Contents

ALARM Fail2Ban Sync Script

This document explains the functionality of the Fail2Ban sync script, which ensures the local Fail2Ban service stays in sync with the ALARM blocklist.

Purpose

- The script fetches the latest blocklist from the ALARM API.
- It applies new bans to Fail2Ban and removes unbanned IPs.
- It reports newly detected bans to the API.

Steps Breakdown

Step 1: Fetch Latest Blocklist

A `GET` request is made to `/api/blocklist/` to retrieve:

- `blocked_ips`: IPs that should be banned.
- `unblocked_ips`: IPs that should be removed from the ban list.

Step 2: Apply Bans in Fail2Ban

For each IP in `blocked_ips`, the script runs:

```
fail2ban-client set sshd banip <IP>
```

Step 3: Remove Unbanned IPs

For each IP in `unblocked_ips`, the script runs:

```
fail2ban-client set sshd unbanip <IP>
```

Step 4: Report Newly Banned IPs to API

The script checks the currently banned IPs in Fail2Ban and reports any new bans to `/api/report_ban/`.

Troubleshooting

Check logs at:

```
/var/log/alarm_agent_install.log
```

ALARM Heartbeat Script

This document explains the functionality of the heartbeat script, which ensures that the ALARM agent maintains a connection with the ALARM API.

Purpose

- The script periodically sends a heartbeat signal to the ALARM API.
- If the access token has expired, the script attempts to refresh it.
- If a new token is obtained, the Vector service is restarted with the updated credentials.

Steps Breakdown

Step 1: Read Stored Tokens

The script reads the stored access and refresh tokens from:

`/etc/alarm/access.token`
`/etc/alarm/refresh.token`

Step 2: Send Heartbeat to API

A `POST` request is made to the API endpoint `/device/heartbeat/` using the access token.

Step 3: Check for Token Expiry

If the response indicates that the token has expired:

1. The script attempts to refresh the token by making a `POST` request to `/device/token/refresh/` using the refresh token.
2. If successful, the new access and refresh tokens are stored securely.

Step 4: Restart Vector with New Token

If new tokens are obtained, Vector is restarted to apply the new credentials.

Troubleshooting

If the heartbeat script fails, check:

`/var/log/alarm_agent_install.log`

This document explains the steps that happens when the ALARM agent installation script is run.

Prerequisites

- The script must be run as `root`
- The machine must have internet access to download required packages and register with the ALARM API
- The script accepts a `UNIQUE_ID` as a parameter for authentication with the API

Steps Breakdown

Step 1: Define Variables

The script defines important variables such as API URLs, token storage paths, log files, and system details.

Step 2: Ensure Root Privileges

If the script is not run as `root`, it exits to prevent permission issues.

Step 3: Create the `alarm-agent` User

A system user `alarm-agent` is created to run ALARM agent-related tasks.

!!! note This user account has login disabled.

Step 4: Grant Required Permissions

The `alarm-agent` user is granted permission to:

- Restart the Vector service (`/bin/systemctl restart vector`)
  - This is required to update the tokens after a refresh
- Manage Fail2Ban (`/usr/bin/fail2ban-client`)

Step 5: Detect Host IP Address

Retrieves the machine's primary IP address.

!!! note If detection fails, the script will exit under the assumption that there is a network error.

Step 6: Install Required Packages

- **Fail2Ban**: Used to manage blocking of IPs and detecting brute-force attacks
- **jq**: Parses JSON responses
- **Vector**: Collects and forwards logs to the API

Step 7: Register Device with API

The machine sends a registration request to `/api/register-device/` with:

- Hostname
- OS Information
- Unique Installation Token

The API returns access and refresh tokens for authentication.

Step 8: Store Tokens Securely

- Tokens are stored in `/etc/alarm`, permissions `640` is set to restrict access
- The environment file `/etc/default/vector` is updated with the current access token

Step 9: Configure Vector

A configuration file (`/etc/vector/vector.yaml`) is created to:

- Monitor authentication logs (`/var/log/auth.log`, `/var/log/secure`)
- Filter for successful/failed SSH login attempts
- Send logs securely to the ALARM API using a Bearer token

Step 10: Enable & Start Vector Service

Ensures Vector starts on boot and is restarted immediately.

