paloalto
TECHDOCS

AI Model Security

docs.paloaltonetworks.com

Contact Information

Corporate Headquarters:
Palo Alto Networks
3000 Tannery Way
Santa Clara, CA 95054
www.paloaltonetworks.com/company/contact-support

About the Documentation

- For the most recent version of this guide or for access to related documentation, visit the Technical Documentation portal docs.paloaltonetworks.com.
- To search for a specific topic, go to our search page docs.paloaltonetworks.com/search.html.
- Have feedback or questions for us? Leave a comment on any page in the portal, or write to us at documentation@paloaltonetworks.com.

Copyright

Last Revised

October 24, 2025

Table of Contents

Secure Your AI Models with AI Model Security

Where Can I Use This?	What Do I Need?
• Prisma AIRS (AI Model Security)	☐ Prisma AIRS AI Model Security License

What is a Model?

Model is the collection of files required to perform a single base inference pass. Model has the following structure:

- **Source**—Where the AI model lives.
- **Model**—Logical entity (like "sentiment-analyzer").
- **ModelVersion**—Specific AI model version (like "v1.2.3").
- **Files**—Artifacts of that AI model version.

Models are the foundational asset of AI/ML workloads and already power many of your key systems in use today. AI model security focuses on securing your models against threats like:

- **Deserialization Threats**: Protecting your models from executing malicious and unknown code at load time.
- **Neural Backdoors**: Detecting Manchurian Candidate like models.
- **Runtime Threats**: Protecting your models from executing malicious and unknown code at inference time.
- **Invalid Licenses**: Ensuring that your models are not using invalid licenses.
- **Insecure Formats**: Ensuring that your models use formats that help prevent threats.

What is AI Model Security?

AI Model Security is an enterprise application designed to enforce comprehensive security standards for both internal and external machine learning models deployed in production environments. The application addresses a critical gap in organizational security practices where machine learning models, despite their significant impact on business operations, often lack the rigorous security validation that is standard for other data inputs and systems.

In most enterprise environments, traditional data inputs such as PDF files undergo extensive security scrutiny before processing, yet machine learning models that drive critical business decisions frequently bypass equivalent security measures. This disparity creates substantial operational risk, as compromised or inadequately validated models can impact business logic, data integrity, and decision-making processes. AI Model Security solves this problem by providing comprehensive model validation through automated security assessments, multi-source support for both internally developed and third-party models, and proactive identification and remediation of model-related security vulnerabilities.

By implementing AI Model Security, organizations can establish consistent security standards across all ML model deployments, significantly reducing operational risk from unvalidated or compromised models. It enables secure adoption of third-party ML solutions while maintaining organizational security rules and industry compliance standards. Additionally, it provides comprehensive audit trails and compliance reporting capabilities, ensuring that AI Model Security assessments meet regulatory requirements and internal governance standards.

Impact of model vulnerabilities

Models can execute arbitrary code and existing tooling is not checking that for you. Models have been found at the root of cloud take over attacks, and can be used to exfiltrate data, or even to execute ransomware attacks. The sensitivity of the data that models are trained on and exposed to at inference time makes them a prime target for attackers.

AI Model Security Core Components

AI Model Security enables you to have flexible controls to secure, validate, and manage AI models across different sources through *Security Groups*, *Sources*, *Rules*, and *Scans*.

AI Model Security delivers a comprehensive framework to establish and enforce security standards for AI models across your organization. Unlike traditional security tools that simply scan for malware, AI Model Security recognizes that AI models require more nuanced security considerations that incorporate license validation, file format verification, and context-specific security checks based on the teams and environments using the models.

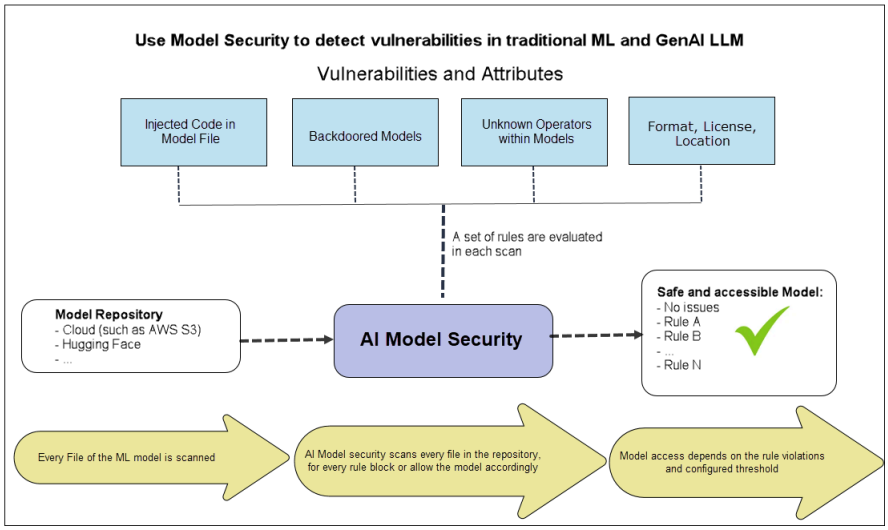
The AI Model Security approach moves beyond the simplified first-party versus third-party model distinction to provide granular security controls that scale with enterprise needs. This approach centers around four key components: *Security Groups*, *Sources*, *Rules*, and *Scans*.

Entity	Description	Examples
Security Groups	Serve as the foundation of your AI Model Security posture, allowing you to combine specific rules and requirements for models from a particular source.	• HuggingFace-Research • S3-Production • Partner-S3-Audit
Source	Each Security Group is assigned to a specific *Source*, which represents where model artifacts reside, such as HuggingFace for external models or Local Storage and Object Storage for internal models. The source designation is crucial as it provides metadata that powers specific security rules applicable to models from that source.	• Hugging Face • S3 • Local Disk
Rules	Within each Security Group, you configure *Rules* that define the specific evaluations performed on models. Rules can verify proper licensing, check for approved file formats, scan for malicious code, and detect architectural backdoors. Each Rule can be enabled or disabled and configured as blocking or non-blocking, giving you precise control over which security issues prevent model usage versus those that simply generate warnings.	• License Existence Check • Serialization Format Safety • Author Verification • Malicious Backdoor Detection
Scan	When models are evaluated against these Rules, a *Scan* is performed, documenting the verdict across all rules. These Scans create an audit trail of security evaluations and serve as decision points to either promote secure models forward or block potentially threatening ones early in your workflow. Here's what a typical scan will look like:	Scan of fraud-detector:v2.1.0 using S3-Production group

Entity	Description	Examples
	Scan of ModelVersion "sentiment-model-v1.2.3": ├── License Compliance Rule (metadata, artifact-level) → PASS ├── Source Verification Rule (metadata, artifact-level) → PASS ├── Serialization Safety Rule (security, file-level: model.pkl) → FAIL (Critical) ├── Serialization Safety Rule (security, file-level: tokenizer.pkl) → PASS └── Aggregate Result: FAIL (Critical severity exceeds threshold)	

AI Model Security leverages rules to help organizations establish sophisticated, scalable security frameworks tailored to their specific requirements. This flexible approach enables teams to enforce strict blocking mechanisms for high-severity threats while maintaining non-disruptive alerting for compliance monitoring—allowing security teams to effectively manage risk without hindering developer productivity. The result delivers dual benefits: end users gain confident access to vetted models through a seamless experience, while security teams receive comprehensive protection for their AI/ML infrastructure.

To implement AI Model Security effectively, you'll typically need at least two Security Groups: one for external models using HuggingFace as a Source, and another for internal models using Local or Object Storage Sources. This separation allows you to apply appropriate security standards based on the origin and intended use of models across your organization.



Get Started with AI Model Security

Where Can I Use This?	What Do I Need?
• Prisma AIRS (AI Model Security)	❑ Prisma AIRS AI Model Security License

AI Model Security Workflow

AI Model Security operates on a two-tier hierarchical structure where each security group encompasses one or more security rules.

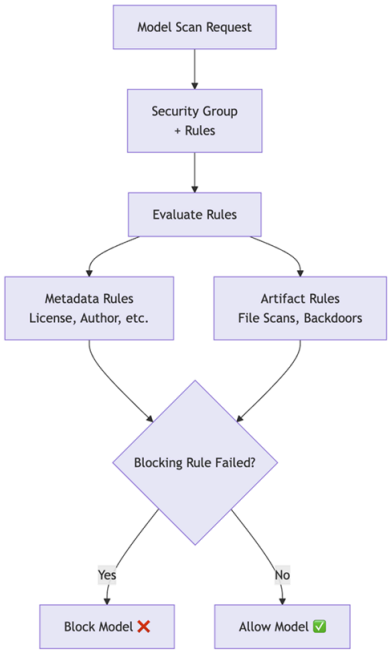
1. **Source Type Binding**: Model Security Groups are initially created and associated with a specific source type (such as, S3 buckets).
2. **Rule Configuration**: Populate these groups with relevant rules. AI Model Security provides intelligent suggestions and validation based on the selected source type, including default rule collections tailored to each source.

   For instance, a "Verify Author" rule wouldn't be available for S3-based groups since S3 doesn't maintain author metadata.

   Rules operate in either blocking or non-Blocking modes, streamlining the previous severity-based threshold system from AI Model Security.
3. **Scanning Process**: During model evaluation, the scan request specifies the applicable model security group to establish context. The system then processes two rule categories:
   - **Metadata Rules**: Validate model metadata from the source platform
   - **Artifact Rules**: Conduct comprehensive analysis of model files
4. **Result Processing**: Each rule produces a binary PASS or FAIL outcome. The final scan verdict aggregates all rule results and determines whether to apply blocking enforcement (for critical issues) or non-Blocking responses (for logging and alerts).

