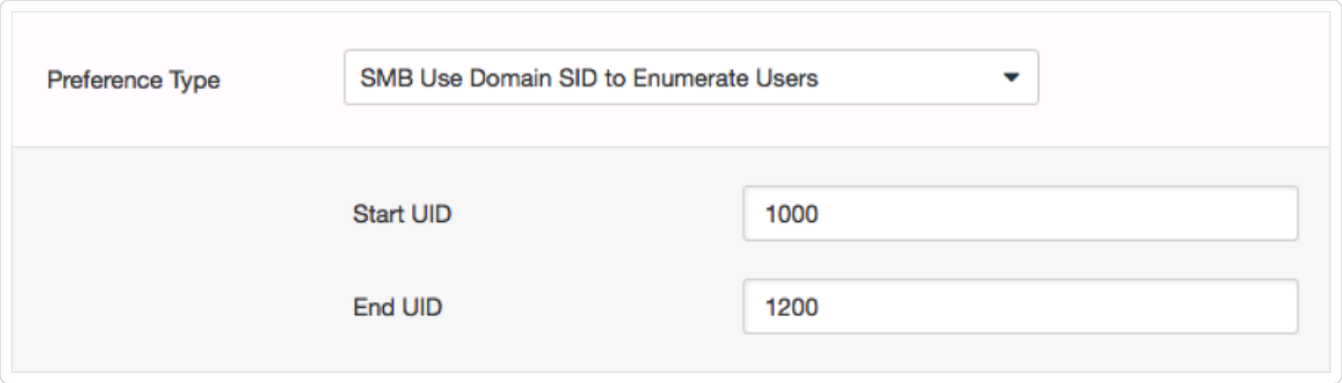
The keyword timestamp allows following values:

- `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.



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>
```

BANNER_CHECK

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

Usage

```
<custom_item>
type: BANNER_CHECK
description: ["description"]
value_type: POLICY_TEXT
value_data: ["banner content"]
reg_key: ["path to registry key"]
reg_item: ["registry item"]
is_substring: [YES|NO]
</custom_item>
```

The following are descriptions of the keywords:

- `value_type:` The value is `POLICY_TEXT`. If you define a check as `POLICY_MULTI_TEXT`, the evaluation will work, but `NULL` displays as the *Remote* value.
- `value_data`: Defines the placement of the banner. New lines are represented by adding an "\n" where the new line should be placed.

- `reg_key` and `reg_item`: The registry key and registry item are combined to identify where the registry banner is located. The most common location will be located at "`HKLM\Software\Microsoft\Windows\CurrentVersion\Policies\System`" key in the "`LegalNoticeText`" item.
- `is_substring`: An optional flag that supports the possibility of location specific information being placed in a banner. If set to `YES`, the expected banner can be a substring of the file content, and not require a full match.

Note: The comparison that the check performs is not case sensitive.

Example

```
<custom_item>
type         : BANNER_CHECK
description  : "Logon banner is configured"
value_type   : POLICY_TEXT
value_data   : "** No Unauthorized Access **"
reg_key      : "HKLM\Software\Microsoft\Windows\CurrentVersion\Policies\System"
reg_item     : "LegalNoticeText"
</custom_item>
```

CHECK_ACCOUNT

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.

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 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)`

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

Example

```
<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

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

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 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

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.

Usage

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

```
</custom_item>
```

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 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"

> **Note:** 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>
```

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

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

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>
```

The allowed type is:

`value_type: POLICY_TEXT or POLICY_MULTI_TEXT`

`value_data: "user1" && "user2" && ... && "usern"`

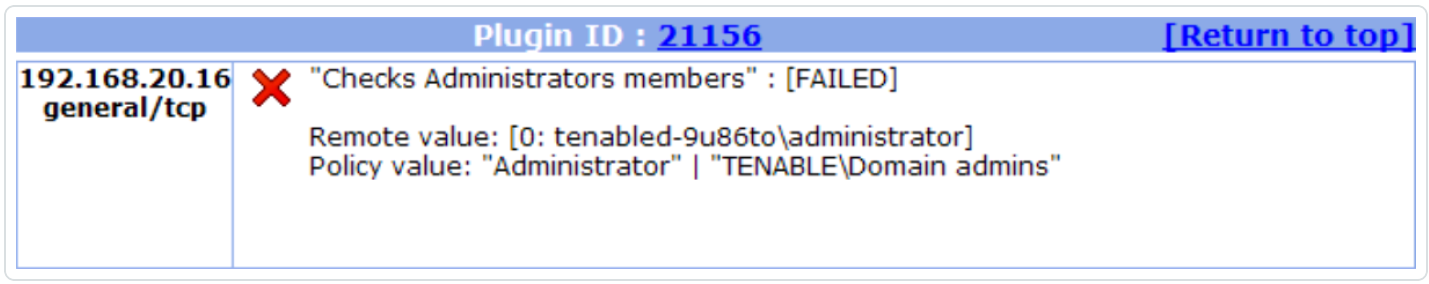
When using this audit, please note that a user name can be specified with the domain name like "MYDOMAIN\John Smith" and the **`group_name`** field specifies a single group for auditing.

