Improving the security process of an Apache project

Why?

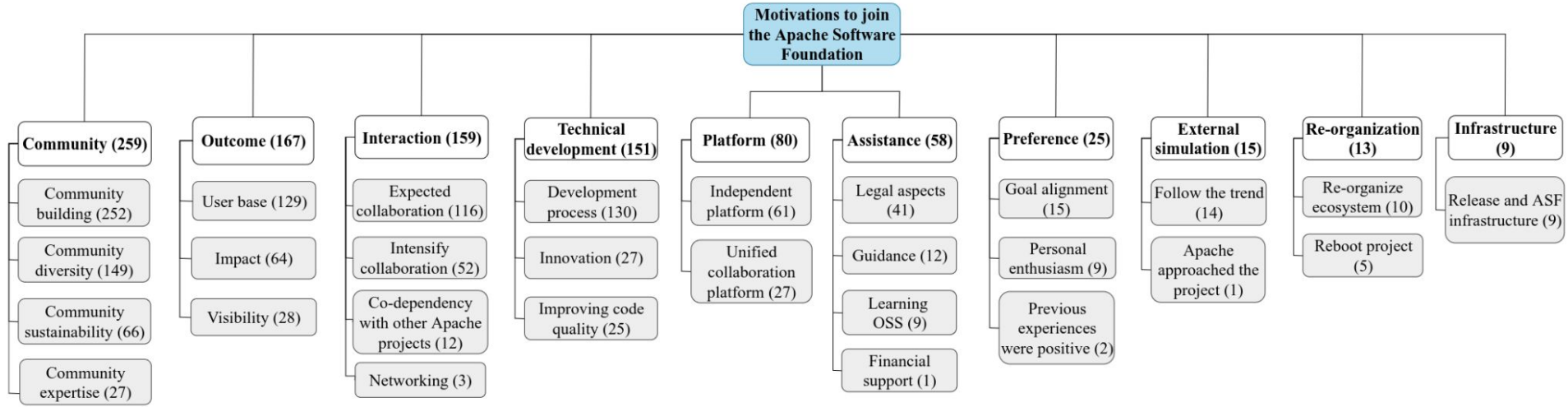
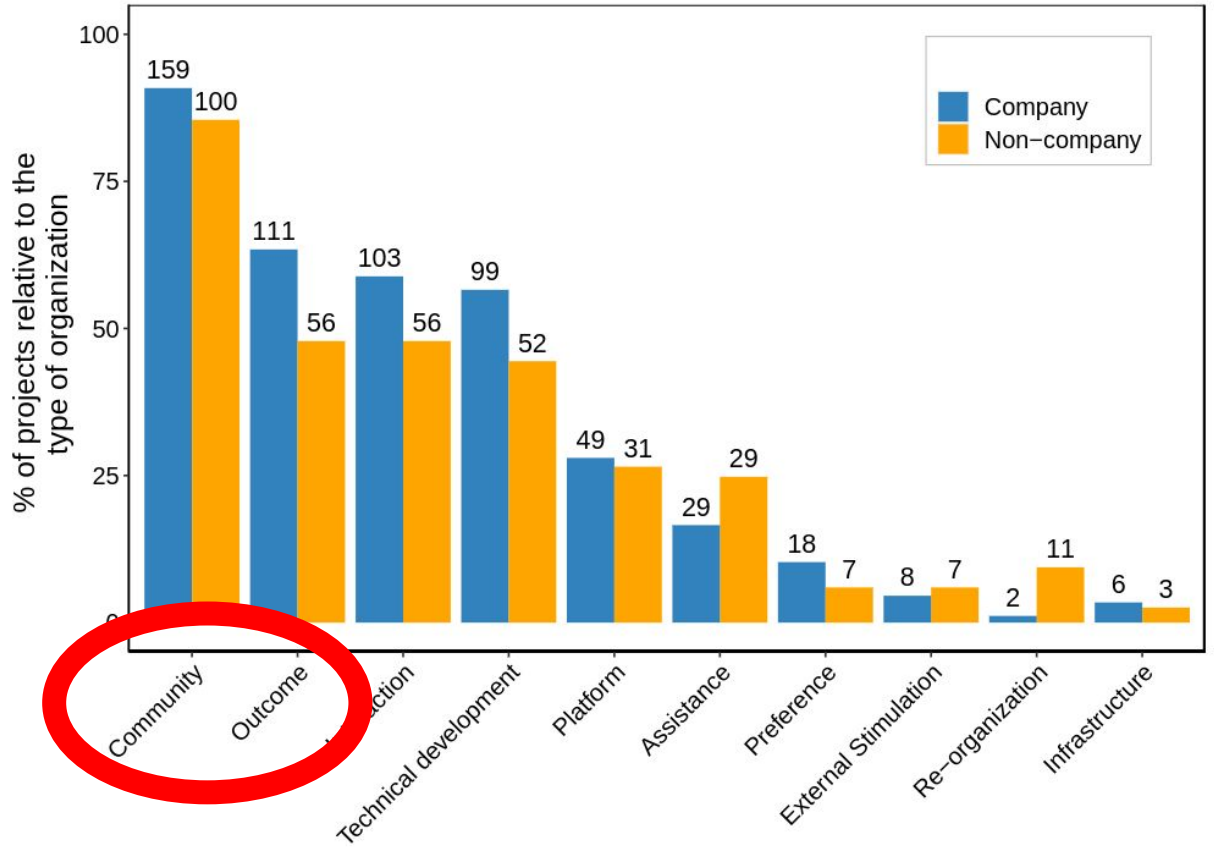


