ivanti

linuxac

Ivanti Application Control for Linux

Utilization Guide

Version 2021.3

Copyright Notice

Contents

Utilization Guide - Ivanti Application Control for Linux

This guide documents how to configure and maintain the Ivanti Application Control for Linux system. It summarizes the configuration settings available via the web console, and the logic of how polices and rules are applied. The document also includes information that may help you debug or troubleshoot across the component parts of your system.

The content is intended for system administrators.

Capabilities

This section highlights what the system can do - or what it is capable of achieving when configured.

Zero-Day Protection

Once installed and running on your Linux endpoints, except for base system functionality allowed via Policy Defaults, the Ivanti Application Control for Linux engine automatically denies all execution operations of binaries, shared objects, scripts, or commands.

Local Whitelisting

After start up, the Application Control for Linux engine scans the local RPM database and automatically whitelists all the packages found based on their contents. It then exposes their trust chain and contents within the web console to enable easier inspection.

Allow/Deny Rules

The default working mode of the Application Control for Linux system is to deny execution operations unless they are permitted via Policy Defaults, locally whitelisted, or explicitly allowed.

Allowing and denying specific paths and/or binaries is done via creating policies inside the Application Control for Linux web console, establishing their rules, and finally selecting a list of registered devices or groups of devices to be protected, and deploying them.

Audit vs. Restrict

A policy in the Application Control for Linux system can be setup into either Audit or Restrict modes:

Audit Mode:

- Instructs the Application Control for Linux engine to monitor all operations on the protected endpoint and to report back to the administrator what would have been the enforcement results based on the rule set of the policy deployed on it.

  This mode enables administrators to audit the possible effects of a policy's rule set and to monitor the actions of users. Review of audit data provides valuable feedback when preparing to implement a new policy.

Restrict Mode:

- Instructs the Application Control for Linux engine to monitor all operations on the protected endpoint, enforce the rule set contained in the deployed policy (this includes policy defaults and local whitelisting), and to report back to the administrator all the operations monitored, and the enforcements applied.

Utilization

This section describes the characteristics that can be configured to influence system capabilities.

Zero-Day Protection

The Policy Defaults rule set is listed below. Its purpose is to ensure base system functionality is allowed. Note, this file comprises paths only; it does not contain binaries. The first path specified is Application Control for Linux

```
/opt/arrow/ac/
/usr/bin/
/usr/libexec/
/usr/lib/xorg/
/usr/lib/gdm3/
/etc/gdm3/
/usr/sbin/
/usr/lib/
/usr/lib/debug/
/usr/lib/debug/lib/
/usr/lib/debug/lib/libc6-prof/x86_64-linux-gnu/
/usr/lib/debug/lib/libc6-prof/
/usr/lib/debug/lib/x86_64-linux-gnu/
/usr/lib/i386-linux-gnu/
/usr/lib/x86_64-linux-gnu/
/sbin/
/etc/
/proc
/var
/etc/init/
/boot/grub2/
/boot/efi/EFI/redhat/
/boot/loader/
/etc/default/
/proc/
/boot/
/boot/efi/
/boot/efi/EFI/
/usr/lib64/
/usr/libexec/
/usr/libexec/sssd/
/usr/libexec/hypervkvpd/
/usr/lib/systemd/
/usr/lib/gdm3/
```

Local Whitelisting

Due to RPM database API technical limitations (https://github.com/rpm-software-management/rpm/issues/1124), the Application Control for Linux engine cannot account for changes occurred in the RPM database while it is running.
Note: If packages are installed or updated on the Linux endpoint, you are advised to restart the engine's daemon. This action is required to keep the engine in sync with the refreshed version of the installed packages in the RPM database.

Allow/Deny Rules

Hashing for binaries

The check box **Use Hashing** helps inform the Application Control for Linux engine that the target binary should be allowed or denied via its signature rather than its location. This means that if the target binary is copied into another location, the enforcement on it cannot be circumvented.

There is no practical method for the system to predict if the specified target is a path or a binary - or whether it exists on all the Linux endpoints where the policy containing the rule might be deployed. The decision to allow or deny is made locally on each endpoint by each Application Control for Linux engine. The system will not prevent the administrator from selecting the Use Hashing option for a path - but if wrongly set, hashing option is ignored.

Whilst it is possible to check the *Use Hashing* check box in the web console for any target – it is up to the administrator to set it correctly. Hashing can only be applied by the engine for existing binaries.

