F-Secure Atlant

Contents

Chapter

1

Introduction

Topics:

F-Secure Atlant is a platform for building applications that are able to scan and detect malicious files.

Atlant provides a REST API for scanning files and managing the product configuration. Applications and services can use the API resources to analyze content using always up-to-date heuristics and statistical techniques.

Atlant provides a scanning service to protect clients against viruses and other harmful files, exploits, network-based attacks, and other security threats. It is based on the ICAP protocol and provides content scanning and reputation services to any application on the network. You can use the scanning service with a HTTP proxy that supports ICAP protocol or create your own ICAP client for it.

This documentation gives you instructions on getting started with using Atlant as well as a reference of the available API resources.

1.1 Features and benefits

The development kit offers the scanning functionality that utilizes multiple scanning engines and cloud-based file and network reputation services.

When deployed, the scanning service:

- scans files, emails and URLs for harmful content,
- protects content in real-time using cloud-based scanning,
- includes cloud-based reputation services for both files and network addresses,
- scans both HTTP requests and responses,
- includes multiple scanning engines, and
- is always up to date.

Benefits

- Multiple scanning engines provide a layered protection against harmful content.
- F-Secure Cloud technology provides real-time and always up-to-date protection for both files and URLs.
- Provides URL filtering that supports multiple different content categories.
- Updates can be configured to meet your needs.
- Easy to deploy and configure for a wide range of purposes.
- The scanning service can be easily integrated with ICAP-compatible software.
- Easy to integrate with your own solutions to provide virus and spam scanning.

Basic and Premium differences

F-Secure Atlant is available in Basic and Premium versions.

Atlant Basic only supports local scanning and is best suited for isolated environments or uses that do not require the best possible level of protection.

Atlant Premium includes all the features of the Basic version, but also provides cloud-based features such as advanced scanning and sandboxing technology.

1.2 Cloud-based reputation services

Security Cloud is a cloud network that houses the various databases and automated analysis systems, which support and enhance the performance of F-Secure security products. The infrastructure for this network is hosted on servers in multiple data centers around the world.

Services connect to Security Cloud to retrieve the most up-to-date details of threats seen in the wild by other protected machines, which makes the response more efficient and effective. The service queries the reputation details of all objects, such as files and URLs. These queries contain anonymous metadata about the object, such as file size and anonymized path, are sent to the Security Cloud for combined data analysis. Security Cloud does not collect IP addresses or or other private information, queries are completely anonymous to maintain the privacy.

By evaluating the combined metadata with information drawn from the in-house databases and various other sources, the automated analysis systems provide a fully-informed, up-to-date risk assessment for the object, immediately blocking threats that have been seen previously by any other service or device that is connected to Security Cloud. This also removes the need to perform any further analysis of the object, which reduces the impact on the system resources that the service uses.

Security Cloud also allows Response Labs analysts to provide critical human intelligence and judgment to complement the automated systems and on-host scanning technology. In addition to creating and maintaining the rules that underpin the databases and automated analysis systems, analysts actively monitor the latest threats and research malware characteristics and behavior patterns to find the most effective ways to identify malicious programs.

1.3 About ICAP

The scanning service supports the Internet Content Adaption Protocol (ICAP). The ICAP is a protocol for sending HTTP requests and responses for another server for modifications.

The ICAP allows any client to pass messages to services for processing. The scanning service processes and modifies these messages when necessary and then sends back responses to clients, which then use the modified content instead of the original.

You can use existing ICAP compatible applications and proxies with the scanning service or create your own ICAP client to scan, remove, and block harmful content and filter unwanted web pages. The scanning service includes scan results as extended ICAP response headers and when used to scan the web traffic, it can show HTML block pages to users when they try to access infected content.

The ICAP protocol is documented in RFC 3507. For more information, see Internet Content Adaptation Protocol (ICAP).

Chapter

2

Installing F-Secure Atlant

Topics:

This section outlines how to install F-Secure Atlant as well as the supported Linux distributions and required dependencies.

2.1 System requirements

The supported Linux distributions and required dependencies for Atlant are listed here.

Atlant supports the following Linux distributions:

- CentOS 7
- CentOS 8
- RHEL 7
- RHEL 8
- Oracle Linux 7
- Oracle Linux 8
- Amazon Linux 2.0
- Debian 9
- Debian 10
- Ubuntu 16.04
- Ubuntu 18.04
- Ubuntu 20.04
- SUSE Linux Enterprise Server 12
- SUSE Linux Enterprise Server 15

Atlant requires the following packages to be installed before installing the product:

- CentOS 7, RHEL 7, Oracle Linux 7, Amazon Linux 2.0: `fuse-libs`, `libcurl`, `python`
- CentOS 8, RHEL 8, Oracle Linux 8: `fuse-libs`, `libcurl`, `python36`
- Debian 9, Ubuntu 16.04: `libfuse2`, `libcurl3`, `python`
- Debian 10, Ubuntu 18.04: `libfuse2`, `libcurl4`, `python`
- Ubuntu 20.04: `libfuse2`, `libcurl4`, `python3`
- SUSE Linux Enterprise Server 12, 15: `libfuse2`, `libcurl4`, `python3`

The hardware requirements for F-Secure Atlant depend on the use case. You can install Atlant on a computer with the following minimum requirements:

- Processor: x86-64 compatible CPU
- Memory: 2 GB RAM
- Disk space: At least 3 GB recommended

  Having sufficient swap memory is highly recommended.

  Additionally, sufficient disk space must be available for the temporary storage of all content submitted for malware analysis.

2.2 Standalone installation steps

Follow the instructions given here if you want to deploy Atlant on a Linux host and use it in standalone administration mode.

1. Run the installation package.
   - For distributions that belong to the Red Hat family, double click the `rpm` installation package or run the following command with root privileges:

     ```
     rpm -Uvh f-secure-atlant-XXX.x86_64.rpm
     ```

   - For Debian or Ubuntu distributions, run the following command with root privileges:

     ```
     dpkg -i f-secure-atlant_XXX.amd64.deb
     ```

2. Activate the product license.

Important: Once the product is activated, it automatically starts listening on port `1344`.

The command for activating the product depends on the kind of license you are using.

- If you are using a license key (a short string similar to `AAA-BBB-CCC-DDD`) to activate the product, run the following command:

```
# /opt/f-secure/atlant/atlant/bin/activate --license-key LICENSE-KEY
```

- If you are using a file-based license to activate the product, run the following command:

```
# /opt/f-secure/atlant/atlant/bin/activate --license-file PATH-TO-LICENSE-FILE
```

If you want to keep using a specific version of the product without automatically updating to the latest available version, use the `--product-version` option to indicate the version that you want in the `activate` command.

Note: To use an HTTP proxy during the activation process, specify the `FSECURE_HTTP_PROXY_HOST` and `FSECURE_HTTP_PROXY_PORT` environment variables. When these variables are set, the activation tool configures Atlant to use this proxy during the operation.

After activating the product license, the ICAP service is running and configured by default.

2.3 Installing Atlant as a virtual appliance

Follow the instructions given here if you want to deploy Atlant as a VMware virtual appliance and use it in standalone mode or manage it through Policy Manager.

Note: If you want to manage the product through F-Secure Policy Manager, you need to create the installation package in Policy Manager Console so that you can deploy it after you have integrated the virtual appliance. For more information, see Creating an installation package in Policy Manager on page 11.

1. Import the Atlant OVA image to your VMware hypervisor.
   For more detailed instructions, see the manual for your hypervisor.
2. Start the virtual machine.
   The `first-boot` setup tool appears.
3. Choose the keyboard layout that you use.
   a) Choose the keymap group from the list.
      Keymaps are grouped in alphabetical order. Choose the group that contains the keymap that you want to use.
   b) Choose the keymap.
   c) Select `y` when asked to set the keymap.
4. Choose the time zone that you use.
   a) Choose the time zone group from the list.
      Time zones are grouped in alphabetical order. Choose the group that contains the time zone that you want to use.
   b) Choose the time zone.
   c) Select `y` when asked to set the time zone.
5. Create the administrator account password.
   Enter the new password and confirm it to change it.
6. Select either the DHCP network type or configure your network settings manually.
7. Select whether or not you want to use an HTTP proxy during the activation.
8. Select the management mode for the appliance.
   To manage the appliance with F-Secure Policy Manager:
   a) Select `1`.
   b) Use the `scp` command shown in the `first-boot` setup tool to upload the installation package that you created in Policy Manager to the hypervisor.
   c) Press Enter.

The host is shown in the **Pending hosts** list in Policy Manager Console after the installation is complete. Import the host from the **Pending hosts** list to complete the activation of the product.

To manage the appliance in standalone administration mode:

a) Select `2`.
b) Select the license type. to enter your license key or to upload a license file.

If you are using a license key (a short string similar to AAA-BBB-CCC-DDD) to activate the product:

1. Select `1`.
2. Enter your license key.
3. Press Enter.

If you are using a file-based license to activate the product:

1. Select `2`.
2. Use the `scp` command shown in the `first-boot` setup tool to upload the license file.
3. Press Enter.

The installation has finished successfully as soon as the location of the license agreement is output. Press Enter to exit the installation.

2.4 Creating an installation package in Policy Manager

Follow the instructions given here if you want to use Atlant as a replacement for F-Secure Scanning and Reputation Server or manage it through F-Secure Policy Manager.

Note: For more information on using Policy Manager, see the Policy Manager administrator's guide.

1. In Policy Manager Console, select **Tools** > **Installation packages** from the menu.
   This opens the **Installation packages** window.
2. Click **Import**.
3. Select the Atlant installation package that you want to use, then click **Import**.
4. Select the imported installation package in the packages list and click **Export**.
5. Enter a name and select a folder for the exported `zip` file.
   A **Remote Installation Wizard** window opens.
6. Click **Next**.
7. Enter your license keycode for the product, then click **Next**.

   Note: This must be the Policy Manager Atlant installer activation keycode. If you do not yet have this keycode, contact customer support.

8. Enter the address for your Policy Manager Server and modify the ports to use for both HTTP and HTTPS communication if necessary.
9. Click **Finish**.

Related tasks

Installing Atlant as a virtual appliance on page 10
Follow the instructions given here if you want to deploy Atlant as a VMware virtual appliance and use it in standalone mode or manage it through Policy Manager.

2.5 Deploying the installation package on a Linux host

Follow these instructions to deploy the installation package that you created in Policy Manager on a target Linux computer. These steps are not needed if you deploy the Atlant virtual appliance.

1. Copy the exported `zip` installation package to the target Linux host in your network.
2. Install the product on the target host:
   a) Log in to the Linux host as `root`.
   b) Check that the required dependencies are installed:

- CentOS 7, RHEL 7, Oracle Linux 7, Amazon 2: `fuse-libs`, `libcurl`, `python`
- CentOS 8, RHEL 8, Oracle Linux 8: `fuse-libs`, `libcurl`, `python36`
- SUSE Linux Enterprise Server: `libfuse2`, `libcurl4`, `python3`
- Debian 9, Ubuntu 16.04: `libfuse2`, `libcurl3`, `python`
- Debian 10, Ubuntu 18.04: `libfuse2`, `libcurl4`, `python`
- Ubuntu 20.04: `libfuse2`, `libcurl4`, `python3`

c) Extract the installation package that you exported from Policy Manager Console to a fresh, empty directory.
d) Run the installation command:

```
bash f-secure-atlant/f-secure-atlant-installer
```

The installation has finished successfully as soon as the location of the license agreement is output.

The host is shown in the **Pending hosts** list in Policy Manager Console after the installation is complete.

2.6 Uninstallation

This topic describes how to uninstall the product.

1. Log in to the Linux host as `root`.
2. Run the uninstallation command:
   - RHEL-based distributions: `rpm -e f-secure-atlant`
   - Debian-based distributions: `dpkg -r f-secure-atlant`

Chapter

3

Setting up F-Secure Atlant

Topics:

This section outlines the steps needed for a minimal Atlant setup.

The steps given here assume that you have already activated Atlant. You also need a valid TLS certificate and corresponding PEM key file.

3.1 Set up HTTPS

All communication with Atlant's REST API happens over secure HTTPS connections, so the first step in setting up Atlant is to provide it with a TLS certificate and a key.

Note: The command path given here is for the current version of the product. If you are using an older version, use `/opt/f-secure/atlant/bin/<command>` as the path instead.

Run the following command to set up TLS:

```
/opt/f-secure/atlant/atlant/bin/atlantctl set tls '{"certificate": "/root/cert.pem", "key":
"/root/key.pem"}'
```

This command example assumes that the path for your certificate is `/root/cert.pem` and the path for the key is `/root/key.pem`.