Examples

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:



USER_GROUPS_POLICY

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”.

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>
```

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

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.

Usage

```
<custom_item>
type: USER_RIGHTS_POLICY
description: ["description"]
value_type: [value type]
value_data: [value]
(optional) check_type: [value]
right_type: [right]
(optional) use_domain : [YES|NO]
</custom_item>
```

Note: 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.

right_type

The `right_type` field corresponds to the right to test. Allowed values are:

`right_type: RIGHT`

Note: There must be no quotes around the `RIGHT` type as it is parsed as a token.

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

value_type

value_type: USER_RIGHT

value_data

value_data: "user1" && "user2" && "group1" && ... && "groupn"

use_domain

The use_domain option is used to add the account domain names to the output of the check.

If you set use_domain to YES, you must modify value_data to include the Windows domain the user or group is a member of.

For example, value_data: "BUILTIN\Administrators" && "NT SERVICE\WdiServiceHost"

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 policy item checks whether the file (**value_data**) exists or not (**file_option**). The check is performed by calling the function **CreateFile**.

> **Note:** 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>
```

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 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.

> **Note:** 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>
```

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 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.

> **Note:** 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>
```

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 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.

> **Note:** 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>
```

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.

Example

```
<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

> **Note:** This check requires remote registry access for the remote Windows system to function properly.

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.

Note: 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.

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>
```

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 **regex** and **expect** fields support POSIX Extended expressions.
- 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 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.

> **Note:** 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>
```

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

Note: 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>
```

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

> **Note:** This check requires remote registry access for the remote Windows system to function properly.

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.

> **Note:** 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**.

```
<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>
```

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>
```

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

> **Note:** 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>
```

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

> **Note:** 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>
```

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`** filed 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.

Example

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 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"
```

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 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.

Examples

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

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

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>
```

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

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

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>
```

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.

Example

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

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.

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>
```

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.

Examples

```
<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.

```
<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`.

```
<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>
```

This section includes the following information:

- Predefined Policies

Predefined Policies

Policy	Usage
Password Policy	name: "Enforce password history" value: POLICY_DWORD name: "Maximum password age"

Policy	Usage
	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
Account Lockout Policy	name: "Account lockout duration" value: TIME_MINUTE or name: "Account lockout duration" value: TIME_SECOND name: "Account lockout threshold" value: POLICY_DWORD name: "Reset lockout account counter after" value: TIME_MINUTE name: "Enforce user logon restrictions"

Policy	Usage
	value: POLICY_SET
Kerberos Policy	name: "Maximum lifetime for service ticket" value: TIME_MINUTE name: "Maximum lifetime for user ticket" value: TIME_HOUR name: "Maximum lifetime for user renewal ticket" value: TIME_DAY name: "Maximum tolerance for computer clock synchronization" value: TIME_MINUTE
Audit Policy	name: "Audit account logon events" value: AUDIT_SET name: "Audit account management" value: AUDIT_SET name: "Audit directory service access" value: AUDIT_SET name: "Audit logon events"

Policy	Usage
	value: AUDIT_SET name: "Audit object access" value: AUDIT_SET name: "Audit policy change" value: AUDIT_SET name: "Audit privilege use" value: AUDIT_SET name: "Audit process tracking" value: AUDIT_SET name: "Audit system events" value: AUDIT_SET
Accounts	name: "Accounts: Administrator account status" value: POLICY_SET name: "Accounts: Guest account status" value: POLICY_SET name: "Accounts: Limit local account use of blank password

Policy	Usage
	to console logon only" value: POLICY_SET name: "Accounts: Rename administrator account" value: POLICY_TEXT name: "Accounts: Rename guest account" value: POLICY_TEXT
Audit	name: "Audit: Audit the access of global system objects" value: POLICY_SET name: "Audit: Audit the use of Backup and Restore privilege" value: POLICY_SET name: "Audit: Shut down system immediately if unable to log security audits" value: POLICY_SET
DCOM	name: "DCOM: Machine Launch Restrictions in Security Descriptor Definition Language (SDDL) syntax" value: POLICY_TEXT name: "DCOM: Machine Access Restrictions in Security Descriptor Definition Language (SDDL) syntax"