   The AI Model Security delivers structured flexibility—the same model can undergo evaluation against multiple Model Security Groups (such as separate development and production

configurations), each with distinct Source Types and rule sets, enabling context-appropriate security rules.



Create a Deployment Profile for Prisma AIRS AI Model Security

Where Can I Use This?	What Do I Need?
• Prisma AIRS	☐ Software NGFW Credits

Use this information to deploy a tenant for AI Model Security. Consider the following when creating a deployment profile:

- AI Model Security is licensed via NGFW credits.
- The deployment profile is accessible to all SCM Pro users.
- The deployment profile is accessible on the Customer Support Portal.
- For this early access release, the deployment profile name is **AI Model Security (preview)**.

You need an active deployment profile and have a proper Identity & Access to view the AI Model Security and its features in Prisma AIRS.

Creating a deployment profile for AI Model Security includes the following steps:

1. Create the deployment profile.
2. Associate the deployment profile with a tenant.

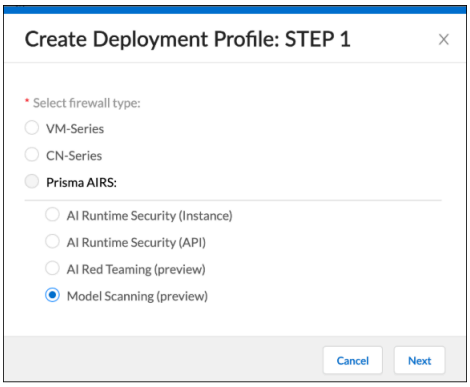
Create a Deployment Profile for Prisma AIRS AI Model Security in the Customer Support Portal

STEP 1 | Log in to the Palo Alto Customer Support Portal (CSP).

STEP 2 | Navigate to **Products** > **Software/Cloud NGFW Credits**.

STEP 3 | Locate your credit pool and click **Create Deployment Profile**.

STEP 4 | Under **Select firewall type**, expand **Prisma AIRS** and select **AI Model Security (preview)**.



STEP 5 | Select **Next**.

STEP 6 | Enter a **Profile Name**.

*Click **Calculate Estimated Cost** to view the credits used for AI Model Security.*

STEP 7 | Click **Create Deployment Profile**.

This takes you to the **Software NGFW Credits** page in the **Customer Support Portal**. The page displays the list of deployment profiles within the credit pool for the selected account.

After creating the deployment profile, you'll associated it with a tenant.

Associate the Deployment Profile with a Tenant

Use the information in this section to associate a deployment profile with a tenant. Consider the following:

- Once the deployment profile is created, you're provided with a link to the Palo Alto Networks Hub to associate the profile with a TSG.
- You can create a new tenant (TSG), or, you can associate the profile with an existing tenant.
- If a new tenant is created, a Strata Cloud Manager (SCM) instance is created; this process can take approximately 15-20 minutes to complete.
- If you are using an existing tenant, a SCM instance is typically associated with it.
- Because AI Model Security is accessible for all SCM Pro users, you'll see SCM and Strata Logging Service (SLS) instances being created; these elements are part of the SCM Pro license.

To associate the deployment profile with a tenant:

STEP 1 | In the CSP dashboard, locate your deployment profile and click **Finish Setup**; this redirects you to the Palo Alto Networks hub.

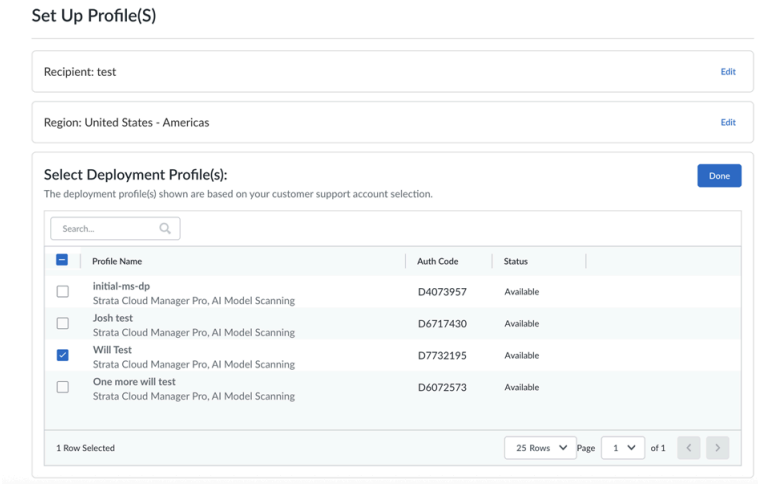
STEP 2 | Select the name of the CSP account in which you created the deployment profile.

If you select a new tenant, also select the region where the tenant should be created. This is referred to as the platform region. AI Model Security determines which regions are supported.

*Only **Region: United States - Americas** is currently supported.*

STEP 3 | Select the deployment profile that you want to associate with the tenant.

STEP 4 | Select **None** in **Additional Services**.



STEP 5 | Agree to the terms, then click **Activate**.

*Deployment Profile activation may take up to two hours. Once activated, the AI model Security will be visible in the Strata Cloud Manager web interface (**Insights** > **Prisma AIRS** > **Model Security**).*

You're redirected to the Tenant Management page which shows the instances for the tenant.

STEP 6 | Verify the TSG association in the Hub. To do this:

1. Navigate to **Common Services** > **Tenant Management**.
2. Select the tenant and switch to **Deployment Profiles**.
3. Confirm that the **Profile Association Status** is **Complete**.

Edit or Update a Deployment Profile

You can edit an existing deployment profile on the CSP as long as AI Model Security supports your updates to the deployment profile. These changes may include changing the number of scans allocated per month.

STEP 1 | Log in to the Palo Alto Customer Support Portal (CSP).

STEP 2 | In the CSP dashboard, click on the three dots (ellipsis or overflow menu) next to your deployment profile.

STEP 3 | Edit, delete or deactivate the deployment profile.

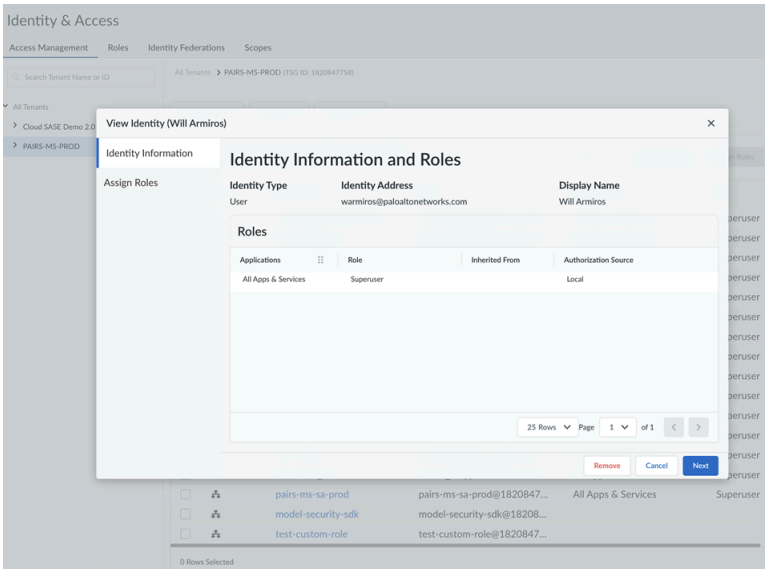
Configure Identity and Access Management

Where Can I Use This?	What Do I Need?
• Prisma AIRS (AI Model Security)	❑ Prisma AIRS AI Model Security License

If you have an active deployment profile but lack IAM permissions for AI Model Security, it may result in user authentication error. Hence it is important to have an active deployment profile and IAM permission to access AI Model Security.

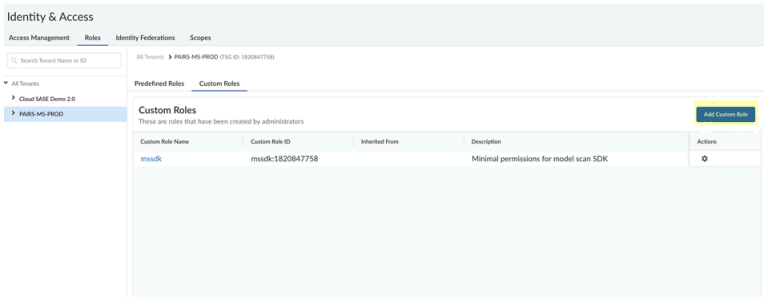
STEP 1 | Navigate to Strata Cloud Manager Identity and Access Management (IAM) settings, **Common Services** > **Identity & Access**.

STEP 2 | Navigate to **Identity & Access** > **Access Management** and select the tenant from the tenants list. Verify the **Identity Information** for the selected tenant to ensure that the **Role** assigned is either **Superuser** for all apps and services, or a custom role with access to AI Model Security.