Note: Make sure that the certificate is trusted by the clients.

3.1.1 About TLS/SSL certificates

A TLS certificate is required for handling the HTTPS communication with Atlant's REST API.

Certificates that are used by public web servers are usually created by some known, trusted certificate authority (CA). For internal servers, you can be your own trusted authority and use a self-signed root certificate that all internal clients trust. As all other certificates rely on the trust on the root certificate, make sure that it is very well secured.

Typically, root certificates are used to sign intermediate CA certificates, which are then used to sign other certificates. Signing certificates this way means that you do not need to access the root private key very often, which keeps it more secure.

Certificates can be revoked if they get compromised. Web browsers should warn users about revoked certificates and even block the access to web sites using them. However, web browsers do not always receive the information about revoked certificates reliably so they may still show that the certificate is valid even when it has been revoked. To prevent this, use short validity times for certificates and replace them often. If a certificate gets compromised, it invalidates itself when its validity expires.

When attaching a certificate to a server, you usually provide a certificate chain and not just the leaf certificate. A certificate chain consists of the needed certificate and the certificates that were used to sign it up to the root certificate. The chain of trust ensures that the leaf certificate and all the intermediate certificates are trusted because the root certificate is trusted.

3.2 Set up Atlant services

When your TLS certificate is set up, the next step is to enable the services that clients use.

Note: The command path given here is for the current version of the product. If you are using an older version, use `/opt/f-secure/atlant/bin/<command>` as the path instead.

Clients can authenticate with Atlant's internal authorization server using OAuth 2.0. Before that, you have to bind the internal authorization server to an address to enable it.

Note: For more complicated setups, you can configure Atlant to use an external authorization server. This can be useful for scenarios where you want to use an external client database.

1. Run the following command to bind the internal authorization server to an address:

   ```
   /opt/f-secure/atlant/atlant/bin/atlantctl add authorization https_endpoints '{"address":
   "localhost", "port": 8081}'
   ```

   This command enables the authorization server to listen on port `8081`. The server uses the TLS certificate specified previously.
2. Run the following command to enable the management server, which allows clients to change Atlant's configuration:

   ```
   /opt/f-secure/atlant/atlant/bin/atlantctl add management https_endpoints '{"address":
   "localhost", "port": 8082}'
   ```

This command enables the management server to listen on port `8082`. The server uses the TLS certificate specified previously.

3. Run the following command to enable Atlant's scanning server:

```
/opt/f-secure/atlant/atlant/bin/atlantctl add scanning https_endpoints '{"address":
"localhost", "port": 8083}'
```

The scanning server is required for clients to scan files. This command enables the scanning server to listen on port `8083`. The server uses the TLS certificate specified previously.

3.3 Create the first client

After you have enabled all the services, you can create the first API client.

Note: The command path given here is for the current version of the product. If you are using an older version, use `/opt/f-secure/atlant/bin/<command>` as the path instead.

Run the following command to create the initial client:

```
/opt/f-secure/atlant/atlant/bin/atlantctl client create '{"scopes": ["management", "scan"]}'
```

This command adds a new client that can inspect and change the Atlant configuration and scan files. You can also specify only the `scan` scope to create a client that scans files but cannot edit the Atlant configuration. The command will return the following type of response:

```
{
  "client_id": "5dbfe97de42bf53a0ae73bff9eba4ecb",
  "client_secret": "87e7b3a61e7fdcdfc03162fdcab82a942b9c62715ff3d612e15adf32d1b1500b"
}
```

The response contains the client ID and the secret of the new client. Store the client secret as this is the only time it is given to you. These are the credentials that the client uses to request access tokens from the authorization server.

Chapter

4

API reference

Topics:

This section contains the details for all API resources available in F-Secure Atlant.

Examples for use with the Atlant API are available online at https://github.com/F-Secure/atlant-api.

4.1 Product license

Resource URI: `/api/atlant/config/v1/license`

Methods: `GET` `PUT`

Data type: `object`

GET

Example request:

```
GET /api/atlant/config/v1/license HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "content": "XXXX-XXXX-XXXX-XXXX-XXXX",
  "remote_license_key": "XXXX-XXXX-XXXX-XXXX-XXXX",
  "type": "key"
}
```

PUT

Example request:

```
PUT /api/atlant/config/v1/license HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

{
  "content": "XXXX-XXXX-XXXX-XXXX-XXXX",
  "remote_license_key": "XXXX-XXXX-XXXX-XXXX-XXXX",
  "type": "key"
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "content": "XXXX-XXXX-XXXX-XXXX-XXXX",
  "remote_license_key": "XXXX-XXXX-XXXX-XXXX-XXXX",
  "type": "key"
}
```

4.2 Authentication

OAuth 2.0 token endpoint for requesting new access tokens. Client credentials is the only supported grant type.

All requests to Atlant REST API require authentication. When making a request to the API, client needs to include `Authorization` header containing a valid access token. Client can request an access token from an authorization server. Atlant installation can be configured to use either the internal authorization server that ships with the product or an external authorization server.

Resource URI: `/api/token/v1`

Methods: `POST`

Data type: `object`

POST

Request a new access token using client credentials grant type. Response will contain access token that must be included in the `Authorization` header when making requests to other Atlant APIs. By default, the returned token will enable token bearer to access all the scopes available to the client.

Example request:

```
POST /api/token/v1 HTTP/1.1
Content-Type: application/x-www-form-urlencoded

grant_type=client_credentials&audience=f-secure-atlant&client_id=CLIENT_ID&client_secret=CLIENT_SECRET
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8

{
  "access_token": "ACCESS_TOKEN",
  "token_type": "Bearer",
  "expires_in": 3600
}
```

4.3 Client management

Client management endpoint allows the creation, deletion, and listing of API clients.

Client management is only available when using the internal authorization server.

Resource URI: `/api/atlant/clients/v1`

Methods: `GET` `POST` `DELETE`

Data type: `object`

GET

Return a list of clients and their scopes.

Example request:

```
GET /api/atlant/clients/v1/ HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Authorization: Bearer TOKEN

{
  "clients": [
    {
      "client_id": "CLIENT_ID_1",
      "scopes": ["scan"]
    },
    {
      "client_id": "CLIENT_ID_2",
      "scopes": ["scan", "management"]
    }
  ]
}
```

POST

Create a client. Caller needs to specify a list of scopes that the client is allowed to access. Response will contain a newly-generated client ID and client secret for the client.

Example request:

```
POST /api/atlant/clients/v1/ HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

{
  "scopes": ["scan", "management"]
}
```

Example response:

```
HTTP/1.1 201 Created
Content-Type: application/json

{
  "client_id": "CLIENT_ID",
  "client_secret": "CLIENT_SECRET"
}
```

DELETE

Remove a client. Caller needs to specify the client ID of the client to be removed.

Example request:

```
DELETE /api/atlant/clients/v1/ HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

{
  "client_id": "CLIENT_ID"
}
```

Example response:

```
HTTP/1.1 204 No Content
```

4.4 File scanning

The scanning API enables clients to scan files for harmful content.

Resource URI: `/api/scan/v1`

Methods: `POST` `GET`

`POST` requests to this path must contain a `multipart/form-data` body, with either one or two parts; the first part must be of type `application/json` and specifies the scan settings and content metadata as a serialized JSON object, structured according to the following JSON schema:

```
{
    "type": "object",
    "properties": {
        "scan_settings": {
            "description":
            "Scan settings.",
            "type": "object",
            "properties": {
                "scan_archives": {
                    "description":
                    "Enable archive scanning.",
                    "type": "boolean",
                    "default": true
                },
                "max_nested": {
                    "description": "Scan nested archives up to this level.",
                    "type": "integer",
                    "minimum": 0,
                    "default": 5
                },
                "max_scan_time": {
```

```
                    "description": "Maximum scan time in seconds.",
                    "type": "integer",
                    "minimum": 1,
                    "default": 600
                },
                "stop_on_first": {
                    "description":
                    "Stop scanning after the first detection.",
                    "type": "boolean",
                    "default": true
                },
                "allow_upstream_metadata": {
                    "description":
                    "Allow upstreaming of file metadata.",
                    "type": "boolean",
                    "default": true
                },
                "security_cloud": {
                    "description":
                    "Use advanced cloud analysis.",
                    "type": "object",
                    "properties": {
                        "allow_upstream_application_files": {
                            "description":
                            "Allow upstreaming of application files.",
                            "type": "boolean",
                            "default": true
                        },
                        "allow_upstream_data_files": {
                            "description":
                            "Allow upstreaming of data files.",
                            "type": "boolean",
                            "default": false
                        }
                    }
                }
            }
        },
        "content_meta": {
            "description":
            "Content metadata.",
            "type": "object",
            "properties": {
                "sha1": {
                    "description":
                    "The content sha1 digest.",
                    "type": "string"
                },
                "uri": {
                    "description":
                    "The content source URI.",
                    "type": "string"
                },
                "content_length": {
                    "description":
                    "The content length.",
                    "type": "integer"
                },
                "content_type": {
                    "description":
                    "The content type.",
                    "type": "string"
                },
                "charset": {
                    "description":
                    "The content character set.",
                    "type": "string"
                }
            }
        }
    }
}
```


The second part, if present, contains the content and must be in binary format. The `Content-Transfer-Encoding` header is not supported.

If the response status is `202`, the analysis is not complete and clients can poll for an updated result by sending a `GET` request for the resource specified by the `Location` header. Responses for both `POST` and `GET` requests contain a body of type `application/json` structured according to the following schema:

```
{
    "type": "object",
    "properties": {
        "scan_result": {
            "description":
            "Scan result.",
            "type": "string",
            "enum": ["clean", "whitelisted", "suspicious", "PUA", "UA", "harmful"]
        },
        "detections": {
            "type": "array",
            "items": {
                "type": "object",
                "properties": {
                    "category": {
                        "description": "Detection category.",
                        "type": "string",
                        "enum": ["suspicious", "PUA", "UA", "harmful"]
                    },
                    "name": {
                        "description": "Detection name.",
                        "type": "string"
                    },
                    "member_name": {
                        "description": "Logical path to the archive member that triggered the
 detection.",
                        "type": "string"
                    }
                },
                "required": [
                    "category",
                    "name"
                ]
            }
        },
        "status": {
            "description":
            "Task status.",
            "type": "string",
            "enum": ["pending", "complete"]
        },
        "warnings": {
            "type": "object",
            "properties": {
                "corrupted": {
                   "description": "The server was unable to fully analyze the content because
 some data is corrupted",
                    "type": "boolean"
                },
                "encrypted": {
                   "description": "The server was unable to fully analyze the content because
 some data is encrypted",
                    "type": "boolean"
                },
                "max_nested": {
                    "description": "The server was unable to fully analyze the content with
the given level limit for nested archives",
                    "type": "boolean"
                },
                "max_scan_time": {
                    "description": "The server was unable to fully analyze the content with
the given scan time limit",
                    "type": "boolean"
                }
            }
        }
    },
    "required": [
        "scan_result",
        "status"
    ]
}
```

The status of the analysis is included in the body. If the status is `"pending"`, the response also includes a `Retry-After` header that indicates how many seconds the client should wait before polling for an updated result. If the license verification fails, `fsicapd` rejects the request and replies with status code `401`.

4.5 Checking the product status

The status API enables clients to query engine and server versions.