Step 11: Create a Heartbeat Script

A heartbeat script (`/usr/local/bin/heartbeat.sh`) is created to:

- Send a heartbeat to the API
- Refresh expired tokens
- Restart Vector when new tokens are obtained

The script is then made executable.

Step 12: Create Fail2Ban Sync Script

A fail2ban sync script (`/usr/local/bin/fail2ban_sync.sh`) is created to:

- Fetch the latest blocklist from the API
- Apply bans to Fail2Ban based on API-provided IPs
- Report newly banned IPs to the API

Step 13: Create ALARM Sync Script

A final script is created that combines the heartbeat and Fail2Ban sync scripts into a single execution (`/usr/local/bin/alarm_sync.sh`).

Step 14: Set Up Cron Job

A cron job is added for the `alarm-agent` user to run the sync script every 60 seconds.

!!! note This cron job is what informs how often the device will report its heartbeat to the API, and how often it will fetch an updated blocklist.

Change the heartbeat/sync interval

To alter how often the managed device will report back its heartbeat, and fetch the latest blocklist, you must edit the install script at the following location:

```
# Step 13: Add Single Cron Job for ALARM Sync
log "Setting up cron job for ALARM sync..."
(crontab -l 2>/dev/null | grep -v "heartbeat.sh" | grep -v "fail2ban_sync.sh"; echo "*/1 * * * * /bin/ba
```

Edit the crontab entry to run as often as you desire.

!!! warning The dashboard will report a degraded connection to a managed device if it doesn't check-in with a hearbeat in more than 5 minutes, it will report a failed conenction after 1 hour without a heartbeat.

Step 15: Configure Fail2Ban for SSH Protection

Creates a jail configuration (`/etc/fail2ban/jail.local`) to:

- Ban IPs after 5 failed login attempts
- Apply an infinite ban to the IP

!!! note IPs can only be unblocked via the dashboard, individual managed devices will not unban an IP after any length of time.

Step 16: Restart Fail2Ban

Ensures Fail2Ban is restarted with the new configuration.

Step 17: Final Confirmation

Logs completion and exits successfully.

Troubleshooting

If the installer fails to complete, see the log that was generated at:

`/var/log/alarm_agent_install.log`

Any issues it encounters will be documented here.

ALARM Sync Script

This document explains the functionality of the ALARM sync script, which ensures that the system periodically sends a heartbeat and updates Fail2Ban rules.

Purpose

- The script runs both the heartbeat script and the Fail2Ban sync script in a single execution.
- It is executed periodically by a cron job.

Steps Breakdown

Step 1: Execute Heartbeat Script

Runs:

`/usr/local/bin/heartbeat.sh`

Step 2: Execute Fail2Ban Sync Script

Runs:

`/usr/local/bin/fail2ban_sync.sh`

Cron Job Execution

This script is executed every 60 seconds by a cron job under the `alarm-agent` user.

To modify the interval, update the crontab entry:

```
*/1 * * * * /bin/bash /usr/local/bin/alarm_sync.sh
```

!!! warning If a managed device does not check in with a heartbeat within 5 minutes, the dashboard will report a degraded connection. After 1 hour of no heartbeat, it will report a failed connection.

Troubleshooting

If synchronisation fails, check:

`/var/log/alarm_agent_install.log`

ALARM Agent Uninstallation Process

This document explains the steps that occur when the ALARM agent uninstallation script is run.

Prerequisites

- The script must be run as `root`
- The machine must have been previously registered with the ALARM system
- The script will attempt to deregister the device from the ALARM API

Steps Breakdown

Step 1: Stop and Disable Vector Service

The script ensures that Vector is stopped and disabled to prevent any further log collection.

Step 2: Remove Vector

Vector is uninstalled from the system, along with any unnecessary dependencies.

Step 3: Remove Fail2Ban

Fail2Ban is removed, ensuring that the system no longer enforces bans based on ALARM's centralized blocklist.

Step 4: Remove `jq`

The `jq` package, which was used for JSON parsing, is removed.

Step 5: Deregister Device from API

If an access token is found, the script sends a `DELETE` request to `/api/deregister/` to inform the ALARM API that the device is being removed.

!!! note If the access token is missing or expired, deregistration is skipped.

Step 6: Remove Vector Configuration

Deletes the Vector configuration file (`/etc/vector/vector.yaml`).