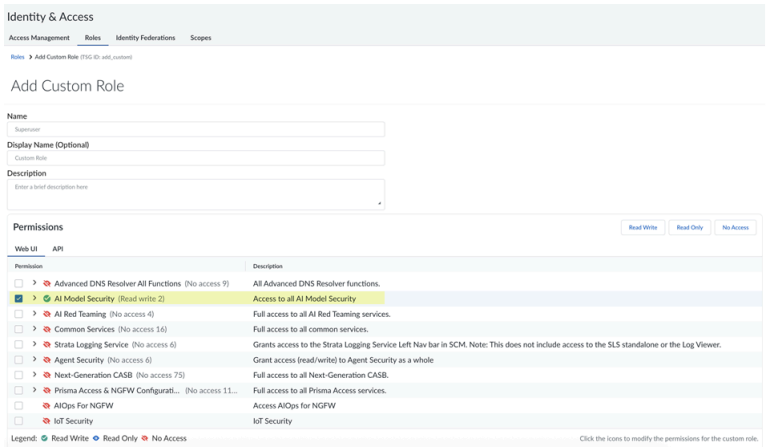


STEP 3 | (Optional) (Administrator Only) Create a custom role.

1. To create a new custom role and assign it to their identity, select **Roles** and then **Custom Roles**.

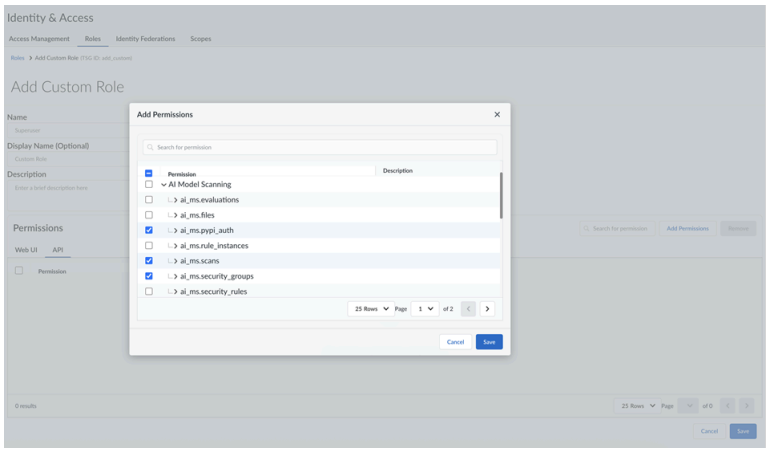


2. Enable **AI Model Security**.

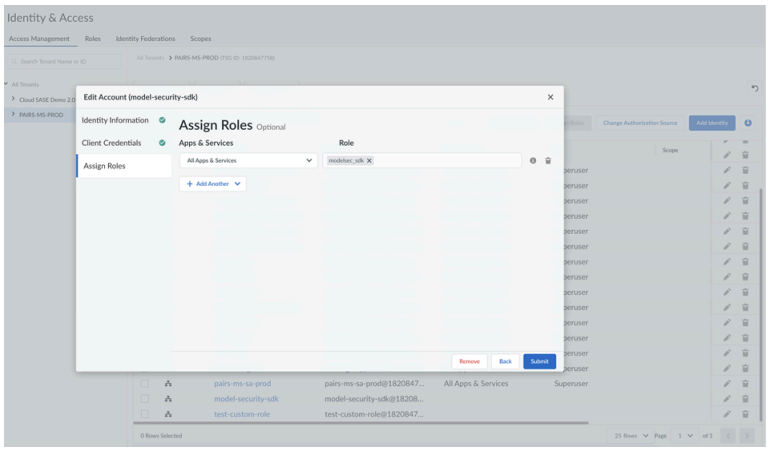


3. If you need a service account with API access, select **API** and **Add permissions** and assign the minimum permissions.

*To use SDK, you need at least the following permissions: **ai_ms_pypi_auth**, **ai_ms.scans**, and **ai_ms.security_groups**.*



4. Enter the **Name** and **Description** for the custom role and **Save** the changes.
5. Select **Assign Roles** and verify if the custom role that you created appears in the **Apps & Services** list.



Install AI Model Security

Where Can I Use This?	What Do I Need?
• Prisma AIRS (AI Model Security)	☐ Prisma AIRS AI Model Security License

To scan both internal and external models, you require either AI Model Security CLI or SDK. AI Model Security is available as a Python package that offers both a command-line interface and a Python SDK. Install the package using your preferred Python package manager.

STEP 1 | Generate the `pip` index link.

Copy the script below and save it to your local environment (alternatively, you can create your own script using this as a reference).

```
#!/bin/bash
#
# Model Security Private PyPI Authentication Script
# Authenticates with SCM and retrieves PyPI repository URL
#

set -euo pipefail

# Check required environment variables
: "${MODEL_SECURITY_CLIENT_ID:?Error: MODEL_SECURITY_CLIENT_ID not
 set}"
: "${MODEL_SECURITY_CLIENT_SECRET:?Error:
 MODEL_SECURITY_CLIENT_SECRET not set}"
: "${TSG_ID:?Error: TSG_ID not set}"

# Set default endpoints
API_ENDPOINT="${MODEL_SECURITY_API_ENDPOINT:-https://
api.sase.paloaltonetworks.com/aims}"
TOKEN_ENDPOINT="${MODEL_SECURITY_TOKEN_ENDPOINT:-https://
auth.apps.paloaltonetworks.com/oauth2/access_token}"

# Get SCM access token
```



```
TOKEN_RESPONSE=$(curl -sf -X POST "$TOKEN_ENDPOINT" \
    -H "Content-Type: application/x-www-form-urlencoded" \
    -u "$MODEL_SECURITY_CLIENT_ID:$MODEL_SECURITY_CLIENT_SECRET" \
    -d "grant_type=client_credentials&scope=tsg_id:$TSG_ID") || {
    echo "Error: Failed to obtain SCM access token" >&2
    exit 1
}

SCM_TOKEN=$(echo "$TOKEN_RESPONSE" | jq -r '.access_token')
if [[ -z "$SCM_TOKEN" || "$SCM_TOKEN" == "null" ]]; then
    echo "Error: Failed to extract access token from response" >&2
    exit 1
fi

# Get PyPI URL
PYPI_RESPONSE=$(curl -sf -X GET "$API_ENDPOINT/mgmt/v1/pypi/
authenticate" \
    -H "Authorization: Bearer $SCM_TOKEN") || {
    echo "Error: Failed to retrieve PyPI URL" >&2
    exit 1
}

PYPI_URL=$(echo "$PYPI_RESPONSE" | jq -r '.url')
if [[ -z "$PYPI_URL" || "$PYPI_URL" == "null" ]]; then
    echo "Error: Failed to extract PyPI URL from response" >&2
    exit 1
fi

echo "$PYPI_URL"
```

STEP 2 | Set up authentication using environment variables.

After placing the script in an executable location, you'll need to set several environment variables before running it. Both the AI Model Security CLI and SDK require authentication credentials set as environment variables. The client automatically manages OAuth2 authentication with the provided credentials.

```
export MODEL_SECURITY_CLIENT_ID=<your-client-id>
export MODEL_SECURITY_CLIENT_SECRET=<your-client-secret>
export TSG_ID=<your-tsg-id>
```

STEP 3 | Install AI Model Security package (both SDK and CLI) with `uv` or `pip`.

1. Install AI Model Security package (both SDK and CLI) using `uv`, or.

   ```
   uv add model-security-client --index $(/path/to/script.sh)
   ```

2. Install AI Model Security package (both SDK and CLI) using `pip`.

   ```
   pip install model-security-client \
     --extra-index-url <URL from Script>
   ```


STEP 4 | Initialize the AI Model Security Python SDK.

To use the Python SDK in your code, import and initialize the AI Model Security client.

```
from uuid import UUID
from model_security_client.api import ModelSecurityAPIClient

# Initialize the client
client = ModelSecurityAPIClient(
    base_url="https://api.sase.paloaltonetworks.com/aims"
)
```

The AI Model Security client uses the same environment variables for authentication as the CLI.

Default Security Groups and Rules

Where Can I Use This?	What Do I Need?
• Prisma AIRS (Model Security)	☐ Prisma AIRS Model Security License

AI Model Security leverages Security Groups to enable you to establish and configure your Model Security controls. Security Groups have a one-to-one relationship with Model Sources and this association is permanent once set.

Each Model Source provides specialized metadata that is used in rules and security validation processes.

You can create a dedicated Security Group for each of our supported Model Sources:

- Local Storage
- HuggingFace
- Azure Blob Storage
- Google Cloud Storage
- Amazon S3

Following are the default security group names:

Security Group Name	Model Source
Default LOCAL	Local Storage
Default HUGGING_FACE	HuggingFace
Default AZURE	Azure Blob Storage
Default GCS	Google Cloud Storage
Default S3	Amazon S3

Scanning Models

Where Can I Use This?	What Do I Need?
• Prisma AIRS (AI Model Security)	❑ Prisma AIRS AI Model Security License

Once your Security Group is configured, you can scan models through either the CLI or Python SDK. The process varies slightly depending on whether you're scanning HuggingFace models or local models.

Scan a HuggingFace Model

To scan a model hosted on HuggingFace, provide the model URI and your Security Group UUID.

Scan using CLI

```
model-security scan \
  --security-group-uuid "12345678-1234-1234-1234-123456789012" \
  --model-uri "https://huggingface.co/microsoft/DialoGPT-medium"
```

Scan using Python SDK

```
from uuid import UUID

result = client.scan(
    security_group_uuid=UUID("12345678-1234-1234-1234-123456789012"),
    model_uri="https://huggingface.co/microsoft/DialoGPT-medium"
)

print(f"Scan completed: {result.eval_outcome}")
```

The AI Model Security automatically fetches the latest version from HuggingFace. To scan a specific version, include the version parameter.