Resource URI: `/api/status/v1`

Methods: `GET`

This retrieves the status information. The response contains an `application/json` body structured according to the following schema:

```
{
    "type": "object",
    "properties": {
        "engines": {
            "description": "Engines.",
            "type": "array",
            "items": {
                "type": "object",
                "properties": {
                    "name": {
                        "description": "Engine name.",
                        "type": "string"
                    },
                    "version": {
                        "description": "Engine version.",
                        "type": "string"
                    },
                    "db_version": {
                        "descripton": "Engine database version",
                        "type": "string"
                    },
                    "release_timestamp": {
                        "description": "Engine timestamp",
                        "type": "integer"
                    }
                }
            },
            "required": [
                "name",
                "version",
                "db_version",
                "release_timestamp"
            ]
        },
        "version": {
            "description": "Server version.",
            "type": "string"
        }
    },
    "required": [
        "engines"
    ]
}
```

4.6 Authorization server settings

Resource URI: `/api/atlant/config/v1/authorization`

Methods: `GET PUT DELETE`

Data type: `object`

GET

Example request:

```
GET /api/atlant/config/v1/authorization HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json
```

```
{
  "external_domains": [
    {
      "audience": "example-audience",
      "issuer": "example-issuer",
      "signing_method": "HS256",
      "verification_key": "cGFzc3dvcmQK"
    }
  ],
  "https_endpoints": [
    {
      "address": "127.0.0.1",
      "port": 8080
    }
  ]
}
```

PUT

Example request:

```
PUT /api/atlant/config/v1/authorization HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

{
  "external_domains": [
    {
      "audience": "example-audience",
      "issuer": "example-issuer",
      "signing_method": "HS256",
      "verification_key": "cGFzc3dvcmQK"
    }
  ],
  "https_endpoints": [
    {
      "address": "127.0.0.1",
      "port": 8080
    }
  ]
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "external_domains": [
    {
      "audience": "example-audience",
      "issuer": "example-issuer",
      "signing_method": "HS256",
      "verification_key": "cGFzc3dvcmQK"
    }
  ],
  "https_endpoints": [
    {
      "address": "127.0.0.1",
      "port": 8080
    }
  ]
}
```

DELETE

Example request:

```
DELETE /api/atlant/config/v1/authorization HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 204 No Content
```

4.6.1 List of external authorization domains

Resource URI: `/api/atlant/config/v1/authorization/external_domains`

Methods: `GET PUT POST DELETE`

Data type: `array`

GET

Example request:

```
GET /api/atlant/config/v1/authorization/external_domains HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "audience": "example-audience",
    "issuer": "example-issuer",
    "signing_method": "HS256",
    "verification_key": "cGFzc3dvcmQK"
  }
]
```

PUT

Example request:

```
PUT /api/atlant/config/v1/authorization/external_domains HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

[
  {
    "audience": "example-audience",
    "issuer": "example-issuer",
    "signing_method": "HS256",
    "verification_key": "cGFzc3dvcmQK"
  }
]
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "audience": "example-audience",
    "issuer": "example-issuer",
    "signing_method": "HS256",
    "verification_key": "cGFzc3dvcmQK"
  }
]
```

POST

Example request:

```
POST /api/atlant/config/v1/authorization/external_domains HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

{
  "audience": "example-audience",
  "issuer": "example-issuer",
  "signing_method": "HS256",
  "verification_key": "cGFzc3dvcmQK"
}
```

Example response:

```
HTTP/1.1 201 Created
Content-Type: application/json
Location: URL

{
  "audience": "example-audience",
  "issuer": "example-issuer",
  "signing_method": "HS256",
  "verification_key": "cGFzc3dvcmQK"
}
```

DELETE

Example request:

```
DELETE /api/atlant/config/v1/authorization/external_domains HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 204 No Content
```

External authorization domain

Keys:

- DOMAIN

  Example: `"string"`

Key should be URL-encoded

Resource URI: `/api/atlant/config/v1/authorization/external_domains/DOMAIN`

Methods: `GET PUT DELETE`

Data type: `object`

GET

Example request:

```
GET /api/atlant/config/v1/authorization/external_domains/DOMAIN HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "audience": "example-audience",
  "issuer": "example-issuer",
  "signing_method": "HS256",
  "verification_key": "cGFzc3dvcmQK"
}
```

PUT

Example request:

```
PUT /api/atlant/config/v1/authorization/external_domains/DOMAIN HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

{
  "audience": "example-audience",
  "issuer": "example-issuer",
  "signing_method": "HS256",
```

```
  "verification_key": "cGFzc3dvcmQK"
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "audience": "example-audience",
  "issuer": "example-issuer",
  "signing_method": "HS256",
  "verification_key": "cGFzc3dvcmQK"
}
```

DELETE

Example request:

```
DELETE /api/atlant/config/v1/authorization/external_domains/DOMAIN HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 204 No Content
```

Domain audience

Keys:

- DOMAIN

  Example: `"string"`

Key should be URL-encoded

Resource URI: `/api/atlant/config/v1/authorization/external_domains/DOMAIN/audience`

Methods: `GET` `PUT`

Data type: `string`

GET

Example request:

```
GET /api/atlant/config/v1/authorization/external_domains/DOMAIN/audience HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

"example-audience"
```

PUT

Example request:

```
PUT /api/atlant/config/v1/authorization/external_domains/DOMAIN/audience HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

"example-audience"
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json
```

```
"example-audience"
```

Domain issuer

Keys:

- DOMAIN

  Example: `"string"`

Key should be URL-encoded

Resource URI: `/api/atlant/config/v1/authorization/external_domains/DOMAIN/issuer`

Methods: `GET` `PUT`

Data type: `string`

GET

Example request:

```
GET /api/atlant/config/v1/authorization/external_domains/DOMAIN/issuer HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

"example-issuer"
```

PUT

Example request:

```
PUT /api/atlant/config/v1/authorization/external_domains/DOMAIN/issuer HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

"example-issuer"
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

"example-issuer"
```

Domain signing method

Value must be one of `"HS256"`, `"RS256"`

Keys:

- DOMAIN

  Example: `"string"`

Key should be URL-encoded

Resource URI:
`/api/atlant/config/v1/authorization/external_domains/DOMAIN/signing_method`

Methods: `GET` `PUT`

Data type: `string`

GET

Example request:

```
GET /api/atlant/config/v1/authorization/external_domains/DOMAIN/signing_method HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

"HS256"
```

PUT

Example request:

```
PUT /api/atlant/config/v1/authorization/external_domains/DOMAIN/signing_method HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

"HS256"
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

"HS256"
```

Domain verification key

Keys:

- DOMAIN

  Example: `"string"`

Key should be URL-encoded

Resource URI:
`/api/atlant/config/v1/authorization/external_domains/DOMAIN/verification_key`

Methods: `GET` `PUT`

Data type: `string`

GET

Example request:

```
GET /api/atlant/config/v1/authorization/external_domains/DOMAIN/verification_key HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

"cGFzc3dvcmQK"
```

PUT

Example request:

```
PUT /api/atlant/config/v1/authorization/external_domains/DOMAIN/verification_key HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

"cGFzc3dvcmQK"
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

"cGFzc3dvcmQK"
```

4.6.2 List of authorization service HTTP endpoints

Resource URI: `/api/atlant/config/v1/authorization/https_endpoints`

Methods: `GET PUT POST DELETE`

Data type: `array`

GET

Example request:

```
GET /api/atlant/config/v1/authorization/https_endpoints HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "address": "127.0.0.1",
    "port": 8080
  }
]
```

PUT

Example request:

```
PUT /api/atlant/config/v1/authorization/https_endpoints HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

[
  {
    "address": "127.0.0.1",
    "port": 8080
  }
]
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "address": "127.0.0.1",
    "port": 8080
  }
]
```

POST

Example request:

```
POST /api/atlant/config/v1/authorization/https_endpoints HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

{
  "address": "127.0.0.1",
  "port": 8080
}
```

Example response:

```
HTTP/1.1 201 Created
Content-Type: application/json
Location: URL

{
  "address": "127.0.0.1",
  "port": 8080
}
```

DELETE

Example request:

```
DELETE /api/atlant/config/v1/authorization/https_endpoints HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 204 No Content
```

Authorization service HTTP endpoint

Keys:

- ENDPOINT

  Example: `{"address":"string","port":0}`

Key should be URL-encoded

Resource URI: `/api/atlant/config/v1/authorization/https_endpoints/ENDPOINT`

Methods: `GET PUT DELETE`

Data type: `object`

GET

Example request:

```
GET /api/atlant/config/v1/authorization/https_endpoints/ENDPOINT HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "address": "127.0.0.1",
  "port": 8080
}
```

PUT

Example request:

```
PUT /api/atlant/config/v1/authorization/https_endpoints/ENDPOINT HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

{
  "address": "127.0.0.1",
  "port": 8080
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "address": "127.0.0.1",
  "port": 8080
}
```

DELETE

Example request:

```
DELETE /api/atlant/config/v1/authorization/https_endpoints/ENDPOINT HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 204 No Content
```

Authorization service HTTP endpoint address

Keys:

- ENDPOINT

  Example: `{"address":"string","port":0}`

Key should be URL-encoded

Resource URI: `/api/atlant/config/v1/authorization/https_endpoints/ENDPOINT/address`

Methods: `GET` `PUT`

Data type: `string`

GET

Example request:

```
GET /api/atlant/config/v1/authorization/https_endpoints/ENDPOINT/address HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

"127.0.0.1"
```

PUT

Example request:

```
PUT /api/atlant/config/v1/authorization/https_endpoints/ENDPOINT/address HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

"127.0.0.1"
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

"127.0.0.1"
```

Authorization service HTTP endpoint port

Keys:

- ENDPOINT

  Example: `{"address":"string","port":0}`

Key should be URL-encoded

Resource URI: `/api/atlant/config/v1/authorization/https_endpoints/ENDPOINT/port`

Methods: `GET PUT`

Data type: `number`

GET

Example request:

```
GET /api/atlant/config/v1/authorization/https_endpoints/ENDPOINT/port HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

8080
```

PUT

Example request:

```
PUT /api/atlant/config/v1/authorization/https_endpoints/ENDPOINT/port HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

8080
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

8080
```

4.7 TLS data

Resource URI: `/api/atlant/config/v1/tls`

Methods: `GET PUT DELETE`

Data type: `object`

GET

Example request:

```
GET /api/atlant/config/v1/tls HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "certificate": "a2V5Cg==",
```

```
  "key": "Y2VydGlmaWNhdGU="
}
```

PUT

Example request:

```
PUT /api/atlant/config/v1/tls HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

{
  "certificate": "a2V5Cg==",
  "key": "Y2VydGlmaWNhdGU="
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "certificate": "a2V5Cg==",
  "key": "Y2VydGlmaWNhdGU="
}
```

DELETE

Example request:

```
DELETE /api/atlant/config/v1/tls HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 204 No Content
```

4.7.1 TLS certificate

Resource URI: `/api/atlant/config/v1/tls/certificate`

Methods: `GET PUT DELETE`

Data type: `string`

GET

Example request:

```
GET /api/atlant/config/v1/tls/certificate HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

"a2V5Cg=="
```

PUT

Example request:

```
PUT /api/atlant/config/v1/tls/certificate HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

"a2V5Cg=="
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

"a2V5Cg=="
```

DELETE

Example request:

```
DELETE /api/atlant/config/v1/tls/certificate HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 204 No Content
```

4.7.2 TLS key

Resource URI: `/api/atlant/config/v1/tls/key`

Methods: `GET` `PUT` `DELETE`

Data type: `string`

GET

Example request:

```
GET /api/atlant/config/v1/tls/key HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

"Y2VydGlmaWNhdGU="
```

PUT

Example request:

```
PUT /api/atlant/config/v1/tls/key HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

"Y2VydGlmaWNhdGU="
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

"Y2VydGlmaWNhdGU="
```

DELETE

Example request:

```
DELETE /api/atlant/config/v1/tls/key HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 204 No Content
```

4.8 Management server settings

Resource URI: `/api/atlant/config/v1/management`

Methods: `GET PUT DELETE`

Data type: `object`

GET

Example request:

```
GET /api/atlant/config/v1/management HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "https_endpoints": [
    {
      "address": "127.0.0.1",
      "port": 8081
    }
  ]
}
```

PUT

Example request:

```
PUT /api/atlant/config/v1/management HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

{
  "https_endpoints": [
    {
      "address": "127.0.0.1",
      "port": 8081
    }
  ]
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "https_endpoints": [
    {
      "address": "127.0.0.1",
      "port": 8081
    }
  ]
}
```

DELETE

Example request:

```
DELETE /api/atlant/config/v1/management HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 204 No Content
```

4.8.1 List of management server endpoints

Resource URI: `/api/atlant/config/v1/management/https_endpoints`

Methods: `GET PUT POST DELETE`

Data type: `array`

GET

Example request:

```
GET /api/atlant/config/v1/management/https_endpoints HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "address": "127.0.0.1",
    "port": 8081
  }
]
```

PUT

Example request:

```
PUT /api/atlant/config/v1/management/https_endpoints HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

[
  {
    "address": "127.0.0.1",
    "port": 8081
  }
]
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "address": "127.0.0.1",
    "port": 8081
  }
]
```

POST

Example request:

```
POST /api/atlant/config/v1/management/https_endpoints HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

{
  "address": "127.0.0.1",
  "port": 8081
}
```

Example response:

```
HTTP/1.1 201 Created
Content-Type: application/json
Location: URL

{
```

```
  "address": "127.0.0.1",
  "port": 8081
}
```

DELETE

Example request:

```
DELETE /api/atlant/config/v1/management/https_endpoints HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 204 No Content
```

4.8.2 Management server endpoint

Keys:

- ENDPOINT

  Example: `{"address":"string","port":0}`

Key should be URL-encoded

Resource URI: `/api/atlant/config/v1/management/https_endpoints/ENDPOINT`

Methods: `GET PUT DELETE`

Data type: `object`

GET

Example request:

```
GET /api/atlant/config/v1/management/https_endpoints/ENDPOINT HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "address": "127.0.0.1",
  "port": 8081
}
```

PUT

Example request:

```
PUT /api/atlant/config/v1/management/https_endpoints/ENDPOINT HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

{
  "address": "127.0.0.1",
  "port": 8081
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "address": "127.0.0.1",
  "port": 8081
}
```

DELETE

Example request:

```
DELETE /api/atlant/config/v1/management/https_endpoints/ENDPOINT HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 204 No Content
```

4.8.3 Management server endpoint address

Keys:

- ENDPOINT

  Example: `{"address":"string","port":0}`

Key should be URL-encoded

Resource URI: `/api/atlant/config/v1/management/https_endpoints/ENDPOINT/address`

Methods: `GET PUT`

Data type: `string`

GET

Example request:

```
GET /api/atlant/config/v1/management/https_endpoints/ENDPOINT/address HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

"127.0.0.1"
```

PUT

Example request:

```
PUT /api/atlant/config/v1/management/https_endpoints/ENDPOINT/address HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

"127.0.0.1"
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

"127.0.0.1"
```

4.8.4 Management server endpoint port

Keys:

- ENDPOINT

  Example: `{"address":"string","port":0}`

Key should be URL-encoded

Resource URI: `/api/atlant/config/v1/management/https_endpoints/ENDPOINT/port`

Methods: `GET` `PUT`

Data type: `number`

GET

Example request:

```
GET /api/atlant/config/v1/management/https_endpoints/ENDPOINT/port HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

8081
```

PUT

Example request:

```
PUT /api/atlant/config/v1/management/https_endpoints/ENDPOINT/port HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

8081
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

8081
```

4.9 HTTP proxy settings

Resource URI: `/api/atlant/config/v1/http_proxy`

Methods: `GET` `PUT` `DELETE`

Data type: `object`

GET

Example request:

```
GET /api/atlant/config/v1/http_proxy HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "enabled": true,
  "host": "example.com",
  "port": 3128
}
```

PUT

Example request:

```
PUT /api/atlant/config/v1/http_proxy HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

{
```

```
  "enabled": true,
  "host": "example.com",
  "port": 3128
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "enabled": true,
  "host": "example.com",
  "port": 3128
}
```

DELETE

Example request:

```
DELETE /api/atlant/config/v1/http_proxy HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 204 No Content
```

4.9.1 HTTP proxy enabled flag

Resource URI: `/api/atlant/config/v1/http_proxy/enabled`

Methods: `GET PUT DELETE`

Data type: `boolean`

GET

Example request:

```
GET /api/atlant/config/v1/http_proxy/enabled HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

true
```

PUT

Example request:

```
PUT /api/atlant/config/v1/http_proxy/enabled HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

true
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

true
```

DELETE

Example request:

```
DELETE /api/atlant/config/v1/http_proxy/enabled HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 204 No Content
```

4.9.2 HTTP proxy host

Resource URI: `/api/atlant/config/v1/http_proxy/host`

Methods: `GET PUT DELETE`

Data type: `string`

GET

Example request:

```
GET /api/atlant/config/v1/http_proxy/host HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

"example.com"
```

PUT

Example request:

```
PUT /api/atlant/config/v1/http_proxy/host HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

"example.com"
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

"example.com"
```

DELETE

Example request:

```
DELETE /api/atlant/config/v1/http_proxy/host HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 204 No Content
```

4.9.3 HTTP proxy port

Resource URI: `/api/atlant/config/v1/http_proxy/port`

Methods: `GET PUT DELETE`

Data type: `number`

GET

Example request:

```
GET /api/atlant/config/v1/http_proxy/port HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

3128
```

PUT

Example request:

```
PUT /api/atlant/config/v1/http_proxy/port HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

3128
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

3128
```

DELETE

Example request:

```
DELETE /api/atlant/config/v1/http_proxy/port HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 204 No Content
```

4.10 Scanning server settings

Resource URI: `/api/atlant/config/v1/scanning`

Methods: `GET PUT DELETE`

Data type: `object`

GET

Example request:

```
GET /api/atlant/config/v1/scanning HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "archive_max_nested": 5,
  "https_endpoints": [
    {
      "address": "127.0.0.1",
      "port": 8082
    }
  ],
```

```
  "icap_endpoints": [
    {
      "address": "127.0.0.1",
      "port": 1344
    }
  ],
  "keepalive_time": 600,
  "max_scan_time": 90,
  "reputation": {
    "file_check": true,
    "timeout": 5,
    "url_check": true
  }
}
```

PUT

Example request:

```
PUT /api/atlant/config/v1/scanning HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

{
  "archive_max_nested": 5,
  "https_endpoints": [
    {
      "address": "127.0.0.1",
      "port": 8082
    }
  ],
  "icap_endpoints": [
    {
      "address": "127.0.0.1",
      "port": 1344
    }
  ],
  "keepalive_time": 600,
  "max_scan_time": 90,
  "reputation": {
    "file_check": true,
    "timeout": 5,
    "url_check": true
  }
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "archive_max_nested": 5,
  "https_endpoints": [
    {
      "address": "127.0.0.1",
      "port": 8082
    }
  ],
  "icap_endpoints": [
    {
      "address": "127.0.0.1",
      "port": 1344
    }
  ],
  "keepalive_time": 600,
  "max_scan_time": 90,
  "reputation": {
    "file_check": true,
    "timeout": 5,
    "url_check": true
  }
}
```

DELETE

Example request:

```
DELETE /api/atlant/config/v1/scanning HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 204 No Content
```

4.10.1 List of HTTP scanning endpoints

Resource URI: `/api/atlant/config/v1/scanning/https_endpoints`

Methods: `GET PUT POST DELETE`

Data type: `array`

GET

Example request:

```
GET /api/atlant/config/v1/scanning/https_endpoints HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "address": "127.0.0.1",
    "port": 8082
  }
]
```

PUT

Example request:

```
PUT /api/atlant/config/v1/scanning/https_endpoints HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

[
  {
    "address": "127.0.0.1",
    "port": 8082
  }
]
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "address": "127.0.0.1",
    "port": 8082
  }
]
```

POST

Example request:

```
POST /api/atlant/config/v1/scanning/https_endpoints HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN
```

```
{
  "address": "127.0.0.1",
  "port": 8082
}
```

Example response:

```
HTTP/1.1 201 Created
Content-Type: application/json
Location: URL

{
  "address": "127.0.0.1",
  "port": 8082
}
```

DELETE

Example request:

```
DELETE /api/atlant/config/v1/scanning/https_endpoints HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 204 No Content
```

HTTP scanning endpoint

Keys:

- ENDPOINT

  Example: `{"address":"string","port":0}`

Key should be URL-encoded

Resource URI: `/api/atlant/config/v1/scanning/https_endpoints/ENDPOINT`

Methods: `GET PUT DELETE`

Data type: `object`

GET

Example request:

```
GET /api/atlant/config/v1/scanning/https_endpoints/ENDPOINT HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "address": "127.0.0.1",
  "port": 8082
}
```

PUT

Example request:

```
PUT /api/atlant/config/v1/scanning/https_endpoints/ENDPOINT HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

{
  "address": "127.0.0.1",
```

```
  "port": 8082
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "address": "127.0.0.1",
  "port": 8082
}
```

DELETE

Example request:

```
DELETE /api/atlant/config/v1/scanning/https_endpoints/ENDPOINT HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 204 No Content
```

HTTP scanning endpoint address

Keys:

- ENDPOINT

  Example: `{"address":"string","port":0}`

Key should be URL-encoded

Resource URI: `/api/atlant/config/v1/scanning/https_endpoints/ENDPOINT/address`

Methods: `GET` `PUT`

Data type: `string`

GET

Example request:

```
GET /api/atlant/config/v1/scanning/https_endpoints/ENDPOINT/address HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

"127.0.0.1"
```

PUT

Example request:

```
PUT /api/atlant/config/v1/scanning/https_endpoints/ENDPOINT/address HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

"127.0.0.1"
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

"127.0.0.1"
```

HTTP scanning endpoint port

Keys:

- ENDPOINT

  Example: `{"address":"string","port":0}`

Key should be URL-encoded

Resource URI: `/api/atlant/config/v1/scanning/https_endpoints/ENDPOINT/port`

Methods: `GET PUT`

Data type: `number`

GET

Example request:

```
GET /api/atlant/config/v1/scanning/https_endpoints/ENDPOINT/port HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

8082
```

PUT

Example request:

```
PUT /api/atlant/config/v1/scanning/https_endpoints/ENDPOINT/port HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

8082
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

8082
```

4.10.2 List of ICAP scanning endpoints

Resource URI: `/api/atlant/config/v1/scanning/icap_endpoints`

Methods: `GET PUT POST DELETE`

Data type: `array`

GET

Example request:

```
GET /api/atlant/config/v1/scanning/icap_endpoints HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "address": "127.0.0.1",
    "port": 1344
```

```
  }
]
```

PUT

Example request:

```
PUT /api/atlant/config/v1/scanning/icap_endpoints HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

[
  {
    "address": "127.0.0.1",
    "port": 1344
  }
]
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "address": "127.0.0.1",
    "port": 1344
  }
]
```

POST

Example request:

```
POST /api/atlant/config/v1/scanning/icap_endpoints HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

{
  "address": "127.0.0.1",
  "port": 1344
}
```

Example response:

```
HTTP/1.1 201 Created
Content-Type: application/json
Location: URL

{
  "address": "127.0.0.1",
  "port": 1344
}
```

DELETE

Example request:

```
DELETE /api/atlant/config/v1/scanning/icap_endpoints HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 204 No Content
```

ICAP scanning endpoint

Keys:

- ENDPOINT

  Example: `{"address":"string","port":0}`

Key should be URL-encoded

Resource URI: `/api/atlant/config/v1/scanning/icap_endpoints/ENDPOINT`

Methods: `GET PUT DELETE`

Data type: `object`

GET

Example request:

```
GET /api/atlant/config/v1/scanning/icap_endpoints/ENDPOINT HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "address": "127.0.0.1",
  "port": 1344
}
```

PUT

Example request:

```
PUT /api/atlant/config/v1/scanning/icap_endpoints/ENDPOINT HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

{
  "address": "127.0.0.1",
  "port": 1344
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "address": "127.0.0.1",
  "port": 1344
}
```

DELETE

Example request:

```
DELETE /api/atlant/config/v1/scanning/icap_endpoints/ENDPOINT HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 204 No Content
```

ICAP scanning endpoint address

Keys:

- ENDPOINT

  Example: `{"address":"string","port":0}`

Key should be URL-encoded

Resource URI: `/api/atlant/config/v1/scanning/icap_endpoints/ENDPOINT/address`

Methods: `GET PUT`

Data type: `string`

GET

Example request:

```
GET /api/atlant/config/v1/scanning/icap_endpoints/ENDPOINT/address HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

"127.0.0.1"
```

PUT

Example request:

```
PUT /api/atlant/config/v1/scanning/icap_endpoints/ENDPOINT/address HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

"127.0.0.1"
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

"127.0.0.1"
```

ICAP scanning endpoint port

Keys:

- ENDPOINT

  Example: `{"address":"string","port":0}`

Key should be URL-encoded

Resource URI: `/api/atlant/config/v1/scanning/icap_endpoints/ENDPOINT/port`

Methods: `GET PUT`

Data type: `number`

GET

Example request:

```
GET /api/atlant/config/v1/scanning/icap_endpoints/ENDPOINT/port HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

1344
```

PUT

Example request:

```
PUT /api/atlant/config/v1/scanning/icap_endpoints/ENDPOINT/port HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN
```

```
1344
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

1344
```

4.10.3 Reputation check settings

Resource URI: `/api/atlant/config/v1/scanning/reputation`

Methods: `GET PUT DELETE`

Data type: `object`

GET

Example request:

```
GET /api/atlant/config/v1/scanning/reputation HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "file_check": true,
  "timeout": 5,
  "url_check": true
}
```

PUT

Example request:

```
PUT /api/atlant/config/v1/scanning/reputation HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

{
  "file_check": true,
  "timeout": 5,
  "url_check": true
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "file_check": true,
  "timeout": 5,
  "url_check": true
}
```

DELETE

Example request:

```
DELETE /api/atlant/config/v1/scanning/reputation HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 204 No Content
```

Enable file reputation checks

Resource URI: `/api/atlant/config/v1/scanning/reputation/file_check`

Methods: `GET PUT DELETE`

Data type: `boolean`

GET

Example request:

```
GET /api/atlant/config/v1/scanning/reputation/file_check HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

true
```