Figure 5: Motivations to join the ASF.

https://research.tue.nl/en/publications/why-do-projects-join-the-apache-software-foundation



https://research.tue.nl/en/publications/why-do-projects-join-the-apache-software-foundation

Goals of a security process

- Collaborate as a community!
- … but deal with sensitive information responsibly

Goals of a security process

- External motivators

New Product Liability Directive
In “A Europe Fit for the Digital Age”

The Apache security process

https://www.apache.org/security/committers.html

Report

1. The person discovering the issue, the *reporter*, reports the vulnerability privately to `security@project.apache.org` or to `security@apache.org`.
2. The Security team ignores any message to this inbox that does not relate to reporting or managing an undisclosed security vulnerability in Apache software.
3. If the report comes to `security@apache.org`, the security team forwards it to the project's security list or, if the project does not have a security list, to the project's private (PMC) mailing list. The Security Team responds to the original reporter that they have done this.

Acknowledge

4. The project team sends an e-mail to the original reporter to acknowledge the report, with a copy to `security@project.apache.org` if it exists, or to `security@apache.org`.
5. The project team investigates the report and either rejects or accepts it.
6. If the project team **rejects** the report, the team writes to the reporter to explain why, with a copy to `security@project.apache.org` if it exists, or to `security@apache.org`.
7. If the project team **accepts** the report, the team writes to the reporter to let them know that they have accepted the report and that they are working on a fix.
8. The project team requests a CVE (Common Vulnerabilities and Exposures) ID from the internal portal, `https://cveprocess.apache.org`; or by sending an e-mail with the subject "CVE request for..." to `security@apache.org`, providing a short (one-line) description of the vulnerability. `security@apache.org` can help determine if a report requires multiple CVE IDs or if multiple reports should be merged under a single CVE ID.
9. The ASF security team allocates a CVE ID and sends to the project team a link to the internal portal where it can enter details of the vulnerability.

