JUPITERONE

JupiterOne Data Security

https://jupiterone.io
November 29, 2018

distributed under NDA

Table of Contents

JupiterOne Data Security

This document describes in detail the data JupiterOne ingests and how your data is protected on our platform.

Data Protection

Encryption

Data is fully encrypted both at rest and in transit. This includes all of your account and user data, as well as operational data imported/ingested into the JupiterOne platform.

Data in Transit is encrypted via TLSv1.2 or later, using SHA-256 with 2048-bit RSA Encryption or equivalent strength cypher.

Production Domains: `*.apps.us.jupiterone.io` is the associated production URL that the SSL/TLS certificate has been issued to.

Data at Rest is hosted in our production AWS environments, using the managed RDS/Neptune, DynamoDB, and S3 services. All database instances, tables, and S3 buckets with customer data have server-side encryption enabled, using AWS KMS for key management. KMS encryption keys are scheduled to rotate annually.

In addition to encryption, managed backup is enabled for the database clusters. For S3 buckets, cross-account replication is enabled to back up data to a different region for disaster recovery. All backup data is fully encrypted the same way as its source.

Multi-tenancy

JupiterOne is a multi-tenancy, software-as-a-service platform hosted in AWS. Customer data is logically partitioned/segregated by software via a unique `accountId` associated with every piece of data. Access to data is restricted to within each unique account for users granted proper access to that account. This is a standard pattern used by cloud infrastructure and SaaS providers.

External Data Ingestion/Import

JupiterOne ingests data from external sources and connected environments primarily via the APIs provided by the target environment/service provider. Objects from these external environments and their corresponding metadata, including configuration properties and tags but never the actual data content, are ingested as "entities". The entity properties and tags are used to perform analysis to build out "relationships" among ingested entities. These entities and relationships are the **JupiterOne CORE Data Model**.

JupiterOne then uses this data model to inventory for and provide insight into your digital infrastructure across all of your connected environments.

More information on the JupiterOne Data Model can be found here.

For more details on data ingested for each managed integration, see their corresponding documentation:

- AWS
- Bitbucket
- Github
- Okta

Access Permissions Needed to Integrated Environments

Access to your environments is needed in order to ingest data, or to enable workflow automation (future capability).

In general, JupiterOne only requires read-only, security-auditor-type access permissions to your environments. Additionally, this read-only access only applies to configurations and meta data, not the actual data content. For example, we do **NOT** read S3 objects data from a connected AWS account, or the actual source code of a connected Bitbucket/Github account.

Additional level of access may be needed for workflow automation. For example, integration with Jira to automatically create an issue when a new Vulnerability finding is added; or to post to a Slack channel/user to send a security alert notification.

You are always in control of the actual permissions granted for each integration. More details of the access permissions required for each managed integration can be found in its corresponding documentation listed above.

Custom Data Import

Additionally, JupiterOne supports the ability for you to add custom data by

- Manually adding entities via Web UI in the Asset Inventory app;
- Adding bulk number of entities via CSV import; or
- Adding custom entities via custom integrations using the public API.

Data Ownership and Access

You retain full ownership of all data that is ingested via integrations, API or manual importing/creation. Data is stored in JupiterOne's production environment in AWS, protected via encryption and replication as specified in the first section.

Infrastructure and Operational Access

JupiterOne infrastructure is built on a **Zero Trust** security model, where access to production is *highly restricted*.

The production environment is virtually “air-gapped” such that there is no SSH, “bastion host”, or VPN connectivity into the production systems to prevent unintended network access to databases and other production servers. We do not allow internal access to production data by any JupiterOne team member. All necessary operational support and maintenance jobs are performed via automation where the automation code is fully documented, reviewed, and approved, ensuring end-to-end traceability.

Our production environment incorporates multiple layers of security monitoring, using JupiterOne itself as well as third party security solutions. Additionally, our software development includes rigorous code analysis and continuous testing practices to ensure we proactively identify any security vulnerability. Our infrastructure-as-code operational model and automated change management process allows us to deploy security patches within minutes of identification and remediation of an issue.

You can review our published security model and corresponding policies and procedures for more details on our operational, infrastructure, and software development security.

Application Access

Access to the JupiterOne application and your accounts/data on the platform is enabled over HTTPS, through either the JupiterOne web apps or the public APIs.

Note: `*.us.jupiterone.io` is the current production domain.

User Logins

Each user has a unique user login to the JupiterOne platform and apps. Users may be invited to one or multiple organizational accounts on JupiterOne.