PUT

Example request:

```
PUT /api/atlant/config/v1/scanning/reputation/file_check HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

true
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

true
```

DELETE

Example request:

```
DELETE /api/atlant/config/v1/scanning/reputation/file_check HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 204 No Content
```

Enable URL reputation checks

Resource URI: `/api/atlant/config/v1/scanning/reputation/url_check`

Methods: `GET PUT DELETE`

Data type: `boolean`

GET

Example request:

```
GET /api/atlant/config/v1/scanning/reputation/url_check HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json
```

```
true
```

PUT

Example request:

```
PUT /api/atlant/config/v1/scanning/reputation/url_check HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

true
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

true
```

DELETE

Example request:

```
DELETE /api/atlant/config/v1/scanning/reputation/url_check HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 204 No Content
```

Timeout for reputation requests in seconds

Resource URI: `/api/atlant/config/v1/scanning/reputation/timeout`

Methods: `GET PUT DELETE`

Data type: `number`

GET

Example request:

```
GET /api/atlant/config/v1/scanning/reputation/timeout HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

5
```

PUT

Example request:

```
PUT /api/atlant/config/v1/scanning/reputation/timeout HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

5
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

5
```

DELETE

Example request:

```
DELETE /api/atlant/config/v1/scanning/reputation/timeout HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 204 No Content
```

4.10.4 Maximum scan time in seconds

Resource URI: `/api/atlant/config/v1/scanning/max_scan_time`

Methods: `GET PUT DELETE`

Data type: `number`

GET

Example request:

```
GET /api/atlant/config/v1/scanning/max_scan_time HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

90
```

PUT

Example request:

```
PUT /api/atlant/config/v1/scanning/max_scan_time HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

90
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

90
```

DELETE

Example request:

```
DELETE /api/atlant/config/v1/scanning/max_scan_time HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 204 No Content
```

4.10.5 Keep-alive time in seconds

Resource URI: `/api/atlant/config/v1/scanning/keepalive_time`

Methods: `GET PUT DELETE`

Data type: `number`

GET

Example request:

```
GET /api/atlant/config/v1/scanning/keepalive_time HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

600
```

PUT

Example request:

```
PUT /api/atlant/config/v1/scanning/keepalive_time HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

600
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

600
```

DELETE

Example request:

```
DELETE /api/atlant/config/v1/scanning/keepalive_time HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 204 No Content
```

4.10.6 Scan nested archives up to this depth

Resource URI: `/api/atlant/config/v1/scanning/archive_max_nested`

Methods: `GET PUT DELETE`

Data type: `number`

GET

Example request:

```
GET /api/atlant/config/v1/scanning/archive_max_nested HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

5
```

PUT

Example request:

```
PUT /api/atlant/config/v1/scanning/archive_max_nested HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

5
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

5
```

DELETE

Example request:

```
DELETE /api/atlant/config/v1/scanning/archive_max_nested HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 204 No Content
```

4.11 Settings for manual scanning

Resource URI: `/api/atlant/config/v1/fsanalyze`

Methods: `GET PUT DELETE`

Data type: `object`

GET

Example request:

```
GET /api/atlant/config/v1/fsanalyze HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "actions": {
    "malware": "remove",
    "pua": "rename",
    "suspected": "none"
  },
  "archive_max_nested": 5,
  "block_archive_max_nested": true,
  "block_encrypted_archives": true,
  "detect_pua": true,
  "scan_archives": true
}
```

PUT

Example request:

```
PUT /api/atlant/config/v1/fsanalyze HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

{
  "actions": {
```

```
    "malware": "remove",
    "pua": "rename",
    "suspected": "none"
  },
  "archive_max_nested": 5,
  "block_archive_max_nested": true,
  "block_encrypted_archives": true,
  "detect_pua": true,
  "scan_archives": true
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "actions": {
    "malware": "remove",
    "pua": "rename",
    "suspected": "none"
  },
  "archive_max_nested": 5,
  "block_archive_max_nested": true,
  "block_encrypted_archives": true,
  "detect_pua": true,
  "scan_archives": true
}
```

DELETE

Example request:

```
DELETE /api/atlant/config/v1/fsanalyze HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 204 No Content
```

4.11.1 What to do with different kinds of malicious files

Resource URI: `/api/atlant/config/v1/fsanalyze/actions`

Methods: `GET PUT DELETE`

Data type: `object`

GET

Example request:

```
GET /api/atlant/config/v1/fsanalyze/actions HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "malware": "remove",
  "pua": "rename",
  "suspected": "none"
}
```

PUT

Example request:

```
PUT /api/atlant/config/v1/fsanalyze/actions HTTP/1.1
Content-Type: application/json
```

```
Authorization: Bearer TOKEN

{
  "malware": "remove",
  "pua": "rename",
  "suspected": "none"
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "malware": "remove",
  "pua": "rename",
  "suspected": "none"
}
```

DELETE

Example request:

```
DELETE /api/atlant/config/v1/fsanalyze/actions HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 204 No Content
```

Manual scanning action for malware

Value must be one of `"none"`, `"rename"`, `"remove"`

Resource URI: `/api/atlant/config/v1/fsanalyze/actions/malware`

Methods: `GET` `PUT` `DELETE`

Data type: `string`

GET

Example request:

```
GET /api/atlant/config/v1/fsanalyze/actions/malware HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

"remove"
```

PUT

Example request:

```
PUT /api/atlant/config/v1/fsanalyze/actions/malware HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

"remove"
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

"remove"
```

DELETE

Example request:

```
DELETE /api/atlant/config/v1/fsanalyze/actions/malware HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 204 No Content
```

Manual scanning action for PUA

Value must be one of "none", "rename", "remove"

Resource URI: /api/atlant/config/v1/fsanalyze/actions/pua

Methods: GET PUT DELETE

Data type: string

GET

Example request:

```
GET /api/atlant/config/v1/fsanalyze/actions/pua HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

"rename"
```

PUT

Example request:

```
PUT /api/atlant/config/v1/fsanalyze/actions/pua HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

"rename"
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

"rename"
```

DELETE

Example request:

```
DELETE /api/atlant/config/v1/fsanalyze/actions/pua HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 204 No Content
```

Manual scanning action for suspicious files

Value must be one of "none", "rename", "remove"

Resource URI: /api/atlant/config/v1/fsanalyze/actions/suspected

Methods: GET PUT DELETE

Data type: `string`

GET

Example request:

```
GET /api/atlant/config/v1/fsanalyze/actions/suspected HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

"none"
```

PUT

Example request:

```
PUT /api/atlant/config/v1/fsanalyze/actions/suspected HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

"none"
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

"none"
```

DELETE

Example request:

```
DELETE /api/atlant/config/v1/fsanalyze/actions/suspected HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 204 No Content
```

4.11.2 Extract and scan files from archives

Resource URI: `/api/atlant/config/v1/fsanalyze/scan_archives`

Methods: GET PUT DELETE

Data type: `boolean`

GET

Example request:

```
GET /api/atlant/config/v1/fsanalyze/scan_archives HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

true
```

PUT

Example request:

```
PUT /api/atlant/config/v1/fsanalyze/scan_archives HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

true
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

true
```

DELETE

Example request:

```
DELETE /api/atlant/config/v1/fsanalyze/scan_archives HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 204 No Content
```

4.11.3 Scan nested archives up to this depth

Resource URI: `/api/atlant/config/v1/fsanalyze/archive_max_nested`

Methods: `GET PUT DELETE`

Data type: `number`

GET

Example request:

```
GET /api/atlant/config/v1/fsanalyze/archive_max_nested HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

5
```

PUT

Example request:

```
PUT /api/atlant/config/v1/fsanalyze/archive_max_nested HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

5
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

5
```

DELETE

Example request:

```
DELETE /api/atlant/config/v1/fsanalyze/archive_max_nested HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 204 No Content
```

4.11.4 Block archives that are not fully scanned because of the configured archive_max_nested value

Resource URI: `/api/atlant/config/v1/fsanalyze/block_archive_max_nested`

Methods: `GET PUT DELETE`

Data type: `boolean`

GET

Example request:

```
GET /api/atlant/config/v1/fsanalyze/block_archive_max_nested HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

true
```

PUT

Example request:

```
PUT /api/atlant/config/v1/fsanalyze/block_archive_max_nested HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

true
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

true
```

DELETE

Example request:

```
DELETE /api/atlant/config/v1/fsanalyze/block_archive_max_nested HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 204 No Content
```

4.11.5 Block encrypted archives

Resource URI: `/api/atlant/config/v1/fsanalyze/block_encrypted_archives`

Methods: `GET PUT DELETE`

Data type: `boolean`

GET

Example request:

```
GET /api/atlant/config/v1/fsanalyze/block_encrypted_archives HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

true
```

PUT

Example request:

```
PUT /api/atlant/config/v1/fsanalyze/block_encrypted_archives HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

true
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

true
```

DELETE

Example request:

```
DELETE /api/atlant/config/v1/fsanalyze/block_encrypted_archives HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 204 No Content
```

4.11.6 Detect potentially unwanted applications

Resource URI: `/api/atlant/config/v1/fsanalyze/detect_pua`

Methods: `GET PUT DELETE`

Data type: `boolean`

GET

Example request:

```
GET /api/atlant/config/v1/fsanalyze/detect_pua HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

true
```

PUT

Example request:

```
PUT /api/atlant/config/v1/fsanalyze/detect_pua HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

true
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

true
```

DELETE

Example request:

```
DELETE /api/atlant/config/v1/fsanalyze/detect_pua HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 204 No Content
```

4.12 Product updates settings

Resource URI: `/api/atlant/config/v1/updates`

Methods: `GET PUT DELETE`

Data type: `object`

GET

Example request:

```
GET /api/atlant/config/v1/updates HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "enabled": true,
  "product_version": "deferred",
  "schedule": {
    "at_date_schedule": 1566565044,
    "regime": "on_arrival",
    "repetition_schedule": {
      "day": "monday",
      "time_of_day": "23:45"
    }
  }
}
```

PUT

Example request:

```
PUT /api/atlant/config/v1/updates HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

{
  "enabled": true,
```

```
  "product_version": "deferred",
  "schedule": {
    "at_date_schedule": 1566565044,
    "regime": "on_arrival",
    "repetition_schedule": {
      "day": "monday",
      "time_of_day": "23:45"
    }
  }
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "enabled": true,
  "product_version": "deferred",
  "schedule": {
    "at_date_schedule": 1566565044,
    "regime": "on_arrival",
    "repetition_schedule": {
      "day": "monday",
      "time_of_day": "23:45"
    }
  }
}
```

DELETE

Example request:

```
DELETE /api/atlant/config/v1/updates HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 204 No Content
```

4.12.1 Toggle automatic updates

Resource URI: `/api/atlant/config/v1/updates/enabled`

Methods: `GET PUT DELETE`

Data type: `boolean`

GET

Example request:

```
GET /api/atlant/config/v1/updates/enabled HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

true
```

PUT

Example request:

```
PUT /api/atlant/config/v1/updates/enabled HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

true
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

true
```

DELETE

Example request:

```
DELETE /api/atlant/config/v1/updates/enabled HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 204 No Content
```

4.12.2 Set the specified product version

Resource URI: `/api/atlant/config/v1/updates/product_version`

Methods: `GET PUT`

Data type: `string`

GET

Example request:

```
GET /api/atlant/config/v1/updates/product_version HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

"deferred"
```

PUT

Example request:

```
PUT /api/atlant/config/v1/updates/product_version HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

"deferred"
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

"deferred"
```

4.12.3 Update schedule

Resource URI: `/api/atlant/config/v1/updates/schedule`

Methods: `GET PUT DELETE`

Data type: `object`

GET

Example request:

```
GET /api/atlant/config/v1/updates/schedule HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "at_date_schedule": 1566565044,
  "regime": "on_arrival",
  "repetition_schedule": {
    "day": "monday",
    "time_of_day": "23:45"
  }
}
```

PUT

Example request:

```
PUT /api/atlant/config/v1/updates/schedule HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

{
  "at_date_schedule": 1566565044,
  "regime": "on_arrival",
  "repetition_schedule": {
    "day": "monday",
    "time_of_day": "23:45"
  }
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "at_date_schedule": 1566565044,
  "regime": "on_arrival",
  "repetition_schedule": {
    "day": "monday",
    "time_of_day": "23:45"
  }
}
```

DELETE

Example request:

```
DELETE /api/atlant/config/v1/updates/schedule HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 204 No Content
```