Scan using CLI

```
model-security scan \
  --security-group-uuid "12345678-1234-1234-1234-123456789012" \
  --model-uri "https://huggingface.co/microsoft/DialoGPT-medium" \
  --model-version "7b40bb0f92c45fefa957d088000d8648e5c7fa33"
```

Scan using Python SDK

```
from uuid import UUID
result = client.scan(
    security_group_uuid=UUID("12345678-1234-1234-1234-123456789012"),
    model_uri="https://huggingface.co/microsoft/DialoGPT-medium",
    model_version="7b40bb0f92c45fefa957d088000d8648e5c7fa33"
)
```


Filter Files in HuggingFace Scans

Large HuggingFace repositories may contain files you don't need to scan. Use global patterns to include or exclude specific files.

Scan using CLI

```
model-security scan \
  --security-group-uuid "12345678-1234-1234-1234-123456789012" \
  --model-uri "https://huggingface.co/microsoft/DialoGPT-medium" \
  --allow-patterns "*.bin" "*.json" \
  --ignore-patterns "*.md" "*.txt"
```

Scan using Python SDK

```
from uuid import UUID
result = client.scan(
    security_group_uuid=UUID("12345678-1234-1234-1234-123456789012"),
    model_uri="https://huggingface.co/microsoft/DialoGPT-medium",
    allow_patterns=["*.bin", "*.json"],
    ignore_patterns=["*.md", "*.txt"]
)
```

Scanning a Local Model

For models stored locally, specify the path to the model directory.

Scan using CLI

```
model-security scan \
  --security-group-uuid "12345678-1234-1234-1234-123456789012" \
  --model-path "path/to/local/model"
```

Scan using Python SDK

```
from uuid import UUID
result = client.scan(
    security_group_uuid=UUID("12345678-1234-1234-1234-123456789012"),
    model_path="path/to/local/model"
)
```

Scanning a Model from Object Storage

To scan a model from object storage (such as S3), provide both the local path and the storage URI.

Scan using CLI

```
model-security scan \
  --security-group-uuid "12345678-1234-1234-1234-123456789012" \
  --model-path "path/to/local/model" \
  --model-uri "s3://your-bucket/model-directory" \
```



```
  --model-name "production-classifier" \
  --model-author "ml-team" \
  --model-version "v2.1"
```

Scan using Python SDK

```
from uuid import UUID
result = client.scan(
    security_group_uuid=UUID("12345678-1234-1234-1234-123456789012"),
    model_path="path/to/local/model",
    model_uri="s3://your-bucket/model-directory",
    model_name="production-classifier",
    model_author="ml-team",
    model_version="v2.1"
)
```

The CLI shows scan results in real-time as they finish. Each scan tests the model against all active rules in your Security Group. The output shows whether the model passes or fails based on your rule configuration.

A model fails if any blocking rule detects a violation. Non-blocking rules record findings without preventing the model from being approved.

Customize Scanning Models

You can configure scan execution and adjust result timeout settings.

Customize Scan using CLI

```
model-security scan \
  --security-group-uuid "12345678-1234-1234-1234-123456789012" \
  --model-uri "https://huggingface.co/microsoft/DialoGPT-medium" \
  --poll-interval-secs 10 \
  --poll-timeout-secs 900 \
  --block-on-errors
```

Customize Scan using Python SDK

```
from uuid import UUID
result = client.scan(
    security_group_uuid=UUID("12345678-1234-1234-1234-123456789012"),
    model_uri="https://huggingface.co/microsoft/DialoGPT-medium",
    poll_interval_secs=10,
    poll_timeout_secs=900,
    scan_timeout_secs=900
)
```

Following are the configuration options to customize the scan for AI models.

Configuration Option	Description	Default Value
`poll_interval_secs`	Specify the frequency of scan status checks.	5 seconds
`poll_timeout_secs`	Specify the maximum wait time for scan completion.	600 seconds
`scan_timeout_secs`	(SDK only) Specify the timeout for local model scanning.	600 seconds
`block_on_errors`	(CLI only) CLI exits with an error code when scan errors occurs.	NA

Viewing Scan Results

Where Can I Use This?	What Do I Need?
• Prisma AIRS (AI Model Security)	☐ Prisma AIRS AI Model Security License

The `scan` command displays results directly, showing the overall verdict and key findings. For deeper analysis, you can retrieve detailed results using the scan ID.

After a model is scanned, it will either pass or fail the scan based on the security checks performed by AI Model Security. If the model passes the scan, it will be downloaded as usual. If the model fails the scan, it will fail to download and return a 403 error as well as a Universally Unique Identifier (UUID) that can be used to view the scan results.

Retrieving a Specific Scan (CLI/SDK)

After a scan completes, note the scan ID from the output. Retrieve the full results at any time.

Retrieve Scan Results using CLI

```
model-security get-scan "87654321-4321-4321-4321-210987654321"
```

Retrieve Scan Results using Python SDK

```
from uuid import UUID
scan_id = UUID("87654321-4321-4321-4321-210987654321")
scan_result = client.get_scan(scan_id)

print(f"Scan Status: {scan_result.eval_outcome}")
print(f"Model URI: {scan_result.model_uri}")
print(f"Created: {scan_result.created_at}")
```

View Scan Summary (CLI/SDK)

View a summary of recent scans to track your security assessments.

View Scan Summary using CLI

```
model-security list-scans --limit 20
```

View Scan Summary using Python SDK

```
scans = client.list_scans(limit=20)

for scan in scans.scans:
    print(f"Scan {scan.uuid}: {scan.eval_outcome} -
 {scan.model_uri}")
```

You can also filter scans by source type, evaluation outcome, or time range.

Filter Scans by source type, evaluation outcome, or time range using CLI

```
model-security list-scans \
  --source-types "HUGGING_FACE" "S3" \
  --eval-outcomes "ALLOWED" "BLOCKED" \
  --start-time "2025-01-01T00:00:00" \
  --limit 50
```

Filter Scans by source type, evaluation outcome, or time range using Python SDK

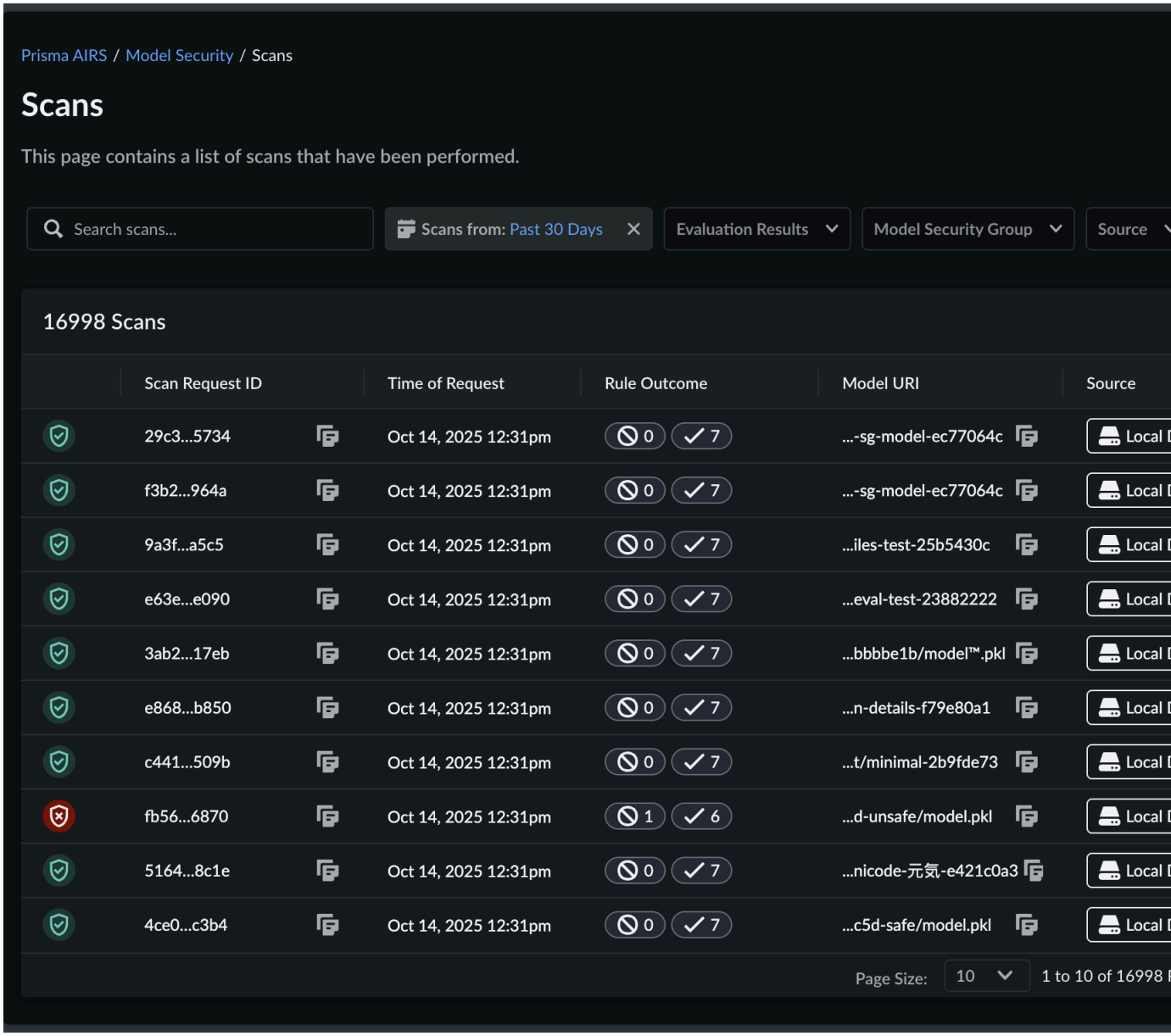
```
from datetime import datetime, timezone
from airs_schemas.constants import SourceType, EvalOutcome


scans = client.list_scans(
    source_types=[SourceType.HUGGING_FACE, SourceType.S3],
    eval_outcomes=[EvalOutcome.ALLOWED, EvalOutcome.BLOCKED],
    start_time=datetime(2025, 1, 1, tzinfo=timezone.utc),
    limit=50
)
```

View Scan Results using Strata Cloud Manager (Web Interface)

Although the CLI delivers complete results, the AI Model Security web interface provides additional tools for analyzing scan findings.