Policy	Usage
	value: POLICY_TEXT
Devices	name: "Devices: Allow undock without having to log on" value: POLICY_SET name: "Devices: Allowed to format and eject removable media" value: DASD_SET name: "Devices: Prevent users from installing printer drivers" value: POLICY_SET name: "Devices: Restrict CD-ROM access to locally logged-on user only" value: POLICY_SET name: "Devices: Restrict floppy access to locally logged-on user only" value: POLICY_SET name: "Devices: Unsigned driver installation behavior" value: DRIVER_SET
Domain Controller	name: "Domain controller: Allow server operators to schedule tasks"

Policy	Usage
	value: POLICY_SET name: "Domain controller: LDAP server signing requirements" value: LDAP_SET name: "Domain controller: Refuse machine account password changes" value: POLICY_SET
Domain Member	name: "Domain member: Digitally encrypt or sign secure channel data (always)" value: POLICY_SET name: "Domain member: Digitally encrypt secure channel data (when possible)" value: POLICY_SET name: "Domain member: Digitally sign secure channel data (when possible)" value: POLICY_SET name: "Domain member: Disable machine account password changes" value: POLICY_SET

Policy	Usage
	name: "Domain member: Maximum machine account password age" value: POLICY_DAY name: "Domain member: Require strong (Windows 2000 or later) session key" value: POLICY_SET
Interactive Logon	name: "Interactive logon: Display user information when the session is locked" value: LOCKEDID_SET name: "Interactive logon: Do not display last user name" value: POLICY_SET name: "Interactive logon: Do not require CTRL+ALT+DEL" value: POLICY_SET name: "Interactive logon: Message text for users attempting to log on" value: POLICY_TEXT name: "Interactive logon: Message title for users attempting to log on" value: POLICY_TEXT

Policy	Usage
	name: "Interactive logon: Number of previous log-ons to cache (in case domain controller is not available)" value: POLICY_DWORD name: "Interactive logon: Prompt user to change password before expiration" value: POLICY_DWORD name: "Interactive logon: Require Domain Controller authentication to unlock workstation" value: POLICY_SET name: "Interactive logon: Require smart card" value: POLICY_SET name: "Interactive logon: Smart card removal behavior" value: SMARTCARD_SET
Microsoft Network Client	name: "Microsoft network client: Digitally sign communications (always)" value: POLICY_SET name: "Microsoft network client: Digitally sign communications (if server agrees)" value: POLICY_SET

Policy	Usage
	name: "Microsoft network client: Send unencrypted password to third-party SMB servers" value: POLICY_SET
Microsoft Network Server	name: "Microsoft network server: Amount of idle time required before suspending session" value: POLICY_DWORD name: "Microsoft network server: Digitally sign communications (always)" value: POLICY_SET name: "Microsoft network server: Digitally sign communications (if client agrees)" value: POLICY_SET name: "Microsoft network server: Disconnect clients when logon hours expire" value: POLICY_SET
Network Access	name: "Network access: Allow anonymous SID/Name translation" value: POLICY_SET name: "Network access: Do not allow anonymous enumeration of SAM accounts"

Policy	Usage
	value: POLICY_SET name: "Network access: Do not allow anonymous enumeration of SAM accounts and shares" value: POLICY_SET name: "Network access: Do not allow storage of credentials or .NET Passports for network authentication" value: POLICY_SET name: "Network access: Let Everyone permissions apply to anonymous users" value: POLICY_SET name: "Network access: Named Pipes that can be accessed anonymously" value: POLICY_MULTI_TEXT name: "Network access: Remotely accessible registry paths and sub-paths" value: POLICY_MULTI_TEXT name: "Network access: Remotely accessible registry paths" value: POLICY_MULTI_TEXT

Policy	Usage
	name: "Network access: Restrict anonymous access to Named Pipes and Shares" value: POLICY_SET name: "Network access: Shares that can be accessed anonymously" value: POLICY_MULTI_TEXT name: "Network access: Sharing and security model for local accounts" value: LOCALACCOUNT_SET
Network Security	name: "Network security: Do not store LAN Manager hash value on next password change" value: POLICY_SET name: "Network security: Force log off when logon hours expire" value: POLICY_SET name: "Network security: LAN Manager authentication level" value: LANMAN_SET name: "Network security: LDAP client signing requirements"

Policy	Usage
	value: LDAPCLIENT_SET name: "Network security: Minimum session security for NTLM SSP based (including secure RPC) clients" value: NTLMSSP_SET name: "Network security: Minimum session security for NTLM SSP based (including secure RPC) servers" value: NTLMSSP_SET
Recovery Console	name: "Recovery console: Allow automatic administrative logon" value: POLICY_SET name: "Recovery console: Allow floppy copy and access to all drives and all folders" value: POLICY_SET
Shutdown	name: "Shutdown: Allow system to be shut down without having to log on" value: POLICY_SET name: "Shutdown: Clear virtual memory pagefile" value: POLICY_SET
System Cryptography	name: "System cryptography: Force strong key protection for user keys stored on the computer" value: CRYPTO_SET