4.12.4 Date of update at epoch time format

Resource URI: `/api/atlant/config/v1/updates/schedule/at_date_schedule`

Methods: `GET PUT DELETE`

Data type: `number`

GET

Example request:

```
GET /api/atlant/config/v1/updates/schedule/at_date_schedule HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

1566565044
```

PUT

Example request:

```
PUT /api/atlant/config/v1/updates/schedule/at_date_schedule HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

1566565044
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

1566565044
```

DELETE

Example request:

```
DELETE /api/atlant/config/v1/updates/schedule/at_date_schedule HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 204 No Content
```

4.12.5 Scheduling type for updates

Value must be one of `"on_arrival"`, `"at_date"`, `"with_repetition"`

Resource URI: `/api/atlant/config/v1/updates/schedule/regime`

Methods: `GET PUT DELETE`

Data type: `string`

GET

Example request:

```
GET /api/atlant/config/v1/updates/schedule/regime HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

"on_arrival"
```

PUT

Example request:

```
PUT /api/atlant/config/v1/updates/schedule/regime HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

"on_arrival"
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

"on_arrival"
```

DELETE

Example request:

```
DELETE /api/atlant/config/v1/updates/schedule/regime HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 204 No Content
```

4.12.6 Update repetition schedule

Resource URI: `/api/atlant/config/v1/updates/schedule/repetition_schedule`

Methods: `GET PUT DELETE`

Data type: `object`

GET

Example request:

```
GET /api/atlant/config/v1/updates/schedule/repetition_schedule HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "day": "monday",
  "time_of_day": "23:45"
}
```

PUT

Example request:

```
PUT /api/atlant/config/v1/updates/schedule/repetition_schedule HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

{
  "day": "monday",
  "time_of_day": "23:45"
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json
```

```
{
  "day": "monday",
  "time_of_day": "23:45"
}
```

DELETE

Example request:

```
DELETE /api/atlant/config/v1/updates/schedule/repetition_schedule HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 204 No Content
```

4.12.7 Scheduled updates day

Value must be one of `"monday"`, `"tuesday"`, `"wednesday"`, `"thursday"`, `"friday"`, `"saturday"`, `"sunday"`, `"daily"`

Resource URI: `/api/atlant/config/v1/updates/schedule/repetition_schedule/day`

Methods: `GET PUT DELETE`

Data type: `string`

GET

Example request:

```
GET /api/atlant/config/v1/updates/schedule/repetition_schedule/day HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

"monday"
```

PUT

Example request:

```
PUT /api/atlant/config/v1/updates/schedule/repetition_schedule/day HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

"monday"
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

"monday"
```

DELETE

Example request:

```
DELETE /api/atlant/config/v1/updates/schedule/repetition_schedule/day HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 204 No Content
```

4.12.8 Scheduled updates time

Resource URI: `/api/atlant/config/v1/updates/schedule/repetition_schedule/time_of_day`

Methods: `GET PUT DELETE`

Data type: `string`

GET

Example request:

```
GET /api/atlant/config/v1/updates/schedule/repetition_schedule/time_of_day HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

"23:45"
```

PUT

Example request:

```
PUT /api/atlant/config/v1/updates/schedule/repetition_schedule/time_of_day HTTP/1.1
Content-Type: application/json
Authorization: Bearer TOKEN

"23:45"
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

"23:45"
```

DELETE

Example request:

```
DELETE /api/atlant/config/v1/updates/schedule/repetition_schedule/time_of_day HTTP/1.1
Authorization: Bearer TOKEN
```

Example response:

```
HTTP/1.1 204 No Content
```

Chapter

5

Configuring settings with the atlantctl utility

Topics:

You can check and edit the settings for Linux Security 64 using the `atlantctl` command-line utility.

Note: The command path given here is for the current version of the product. If you are using an older version, use `/opt/f-secure/atlant/bin/<command>` as the path instead.

To use the `atlantctl` utility:

1. Log in to the Linux host as `root`.
2. Use the following commands:
   - To see the current value of a setting, run `/opt/f-secure/atlant/atlant/bin/atlantctl get [OPTIONS] [SETTING]`
   - To assign a new value to a setting, run `/opt/f-secure/atlant/atlant/bin/atlantctl set [OPTIONS] [SETTING] [VALUE]`
   - To add an entry to a setting that holds several values, run `/opt/f-secure/atlant/atlant/bin/atlantctl add [OPTIONS] [SETTING] [VALUE]`
   - To remove an entry from a setting that holds several values, run `/opt/f-secure/atlant/atlant/bin/atlantctl del [OPTIONS] [SETTING] [KEY]`
   - To clear the value of a setting, run `/opt/f-secure/atlant/atlant/bin/atlantctl reset [OPTION] [SETTING]`
   - To see a list of the operations available for a setting, run `/opt/f-secure/atlant/atlant/bin/atlantctl help [OPTIONS] [SETTING]`
   - To get a list of all available settings, run `/opt/f-secure/atlant/atlant/bin/atlantctl -h`

The `get`, `set`, `add` and `del` commands support the following options:

--json, -j Treat input as JSON and produce output in JSON format.

--raw, -r This is currently the default. Raw is an alternative input/output format that is similar to JSON with the following changes:

- String values are accepted and returned without surrounding quotes.
- Boolean values accept additional expressions such as `yes` and `no` in addition to `true` and `false`.

Note: The default input/output format for `atlantctl` is subject to change. If you rely on the format of `atlantctl`, specify either `--json` or `--raw` explicitly when `atlantctl` is run. In addition to setting values, the keys for array-type settings are considered as input and need to be entered in the correct format.

The `get` command accepts the following additional option:

--sensitive, -s Show all values for settings, including any that may contain private information. If this option is not included in the command, values that are likely to contain private information are not shown.

The set and add commands accept the following options:

--prompt, -p Prompt for input using a text editor.

--file, -f [PATH] Read the input value from a file. If the path is set to `-`, the input value is read from the standard input.

If you use the `--prompt` or `--file` options, do not enter a `[VALUE]` for the command.

5.1 Command-line settings for product license management

The settings related to product license management that are available for `atlantctl`.

Tip: To see an example of the structure for any of the settings, you can check the output for the following command: `/opt/f-secure/atlant/atlant/bin/atlantctl get [--json|--raw] [--sensitive] [SETTING]`

license

Type: JSON object

The license `type` and `content`. The `type` value must be one of `file` or `key`.

For `set` commands, the `content` value should be the path to the license file (if `type` is `file`) or the code in data URL form (if `type` is `key`).

Examples:

```
atlantctl set license '{"type": "file", "content": "path_to_local_license_file"}'
```

```
atlantctl set license '{"type": "key", content": "data:,aaa-bbb-ccc"}'
```

The response for `get` commands includes the product license content in data URL form.

Note: Unlike most of the `atlantctl` commands, there are no separate command options for the individual components of the `license` setting objects.

5.2 Command-line settings for client management

The settings related to client management that are available for `atlantctl`.

Note: The command path given here is for the current version of the product. If you are using an older version, use `/opt/f-secure/atlant/bin/<command>` as the path instead.

Note: The `client` command uses a slightly different syntax than the other `atlantctl` commands.

atlantctl client create [SCOPES]

Type: JSON object

Creates a new client with the specified scopes. The available scopes are:

- `management` - allows the client to inspect and change the Atlant configuration
- `scan` - allows the client to scan files

The response is a JSON object containing the `client_id` and `client_secret` values, which are the credentials that the client uses to request access tokens from the authorization server.

Example:

```
/opt/f-secure/atlant/atlant/bin/atlantctl client create '{"scopes":
["management", "scan"]}'
```

Example response:

```
{
  "client_id": "5dbfe97de42bf53a0ae73bff9eba4ecb",
  "client_secret":
"87e7b3a61e7fdcdfc03162fdcab82a942b9c62715ff3d612e15adf32d1b1500b"
}
```

atlantctl client list

Type: JSON object

Returns a list of the clients and the scopes for each of them.

atlantctl client delete [CLIENT_ID]

Type: JSON object

Delete a client. The request must specify the `client_id` value for the client that you want to remove.

Example:

```
/opt/f-secure/atlant/atlant/bin/atlantctl client delete '{"client_id":
 "5dbfe97de42bf53a0ae73bff9eba4ecb"}'
```

5.3 Command-line settings for authorization server

The settings related to authorization that are available for `atlantctl`.

Tip: To see an example of the structure for any of the settings, you can check the output for the following command: `/opt/f-secure/atlant/atlant/bin/atlantctl get [--json|--raw] [--sensitive] [SETTING]`

authorization	Type: JSON object All authorization server settings.
authorization external_domains	Type: JSON array The list of external authorization domains.
authorization external_domains [AUDIENCE]	Type: JSON object All information for the external authorization domain specified in `[AUDIENCE]`.
authorization external_domains [AUDIENCE] audience	Type: string The audience for the external authorization domain specified in `[AUDIENCE]`.
authorization external_domains [AUDIENCE] issuer	Type: string The issuer for the external authorization domain specified in `[AUDIENCE]`.
authorization external_domains [AUDIENCE] signing_method	Type: string The signing method for the external authorization domain specified in `[AUDIENCE]`. The value must be either `HS256` or `RS256`.
authorization external_domains [AUDIENCE] verification_key	Type: string The verification key for the external authorization domain specified in `[AUDIENCE]`.
authorization https_endpoints	Type: JSON array The list of authentication server HTTP endpoints.
authorization https_endpoints [ENDPOINT]	Type: JSON object All information for the HTTP endpoint specified in `[ENDPOINT]`.
authorization https_endpoints [ENDPOINT] address	Type: string The address for the HTTP endpoint specified in `[ENDPOINT]`.
authorization https_endpoints [ENDPOINT] port	Type: integer The port for the HTTP endpoint specified in `[ENDPOINT]`.

5.4 Command-line settings for TLS

The settings related to TLS that are available for `atlantctl`.

Note: The command path given here is for the current version of the product. If you are using an older version, use `/opt/f-secure/atlant/bin/<command>` as the path instead.

Tip: To see an example of the structure for any of the settings, you can check the output for the following command: `/opt/f-secure/atlant/atlant/bin/atlantctl get [--json|--raw] [--sensitive] [SETTING]`

tls

Type: JSON object

All TLS-related settings used in Atlant.

tls certificate

Type: string

The TLS certificate used in Atlant.

For `set` commands, enter either the local file path to the certificate or the base64-encoded data URL for the certificate contents.

The response for `get` commands includes the base64-encoded data URL with the contents of the certificate.

Examples:

```
/opt/f-secure/atlant/atlant/bin/atlantctl set tls certificate
<local_path_to_certificate>
```

```
/opt/f-secure/atlant/atlant/bin/atlantctl set tls certificate
'data;base64,Y2VydGlmaWNhdGU='
```

tls key

Type: string

The TLS key used in Atlant.

For `set` commands, enter either the local file path to the key or the base64-encoded data URL for the key contents.

The response for `get` commands includes the base64-encoded data URL with the contents of the key.

Example:

```
/opt/f-secure/atlant/atlant/bin/atlantctl set tls key <local_path_to_key>
```

5.5 Command-line settings for management server

The settings related to management server setup that are available for `atlantctl`.

Tip: To see an example of the structure for any of the settings, you can check the output for the following command: `/opt/f-secure/atlant/atlant/bin/atlantctl get [--json|--raw] [--sensitive] [SETTING]`

management

Type: JSON object

All management server settings.

management https_endpoints

Type: JSON array

The list of management server endpoints.

management https_endpoints [ENDPOINT]

Type: JSON object

All information for the management server endpoint specified in `[ENDPOINT]`.

management https_endpoints [ENDPOINT] address

Type: string

The address for the management server endpoint specified in `[ENDPOINT]`.

management https_endpoints [ENDPOINT] port

Type: integer

The port for the management server endpoint specified in `[ENDPOINT]`.

5.6 Command-line settings for HTTP proxy

The settings related to HTTP proxy that are available for `atlantctl`.

Tip: To see an example of the structure for any of the settings, you can check the output for the following command: `/opt/f-secure/atlant/atlant/bin/atlantctl get [--json|--raw] [--sensitive] [SETTING]`

http_proxy	Type: JSON object All HTTP proxy settings.
http_proxy enabled	Type: boolean Controls whether or not to use an HTTP proxy for features that require network access.
http_proxy host	Type: string The host name of the HTTP proxy.
http_proxy port	Type: integer The HTTP proxy port.

5.7 Command-line settings for scanning server

The settings related to the scanning server that are available for `atlantctl`.