Enabling or disabling individual rules:

Rules can be enabled or disabled at an individual level. This allows administrators to test settings and configurations before finalizing policies.

Audit vs. Restrict

Viewing results of deployed policies:

The effects of a deployed policy, in either Audit or Restrict mode, can be observed in the Audit Log or Debug Info tabs available on each device in the Devices page.

Note, operation identifiers are different for each mode:

- Audit mode - Allowed or Denied
- Restrict mode - Permitted or Blocked

Use the Search, Filter, and Order functions available within each tab to inspect the results of interest.

Root User Exclusion

The root user is excluded from all the enforcements performed by the Application Control for Linux engine.

Deployment - Additional Notes

The section provides additional user guidance relating to policy deployment actions.

Saving without Deploying

A Linux endpoint can have only one policy deployed/active at any given moment. The active policy includes, by default, the latest policy defaults and local whitelisting specified.

- While working on the structure of your policy you can perform intermediate Save operations to ensure that your work is not lost, and you can continue it later before deciding to deploy.
- Any new policy deployed successfully on an endpoint replaces the previous one as the active policy.
- Saved policies can be edited further until they are deployed (and become active), as opposed to deployed/active policies which can only be viewed or duplicated after deployment.

Deployments on Groups

Device Groups are a way of organizing the devices registered in your system into logical groups that make sense to your business. Grouping allows for easier handling and maintenance.

- A device cannot be present multiple times or included in different logical groups.
- Groups cannot be further nested or otherwise classified.
- When adding a Device Group as a deployment target into a policy, the system will add a list of all the devices in it, ensuring they are uniquely inserted into the policy.
- Within the web console, when creating or editing Device Groups, the system displays only the groups permitted for insertion (groups that contain any devices that you have previously added are not listed as available for selection).

Managing Deployments

Following successful deployment, policies can only be viewed, duplicated, or removed.
Note that removing a deployed policy will not remove it or inactivate it from the endpoints where it was active – such a policy can only be replaced later via a new deployment.

To create a new version of an existing, successfully deployed policy, you must use the *Duplicate* action to create a new draft, and then *Edit* the new draft.

- Duplicate will copy the policy's main properties, the complete list of rules, and the complete list of added devices.
- Manully including a version number in the name of your policy is recommended.

Administration

Effectiveness of your Ivanti Application Control for Linux system requires the administration of policies, devices and rules tailored to meet the requirements of your organization. Administration is performed using the web console and this section of the documentation describes the concepts and functionality available.

Web Console

Home Tab

The Home tab is the entry point to the web console. It acts as a dashboard, displaying the most relevant data from your system. The Home tab includes links to further resources such as Help, Support, Community Articles and Ivanti-specific trainings.

Policies Tab

The Policies tab is where you create new policies and manage existing ones.

All Device Policies Section

The All Device Policies section lists all policies, displaying their deployment, contents, current status and execution mode.

- Select the ellipses icon menu to access relevant functions. The menu includes options to view deployed/active policies, duplicate existing ones, edit draft policies, and remove policies.
- Policies can be edited or removed only until they are deployed. Once successfully deployed, the policy becomes active.
- Deployed (active) policies can only be viewed or duplicated; they cannot be edited or removed.

New Policy Section

Select New Policy to access the section where you create policies, compose a rule sets, set up the list of devices where they the policy will be deployed and establish the mode (Audit or Restrict) in which the new policy will behave on the targeted Linux endpoints.

On an individual rule you can specify the targets required. This includes full or relative paths or binaries, and if execution should be allowed or denied.

If required, you can use environment variables and symbolic links. To view examples, refer to the "Rules System" on page 16 section.

For binaries, the Use Hashing option allows you to ensure the target is identified by signature rather than location. This means even if the target is copied into another location, the enforcement of the rule cannot be circumvented. Further information on the use of hashing is available here.

The list of rules allows you to filter rules and further inspect them. Functions allow you to edit previously existing rules, modify their type (Allowed or Denied), enable, or disable the use of hashing, enable, or disable a rule entirely, or remove it from the list.

Add Devices Tab

The Add Devices tab enables you to specify where you want your policy to be deployed and activated. You can add individual devices or groups of devices. For further information refer to Deployments on Groups.

The list of Added Devices [check name] indicates if a device belongs to any group, and if a policy is currently active for that device. You can filter the list, and/or remove individual or multiple devices.