Step 7: Remove Tokens and Environment File

- The ALARM token directory (`/etc/alarm`) is removed
- The Vector environment file (`/etc/default/vector`) is deleted

Step 8: Remove Heartbeat Script

The heartbeat script (`/usr/local/bin/heartbeat.sh`), which was responsible for keeping the device in sync with the ALARM API, is deleted.

Step 9: Remove Heartbeat Cron Job

The cron job that executed the heartbeat script is removed from the `alarm-agent` user's crontab.

Step 10: Remove `alarm-agent` sudoers Entry

The `alarm-agent` user was granted specific permissions for managing Vector and Fail2Ban. This entry is now removed from the `/etc/sudoers.d/` directory.

Step 11: Remove `alarm-agent` User

The `alarm-agent` system user is removed, along with its home directory.

!!! note Any remaining files owned by `alarm-agent` that were not removed by previous steps may still exist.

Step 12: Clean Up Logs

The installation log file (`/var/log/alarm_agent_install.log`) is deleted to remove traces of the initial setup.

Step 13: Final Confirmation

Logs the completion of the uninstallation and exits successfully.

Troubleshooting

If the uninstaller encounters issues, check the log file at:

`/var/log/alarm_agent_uninstall.log`

All uninstallation steps and any errors encountered will be recorded there.

ALARM API Documentation

Welcome to the ALARM API documentation. This API enables managed devices to report login attempts, register, deregister, refresh tokens, send heartbeats, and manage blocked IPs.

Base URL

`https://alarm.sgt.me.uk/api/`

Log Submission Endpoint

POST /log/

Description: Accepts log entries from a managed device.

Request Headers:

Header	Type	Required	Description
Authorization	Bearer Token		Access token for authentication

Request Body (JSON):

```
{
    "message": "Failed password for root from 192.168.1.1",
    "timestamp": "2025-02-12T10:00:00Z",
```

```
    "host": "server-01"
}
```

Response (Success - 201 Created):

```
{
    "status": "Log entries created"
}
```

Response (Error - 400 Bad Request):

```
{
    "error": "Missing 'message' or 'timestamp' in log data."
}
```

Register a New Device

POST /register-device/

Description: Registers a new managed device using an installation token.

Request Body (JSON):

```
{
    "hostname": "device-01",
    "os": "Ubuntu 22.04",
    "install_token": "abcd1234"
}
```

Response (Success - 201 Created):

```
{
    "message": "Device registered successfully.",
    "unique_id": "123e4567-e89b-12d3-a456-426614174000",
    "tokens": { "access": "token123", "refresh": "token456" }
}
```

Response (Error - 400 Bad Request):

```
{
    "error": "Invalid or expired token."
}
```

Deregister a Device

DELETE /deregister-device/

Description: Removes a registered device from the system.

Request Headers:

Header	Type	Required	Description
Authorization	Bearer Token		Access token for authentication

Response (Success - 200 OK):

```
{
    "detail": "Device deregistered successfully."
}
```

Response (Error - 404 Not Found):

```
{
    "detail": "Device not found or already deregistered."
}
```

Refresh Device Token

POST /refresh-token/

Description: Refreshes an access token for a registered device.

Request Headers:

Header	Type	Required	Description
Authorization	Bearer Token		Refresh token for authentication

Response (Success - 200 OK):

```
{
    "message": "Tokens refreshed successfully.",
    "tokens": { "access": "new_access_token", "refresh": "new_refresh_token" }
}
```

Response (Error - 401 Unauthorized):

```
{
    "error": "Device not found."
}
```

Send Device Heartbeat

POST /heartbeat/

Description: Updates the last check-in time of a registered device.

Request Headers:

Header	Type	Required	Description
Authorization	Bearer Token		Access token for authentication

Response (Success - 200 OK):

```
{
    "message": "Heartbeat received."
}
```

Get Device Status

GET /device-status/{token}/

Description: Retrieves the registration status of a device.

Response (Success - 200 OK):

```
{
    "status": "Healthy"
}
```

Response (Error - 404 Not Found):

```
{
    "error": "Device not found."
}
```

Get Blocked IP List

GET /get-blocklist/

Description: Retrieves the list of blocked and unblocked IPs.