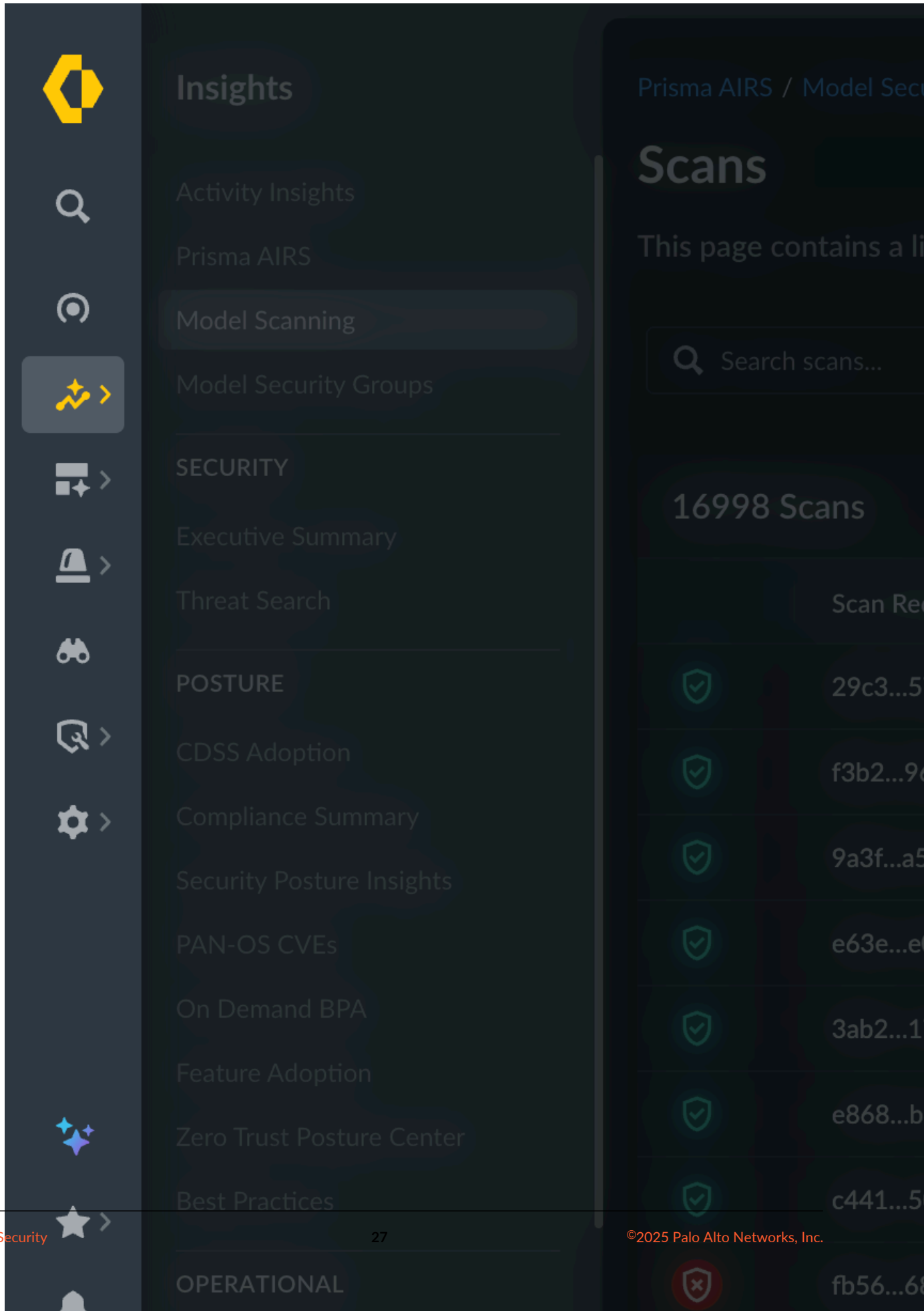
1. Log in to Strata Cloud Manager.

2. Navigate to the **Insights** > **Prisma AIRS** > **Model Security** > **Scans**

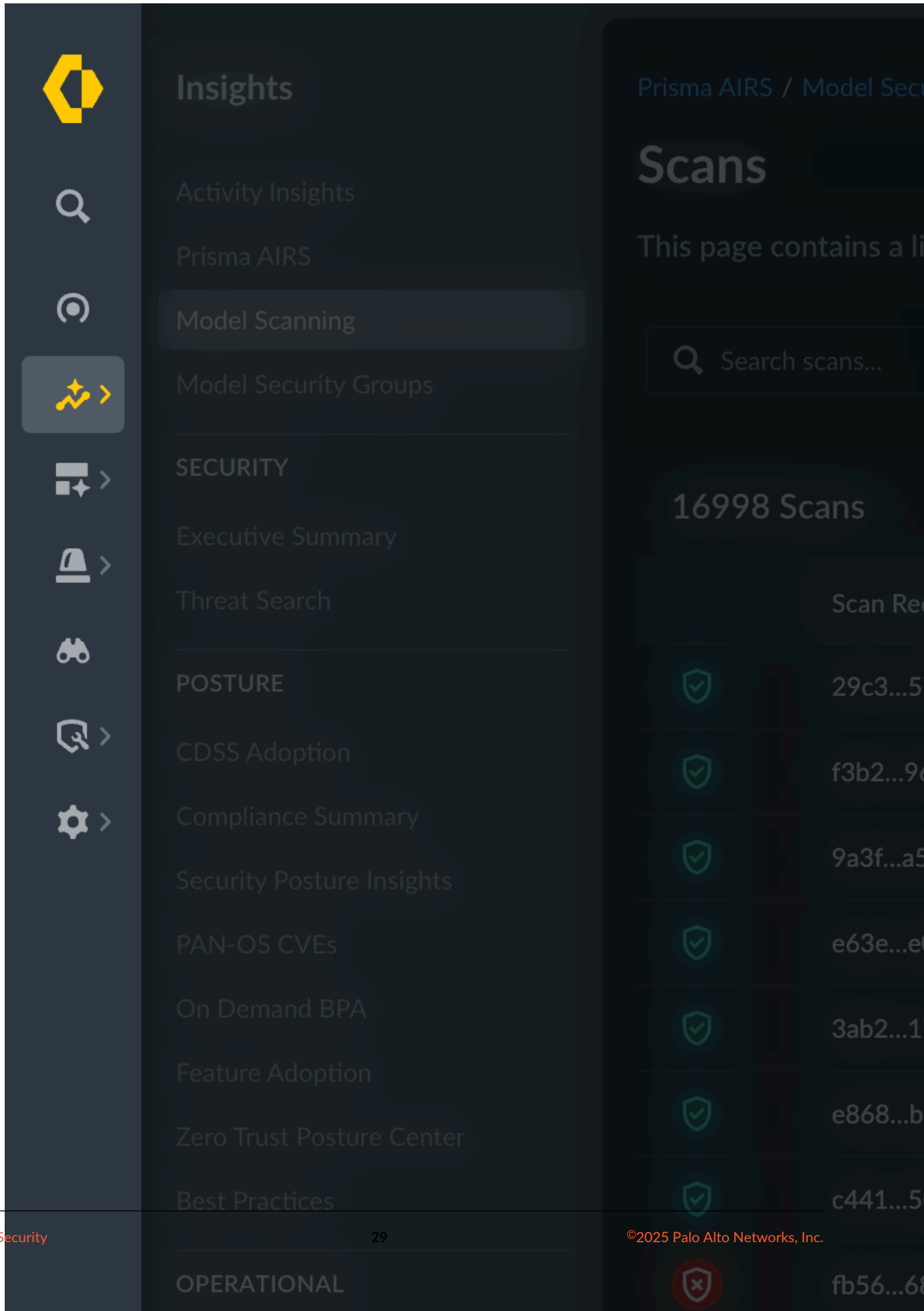


3. Locate your scan by ID or filter the scan list. Review detailed findings for each rule evaluation. Export results or share them with your team.