Notes

- Until a policy is deployed it can be edited. You are advised to periodically save your changes (for further information refer to Save Without Deploying.
- In the Deploy Policy tab you can see a summary of your policy, and save or deploy the created rules to the devices specified.

Devices Tab

The Devices tab lists devices that are registered to your server instance.

In the list of devices you can view whether the device belongs to a group, whether they have an active policy, and the deployment progress and/or status of that policy.

The list can be filtered for ease of navigation and administration.

To view device-specific details select the required device name from those listed. Device details are displayed in four tabs:

Device Summary

The Device Summary summarizes relevant hardware and software information. The information is extracted from the Linux distribution on the device at the point of registration .

Debug Info Tab

Debug Info enables you to review detailed debug information.
The data is supplied from the Application Control for Linux engine running on that device. The data can be filtered for ease of navigation and further inspection. It can be valuable in troubleshooting the current state of your endpoint.

See also Audit vs Restrict for further information.

Audit Log Tab

Audit logs detail Application Control-specific events as they are audited and reported from the device's Application Control for Linux engine. The audit log events are based upon the operations executed. The operations could be executed by the system, various systems services, individual applications, or by users.

Events are audited whether the deployed policy mode is Audit or Restrict. They apply to the currently active policy (including policy defaults and local whitelisting).

Audit log data can be filtered for ease of navigation and inspection and is valuable in auditing the current behavior of your endpoint - whether allowed and denied. For further information, refer to er to Audit vs Restrict.

Device Contents Tab

Device Contents displays the applications and packages installed on the Linux endpoint. This includes a visual structure of all binaries present and their locations, and a list of installed packages, their contents, and their security characteristics such as hash values and signature keys.

Device Groups

The Device Groups tab allows you to organize your registered devices into logical groups.

The groups are listed and the number of devices included currently is displayed. To remove a device group select the ellipses icon to open the context menu, then select Remove.

The list can be filtered for ease of navigation and administration.

Create a Group Section

The Create a Group section allows you to organize your devices into a new logical group.

Add Devices

Devices can only belong to one group at a time.

The add devices search returns only relevant registered devices, note that this will exclude any device that is currently a group member.

Note: If a device belonging to one group is added to a new group its original group membership is removed.

Devices (list)

The list of devices allows you to filter, and then further inspect added devices. The list displays current group membership for the devices and their currently active policies.

Group member devices can be selected individually or in multiples and removed using the Remove button.

Rules System

This section describes the concept of how Ivanti Application Control for Linux uses and applies rule sets. It summarizes how to create rules, the logic used to apply those rules, and includes a range of specific examples.

Concepts

The Policy Rule System in Application Control for Linux is based on three fundamental concepts and a few functional details.

- The concepts we are using are called target identification, operator precedence and hierarchical specialization.
- The functional details are denied by default for zero-day protection, policy defaults and local whitelisting, and root user exclusion.

  Understanding the concepts will help you grasp how to correctly compose your rule system to obtain the expected results.

Target Identification:

- Rules apply concretely only on execution operations as final targets (binaries, shared objects, scripts, or commands).
    - We will often use the terms binary or executing binary to represent these throughout the rest of the documentation.
- From a user's perspective, a target is your first input into a rule, specifying where or exactly on what do you want the rule to apply.
- You can specify a location (relative or full path) as a target, meaning that the rule will apply on all execution operations started under or by the current user in that location.
- You can specify a binary (via a relative or full path to it) as a target, meaning that you set the rule to apply exactly on the execution operation identified, started under or by the current user.
- Also, as a user, besides specifying a target as your first input into a rule, you can also set the hashing option (only for binary targets, see here for a few details on how this works), telling the system that it should compute a hashing value for the final target residing at the specified location, and use that further to globally identify the final target, instead of the original location.
- In the end, the system can identify a final target in two ways, by location or by signature, making the identification and rule enforcement local or global.

Operator Precedence:

- There are three operators considered in the application control environment: allow, deny, and hashing.
- Allow and deny are mandatory Boolean operators, hashing is an optional, conditional operator.
- Allow and deny set the enforcement type (permit or block), while hashing changes the enforcement's coverage (from local to global).
- Allow specifies that a certain final target (binary, shared objects, script, or command) should be allowed to execute under or by the current user, at its current location (specified or inherited) or by its signature (globally).
- Deny specifies that a certain final target (binary, shared object, script, or command) should be denied execution under or by the current user, at its current location (specified or inherited) or by its signature (globally).