Policy	Usage
	name: "System cryptography: Use FIPS compliant algorithms for encryption, hashing, and signing" value: POLICY_SET
System Objects	name: "System objects: Default owner for objects created by members of the Administrators group" value: OBJECT_SET name: "System objects: Require case insensitivity for non-Windows subsystems" value: POLICY_SET name: "System objects: Strengthen default permissions of internal system objects (e.g. Symbolic Links)" value: POLICY_SET
System Settings	name: "System settings: Optional subsystems" value: POLICY_MULTI_TEXT name: "System settings: Use Certificate Rules on Windows Executables for Software Restriction Policies" value: POLICY_SET
Event Log	name: "Maximum application log size" value: POLICY_KBYTE

Policy	Usage
	name: "Maximum security log size" value: POLICY_KBYTE name: "Maximum system log size" value: POLICY_KBYTE name: "Prevent local guests group from accessing application log" value: POLICY_SET name: "Prevent local guests group from accessing security log" value: POLICY_SET name: "Prevent local guests group from accessing system log" value: POLICY_SET name: "Retain application log" value: POLICY_DAY name: "Retain security log" value: POLICY_DAY

Policy	Usage
	name: "Retain system log" value: POLICY_DAY name: "Retention method for application log" value: EVENT_METHOD name: "Retention method for security log" value: EVENT_METHOD name: "Retention method for system log" value: EVENT_METHOD

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 File Content Global Settings

This section describes global settings for Windows File Contents audit files. If you have multiple audit files in the scan or policy, the global setting applies to all Unix audit files. If you want to use different settings for different audits, create a separate scan or policy for each one.

To access global settings in Tenable Nessus or Tenable Vulnerability Management:

1. In a scan or policy, open the **Compliance** tab.
2. In the **Categories** drop-down box, select **Windows File Contents**.

   Tenable Nessus shows a list of Windows File Contents audit files, or you can upload a custom Windows File Contents audit file.
3. Select any Windows File Contents audit file to add to the scan or policy.
4. Edit the **Global Settings**.

Performance Options

This setting controls the method that the plugin uses to enumerate the files on the target host.

- **__WMI File Enumeration__** – Uses a combination of WMI and SMB to query the directories and enumerate the files on the target. Tends to have higher network activity and lower target utilization.
- **__Powershell File Enumeration__** – Uses Powershell command-lets to query the directories and enumerate the files on the target. Tends to have lower network activity and higher target utilization.
- **__Powershell File Enumeration with local cache__** – Uses Powershell command-lets to query the directories and enumerate the files on the target, with a local tempoary file is used to keep track of directores to be evaluated. Tends to have lower network activity and higher target utilization with small temporary storage file.

The default value for this setting is to use **__WMI File Enumeration__**.

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.

This section includes the following information:

- Check Type
- Item Format
- Windows Content Command Line Examples
- Performance Considerations