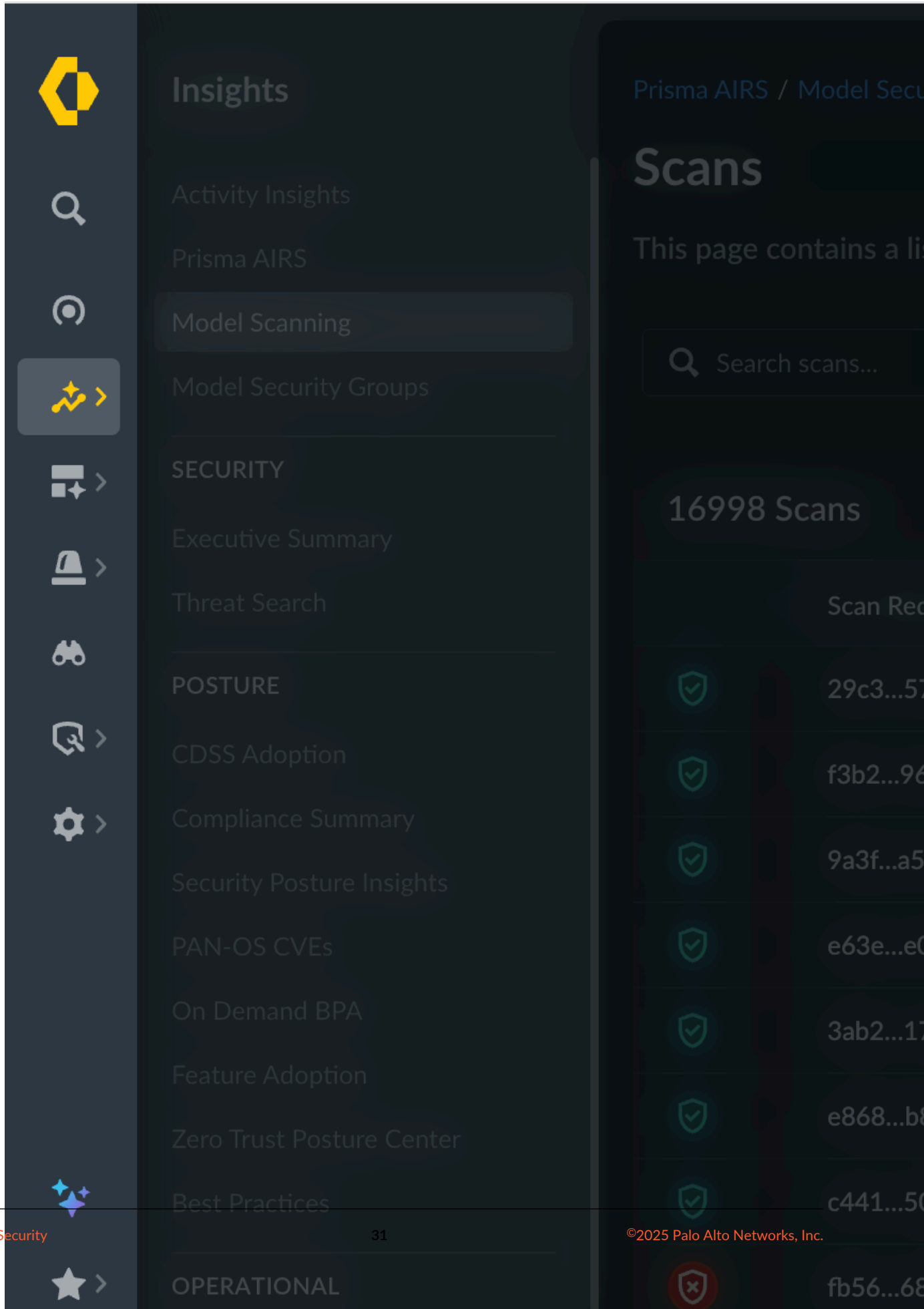
   Following is an example scan result that is **Allowed**. After locating the specific scan, select **Overview** to review the evaluation details.



Following is an example scan result that is **Blocked**. After locating the specific scan, select **Overview** to review the evaluation details.



Select **JSON** of the specific Scan, to review the details of the scan results in JSON format.



The Strata Cloud Manager displays rule violations visually, identifies the specific files or model components that caused findings, and offers remediation guidance.

Understanding Model Security Scan Results

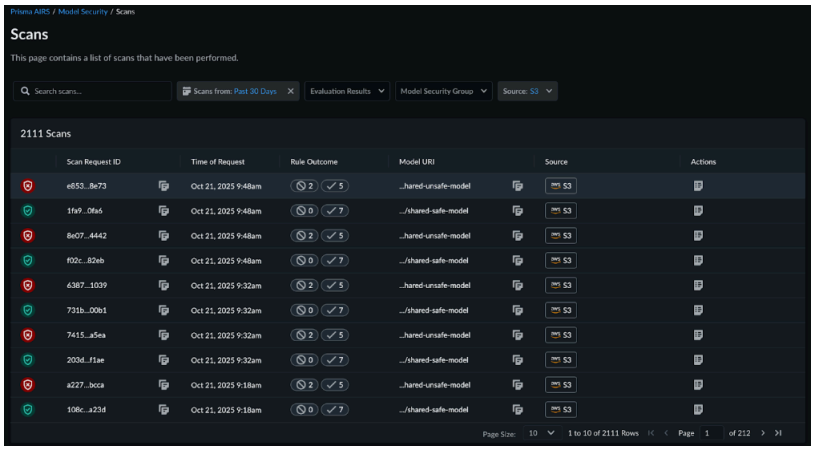
Where Can I Use This?	What Do I Need?
• Prisma AIRS (Model Security)	☐ Prisma AIRS Model Security License

Once a scan finishes, you can retrieve the results through either the SDK or the Strata Cloud Manager web interface. The primary check involves determining whether your scanned model is permitted or restricted.

Execute the following command to retrieve the scan result using the SDK:

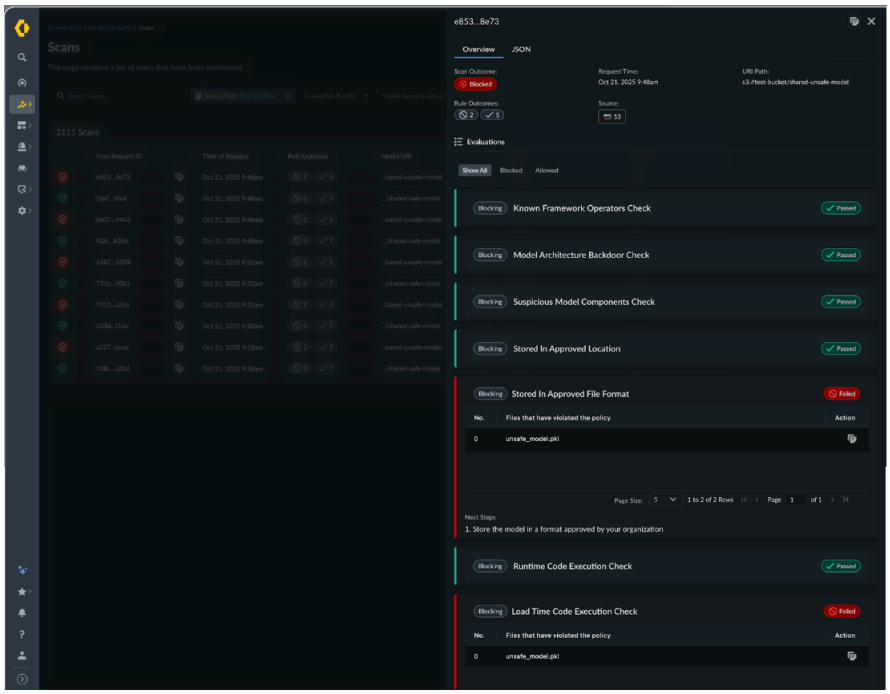
```
scan_result.eval_outcome
```

To view the scan results in the Strata Cloud Manager, navigate to **Insights** > **Prisma AIRS** > **Model Security** > **Scans** and locate your recently scanned model.



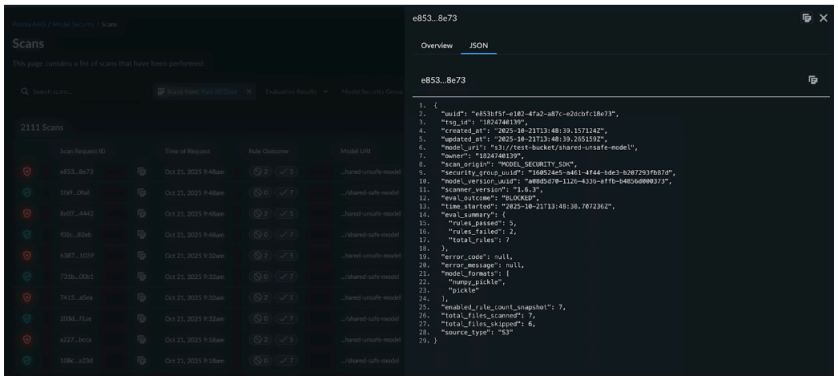
A red shield indicates the model was blocked, while a green shield shows the model is allowed.

Select the **Scan Request ID** to open a detailed flyover with comprehensive scan information:



This example shows a result of a scan that is blocked due to failure in compliance to the rules: the use of non-approved file formats and detection of code execution in `unsafe_model.pkl` that runs during model loading, indicating a potential deserialization attack.

To view the complete JSON response that your SDK would receive, select **JSON**:



Understanding Model Security Rules

Where Can I Use This?	What Do I Need?
• Prisma AIRS (Model Security)	❑ Prisma AIRS Model Security License

Model security rules serves as the central mechanism for securing model access. Model security scanning covers several key areas: thread and metadata. To learn more about threat categories and risk mitigation strategies for AI and machine learning systems, refer Model Threats.