Tip: To see an example of the structure for any of the settings, you can check the output for the following command: `/opt/f-secure/atlant/atlant/bin/atlantctl get [--json|--raw] [--sensitive] [SETTING]`

scanning	Type: JSON object All scanning server settings.
scanning https_endpoints	Type: JSON array The list of scanning server HTTP endpoints.
scanning https_endpoints [ENDPOINT]	Type: JSON object The settings for the HTTP endpoint specified in `[ENDPOINT]`.
scanning https_endpoints [ENDPOINT] address	Type: string The address for the HTTP endpoint specified in `[ENDPOINT]`.
scanning https_endpoints [ENDPOINT] port	Type: integer The port for the HTTP endpoint specified in `[ENDPOINT]`.
scanning icap_endpoints	Type: JSON array The list of scanning server ICAP endpoints.
scanning icap_endpoints [ENDPOINT]	Type: JSON object The settings for the ICAP endpoint specified in `[ENDPOINT]`.
scanning icap_endpoints [ENDPOINT] address	Type: string The address for the ICAP endpoint specified in `[ENDPOINT]`.
scanning icap_endpoints [ENDPOINT] port	Type: integer The port for the ICAP endpoint specified in `[ENDPOINT]`.

5.8 Command-line settings for automatic updates

The settings related to automatic updates that are available for `atlantctl`.

Tip: To see an example of the structure for any of the settings, you can check the output for the following command: `/opt/f-secure/atlant/atlant/bin/atlantctl get [--json|--raw] [--sensitive] [SETTING]`

updates

Type: JSON object

All settings for automatic updates.

updates enabled

Type: boolean

Controls whether or not automatic updates are in use.

updates product_version

Type: string

If not empty, this sets a specific product version to use instead of automatically updating to the latest available version.

updates schedule

Type: JSON object

The automatic update schedule.

updates schedule regime

Type: string

The type of scheduling for automatic updates. The allowed values are:

- `on_arrival` - updates are applied as soon as they become available
- `at_date` - updates are postponed until the time specified in the `at_date_schedule` setting
- `with_repetition` - updates are applied regularly according to the `repetition_schedule` settings

updates schedule at_date_schedule

Type: integer

The date to apply updates when `regime` is set to `at_date`. The value must be a timestamp in epoch time format.

updates schedule repetition_schedule

Type: JSON object

Settings related to automatic updates with a fixed repetition schedule.

updates schedule repetition_schedule day

Type: string

The scheduled day for repeated updates. Allowed values are `daily`, to check for updates every day at the specific time, or `monday`, `tuesday`, `wednesday`, `thursday`, `friday`, `saturday`, or `sunday` to check for updates on the given day each week.

updates schedule repetition_schedule time_of_day

Type: string

The scheduled time of day for applying updates. The value is a 24-hour UTC time value (`HH:MM`).

Chapter

6

Command line usage

Topics:

This section describes the commands that you can run from the command line in F-Secure Atlant.

6.1 Scanning the computer manually from the command line

You can scan the computer for malware manually from the command line of a Linux host that has the product installed.

Note: The command path given here is for the current version of the product. If you are using an older version, use `/opt/f-secure/atlant/bin/<command>` as the path instead.

The `fsanalyze` program scans files with the privileges of the user who runs it. The user must have read access to files, and read and execute access to all directories to be scanned. To rename or remove infected files, the user must have write access to the directories where the files are located.

The `fsanalyze` command scans specified targets (files or directories) and reports any malicious code it detects.

If no files are specified on the command line, `fsanalyze` reads the standard input for content to analyze. Otherwise, every given file is analyzed and every given directory is scanned recursively, with the following exceptions:

- Special files (such as socket, FIFO, or device files), and files on the `/proc` and `/sys` special file systems are skipped.
- Recursive scanning does not follow symbolic links unless this is explicitly requested with the `--follow` command line option.
- Recursive scanning does not automatically cross file system boundaries. To scan file systems mounted on other file systems, specify the mount points explicitly on the command line.

Run the following command from the shell to start the manual scan:
`/opt/f-secure/atlant/atlant/bin/fsanalyze [OPTIONS] [FILE...]`

Usage: `fsanalyze [OPTIONS] [FILE...]`

Analyze each `FILE` for potential malicious content. If no file is specified, analyze standard input, in which case the only supported action is `none`.

-h, --help	Show help.
--malware=ACTION	Action to take when a file is detected as malware. If ACTION is `remove` the file is removed; if ACTION is `rename` the file is renamed adding the detection type as an extension; if ACTION is `none`, the infection is only reported on standard output. Default: `rename`
--pua=ACTION	Action to take when a file is detected as a potentially unwanted application. Options for the action are the same as for malware. Default: `none`
--suspected=ACTION	Action to take when a file is detected as suspicious. Options for the action are the same as for malware. Default: `none`
-L, --follow	Follow symbolic links.
-l, --list	List all scanned files.
--preserve-atime=VALUE	Preserve the access time of files that are scanned. Value can be `yes` or `no`. Default: `no`
-q, --quiet	Only output the scan results.
--scan-archives=VALUE	Enable archive scanning. Value can be `yes` or `no`. Default: `no`
--max-nested=NUMBER	Set the maximum allowed nesting level of scanned archives. Default: `5`
--detect-max-nested=VALUE	If enabled, treat archives that exceed the maximum depth as malware. Value can be `yes` or `no`. Default: `no`
--detect-encrypted-archives=VALUE	Treat encrypted archives as malware. Value can be `yes` or `no`.

	Default: `no`
--detect-pua=VALUE	If disabled, potentially unwanted applications are ignored. Value can be `yes` or `no`. Default: `yes`
--use-orsp=VALUE	If enabled, use the reputation service (ORSP) from Security Cloud. Value can be `yes` or `no`. Default: `yes`
-v, --version	Display program version and exit.
--quoting-style=STYLE	Set how to quote the output. If STYLE is `url`, the output is URL-encoded; if STYLE is `escape`, ASCII control characters in the output are escaped as `<NAME>`, and if the output is not redirected to a file, they are also highlighted; if STYLE is `literal`, the output is printed as is. Default: `url`

The scanning exit codes are as follows:

- `0` = scanning was completed with no errors, and no malware, PUA, or suspicious content was detected
- `1` = scanning failed for one or more files
- `2` = scanning was completed with no errors, but malware, PUA, or suspicious content was detected

Scanning detections are printed for each file on the standard output as separate lines in the following format:

```
<FILE>: result=<RESULT> infection=<DETECTION NAME> member-name=<MEMBER>
```

where:

- `<FILE>` is the path name of the scanned file.
- `<RESULT>` is one of `clean`, `infected`, `pua`, or `suspected`. Clean files are reported in the output only if the `--list` option is specified on the command line.
- `<DETECTION NAME>` is the name of the infection that was detected in the file. If the file is clean, the `infection` field is omitted.
- `<MEMBER>` is the name of the file scanned in an archive (if `<FILE>` is an archive file). If `<FILE>` is not an archive, the `member-name` field is omitted.

Multiple detections on the same file are printed on separate lines of output.

Note: Nonprintable or non-ASCII characters in all file names shown in the output are URL-encoded by default. For example, the space character is represented as a plus sign (+) and the é character is represented as `%C3%A9`. You can change the output format for path names with the `--quoting-style` option.

Unless the action assigned to the type of detection is `none`, the detection information is followed by a line in the following form:

```
<FILE>: action=<ACTION>
```

where:

- `<FILE>` is the name of the file that was scanned
- `<ACTION>` is either `deleted` or `renamed`

Sample output for detections:

```
archive.tgz: result=infected infection=EICAR_Test_File member-name=eicar.txt
archive.tgz: action=renamed
cloudcar.exe: result=infected infection=Malware:W32/Cloudcartestfile
cloudcar.exe: action=renamed
```

If the `--quiet` option was specified on the command line, `fsanalyze` does not produce any further output. Otherwise, the list of detections is followed by a list of engines used for scanning and a summary of scan results in the following form:

```
Engine versions: F-Secure Corporation Aquarius/18.0.676/2020-04-14_03 F-Secure Corporation
Hydra/6.0.216/2020-04-13_01 F-Secure Corporation FMLib/17.0.607.475  (cf1875a)/2020-02-04_01
 fsicapd/2.0.84

15 files scanned
7 files infected
3 files contain PUA
1 files suspected
2 files could not be scanned
```

The summary always includes the number of input files that were scanned successfully. If there were no infected files, files with PUA, suspected files, or files that could not be scanned, the corresponding lines are omitted. Archive files count as single input files.

Scanning errors are reported on the standard error using messages in the following form:

```
fsanalyze: <MESSAGE> '<FILE>' (<DETAILS>)
```

where:

- `<MESSAGE>` identifies the operation that failed
- `<FILE>` is the name of the file for which the operation failed
- `<DETAILS>` gives the operating system-level details about the error

Note: The `fsanalyze` program fills the same purpose as the `fsav` program included in F-Secure Linux Security 11. The main differences compared to the output of `fsav` are that `fsanalyze` shows path names with nonprintable or non-ASCII characters as URL-encoded by default, does not display the name of the scanning engine separately for every detection, and does not display time stamps for the scanning process.

Chapter

7

Using Atlant as a virtual appliance

Topics:

Atlant is available as a VMware-compatible OVA image for use in virtual environments, and you can also build an Atlant virtual appliance image for other hypervisors.

The Atlant VA includes a `first-boot` wizard for handling the initial setup when you first start Atlant. For installation instructions, see Installing Atlant as a virtual appliance on page 10.

You can change basic settings later with the `admin-menu` tool.

7.1 Building an Atlant VA image

This topic describes how to set up your own Atlant virtual appliance image.

Note: If you are using a VMware hypervisor, you can use the Atlant VMware-compatible OVA image instead of building your own image. For instructions on installing the OVA image, see Installing Atlant as a virtual appliance on page 10.

To set up your own Atlant VA image, you have to create a CentOS 8.2 virtual machine. The recommended system properties are:

- CPU: 4 cores
- RAM: 4 GB
- Hard drive: 100 GB (the `/boot` partition must be at least 1 GB)

In addition to the system packages, you have to install the following packages:

- `fuse-libs`
- `libcurl`
- `python36`

The command to install these packages is listed in the steps below.

1. Download the following packages from the F-Secure website and upload them to the virtual machine:
   - `f-secure-atlant.rpm` - this package contains the F-Secure Atlant bootstrapper, which is needed to install the product on the virtual machine. Upload the package to `/root` on the virtual machine and rename it to `atlant.rpm`.
   - f-secure-atlant-va.noarch.rpm - this package contains the Atlant-VA admin menus, first boot activator, and all necessary configurations (such as F-Secure RPM repository and log rotation configuration). Upload the package to `/tmp` on the virtual machine and rename it to `atlantva.rpm`.

   To upload the file from the host to the virtual machine using PuTTY:
   a) Open the command prompt.
   b) Enter `where pscp` and check that it does not raise a `Could not find files for the given pattern(s)` error.
   c) To update the file, enter the following command:

   ```
   pscp <path-to-the-file-on-host> root@<ip-address>:<path-on-virtual-machine>
   ```

   For example:

   ```
   pscp C:\Users\user\Downloads\f-secure-atlant.rpm root@192.168.1.100:/root/atlant.rpm
   pscp C:\Users\user\Downloads\f-secure-atlantva.rpm root@192.168.1.100:/tmp/atlantva.rpm
   ```

2. After you have uploaded the files, log in to the virtual machine as `root`.
3. Run the following commands to check that the necessary permissions and files are available:

   ```
   whoami
   ```

   The output must be `root`.

   ```
   ls /root/atlant.rpm
   ls /tmp/atlantva.rpm
   ```

   Make sure that neither of these commands raises a `No such file or directory` error message.
4. Run the following command to install the required additional packages:

   ```
   dnf install fuse-libs libcurl python36
   ```

5. Run the following command to install the `atlantva` package:

   ```
   dnf install -y /tmp/atlantva.rpm
   ```

6. Run the following command to remove the `atlantva.rpm` file as it is no longer needed:

```
rm -f /tmp/atlantva.rpm
```

7. Run the following command to disable logging in as `root` via `ssh`:

```
sed -i 's/PermitRootLogin yes/PermitRootLogin no/' /etc/ssh/sshd_config
```

8. Run the following command to lock the `root` account and prevent the local user from logging in as `root`:

```
passwd -l root
```

9. Set up and update the operating system using F-Secure repositories installed by RPM.
   a) Disable automatic `dnf` caching:

```
systemctl disable --now dnf-makecache.timer
```

   b) Clean up all remaining entries from any previously triggered `dnf-makecache` operations:

```
dnf clean all
```

   c) Decrease the number of kernels to 2:

```
sed -i 's/installonly_limit=3/installonly_limit=2/' /etc/dnf/dnf.conf
```

   d) Update the system using F-Secure repositories:

```
dnf --assumeyes --refresh --nodocs upgrade
```

   Your virtual machine is now ready to be exported or deployed, depending on your hypervisor.