Response (Success - 200 OK):

```
{
    "blocked_ips": ["192.168.1.1", "10.0.0.5"],
    "unblocked_ips": ["172.16.0.2"]
}
```

Report a New Banned IP

POST /report-ban/

Description: Receives a newly banned IP from a managed device.

Request Body (JSON):

```
{
    "ip": "192.168.1.100",
    "reason": "Too many failed SSH login attempts"
}
```

Response (Success - 201 Created):

```
{
    "status": "IP added to blocklist"
}
```

Response (Error - 400 Bad Request):

```
{
    "error": "IP address is required."
}
```

Notes

- **Authentication:** Most endpoints require a **Bearer token** for authentication.
- **Timestamps:** Follow the format `YYYY-MM-DDTHH:MM:SSZ`.
- **Error Handling:** All errors return standard HTTP status codes with JSON error messages.

This document explains the process that the ALARM agent goes through to generate an access token to communicate with the API.

!!! warning “Important” Communication with the API is only authorised when the correct access token is passed along with the request body.

Secured endpoints

All endpoints that a managed device (a machine with the ALARM agent installed) send data to, require an authorisation bearer token to be sent along with the request body. The following endpoints require a bearer token to communicate with:

- `/logs/`
- `/deregister/`
- `/device/token/refresh/`
- `/device/heartbeat/`

!!! warning “Note” It is not intended for a user to directly interact with the API, only the ALARM agent. This is due to the complex token management that has to happen to authenticate with the API.

Token management

There are 2 tokens issued to a managed device.

Token	Expiry	Stored	Description
Access	5 mins	/etc/alarm/access.tken	Access token used as bearer token, sent with secure requests to the API
Refresh	24 hours	/etc/alarm/refresh.tken	Refresh token used to get a new set of access and refresh tokens upon access token expiration

These token are initially aquired when the device first registers with the API. The ALARM heartbeat script runs every 60 seconds to check-in with the device, during this process, if the token it currently holds has expired, it will use its refresh token to request a new set of tokens. It does this by calling `/device/token/refresh/` with the refresh token as the bearer authorisation header.

When the device is deregistered, the tokens are removed from the device.

Changing token expirations

The token expiration durations can be altered in the projects global `settings.py` file. This is found in `app/alarm/`. These times can be found under the `SIMPLE_JWT` settings.

ALARM Dashboard Documentation

This document provides an overview of the various dashboard views available in the ALARM project. These views enable users to monitor login attempts, manage devices, configure system settings, and review alerts.

Dashboard Home

View: `dashboard_home`

Template: `dashboard/dashboard_home.html`

Description: Displays an overview of failed and successful login attempts, insights, and top login sources over different time periods.

Key Data Displayed: - Failed logins (last 24 hours, last week, all-time) - Successful logins (last 24 hours, last week, all-time) - Top failed/successful login sources - Hourly login trends for visualisation - Security insights based on login patterns

---

Login Attempt List

View: `login_attempt_list`

Template: `dashboard/login_attempt_list.html`

Description: Provides a paginated list of login attempts with filtering options.

Filters Available: - **Search Query (`q`)**: Filter by IP address or action (failed/successful login) - **Start Date (`start_datetime`)**: Filter logins from a specific timestamp - **End Date (`end_datetime`)**: Filter logins up to a specific timestamp

Pagination: - Displays 15 login attempts per page

Blocked IPs Management

View: `blocked_ips`

Template: `dashboard/blocked_ips.html`

Description: Displays a list of blocked IPs with search and filter options. Allows manually blocking new IPs.

Filters Available: - **Search Query (`q`)**: Filter by IP address or reason - **Start Date (`start_datetime`)**: Filter blocked IPs from a specific date - **End Date (`end_datetime`)**: Filter blocked IPs up to a specific date

Manual IP Blocking: - Users can submit an IP address with an optional reason to block it. - Invalid IP formats will show an error message.

Toggle IP Ban Status

View: `toggle_ban_ip`

Template: Uses `redirect("blocked-ips")`

Description: Toggles the ban status of a given IP address when a POST request is made.

Behavior: - If an IP is currently banned, it will be unbanned. - If an IP is not banned, it will be marked as banned. - Redirects back to the blocked IPs page after updating.

Alerts View

View: `alerts_view`

Template: `dashboard/alerts.html`

Description: Displays a list of security alerts with filtering options.