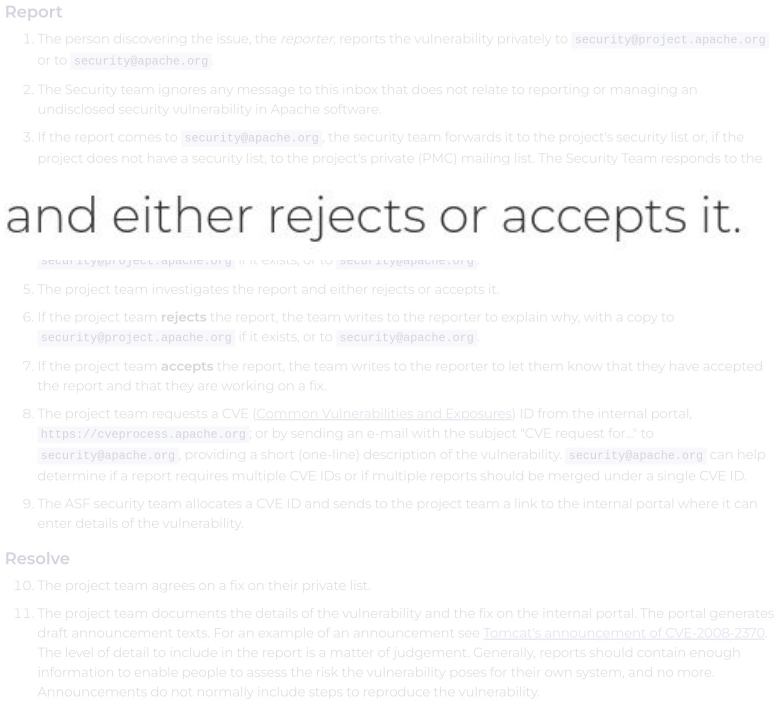
Resolve

10. The project team agrees on a fix on their private list.
11. The project team documents the details of the vulnerability and the fix on the internal portal. The portal generates draft announcement texts. For an example of an announcement see Tomcat's announcement of CVE-2008-2370. The level of detail to include in the report is a matter of judgement. Generally, reports should contain enough information to enable people to assess the risk the vulnerability poses for their own system, and no more. Announcements do not normally include steps to reproduce the vulnerability.

The Apache security process

https://www.apache.org/security/committers.html

5. The project team investigates the report and either rejects or accepts it.



Report

1. The person discovering the issue, the *reporter*, reports the vulnerability privately to `security@project.apache.org` or to `security@apache.org`.
2. The Security team ignores any message to this inbox that does not relate to reporting or managing an undisclosed security vulnerability in Apache software.
3. If the report comes to `security@apache.org`, the security team forwards it to the project's security list or, if the project does not have a security list, to the project's private (PMC) mailing list. The Security Team responds to the

`security@project.apache.org` if it exists, or to `security@apache.org`.

5. The project team investigates the report and either rejects or accepts it.
6. If the project team **rejects** the report, the team writes to the reporter to explain why, with a copy to `security@project.apache.org` if it exists, or to `security@apache.org`.
7. If the project team **accepts** the report, the team writes to the reporter to let them know that they have accepted the report and that they are working on a fix.
8. The project team requests a CVE (Common Vulnerabilities and Exposures) ID from the internal portal, `https://cveprocess.apache.org`; or by sending an e-mail with the subject "CVE request for..." to `security@apache.org`, providing a short (one-line) description of the vulnerability. `security@apache.org` can help determine if a report requires multiple CVE IDs or if multiple reports should be merged under a single CVE ID.
9. The ASF security team allocates a CVE ID and sends to the project team a link to the internal portal where it can enter details of the vulnerability.

Resolve

10. The project team agrees on a fix on their private list.
11. The project team documents the details of the vulnerability and the fix on the internal portal. The portal generates draft announcement texts. For an example of an announcement see Tomcat's announcement of CVE-2008-2370. The level of detail to include in the report is a matter of judgement. Generally, reports should contain enough information to enable people to assess the risk the vulnerability poses for their own system, and no more. Announcements do not normally include steps to reproduce the vulnerability.

Airflow starting point

Airflow in March 2023

airflow: 20 open issues, ages in days [0, 0, 3, 3, 4, 15, 16, 17, 21, 23, 23, 30, 32, 40, 70, 88, 89, 89, 154, 180]
There are 20 open issues across 1 projects
with median age 23 days
10 of those issues have CVE names

Observations

- Airflow is added to HackerOne Open Source bounty programme
- We got some really old issues (Some of them were older than 6 months)
- Little activity / discussions (not easy to track the issues)
- No knowledge to discuss about them
- A lot of similar issues (Connections)

What could be improved?