Password Policy

Users are required to select a strong password meeting the following password policy requirements in order to create a login and authenticate to the system:

- Minimum of 8 characters
- Must contain an uppercase letter
- Must contain a lowercase letter
- Must contain a number
- Must contain a special character

Single Sign On (SSO)

JupiterOne currently supports single sign on (SSO) via:

- Google
- SAML

Multi-Factor Authentication (MFA) / Two-Step Verification (2SV)

Multi-Factor Authentication (MFA) or Two-Step Verification (2SV) is strongly recommended for all users on the JupiterOne platform. This needs to be enabled and configured via your SSO provider (Google or your SAML IdP such as Okta or OneLogin).

Access Control

In order to support potential complex access control use cases, JupiterOne platform implements Attribute Based Access Control (ABAC).

A good general overview of ABAC is sections 1 and 2 of NIST's Guide to Attribute Based Access Control. The absolute basics of ABAC are that you have a subject (e.g. a user) who wants to perform some operation (e.g. download) on an object (e.g. a file) in some environment. The subject, object and environment all have attributes (i.e. key/value pairs), and there are policies that control the privileges (i.e. what operations the subject can perform) given the attributes.

Access policies defined in JupiterOne are associated with a **User Group** and **Users** are invited/added as members to one or more groups.

- A *Read-Only* access policy is predefined and associated with the default **Users** group.
- A *Full-Access* policy is also predefined and associated with the default **Administrators** group.
- The ability to customize and add granular access control policies is to be released in 1Q2019.

API Access

JupiterOne API is available at: https://api.us.jupiterone.io/

We use OAuth 2.0 for authorization, which means in order to access data a user must authenticate and the requesting app must be authorized. Implicit grant, authorization code, and client credentials flows are supported. Authorization code is recommended for web apps, which involves utilizing both the authorize and token API resources. When using the authorization code grant flow, it is also recommended to use Proof Key for Code Exchange (PCKE) to mitigate authorization code intercept attacks. Contact us if building a native app which can securely perform client credentials flow.

Additionally, each user on the platform can create an API key that can be passed along with request to act on behalf of that user.

Note: the UI for self-service configuration of OAuth and user API key is targeted to be available in 1Q2019.

Support Access to Your JupiterOne Account(s)

A JupiterOne Support User is by default added to a new account during free trial, proof-of-concept evaluation, or initial account onboarding. This is to facilitate better support and training to using the platform.

- The support user's login can either be the individual Security Engineer/Architect designated to your account (e.g. `firstname.lastname@jupiterone.io`) or the general support login (i.e. `callisto@jupiterone.io`).
- The support user can be removed by an account administrator at any time, should you determine that ongoing regular support is no longer needed.
- You have the option and administrative privilege to add the support user back at any time, when support is needed in the future. # AWS Data and Integration Details

AWS Data and Integration Details

Overview

JupiterOne provides a managed integration with Amazon Web Services. The integration connects directly to AWS APIs to obtain infrastructure metadata and analyze resource relationships. Customers authorize read-only, security audit access by establishing an IAM trust relationship that allows JupiterOne to assume a role in their account.

Information is ingested from all AWS regions that do not require additional contractual arrangements with AWS. Please submit a JupiterOne support request if you need to monitor additional regions.

Integration Instance Configuration

The integration is triggered by an event containing the information for a specific integration instance.

The integration instance configuration requires the customer's `roleArn` to assume in order to read infrastructure information through AWS APIs. The role is configured to require an `externalId`; this also must be maintained in the instance configuration.

Detailed setup instructions and a pre-built CloudFormation Stack are provided in the application and maintained in the public JupiterOne AWS Integration project on Github.

Entities

The following entity resources are ingested when the integration runs:

AWS Service	AWS Entity Resource	_type : _class of the Entity
Account	n/a	`aws_account` : `Account`
API Gateway	REST API	`aws_apigateway_rest_api`: `Gateway`
EC2	EC2 Instance	`aws_ec2_instance` : `Host`
	EBS Volume	`aws_ec2_volume` : `DataStore, Disk`
	Security Group	`aws_ec2_security_group` : `Firewall`
	VPC	`aws_ec2_vpc` : `Network`
	Subnet	`aws_ec2_subnet` : `Network`
IAM	IAM User	`aws_iam_user` : `User`
	IAM Group	`aws_iam_group` : `UserGroup`
	IAM Role	`aws_iam_role` : `AccessRole`

	IAM User Policy	aws_iam_user_policy : AccessPolicy
	IAM Group Policy	aws_iam_group_policy : AccessPolicy
	IAM Role Policy	aws_iam_role_policy : AccessPolicy
	IAM Managed Policy	aws_iam_managed_policy : AccessPolicy
S3	S3 Bucket	aws_s3_bucket : DataStore
Lambda	Lambda Function	aws_lambda_function : Function, Workload
Config	Config Rule	aws_config_rule : ControlPolicy