Check Type

All Windows content compliance checks must be bracketed with the **`check_type`** encapsulation and the "**`WindowsFiles`**" designation. This is very similar to all other **`.audit`** files. The basic format of a content check file is as follows:

```
<check_type: "WindowsFiles">
<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
`type`	This must always be set to FILE_CONTENT_CHECK
`description`	This keyword provides the ability to add a brief description of the check that is being performed. It is strongly recommended that the **`description`** field be unique and no distinct checks have the same description field. Tenable uses this field to automatically generate a unique plugin ID number based on the description field.
`file_extension`	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 regex and expect are not included in the audit, files with the file_extension specified are displayed in the audit output.
`regex`	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. **Note:** The **`regex`** keyword must be run with the expect keyword described below.

Keyword	Description
	Note: Unlike Windows Compliance Checks, Windows File Content Compliance Check **regex** and **expect** do not have to match the same data string(s) within the searched file. Windows File Content checks simply require that both the **regex** and **expect** statements match data within the <max_size> bytes of the file searched.
expect	The **expect** 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. Multiple patterns are listed in quotes and separated with pipe characters. Simple pattern matching is also supported in this keyword with the period. When matching the string “C.T”, the **expect** statement would match “CAT”, “CaT”, “COT”, “C T” and so on. **Note:** The **expect** keyword may be run standalone for single pattern matching, however, if the **regex** keyword is used, **expect** is required. **Note:** Unlike Windows Compliance Checks, Windows File Content Compliance Check **regex** and **expect** do not have to match the same data string(s) within the searched file. Windows File Content checks simply require that both the **regex** and **expect** statements match data within the <max_size> bytes of the file searched.
file_name	Whereas the **file_extension** 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. 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”.
max_size	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. Only values up to 5M will be

Keyword	Description
	honored and any files found over 5M will be skipped in the resulting scan.
`only_show`	This keyword supports revealing a specific number of characters specified by policy. When matching sensitive data such as credit card numbers, your organization may require that only a limited number of digits be made visible in the report. The default is 4 or half of the matched string, whichever is smaller. For example, if a matched string is 10 characters long and `only_show` is set to 4, only the last 4 characters are shown. If the matched string is 6 characters long, only 3 characters will be shown. **Note:** When you match against US Social Security numbers (SSNs), the specified number of digits are revealed *in front* of the string (for example, `123-XX-XXXX`).
`regex_replace`	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.
`include_paths`	This keyword allows for directory or drive inclusion within the search results. This keyword may be used in conjunction with, or independently of the "**`exclude_paths`**" 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. You can only specify the top-level directory of a drive (for example, `E:\<top-level directory>`). Using more than one directory level (for example, `E:\<top-level directory>\<directory>`) returns an error. **Note:** Only drive letters or folder names can be specified with the "**`include_paths`**" keyword. File names cannot be included in the "**`include_paths`**" value string.
`exclude_paths`	This keyword allows for drive, directory or file exclusion from search

Keyword	Description
	results. This keyword may be used either in conjunction with, or independently of the "**include_paths**" 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.
see_also	This keyword allows to include links to a reference. Example: see_also: "https://benchmarks.cisecurity.org/tools2/linux/CIS_Redhat_Linux_5_Benchmark_v2.0.0.pdf"
solution	This keyword provides a way to include "Solution" text if available. Example: solution : "Remove this file if it is not required"
reference	This keyword provides a way to include cross-references in the .audit. The format is "ref|ref-id1,ref|ref-id2". Example: reference : "CAT|CAT II,800-53|IA-5,8500.2|IAIA-1,8500.2|IAIA-2,8500.2|IATS-1,8500.2|IATS-2"

Windows Content Command Line Examples

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

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

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

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

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

This section includes the following information:

- Target Test File
- Search Examples

Target Test File

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

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

Take this data and copy it to any Windows system you have credentialed access to. Name the file “Tenable_Content.tns”.

Search Examples

The following examples describe how to search for specific `.tns` and `.doc` documents.

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:"WindowsFiles">
<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:"WindowsFiles">
<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:"WindowsFiles">
<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:"WindowsFiles">
<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:"WindowsFiles">
<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:"WindowsFiles">
<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    (XXXXXXX7890)
```

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:"WindowsFiles">
<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:"WindowsFiles">
<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:"WindowsFiles">
<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:"WindowsFiles">
<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:"WindowsFiles">
<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:"WindowsFiles">
<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 “\.t” 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:`**).

Performance Considerations

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

Start small and build up

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

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

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

Which extensions should we search for?

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

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

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

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

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

How much data should be scanned?

If a `max_size` is not specified, Tenable Nessus will attempt to retrieve entire files. Since these files traverse the network, there is more network traffic with these audits than with typical scanning or configuration auditing. By specifying a `max_size` in the audit files, the amount of data transferred over the network can be limited.

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

How many policies are being evaluated?

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

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

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

Can an agent be used?

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

Additional Information

This section contains the following resources:

- All Compliance and Audit Files
- Conditional Auto Else and Rollup
- Credentialed Scanning and Privileged Account Use
- XSL Transform to .audit Conversion

All Compliance and Audit Files

To see the full list of compliance and audit files, see the Tenable Audits Page.

Compliance Data Export Plugins

This document describes plugins you can use to format compliance results into data formats that both Tenable and third-party tools can use for integrations.

Available Formats

- Gold Image Audit
- JSON
- XCCDF

Compliance Export Gold Image

- Plugin ID: 174791
- Plugin Name: Compliance Export Gold Image Audit

This plugin creates a Gold Image Audit file of compliance scanning results.

The Gold Image Audit

The Gold Image Audit uses a `known_good` feature in the Tenable Audit Language syntax that allows a prior known good value to evaluate as a compliance result, even if the original audit would have failed the check.

Some things to consider:

- Actual values tend not to be applied to checks that are within the conditional sections of audit if/then/else structures. This was done to maintain consistency with the conditional logic.
- Any checks that use custom commands, such as `CMD_EXEC` or `SQL_POLICY`, may have inconsistent or unsupported output. SQL statements may return values in different orders if not sorted or system commands may output Unicode characters in their output. If you are developing checks, consider sorting the output where possible into a consistent order and sanitizing any non-ASCII characters from the output.
- Checks that use static report items to post a `WARNING` or `FAILED` are not modified by `known_good` values.

Interpreting the Results

The theory behind the gold image scanning is that if you scan a target that is considered a “gold image," run this script with that audit and results, and rescan the target with the new audit, every item should be a PASSED/info result. But there are a number of factors that do not allow 100% passes on most audits.