- As said earlier, in the target identification section, Hashing specifies that a certain final target (binary, shared object, script, or command) should be identified by its signature instead of its location, and globally consider further the enforcement of its execution, under or by the current user.
- Given the details above, the operator precedence is as follows:
    - Initially, the hashing is considered, first the allows, then the denies.
    - Second, the allow operator is considered, by final target location.
    - Third, the deny operator is considered, by final target location.
    - Note: although the hashing operator is optional, it is considered first otherwise its globality would not be satisfied.
    - Note: since this is a deny by default system, allow are considered second, otherwise the whole system would remain blocked.
- Operator Precedence TL/DR:
    1. Allows with Hashing (a.k.a. allows by signature)
    2. Denies with Hashing (a.k.a. denies by signature)
    3. Allows without Hashing (a.k.a. allows by location)
    4. Denies without Hashing (a.k.a. denies by location)

Hierarchical Specialization:

- It represents the way a rule for a final target is decided in the rules system.
- Note: initially, all final targets will be localized (found by location), so the system can know where and who they are, and then identified, either by location or, if hashing is enabled, by signature.

  When deciding a rule for a final target, the system will perform these steps:
    - compute the hash of the executed binary and search the internal rules hashes table for allows or denies (that's the first reason it always needs to know where it is, to ensure the creation of that table).

- if the previous step doesn't yield a rule, it will search for a non-hashing allow/deny rule for the exact location of the executed binary.
    - This is called the most specialized rule for the final target.
- if the previous step doesn't yield a rule either, the system will start reducing the path to the executed binary backwards, by removing the right-most element (first the binary itself, then the previous directory, and so on, up to the root directory a.k.a. "/") and try to find a rule in the rules list that applies to that reduced location. If this search yields, that resulting rule will be applied to the initial executing binary.
    - This is called the closest specialized rule for the final target.
- while performing the previous two steps the system will also consider the policy defaults and the local whitelisting tables in establishing the correct definitive rule for an executing binary.
- in the end, if no rule can be found for the final target, its execution will be denied by default.

  Knowing the functional details will allow you to get a better overview on the behavior of the application control system as a whole and its capabilities.

Please revisit the Capabilities and Utilization sections for a reminder on these topics.

Examples

Simple rules for a binary

Simple rules for a binary called "myApp", located in "/home/Desktop/":

Setup:

- Rule 1: Allow – "/home/user/Desktop/myApp"
- Rule 2: Deny – "/home/user/Desktop/myApp"

Result:

Allow execution of "/home/user/Desktop/myApp"

Logic:

Allow by location takes precedence to deny by location (the level of specialization here is the same for both rules).

Simple rules for a binary - unoptimized and optimized

Same simple rules for a binary called "myApp", located in "/home/Desktop/", with an optimization suggestion:

Unoptimized example

Setup:

- Rule 1: Deny – "/home/user/Desktop/myApp"
- Rule 2: Allow – "/home/user/Desktop/myApp"

Result:

Allow execution of "/home/user/Desktop/myApp"

Logic:

Allow by location takes precedence to deny by location (the level of specialization here is the same for both rules).
Deny is analyzed, as it's specified first, and the system must consider it, not knowing what will follow.

Optimized example:

Setup:

- Rule 1: Allow – "/home/user/Desktop/myApp"
- Rule 2: Deny – "/home/user/Desktop/myApp"

Result:

Allow execution of "/home/user/Desktop/myApp"

Logic:

Allow by location takes precedence to deny by location (the level of specialization here is the same for both rules).
Deny is ignored, the result is immediate, because allows have precedence to denies.

Rules for a binary using full path and hashing

Simple rules for a binary called "myApp", located in "/home/Desktop/", by location (full path) and by signature (hashing):

Setup:

- Rule 1: Allow – "/home/user/Desktop/myApp"
- Rule 2: Deny – "/home/user/Desktop/myApp" – Use hashing is checked

Result:

Deny execution of "/home/user/Desktop/myApp"

Logic:

Deny by hashing takes precedence to allow by location (the level of specialization here is the same for both rules, but the globality differs).

Rules for a binary and parent location

Simple rules for a binary called "myApp", and its parent location "/home/user/Desktop/":

Setup:

- Rule 1: Deny – "/home/user/Desktop/"
- Rule 2: Allow – "/home/user/Desktop/myApp"

Result:

Allow execution of "/home/user/Desktop/myApp"

Logic:

Allow set by the most specialized rule on the executing binary takes precedence to the deny for its parent location.