Relationships

The following relationships are created/mapped:

Basic relationships within the integration instance account/resources

aws_account **HAS** aws_apigateway

aws_account **HAS** aws_ec2

aws_account **HAS** aws_iam

aws_account **HAS** aws_lambda

aws_account **HAS** aws_s3

aws_account **HAS** aws_config

aws_apigateway **HAS** aws_apigateway_rest_api

aws_apigateway_rest_api **TRIGGERS** aws_lambda_function

aws_config **HAS** aws_config_rule

aws_config_rule **EVALUATES** aws_account

aws_config_rule **EVALUATES** aws_ec2_instance

aws_config_rule **EVALUATES** aws_ec2_security_group

aws_config_rule **EVALUATES** aws_ec2_volume

aws_config_rule **EVALUATES** aws_iam_user

aws_config_rule **EVALUATES** aws_iam_group

aws_config_rule **EVALUATES** aws_iam_role

aws_config_rule **EVALUATES** aws_s3_bucket

aws_ec2 **HAS** aws_ec2_instance

aws_ec2 **HAS** aws_ec2_security_group

aws_ec2 **HAS** aws_ec2_subnet

aws_ec2 **HAS** aws_ec2_volume

aws_ec2 **HAS** aws_ec2_vpc

aws_ec2_instance **USES** aws_ec2_volume

aws_ec2_security_group **PROTECTS** aws_ec2_instance `aws_ec2_vpc` **CONTAINS** `aws_ec2_subnet`

`aws_iam` **HAS** `aws_iam_managed_policy`

`aws_iam` **HAS** `aws_iam_role`

`aws_iam` **HAS** `aws_iam_role_policy`

`aws_iam` **HAS** `aws_iam_user`

`aws_iam` **HAS** `aws_iam_user_policy`

`aws_iam` **HAS** `aws_iam_group`

`aws_iam` **HAS** `aws_iam_group_policy`

`aws_iam_group` **HAS** `aws_iam_group_policy`

`aws_iam_group` **CONTAINS** `aws_iam_user`

`aws_iam_group` **HAS** `aws_iam_managed_policy`

`aws_iam_role` **HAS** `aws_iam_role_policy`

`aws_iam_role` **HAS** `aws_iam_managed_policy`

`aws_iam_user` **HAS** `aws_iam_managed_policy`

`aws_iam_user` **HAS** `aws_iam_user_policy`

`aws_lambda` **HAS** `aws_lambda_function`

`aws_lambda_function` **HAS** `aws_iam_role`

`aws_s3` **HAS** `aws_s3_bucket`

Connections to broader entity resources

`aws_iam_user` **IS** `Person` Note: This is mapped automatically only when the username of the IAM User is an email that matches the Person's.

Advanced mappings

The AWS integration performs analysis of security group rules, IAM policies, and assume role trust policies to determine the following mapping:

`User|Service|AccessRole` **CAN_ASSUME** `aws_iam_role`

Bitbucket Data and Integration Details

Overview

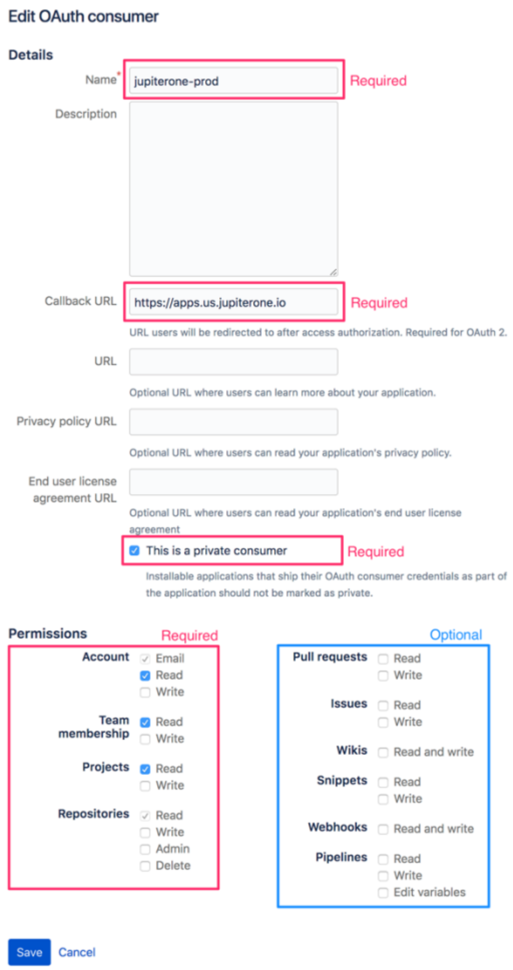
JupiterOne provides a managed integration with Bitbucket. The integration connects directly to Bitbucket APIs to obtain account metadata and analyze resource relationships. Customers authorize access by creating a Bitbucket OAuth App in their account and providing the app credentials to JupiterOne.