Some of the factors include:

- Some checks in the audit are a direct report of a certain result, most notably with WARNING/medium. If the audit item is a report WARNING, it cannot change in the gold image audit. If the audit is a Tenable-published audit, these items can be identified by having a “NOTE:” in the description that the check was not run. The code inside the audit has an opening tag similar to `<report type:"WARNING">`.
- Audits can contain conditional logic that provides results based on a setting on the target being scanned. When rescanning the “gold image” system with a gold image audit, the same conditional logic works, and the results become a PASSED/info. But, when scanning other hosts, they may take a different conditional path that can provide results that were not present in the original “gold image” results.
- If the output of an audit check includes dynamic data, such as timestamps, a known good value of a gold image does not work. Since the value of a time stamp changes with every execution of the scan, matches against the static known good fail.

- When creating a gold image audit, the original range or regular expression is abandoned and an absolute value is used in its place. An example would be a benchmark that accepted a password length of greater than 8 characters; the “gold image” had 7 characters set. The new gold image audit fails on anything that is not exactly 7 characters. You can adjust this by adding more known good values, but the end result is always an audit looking for absolute values.

Custom Audit Content

When creating a custom audit for use as a gold image, use the following tips to get quality results:

- Make the audit relatively flat and use few to no conditionals. Conditionals may change the results presented on different targets.
- Use items that provide computed results and not static reports. Static reports never change results.
- Create results that do not contain time-based output. If a time stamp shows in the results, it is impossible to create a known good value.
- If creating a custom audit item using a command (`CMD_EXEC, AUDIT_POWERSHELL`), make sure the output is consistently generated. This means that all outputs should be sorted. If the output comes out in a random order, the gold image does not have a consistent known good to compare with.

How To Enable Gold Image Audit

By default, Gold Image audit for **Policy Configuration Auditing** is disabled. To enable the Gold Image audit feature.

1. Select the **Policy Compliance Auditing** scan template.
2. From the **Settings** tab, select **Advanced** and then **Custom** from the **Scan Type** drop-down menu.
3. Select **General**.
4. Under **Compliance Output Settings**, select the checkbox next to **Generate Gold Image Audit file**.

The option is also available for any scan template that allows the selection of audit files and generation of audit results, including the **Advanced Scan** template.

How to Access Gold Image Audit

The data is provided as attached files to the plugin output. The attached files are named as `gold_<host_address>_<audit_file>.json`. Each audit used generates a different results file. To retrieve the data, navigate to the scan vulnerabilities, find the **Compliance Export Gold Image Audit** plugin, and find the file attachments. Click to download the attachments.

Notes:

- Enabling this plugin increases the storage and size of your scan results.
- The plugin only runs once per audit per scan.
- If known good values exist in an audit used to create the Gold Image Audit, no new values are added.

Compliance Export JSON Results

Plugin ID: 174790

Plugin Name: Compliance Export JSON

This plugin creates a JSON-formatted data file of compliance scanning results. The JSON data produced by this plugin represents the same results that would be exported using the ".nessus" XML data format.

The JSON Format

The JSON file contains properties for the following items:

- **audit** - Information on the audit file that is used.
- **host** - Information on the target that is being audited.
- **results** - The compliance results.
- **scan** - Information on the scanner in use and the start and stop times of the scan.

Each result contains the following data:

- **check_name** - The name of the recommendation that was audited.
- **result** - The result that is posted to the Tenable product ("PASSED," "WARNING," "FAILED," or "ERROR").
- **actual_value** - The value that the audit check produced.
- **policy_value** - A representation to identify the policy used in checking the recommendation.
- **audit_file** - The audit file that is in use, as reported by the scanner.
- **benchmark_name** - The benchmark name identified by the audit.
- **benchmark_version** - Then benchmark version identified by the audit.
- **see_also** - Link to source benchmark guidance.

Additional data points may be included based on what audit file is in use.

How To Enable JSON Audit Results

By default, JSON results for **Policy Configuration Auditing** is disabled. To enable the JSON results feature:

1. Select the **Policy Compliance Auditing** scan template.
2. From the **Settings** tab, select **Advanced** and then **Custom** from the **Scan Type** drop-down menu.
3. Select **General**.
4. Under **Compliance Output Settings**, select the checkbox next to **Generate JSON result file**.

The option is also available for any scan template that allows the selection of audit files and generation of audit results, including the **Advanced Scan** template.

How to Access JSON Audit Results

The data is provided as attached files to the plugin output. The attached files are named as `results_<host_address>_<audit_file>.json`. Each audit used generates a different results file.

To retrieve the data, navigate to the scan vulnerabilities, find the **Compliance Export JSON** plugin, and find the file attachments. Click to download the attachments.