Related tasks

Installing Atlant as a virtual appliance on page 10
Follow the instructions given here if you want to deploy Atlant as a VMware virtual appliance and use it in standalone mode or manage it through Policy Manager.

7.2 Configuring settings from the administrator menu

You can change the basic settings using the `admin-menu` tool, which appears when you log in to the Atlant virtual appliance.

1. Log in to the Atlant virtual appliance using the `admin` login name.
   The administrator menu opens.
2. Select the setting that you want to edit.

Option	Description
1	Change the administrator account password
2	Configure the centralized management settings
3	Create a diagnostics package
4	Manage weekly updates
5	Edit advanced settings

To change your administrator account password:

a) Select `1`.
b) Enter the new password and confirm it to change it.

To change the centralized management configuration:

a) Select `2`.

b) Choose one of the following options:

- Select 1 to view the current configuration.
- Select 2 to change the address of the management server.
- Select 3 to reset the address of the management server.

To create a diagnostics package, select 3.

To manage updates:

a) Select 4.
b) When asked to change the update schedule, answer y.
c) Enter the name of the weekday when you want the weekly update to run.
d) Enter the time of the day when you want the update to run.
The new update schedule is taken into use.

Note: The virtual appliance reboots automatically after a successful update.

To edit advanced settings, select 5.

7.3 Editing advanced settings

You can change the settings that you configured during the initial setup with the advanced settings in the administrator menu.

1. Select the setting that you want to edit from the advanced administrator menu.
The menu contains the following options:

Option	Description
1	Network settings
2	Configure HTTP proxy
3	Reboot or shut down appliance
4	Select keyboard layout
5	Select timezone
6	Go to Linux shell

2. To change the network settings, select 1.
   - Select 1 to view the current network configuration.
   - Select 2 to change the appliance's hostname.
   - Select 3 to change the DNS server that the appliance uses.
   - Select 4 to configure the IP address.
3. To configure HTTP proxy, select 2.
   - Select 1 to view the current HTTP proxy configuration.
   - Select 2 to enter the address of the HTTP proxy.
   - Select 3 to turn the HTTP proxy on or off.
   - Select 4 to remove the HTTP proxy.
4. To reboot or shut down the appliance, select 3.
5. To select the keyboard layout that you use, select 4.
   a) Choose the keymap group from the list.
   Keymaps are grouped in alphabetical order, choose the group that contains the keymap that you want to use.
   b) Choose the keymap.
   c) Select y when asked to set the keymap.
6. To select the time zone, select 5.

a) Choose the time zone group from the list.

   Time zones are grouped in alphabetical order, choose the group that contains the time zone that you want to use.

b) Choose the time zone.

c) Select `y` when asked to set the time zone.

7. To open the Linux shell, select `6`.

Note: To return to the administrator menu from the command line, press `CTRL-D` or type `exit`.

Chapter

8

Using Atlant as a replacement for F-Secure Scanning and Reputation Server

Topics:

You can use Atlant to provide scanning and content reputation services to F-Secure client software installed in virtual environments.

Similarly to Scanning and Reputation Server, which works as a virtual appliance that is deployed in the virtualization environment, you can use Atlant to minimize the impact on the performance of virtual desktops and servers. This approach offloads the malware scanning and content reputation checking to a dedicated server that runs Atlant, instead of performing the requests directly on each virtual machine.

To deploy Atlant to a VMware hypervisor for use in this role:

1. Create the installation package in F-Secure Policy Manager.
2. Set up the Atlant virtual appliance.

To deploy Atlant to a physical Linux host for use in this role:

1. Create the installation package in F-Secure Policy Manager.
2. Install the package on the target Linux host.

After deployment, you then need to configure the settings in Policy Manager Console.

Related concepts

Using Atlant as a virtual appliance on page 83

8.1 Configuring Atlant for virtual environments

To set up Atlant for use as a replacement for Scanning and Reputation Server, you need to configure it in Policy Manager Console to handle the offloaded requests from virtual desktops and servers.

After you have created the installation package in Policy Manager and installed Atlant:

1. In the Standard view of Policy Manager Console, select the **Root** domain.
2. Go to the **Settings** tab and select **Virtual security** > **Offload Scanning Agent**.
3. Select **Offload file scanning** and enter the address for your Atlant host.
4. Select **Virtual security** > **Atlant scanning service**.
5. Enter your Atlant Premium subscription key if you have one and want to use the premium features, such as Security Cloud scanning.
6. Configure the **General** settings and timeout values as needed.
7. Under **Endpoints**:
   - Click **Add** if you want add more ICAP service endpoints.
   - Select any of the entries and click **Edit** if you need to change the address or port for the endpoint.

   By default, Policy Manager creates a single ICAP scanning endpoint that uses the address `0.0.0.0` and port `1344`.
8. Click the following icon to distribute the policy:

Appendix A

Services installed with Atlant

F-Secure Atlant installs several services that are used to handle various product features.

Note: The services listed here are subject to change without advance notice.

Important: Do not manually start or stop any of the services listed here.

Active services

f-secure-atlant-atlant-atlantpmd.service	Locally distributes centrally managed settings to Atlant services.
f-secure-atlant-atlant-webserver.service	Provides an HTTP API for centrally managed settings.
f-secure-atlant-baseguard-as.service	A BaseGuard facility for email spam scanning. This service is running, but is not used by the product.
f-secure-atlant-baseguard-authorize.service	An OAuth 2 authorization server used with HTTP APIs. This service is inactive unless you have configured authorization endpoints.
f-secure-atlant-baseguard-av.service	A service stub required for backward-compatibility.
f-secure-atlant-baseguard-cleanup.service	Makes sure that older channel updates are cleaned up to save disk space.
f-secure-atlant-baseguard-doormand.service	Facilitates handling subscriptions, registration and cloud service lookups.
f-secure-atlant-baseguard-icap.service	The malware analysis service used for realtime, scheduled and manual scanning.
f-secure-atlant-baseguard-orspgw.service	A local proxy for F-Secure's Online Reputation Service.
f-secure-atlant-baseguard-tokenverify.service	An OAuth 2 authentication manager used with HTTP APIs.
f-secure-atlant-baseguard-update.service	Monitors F-Secure's GUTS2 service for channel updates and sends notifications to `fsbg-updated.service`.
fsbg-atlant-db-update.service	Triggered by the update system to run when needed

	during the loading or installation of channel updates.
fsbg-atlant-report-updates.service	Triggered by the update system to run when needed during the loading or installation of channel updates.
fsbg-atlant-update-all.service	Triggered by the update system to run when needed during the loading or installation of channel updates.
fsbg-atlant-update-expired.service	Triggered by the update system to run when needed during the loading or installation of channel updates.
fsbg-atlant-update-notifier.service	Triggered by the update system to run when needed during the loading or installation of channel updates.
fsbg-atlant-updated.service	Schedules the installation of online channel updates.
fsbg-atlant-statusd.service	Collects status and statistics information from BaseGuard services and relays them to the central management agent (`fsma2`).
fsbg-atlant-pmd.service	Locally distributes centrally managed settings to BaseGuard services.
atlant-fsma2.service	Handles centrally managed settings and communication.

Inactive services

These services are included during installation, but are not loaded by the product.

f-secure-atlant-baseguard-accd.service	Real-time scanner service that handles file access notifications.

Appendix B

ICAP extension reference

The ICAP service in Atlant extends the standard protocol with a number of extensions.

URI options

The REQMOD and RESPMOD requests can include the following URI options:

Option	Values	Default	Description
allow_upstream_metadata	[01]	1	Allow logging and upstreaming of metadata.
allow_upstream_application_files	[01]	1	Allow upstreaming of application files.
allow_upstream_data_files	[01]	0	Allow upstreaming of data files.
antispam	[01]	0	Scan content for spam.
scan_embedded_urls	[01]	0	Scan URLs embedded in the content.
security_cloud	[01]	0	1 to allow call to Security Cloud to process request.

Request headers

For REQMOD and RESPMOD requests, the client can use the following extension headers:

X-Forbidden-URI-Categories If the URLs that are checked in this request match any of the provided categories, a detection is reported. See **URL categories** for a list of supported categories.

```
value = token *( ',' token )
```

X-Meta-Charset

The MIME content character set (for example `UTF8`). Several commonly used alternate spellings are supported.

```
value = token ; RFC 7230
```

X-Meta-Content-Length

The content size.

```
value = 1*DIGIT
```

X-Meta-Content-Type

The MIME type of the content (for example `text/html`).

```
value = token "/" token ; RFC 7230
```

X-Meta-SHA1

The SHA1 digest for the content.

```
value = 40*HEXDIG
```

X-Meta-URI

The source URI for the content.

```
value = URI
```

X-Meta-IP

The IP address of the content source or originating client.

```
value = IPv4address ; RFC 3986
```

X-Meta-Sender

The email address of the sender of the content.

```
value = addr-spec ; RFC 822
```

X-Meta-Recipients

The email addresses of the recipients of the content.

```
value = addr-spec *( ',' addr-spec )
```

Response headers

These extended response headers are included by default in the ICAP responses:

X-FSecure-Scan-Result

Reports the final verdict of the scanning process. This header is included in all REQMOD and RESPMOD responses. The following values are possible:

- `clean`: No detection found
- `infected`: Content detected as harmful
- `suspected`: Content detected as suspicious
- `grayware`: Content detected as PUA/UA
- `spam`: Content detected as spam

Example:

```
X-FSecure-Scan-Result: infected
```

X-FSecure-Infection-Name

Reports the infection name. This header is not included if the scan result is `clean`. The infection name always appears in the quoted format described in RFC 2822, surrounded by double quotes, with `"` and `'` characters escaped with an extra `'`.

Example:

```
X-FSecure-Infection-Name:
"Eicar_Test_File"
```

X-FSecure-Transaction-Duration

Reports the total time used to process a single request. This is the number of seconds between the time the server finished receiving the ICAP request headers and the time the ICAP response headers were generated. This is more than the sum of scan operation durations, because it also includes the time taken to parse requests and the time spent by the request in various processing queues.

Example:

```
X-FSecure-Transaction-Duration:
 0.43920
```

X-FSecure-Infected-Filename

If file is detected as `clean`, `suspected`, or `grayware`, this header may report the name of the file that caused the detection. This header is omitted if the name of the file is not known. Currently, the file name can be reported only if the detection was caused by a file inside an archive or in a MIME email attachment. The file name is URL-encoded to make it possible to report file names that contain non-ASCII characters.

Example:

```
X-FSecure-Infected-Filename:
"infected file %E5.exe"
```

X-FSecure-Versions

This header returns version information about the antivirus engines and associated components. The string is not designed to be machine-readable, and the format may change at any time.

Example:

```
X-FSecure-Versions: F-Secure
Corporation Hydra/5.15 build
154/2017-01-26_01 F-Secure
Corporation Aquarius/1.0 build
8/2017-01-26_11 F-Secure
Corporation FMLib/4.66.50.16
build e70eac0/2016-11-15_01
fsicapd/1.1.0-7f7ae6a
```

X-Attribute: <string list>

This header contains a comma-separated list of categories matching the reputation for the provided URI. The URI can be specified with the `X-Meta-URI` header.

See **URL categories** for a list of possible values.

If Atlant is allowed to call Security Cloud, the response may also contain the following ICAP headers:

X-Location: <string>	If this header is not present, all analysis actions are completed and the result is final. Otherwise, the analysis is incomplete and clients can query for an updated result by sending a REQMOD or RESPMOD request for the resource specified by this header. If scanning a local file, the client should ensure that the local file exists until either the task is complete or the client no longer cares about the result. Failure to access the local file may cause the ICAP request to fail with a `500` error.
X-Retry-After: <seconds>	If present, this header specifies the number of seconds that the client should wait before polling for an updated result.

URL categories

The following categories are valid in the context of the `X-Forbidden-URI-Categories` and `X-Attribute` headers:

- `abortion`
- `adserving`
- `adult`
- `alcohol_and_tobacco`
- `anonymizers`
- `auctions`
- `banking`
- `blogs`
- `chat`
- `dating`
- `disturbing`
- `drugs`
- `entertainment`
- `file_sharing`
- `forum`
- `gambling`
- `games`
- `hate`
- `illegal`
- `warez`
- `job_search`
- `paymentservice`
- `search_engines`
- `shopping`
- `social_networking`
- `software_download`
- `spam`
- `streaming_media`
- `tracking_cookie`
- `tracking_domain`
- `tracking_script`
- `tracking_object`
- `travel`
- `violence`

- weapons
- webmail