Filters Available: - **Search Query (`q`)**: Filter by title or message - **Start Date (`start_datetime`)**: Filter alerts from a specific timestamp - **End Date (`end_datetime`)**: Filter alerts up to a specific timestamp - **Severity (`severity`)**: Filter alerts by severity (INFO, WARNING, ERROR, etc.)

Pagination: - Displays 15 alerts per page

System Settings

View: `system_settings`

Template: `dashboard/system_settings.html`

Description: Allows administrators to update and manage system installation/uninstallation scripts.

Actions: - Users can edit and save the install and uninstall scripts. - Scripts are stored in the database and updated on submission. - Successful updates trigger an alert message.

Managed Devices

View: `managed_devices`

Template: `dashboard/managed_devices.html`

Description: Displays a paginated list of all registered devices along with their last check-in time.

Displayed Data: - Device hostname - Last check-in time - Status based on time since last check-in

Pagination: - Displays 10 devices per page

Get System Script

View: `get_script`

Response Type: `text/plain` (file download)

Description: Fetches and downloads a system script (install/uninstall) based on the provided script type.

Response Headers: - `Content-Disposition: attachment; filename="install_script.sh"` - `Content-Type: text/plain`

Generate Install Command

View: `generate_install_command`

Response Type: `JSON`

Description: Generates a unique install token and returns the installation command for a new device.

Example Response:

```
{
    "command": "curl -sSL https://alarm.sgt.me.uk/install.sh | sudo bash -s -- abc123",
    "token": "abc123"
}
```

Ethics Page

View: `ethics`

Template: `dashboard/ethics.html`

Description: Displays the project's ethical guidelines and principles.

Generate an install command

URL: `/dashboard/managed-devices/`

Add New Device: Click blue "Add New Device" button, above the table of managed devices. By clicking this button, a new install token is generated in the database and is ready to be used to validate a new ALARM agent. **This token can only be used once.**

Copy Generated Command: By either selecting the whole command and copying, or pressing the "Copy Command" grey button, copy the whole generated command and install token.

Run command on new machine

!!! warning "Important" Only Ubuntu 20.04 is supported.

SSH into new machine:

It is reccomended to SSH into the new machine using the same device that has the dashboard open, this ensures you can easily copy the isntall command to the new device.

Ensure you have the correct permissions:

This script requires `sudo` to run. See the documentation about the scripts to view exactly what they do and why they need this root permission. You can check if you have sudo permissions by running:

```
sudo -v
```

If this doesn't return an error, you will likely have the correct permissions to run the install script.

Ensure the correct dependancies are installed:

This script requires the following dependancies to be installed:

- `sudo`
- `curl`

Please ensure these are isntalled before trying to run the install script.

Paste the copied install command:

Paste the whole command into the terminal, and press Enter. This will ask for the password for the user to verify you have sudo access. Enter the correct password, press Enter. The install script will now run and show a log of what it is doing in the terminal.

!!! success When it is completed it will display a message to inform the user and drop them back into the terminal.

Return to the dashboard:

When the install script has finished running it will mark the new device as registered on the ALARM dashboard. Upon returning to the dashboard you will see a green button marked "Complete Setup". Pressing this will take you back to the managed device list, where the new device will be present.

Important Notes

Compatible Devices

This version of ALARM has only be tested to successfully deploy an agent on an Ubuntu 20.04 machine. Other installs are not supported at this time.

Generate the uninstall command

URL: `/dashboard/managed-devices/`

Remove device: By selecting the "Remove Device" action for the appropriate managed device, a popup will display the uninstall command.

Copy Generated Command: By either selecting the whole command and copying, or pressing the "Copy Command" grey button, copy the whole generated command.

Run command on new machine

!!! warning "Important" Only run this script on machines that have the ALARM agent installed.

SSH into new machine:

It is reccomended to SSH into the new machine using the same device that has the dashboard open, this ensures you can easily copy the command to the new device.

Paste the copied uninstall command:

Paste the whole command into the terminal, and press Enter. The uninstall script will now run and show a log of what it is doing in the terminal.

!!! success When it is completed it will display a message to inform the user and drop them back into the terminal.

Return to the dashboard:

When the uninstall script has finished running it will mark the new device as de-registered in the ALARM system and delete its entry in the database. Upon returning to the dashboard, refresh the managed device page, the uninstalled device will now be removed.