Notes:

- The JSON results may contain additional results that were used in conditional evaluation of audit checks.
- Enabling this plugin increases the storage and size of your scan results.

Compliance Export XCCDF Results

- Plugin ID: 174792
- Plugin Name: Compliance Export XCCDF

A feature available within Tenable Policy Compliance Auditing is the capability of downloading XCCDF results after an audit scan has completed. This feature has always been capable within SCAP and OVAL Auditing but is now available within our Policy Compliance Auditing. The feature is only available with use of our DISA STIG audits.

XCCDF Standard

XCCDF (Extensible Configuration Checklist Description Format) is a standards component that is found within the SCAP (Security Content Automation Protocol) standards family. The XCCDF standard is a language that is used to describe security checklists. The XCCDF standard provides a standardized reporting format for expressing and storing results. The XCCDF XML results file can contain information such as target details, the results of each security check, the XCCDF default score and more. You can import these results with other tools, such as the DISA STIG viewer, where the results can be viewed and modified based on the user's preferences.

How To Enable XCCDF Audit Results

By default, XCCDF results for **Policy Configuration Auditing** is disabled (enabled by default with SCAP scanning). To enable the XCCDF results feature:

1. Select the **Policy Compliance Auditing** scan template.
2. From the **Settings** tab, select **Advanced** and then **Custom** from the **Scan Type** drop-down menu.
3. Select **General**.

4. Under **Compliance Output Settings**, select the checkbox next to **Generate XCCDF result file**.

The option is also available for any scan template that allows the selection of audit files and generation of audit results, including the **Advanced Scan** template.

How to Access XCCDF Audit Results

The data is provided as attached files to the plugin output. The attached files are named as `xccdf_<host_address>_<audit_file>.json`. Each audit used generates a different results file.

To retrieve the data, navigate to the scan vulnerabilities, find the **Compliance Export XCCDF** plugin, and find the file attachments. Click to download the attachments.

Notes:

- This capability is only available when scanning using a DISA STIG audit.
- Enabling this plugin increases the storage and size of your scan results.

Conditional Auto Else and Rollup

Auto Else

Audit files use condition tags to assess an if/then/else logical evaluation. See the following example:

```
If
      "Service installed/enabled"
then
      "Check for configuration"
else
      "Report that the service isn't installed, this check doesn't apply."
```

If you omit the `else` section and the conditional fails, nothing is reported. As a result, many cases of content duplication occur in published audit content. This has traditionally been required to achieve full transparency and parity with industry guidance.

The following is an example of content duplication that might be found in a Unix configuration audit:

```
<if>
```

```
      <condition type:"AND">
            <custom_item>
                  system      : "Linux"
                  type        : CMD_EXEC
                  description : "NTP is installed"
                  cmd         : "/bin/systemctl is-enabled ntp"
                  expect      : "enabled"
            </custom_item>
      </condition>
      <then>
            <report type:"PASSED">
                  description : "Ensure time synchronization is in use"
                  info        : "Time should be synchronized"
            </report>
      </then>
      <else>
            <report type:"FAILED">
                  description : "Ensure time synchronization is in use"
                  info        : "Time should be synchronized"
            </report>
      </else>
</if>
```

The `auto else` functionality eliminates this need to duplicate content by automatically generating `else` content from the checks or reports provided in the `then` section of a conditional. See the following usage example:

```
<if>
      <condition type:"AND" auto:"FAILED">
            <custom_item>
                  system      : "Linux"
                  type        : CMD_EXEC
                  description : "NTP is installed"
                  cmd         : "/bin/systemctl is-enabled ntp"
                  expect      : "enabled"
            </custom_item>
      </condition>
      <then>
            <report type:"PASSED">
                  description : "Ensure time synchronization is in use"
```

```
                info        : "Time should be synchronized"
            </report>
      </then>
</if>
```

The `auto` attribute in the condition tag has a status of `FAILED`. If the conditional check for NTP fails, the report inside the `<then>` section converts to `FAILED`. This allows you to de-duplicate content and reduce the complexity of an audit.

The `auto` attribute accepts `FAILED`, `PASSED`, and `WARNING`. These are the same status results as existing audit checks. Both `AND` and `OR` condition types support `auto else` functionality.

Rollup

It is common within industry guidance, such as CIS benchmarks and DISA STIGs, to evaluate a single recommendation with multiple tests. Traditionally, to achieve parity with industry guidance, Tenable's published audit files duplicate this recommendation content once per test and add a unique modifier to the description for each duplication. This modifier is used to show that a check is aligned with a specific recommendation, but is still a separate test. While this works from a functional standpoint, it can cause policy-related issues when you try to align assessed recommendations with a benchmark checklist or other external tools.