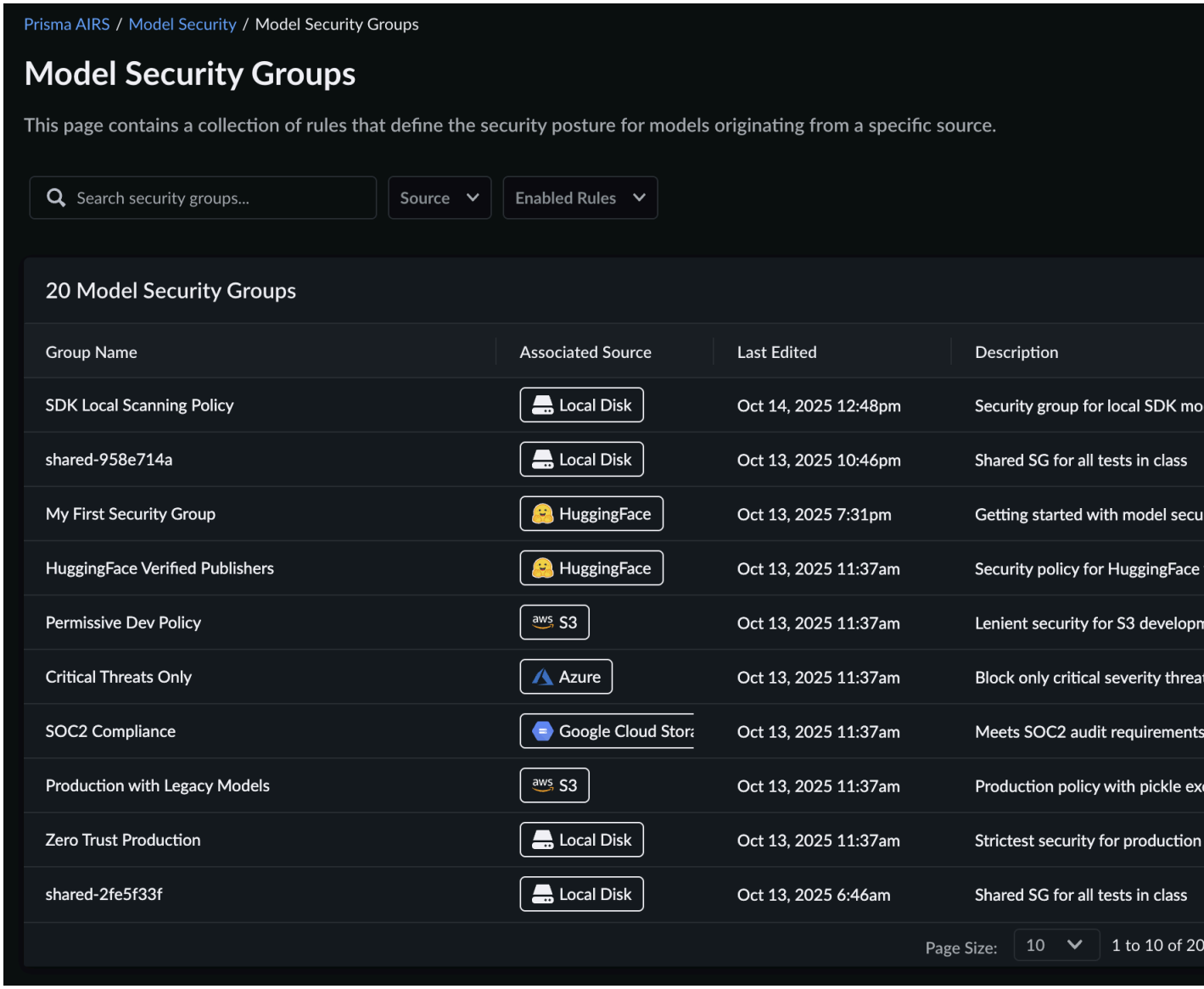
Following are the three major threat categories:

Threat	Description
Deserialization threats	Issues that arise when you load a model into memory.
Backdoor Threats	Issues that arise when a model was specifically designed to support alternative or hidden paths in its behavior.
Runtime Threats	Issues that arise when you use a model to perform inference.

Model security also examine specific metadata fields in models to address security considerations, such as verifying a model's license to ensure it's appropriate for commercial use. Model security rules enables you to validate the following concerns:

Metadata in Models	Security Rule Validation
Open Source License	Ensures that the model you are using is licensed for your use case.
Model Format	Ensures that the model you are using is in a format that is supported by your environment.
Model Location	Ensures that the model you are using is hosted in a location that is secure and trusted.
Verified Organizations in Hugging Face	Ensures that the model you are using is from a trusted source.
Model is Blocked	Overrides all other checks to ensure that a model is blocked no matter what.

Our managed rules integrate all of these checks to provide comprehensive security for model usage. To get started, you can explore your defaults in your Model Security at **Insights** > **Prisma AIRS** > **Model Security** > **Model Security Groups**.



How Scans, Security Groups, and Rules Work Together

With Model Security, you initiate a model scan and associate it with a security group. Model Security evaluates the model against all enabled rules within that security group to assess whether it satisfies your security requirements. This assessment relies on the results from enabled rules in the security group and the group's configured threshold.

When you scan a model as follows:

```
# Import the Model Security SDK/Client
from model_security_client.api import ModelSecurityAPIClient
# Load your scanner URL
```



```
scanner_url = os.environ["MODEL_SECURITY_TOKEN_ENDPOINT"]
# Define your model's URI
model_uri = "s3://demo-models/unsafe_model.pkl"
# Set your security group's UUID
security_group_uuid = "6e2ccc3a-db57-4901-a944-ce65e064a3f1"
# Create a Model Security Client
guardian = ModelSecurityAPIClient(base_url=scanner_url)
# Scan your model
response = client.scan(model_uri=model_uri,
 security_group_uuid=security_group_uuid)
```

The response will appear as follows (showing the security rules results):

```
{
  "http_status_code": 200,
  "scan_status_json": {
    "aggregate_eval_outcome": "FAIL",
    "aggregate_eval_summary": {
      "critical_count": 1,
      "high_count": 0,
      "low_count": 0,
      "medium_count": 1
    },
    "violations": [
      {
        "issue": "Model file 'ykilcher_totally-harmless-model/
retr0reg.gguf' is stored in an unapproved format: gguf",
        "threat": "UNAPPROVED_FORMATS",
        "operator": null,
        "module": null,
        "file": "ykilcher_totally-harmless-model/retr0reg.gguf",
        "hash": "f59ad9c65c5a74b0627eb6ca5c066b02f4a76fe6",
        "threat_description": "Model is stored in a format that your
 Security Group does not allow",
        "policy_name": "Stored In Approved File Format",
        "policy_instance_uuid": "34ef1ddc-0b7a-45b8-a84a-
c96b1d8383d0",
        "remediation": {
          "steps": [
            "Store the model in a format approved by your
 organization"
          ]
        }
      },
      {
        "issue": "The model will execute remote code since it
 contains operator `__class__` in Jinja template.",
        "threat": "PAIT-GGUF-101",
        "operator": "__class__",
        "module": null,
        "file": "ykilcher_totally-harmless-model/retr0reg.gguf",
        "hash": "f59ad9c65c5a74b0627eb6ca5c066b02f4a76fe6",
        "threat_description": "GGUF Model Template Containing
 Arbitrary Code Execution Detected",
        "policy_name": "Load Time Code Execution Check",
```



```
        "policy_instance_uuid": "09780b9f-c4f7-4e0b-
ad21-7ff779472283",
        "remediation": {
          "steps": [
            "Use model formats that disallow arbitrary code
 execution"
          ]
        }
      }
    ]
  }
}
```


Note the `FAIL` status in the `aggregate_eval_outcome` field. This indicates the model did not pass the scan because security rule failures surpassed your security group's threshold, with the `violations` field providing details about which rules were breached.

Each model security rule contains the following fields.

Rule Field	Description
Rule Name	Specifies the name of the security rule.
Rule Description	Specifies the description of the security rule.
Compatible Sources	Specifies the model source types that this rule is compatible with.
Status	Specifies the status of the security rule, either `Enabled` or `Disabled`. This can be set globally, or at the security group level.

Connecting Scans to Rules

When scanning a model, model security first identifies its format through introspection. After determining the model type, model security maps it to the taxonomy of model vulnerability threats and coordinate the specific deeper scans required for that model.

A series of specific threats like `Arbitrary Code Execution At Runtime` are grouped together when they are reported and are shown in a specific rule. This allows you to block specific types of threats without having to manage the complexity of all the various formats and permutations.

You can also configure common rules for all models regardless of their source type.

Security Rule Checks

Rule Name	Status	Description	Example
Runtime Code Execution Check	**Status:** `Enabled`	This rule detects Arbitrary Code Execution that can occur during model inference through various methods.	• Keras Model Lambda Layer Arbitrary Code Execution Detected at Model Run Time • TensorFlow SavedModel Contains Arbitrary Code Execution at Model Run Time These attacks mean the model will execute code without your knowledge during use, making this a `Critical` issue to block. Learn more about this threat type here: Runtime Threats.
Known Framework Operators Check	**Status:** `Enabled`	Machine learning model formats often include built-in operators to support common data science tasks during model operation. Some frameworks allow custom operator definitions, which poses risks when executing unknown third-party code.	When `TensorFlow SavedModel Contains Unknown Operators` is detected, it indicates that the model creator is

Rule Name	Status	Description	Example
			using non-standard tooling approaches. For more information refer, SavedModel Contains Unknown Operators.
Model Architecture Backdoor Check	**Status**: `Enabled`	A model's behavior can contain a backdoor embedded in its architecture, specifically through a parallel data flow path within the model. For most inputs, the model operates as expected, but for certain inputs containing a trigger, the backdoor activates and effectively alters the model's behavior.	When `ONNX Model Contains Architectural Backdoor` is detected, it warns you that a model has at least one non-standard path requiring further investigation. For more information refer, ONNX Model Contains Architectural Backdoor.
Load Time Code Execution Check	**Status**: `Enabled`	This rule is similar to the runtime attacks, but these attacks execute immediately upon model loading. For example, the below python snippet is sufficient to trigger the exploit without your knowledge: `with open('path_to_your_model.pkl', 'rb') as file: model = pickle.load(file)`	When `Pickle, Keras Lambda Layers, PyTorch models, and more are all vulnerable to this threat` is detected in a