Rules for a binary - use of syntax

Simple rules for a location, "/home/user/Desktop/", with a minor difference in syntax, but lexically identical:

Setup:

- Rule 1: Allow – "/home/user/Desktop"
- Rule 2: Deny – "/home/user/Desktop/"

Result:

Allow execution of anything in "/home/user/Desktop/"

Logic:

Allow by location takes precedence to deny by location (the level of specialization here will be the same for both rules).
While searching for the most specialized rule by location, the system knows and accounts for the fact that the last "/" is not changing it, so both syntaxes are treated as the same location.

Complex rules for a binary - example 1

More complex scenario, with rules for a binary called "myApp", located in "/home/user/Desktop/", its parent location, the "home" directory, and the system binary "ls":

Setup:

- Rule 1: Allow – "/home/user/Desktop/"
- Rule 2: Deny – "/home/"
- Rule 3: Deny – "/home/user/Desktop/myApp"
- Rule 4: Deny – "/usr/bin/ls" – Use hashing is checked

Result:

Deny execution of anything in "/home/", except allowing execution of anything in "/home/user/Desktop/", but again deny the execution of "/home/user/Desktop/myApp", and finally deny the execution of "ls" globally, not only from "/usr/bin/" (e.g., even if "ls" would be copied in "/home/user/Desktop/").

Logic:

Deny for "ls" is global, by hashing, deny set by the most specialized rule on the executing binary for "myApp" takes precedence to the allow for its parent location, then anything else in its parent location "/home/user/Desktop/" would be allowed, taking precedence by specialization to the deny rule for its parent folder "/home/", and finally, the deny on "/home/" is enforced.

Rules to block an app allowed by its parent location

Another complex scenario, with rules to block an app, "/usr/bin/ls", implicitly allowed by the policy default rules by allowing its parent location "/usr/bin/":

Default:

Rule x: Allow – "/usr/bin/"

Note, the default exists for both setups presented below:

Setup 1:

- Rule 1: Allow – "/home/user/Desktop/"
- Rule 2: Deny – "/usr/bin/ls"

Result 1:

Allow execution of anything in "/usr/bin/", also allow execution of anything in "/home/user/Desktop/" but deny the execution of "/usr/bin/ls", locally, only if executed from "/usr/bin/" (e.g., if "ls" would be copied in "/home/user/Desktop/" it would be allowed to execute).

Logic 1:

Deny for "ls" is local but set by the most specialized rule on the executing binary, so it takes precedence to the allow in policy defaults for its parent location, "/usr/bin/".

Allow for "/home/user/Desktop/" will allow execution of any binary in it, unless explicitly denied, so since the deny for "ls" is not global, by hashing, and since there is no local deny for a copy of "ls" in "/home/user/Desktop/", if that copy would exist, it would be allowed to execute.

Remember, the default is still there for this setup:

Setup 2:

- Rule 1: Allow – "/home/user/Desktop/"
- Rule 2: Deny – "/usr/bin/ls" – Use hashing is checked

Result 2:

Allow execution of anything in "/usr/bin/", also allow execution of anything in "/home/user/Desktop/" but deny the execution of "/usr/bin/ls", globally, not only from "/usr/bin/" (e.g., even if "ls" would be copied in "/home/user/Desktop/").

Logic 2:

Deny for "ls" is global, set by hashing, and by the most specialized rule on the executing binary, so it takes full precedence to the allow in policy defaults for its parent location, "/usr/bin/", and to the allow for "/home/user/Desktop/", so if a copy of it were to exist there, it would be denied execution.

Environment variable examples

A few examples of expanding Environment Variables and how they are treated:

Case 1 – simple or composed:

Setup:

- $OTHER = "/home/Desktop/Folder/"
- $ENV = "/home/usr/$OTHER"

Result:

→ "/home/usr/home/Desktop/Folder/"

Logic:

will expand, composing the sub-paths, and removing extra "/".

Case 2 – undefined variable:

Setup:

$UNDEFINED – is not defined on the system

Result:

→ N/A

Logic:

will not expand and the rule will be ignored for security reasons.

Case 3 – garbage variable

Setup:

$GARBAGE = "#847e08" – is defined, but unusable

Result:

→ N/A

Logic:

will not expand and the rule will be ignored for security reasons.

Case 4 – not found variable:

Setup:

$NOTFOUND = "/home/<none>/" – is defined, but path is not found

Result:

→ N/A

Logic:

will not expand and the rule will be ignored for security reasons.