Consider the following example from the CIS Ubuntu 20.04 audit (some fields removed for brevity):

```
<custom_item>
      system      : "Linux"
      type        : CMD_EXEC
      description : "1.1.1.1 Ensure mounting of cramfs filesystems is disabled -
modprobe"
      cmd         : "/sbin/modprobe -n -v cramfs | /bin/grep -E '(cramfs|install)'"
      expect      : "install /bin/(true|false)"
</custom_item>
<custom_item>
      system      : "Linux"
      type        : CMD_EXEC
      description : "1.1.1.1 Ensure mounting of cramfs filesystems is disabled - lsmod"
      cmd         : "/sbin/lsmod | /bin/grep cramfs | /usr/bin/awk \'{print} END {if
(NR == 0) print \"pass\"; else print \"fail\"}\'"
      expect      : "pass"
```

```
</custom_item>
```

In this example, the two items are separate tests, but they relate to the same benchmark recommendation. The modifiers in this case are the description tags: `- modprobe` and `- lsmod`.

To improve achieving parity with industry guidance, conditionals can now return the output of multiple conditional tests in a single report.

Taking the previous example, you can combine these items to return a single report using a conditional:

```
<if>
      <condition type:"AND">
            <custom_item>
                  system      : "Linux"
                  type        : CMD_EXEC
                  description : "modprobe"
                  cmd         : "/sbin/modprobe -n -v cramfs | /bin/grep -E '
(cramfs|install)'"
                  expect      : "install /bin/(true|false)"
            </custom_item>
            <custom_item>
                  system      : "Linux"
                  type        : CMD_EXEC
                  description : "lsmod"
                  cmd         : "/sbin/lsmod | /bin/grep cramfs | /usr/bin/awk \'
{print} END {if (NR == 0) print \"pass\"; else print \"fail\"}\'"
                  expect      : "pass"
            </custom_item>
      </condition>
      <then>
            <report type:"PASSED">
                  description : "1.1.1.1 Ensure mounting of cramfs filesystems is
disabled"
                  show_output : YES
            </report>
      </then>
      <else>
            <report type:"FAILED">
```

```
                  description : "1.1.1.1 Ensure mounting of cramfs filesystems is
disabled"
                  show_output : YES
            </report>
      </else>
</if>
```

The show_output tag within a report gathers the returned values from the checks inside the conditional section and shows them in a report's output :

```
"1.1.1.1 Ensure mounting of cramfs filesystems is disabled"  : [PASSED]

Policy Value:
PASSED
Actual Value:
All of the following must pass to satisfy this requirement:

-------------------------
PASSED - modprobe
Output of the command

-------------------------
PASSED - lsmod
Output of the command
```

This rollup functionality supports both AND and OR condition types. AND shows a message that "All" of the following must pass, and OR shows that "Any" of the following must pass.

Combining Both Features

You can combine these features or use them independently. The following is an example of combining the auto attribute and the show_output tag:

```
<if>
      <condition type:"AND" auto:"FAILED">
            <custom_item>
                  system      : "Linux"
                  type        : CMD_EXEC
                  description : "modprobe"
```

```
                  cmd         : "/sbin/modprobe -n -v cramfs | /bin/grep -E '
(cramfs|install)'"
                  expect      : "install /bin/(true|false)"
            </custom_item>
            <custom_item>
                  system      : "Linux"
                  type        : CMD_EXEC
                  description : "lsmod"
                  cmd         : "/sbin/lsmod | /bin/grep cramfs | /usr/bin/awk \'
{print} END {if (NR == 0) print \"pass\"; else print \"fail\"}\'"
                  expect      : "pass"
            </custom_item>
      </condition>
      <then>
            <report type:"PASSED">
                  description : "1.1.1.1 Ensure mounting of cramfs filesystems is
disabled"
                  show_output : YES
            </report>
      </then>
</if>
```

The result of this conditional is evaluated, and if both items pass (`AND` condition), a `PASSED` report returns containing the results of the conditional evaluation. Otherwise, an automatic `else` with a `FAILED` result returns with the results of the conditional evaluation.

XSL Transform to .audit Conversion

Several compliance check plugins rely on auditing XML content, such as Palo Alto, VMware, and Unix compliance checks. To take advantage of these capabilities, it is beneficial to become familiar with creating XSL Transforms. In some cases, building an XSL Transform requires 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 topic provides proper guidance on how to build and utilize custom XSL Transforms, and convert them into **`.audit`** files.

Several audit checks (for example, 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, you can follow these steps to generate custom **`.audit`** files for XML content:

1. Install xsltproc
2. Identify the XML File to Use
3. Become Familiar with XSL Transforms and XPath
4. Create the XSLT Transform
5. Verify the XSLT Transform Works
6. Copy the XSLT to the .audit
7. Final Audit