Rule Name	Status	Description	Example
			model, there's no reliable method to eliminate the threat. However, you can investigate the model's history or, if it's your own model, examine the build process to identify the source of the malicious code. For more information refer, PyTorch Model Arbitrary Code Execution
Suspicious Model Components Check	**Status:** `Enabled`	Not all rules target specific threats; some, like this one, assess a model's potential for future exploitation. This check identifies components within the model that could enable malicious code execution later. Example: • Remote code execution being called by fetching external data from Protobuf files or others over the internet. A violation here should prompt you to be cautious and to evaluate all relevant components around the model before making a decision on whether or not the model is safe for use. More information can be found here: Keras Lambda Layers Can Execute Code	`Remote code execution being called by fetching external data from Protobuf files or others over the internet.` This violation prompts caution and thorough evaluation of all relevant model

Rule Name	Status	Description	Example
			components before determining whether the model is safe for use. For more information refer, Keras Lambda Layers Can Execute Code
Stored In Approved File Format	**Status**: `Enabled`	This rule verifies whether the model is stored in a format that you've approved within the rule's list. We recommend enabling these formats by default: • Safetensors • JSON By default Model Security reports that `pickle` and `keras_metadata` are not approved formats.	—

Hugging Face Model Rules

Lastly, there are rules specifically scoped to Hugging Face models. These rules target the particular metadata you control or that is consistently provided by Hugging Face.

Hugging Face Model Rule	Status	Description
License Exists	**Status**: `Enabled`	The simplest rule in the application. This rule checks to see if the model has a license associated with it. If no license is present, then this rule will fail.
License Is Valid For Use	**Status**: `Enabled`	This rule gives you more control for the models that your organization will run or test. As a commercial entity, you may not want to run models with non-permissive open-source licenses like GPLv3 or others.

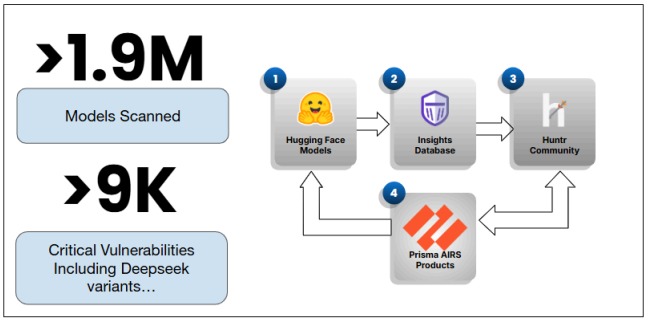
Hugging Face Model Rule	Status	Description
		Adding licenses to the rule will expand the list of licenses that are allowed for use. A reasonable set of defaults are: • apache-2.0 • mit • bsd-3.0 You can find a full list of license options here: Hugging Face Licenses Note we use the `License identifier` field to map to the license in the model metadata.
Model Is Blocked	**Status**: `Enabled`	Sometimes you just cannot tolerate a model for whatever reason. This rule allows you to block a specific model from being used. The format for blocking a model from Hugging Face relies on the Organization and Model Name. For example `opendiffusion/sentiment-check` can be entered and it would block the model `sentiment-check` from `opendiffusion`. *When a model is both blocked and allowed simultaneously, the block takes precedence.*
Organization Verified By Hugging Face	**Status**: `Enabled`	Hugging Face is an excellent site for the latest models from all over the world, giving you access to cutting edge research. You may want to restrict organizations from providing unverified models from Hugging Face. This prevents accidentally running models from deceptively similar sources like `facebook-llama` instead of the `legitimate meta-llama` (where `legitimate meta-llama` is the correct model). This rule simply checks that Hugging Face has verified the organization, if that passes

Hugging Face Model Rule	Status	Description
		the models from the organization will pass this check.
Organization Is Blocked	**Status**: `Enabled`	If you find a particular organization that just delivers problematic models or for any other reason, you'd like to block them, this is your rule. Enter the organization name into the rule and all models from that organization will be blocked. For example `facebook-llama` would block ALL of the models provided by that organization.

Reporting New Threats

Where Can I Use This?	What Do I Need?
• Prisma AIRS (Model Security)	❑ Prisma AIRS Model Security License

Our comprehensive vulnerability detection begins with the industry's most extensive dataset of model security issues. This intelligence is driven by our Huntr community—a network of over 16,000 security researchers who continuously discover novel attack vectors through our bug bounty program and strategic partnership with Hugging Face.



You can explore our most recent discoveries in the Insights DB, where we publish vulnerability assessments from scanning every public model in the Hugging Face repository. The platform includes mechanisms for security researchers and ML practitioners to challenge or validate our findings.

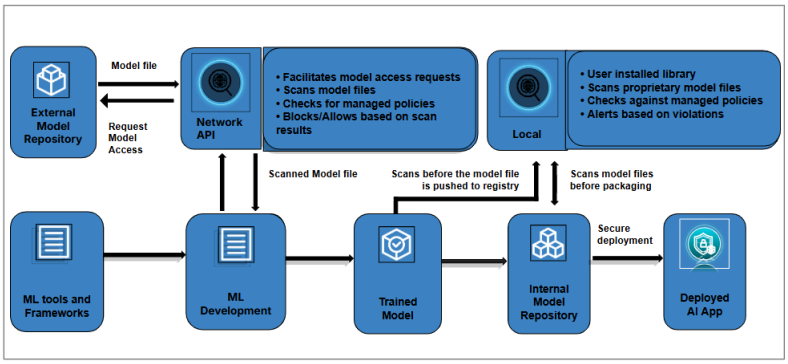
InsightsDB classifies and lists the models as **Safe**, **Unsafe**, and **Suspicious**. You can dispute a finding in Insights DB by selecting the specific model from the list and **Report an issue**:

- **Report your finding** if you've found a new threat.
- **Report an incorrect threat** if you disagree with our findings

This continuous intelligence stream enables us to identify a broad spectrum of vulnerabilities in your models, including many AI-specific security issues that fall outside the scope of traditional software vulnerabilities—threats that typically won't appear in the National Vulnerability Database (NVD) or conventional security feeds.

Model Security operates as a centralized repository of models from third-party sources like Hugging Face.

Additionally, we provide an on-premises scanning solution that deploys directly within your infrastructure, enabling you to assess your proprietary models locally without data transmission to our systems—ensuring complete privacy and security of your assets.



Supported Model Formats

Where Can I Use This?	What Do I Need?
• Prisma AIRS (Model Security)	❑ Prisma AIRS Model Security License

AI Model Security checks are supported on the following formats:

1. **CNTK Models**: Models saved in Microsoft Cognitive Toolkit format.
2. **Flax Models**: Models created with Flax, a neural network library for JAX.
3. **GGUF Models**: General-purpose model format using GGUF.
4. **Keras Models**:
   - **Legacy Keras Models**: Older Keras models, often saved with HDF5.
   - **Keras 3 Models**: Newer Keras models using the latest Keras version.
   - **Keras Pickle Models**: Keras models saved with Python pickle.
   - **Keras H5 Models**: Models in HDF5 format, compatible with legacy and newer versions.
   - **Keras Weights**: Separate files storing only model weights.
   - **Keras JSON**: Models saved in JSON format for architecture storage.
   - **Keras Metadata**: Auxiliary files that store metadata for Keras models.
5. **KModel**: KModel files specific to Keras.
6. **LightGBM Models**: Gradient boosting models using LightGBM.
7. **MS Lite Models**: Microsoft Lite format for lightweight models.
8. **MXNet Models**: Models saved in Apache MXNet format.
9. **Numpy Models**:
   - **Numpy Array Files**: Arrays saved in .npy format.
   - **Numpy Zip Files**: Arrays compressed in .npz format.
   - **Numpy Pickle Files**: Arrays serialized with pickle.
10. **OM Models**: Models in Huawei Ascend's OM format.
11. **ONNX Models**: Models saved in Open Neural Network Exchange format.
12. **OpenVINO Models**:
    - **OpenVINO Binary Files**: Compiled binary files for OpenVINO.
    - **OpenVINO XML Files**: XML files storing OpenVINO model metadata.
13. **Pickle Files**: Models serialized using Python's pickle.
14. **PyTorch Models**:
    - **Various PyTorch Versions**: Models saved with different PyTorch versions.
    - **TorchScript**: PyTorch's format for serializing models.
    - **PyTorch Archives**: Archived files containing serialized models.

15. **RKNN Models**: Models saved in Rockchip Neural Network (RKNN) format.
16. **Safetensors**:
    - **Safetensors Models**: Models saved using safetensors format for secure tensor storage.
    - **Safetensors Index**: Index files for safetensors.
17. **SKLearn Models**: Scikit-learn models serialized for deployment.
18. **TensorRT Models**: NVIDIA's TensorRT models optimized for inference.
19. **TensorFlow Models**:
    - **SavedModel**: TensorFlow's standard saved model format.
    - **TFHub**: Models from TensorFlow Hub.
    - **MetaGraph**: TensorFlow's MetaGraph format for exporting graphs.
    - **TFLite**: Lightweight format for mobile and embedded devices.
    - **TFJS**: TensorFlow.js format for models running in the browser.
20. **Torch Models**: General format for PyTorch models.
21. **JSON Files**: JSON-based configurations or model descriptions.