Case 5 – empty variable:

Setup:

- $EMPTY = "" or "/" – is defined, but has no impact
- $OTHER = "/home/Desktop/Folder/$EMPTY"

Result:

→ "/home/Desktop/Folder/"

Logic:

will expand, composing the sub-paths, and removing extra "/".

Case 6 – semantic identity:

Setup:

- Rule 1: Deny – "/home/$VAR", where $VAR = "/user/Desktop/"
- Rule 2: Allow - "/home/user/Desktop/"

Result:

Allow execution of anything in "/home/user/Desktop/"

Logic:

Allow by location takes precedence to deny by location.
Rule 1's path will expand, compose sub-paths, and remove extra "/".

Maintenance

This section provides information and guidance that can help you maintain and debug your system.

Logging & Debugging

Please find below details on the logging locations and their debugging purposes for your entire experience with the AC for Linux system, from initial installation to deploying policies in production.

Windows Server

Installation

The installation of the AC for Linux Master Installer is logged in "C:\ProgramData\Ivanti\ACServer\", in files named similar to "master_20211028113915.txt",
where "20211028113915" is the installation time-stamp, and the AC for Linux sub-installer for the AC & AF Servers is logged also in "C:\ProgramData\Ivanti\ACServer\", in files named similar to "master_20211028113915_002_AFServerHostSetup.msi.txt" where "20211028113915" is the installation timestamp and the "002" is the current rotation number.

Utilization

The communication part of the AC for Linux solution, via the AF Server, is activelly logged as follows:

- C:\ProgramData\Ivanti\ACServer contains runtime logging information, detailing the execution of the AF Server instance.

Linux Endpoints

AC Agent Logs

Location

Current log files are located here: "/opt/ivanti/ac/logs/".

Rotation

Older log files are rotated here: "/opt/ivanti/ac/logs/old".

Older log files are moved to this location and new versions are generated in the main one. Versioning adds an "<underscore_number>" to the log file name.

Log Files

"acengd_0.log"

Purpose
Agent-Engine supervisory log, detailing the management of the Engine by the Agent.

Example
In connection error scenarios, when the Engine doesn't appear to receive any data, the Agent will be aware of and report the issues.

"stagentctl_0.log"

Purpose
Agent registration and control log, detailing the registration of the Agent with the Windows Backend, including secured communication status.

Example
During registration, SSL_HANDSHAKE error scenarios can appear, and they be reported here.

"stagentd_0.log"

Purpose
Agent daemon main log, detailing the secured communication events between the Linux Endpoint and the Windows Backend.

Example
At runtime, any communication errors will be reported here (the Agent handles and secures all communication between the AC Engine and the Windows Backend).

"stagentlistener_0.log"

Purpose
Agent daemon listener log, detailing the status of commands and package exchanges via all communication channels (HTTPS, MQTT).

"stagentupdater_0.log"

Purpose
Agent daemon updater log, detailing the status of updates and configuration arrivals from the Windows Backend infrastructure.

“stmqttservice_0.log”

Purpose
Agent MQTT communication log, detailing all the communication via MQTT between the Linux Endpoint and the Windows Backend.

“sttelemetryreporter_0.log”

Purpose
Agent daemon telemetry log, providing detailed telemetry for all the operations executed by the agent.

Example
At runtime, this is the best starting point for debugging all types of business logic and communication error scenarios.

AC Engine Logs

Location

Current log files are located here: "/opt/ivanti/ac/engines/ivanti-ac-engine-<distro_name>/logs/".
Logs are located per your running distribution, so <distro_name> can be: "centos-8", or "oracle-8", or "redhat-8", quotes excluded.

Rotation

Older log files are rotated here: "/opt/ivanti/ac/engines/ivanti-ac-engine-<distro_name>/logs/old".

Logs are also rotated per your running distribution, so <distro_name> can be: "centos-8", or "oracle-8", or "redhat-8", quotes excluded.

Older log files are archived and moved to this location and new files are generated in the main one. Rotation changes the file name as follows: “<log_file_name><underscore_number><.log.gzip>”.

Log Files

“acengd.log”

Purpose

Engine general log, detailing all the telemetry and debug-mode information for the engine at runtime, with maximum verbosity.

Example

Any engine business logic event (such as local whitelist caching and decision engine policy handling scenarios), can be inspected and debugged with the help of this log file.