Integration Instance Configuration

The integration is triggered by an event containing the information for a specific integration instance.

The integration instance configuration requires the customer's Bitbucket OAuth App `clientId` and `clientSecret` to authenticate requests to the Bitbucket REST APIs. The integration requires Read access to the target Account, Team Membership, Projects, and Repositories.

See the following screenshot for an example configuration within a Bitbucket Team Settings, note the required and optional settings.



Edit OAuth consumer

Details

Name*: jupiterone-prod — Required

Description:

Callback URL: https://apps.us.jupiterone.io — Required

URL users will be redirected to after access authorization. Required for OAuth 2.

URL:

Optional URL where users can learn more about your application.

Privacy policy URL:

Optional URL where users can read your application's privacy policy.

End user license agreement URL:

Optional URL where users can read your application's end user license agreement

☑ This is a private consumer — Required

Installable applications that ship their OAuth consumer credentials as part of the application should not be marked as private.

Permissions — Required

- **Account**: ✓ Email, ☑ Read, □ Write
- **Team membership**: ☑ Read, □ Write
- **Projects**: ☑ Read, □ Write
- **Repositories**: ✓ Read, □ Write, □ Admin, □ Delete

Optional

- **Pull requests**: □ Read, □ Write
- **Issues**: □ Read, □ Write
- **Wikis**: □ Read and write
- **Snippets**: □ Read, □ Write
- **Webhooks**: □ Read and write
- **Pipelines**: □ Read, □ Write, □ Edit variables

BitBucket OAuth Example Config

Entities

The following entity resources are ingested when the integration runs:

Bitbucket Entity Resource	_type : _class of the Entity
Team	`bitbucket_team` : `Account`
Project	`bitbucket_project`: `Project`
Repository	`bitbucket_repo` : `CodeRepo`
User	`bitbucket_user` : `User`

Relationships

The following relationships are created/mapped:

Basic relationships within the integration instance account/resources

`bitbucket_team` **HAS** `bitbucket_project`

`bitbucket_team` **HAS** `bitbucket_user`

`bitbucket_project` **HAS** `bitbucket_repo`

GitHub Data and Integration Details

Overview

JupiterOne provides a managed integration with GitHub. The integration connects directly to GitHub APIs to obtain account metadata and analyze resource relationships. Customers authorize access by creating a GitHub OAuth App in their account and providing the app credentials to JupiterOne.

Integration Instance Configuration

The integration is triggered by an event containing the information for a specific integration instance.

The integration instance configuration requires the customer's GitHub OAuth App `clientId` and `clientSecret` to authenticate requests to the GitHub REST APIs. Detailed instructions for creating the OAuth App are provided by GitHub.

Entities

The following entity resources are ingested when the integration runs:

GitHub Entity Resource	_type : _class of the Entity
Account	`github_account` : `Account`
Repository	`github_repo` : `CodeRepo`
User	`github_user` : `User`

Relationships

The following relationships are created/mapped:

Basic relationships within the integration instance account/resources

`github_account` **OWNS** `github_repo`

`github_account` **HAS** `github_user`

Okta Data and Integration Details

Overview

JupiterOne provides a managed integration with Okta. The integration connects directly to Okta APIs to obtain account metadata and analyze resource relationships. Customers authorize access by creating an API token in your target Okta account and providing that credential to JupiterOne.

Integration Instance Configuration

The integration is triggered by an event containing the information for a specific integration instance.

Instructions on creating an API token within your Okta account can be found here.

Entities

The following entity resources are ingested when the integration runs:

Okta Entity Resource	_type : _class of the Entity
Account	`okta_account` : `Account`
Application	`okta_application` : `Application`
Group	`okta_group` : `Group`
User	`okta_user` : `User`

Relationships

The following relationships are created/mapped:

Basic relationships within the integration instance account/resources

`okta_account` **HAS** `okta_application`

`okta_account` **HAS** `okta_group`

`okta_group` **HAS** `okta_user`

`okta_group` **ASSIGNED** `okta_application`

`okta_user` **ASSIGNED** `okta_application`