- Lack of experience and “security thinking”
- No good understanding how to map ASF process into Airflow context
- Complex “issue management” . Email is a terrible tracker
- No-one to drive it internally

What the PMC did

Team “setup”

- Limit the “security@” to only interested people
- Open up for people outside PMC / committers
  - Interested people on the way to committership
  - Individuals who work on security from the stakeholders
  - Interested security researchers
- Solving chicken-egg problem
- Committed leader to drive it

Tooling

- Github Issues are great tracking tool
- Private project with issues
- Currently a private project outside of the main repo
- Quite a bit of copy & paste (unfortunately)
- Issue templates to make sure to capture the important information
- Links for cross-navigation emails < > issues

Apache Airflow FTP Provider: Arbitrary File Read due a permission mismanagement #50

Closed potiuk opened this issue on Aug 12 · 1 comment

potiuk commented on Aug 12

The issue description

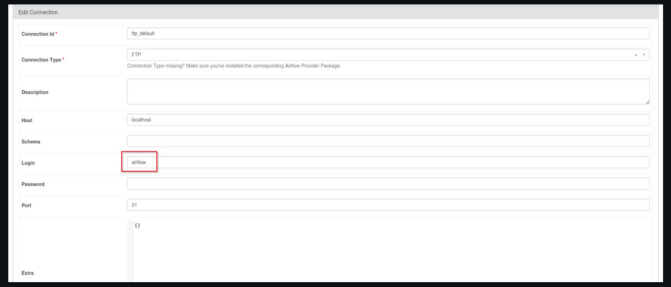
last Thursday I reported my first bug to you and today I found another one in the FTP provider. The vulnerability is basically that the permissions to read files from the system apply to the user running Airflow (user:archyxsec) and not the user of the ftp connection (user:airflow) therefore we can read files where only the user archyxsec has read permissions and not the user airflow.

Environment:

- Apache-airflow-providers-ftp 3.5.0
- Apache Airflow 2.6.3
- Running with local user (archyxsec)

Reproduction Steps:

First, we must configure our FTP connection, for this I have created a local user in the system (airflow) to access ftp and use it in the airflow connection.



To continue I have created a file in the home directory of archyxsec that can only read and write it, so if I try to copy the file to a directory via ftp I get a permissions error:

```
        else:
            input_handle = local_full_path_or_buffer
        remote_path, remote_file_name = os.path.split(remote_full_path)
        conn.cwd(remote_path)
        conn.storbinary(f"STOR {remote_file_name}", input_handle, block_size)

        if is_path:
            input_handle.close()
```

Impact:

A user could read any file that the user running the server has permissions to even if a special user has been configured for ftp. If the user running airflow is the root user an attacker could exploit a dag to read any file that the root user could read such as /etc/shadow or other critical system files and result in privilege escalation.

Kind regards Team.

Tomi García

Cybersecurity Engineer | Hacker | Bug Bounty

Short public summary for publish

No response

Security mailing list thread

https://lists.apache.org/thread/dfp5dx0h25tsnoy2htyox56y46ybkqfg

Reporter credited as

No response

PR with the fix

No response

CWE

No response

Severity

None

CVE tool link

No response

Doing it the Apache way

- A little bit non standard and (out-of-the usual rules)
- Talking to many people and discussing in a few discussion groups
- Opening JIRA tickets to see what we can do “within the processes”
- Prototyping the tooling and team solution
- Discussed with multiple stakeholders
- [PROPOSAL] [DISCUSS] [VOTE] on documented process
- Good feedback: number of corrections, adding release managers etc.

Involving the ASF security team

- A bit non-standard process (non-committers)
- Explaining the idea
- Getting the “go, let’s try”
- Promise of cooperation (fulfilled)

What ASF security team did

What the ASF security team did

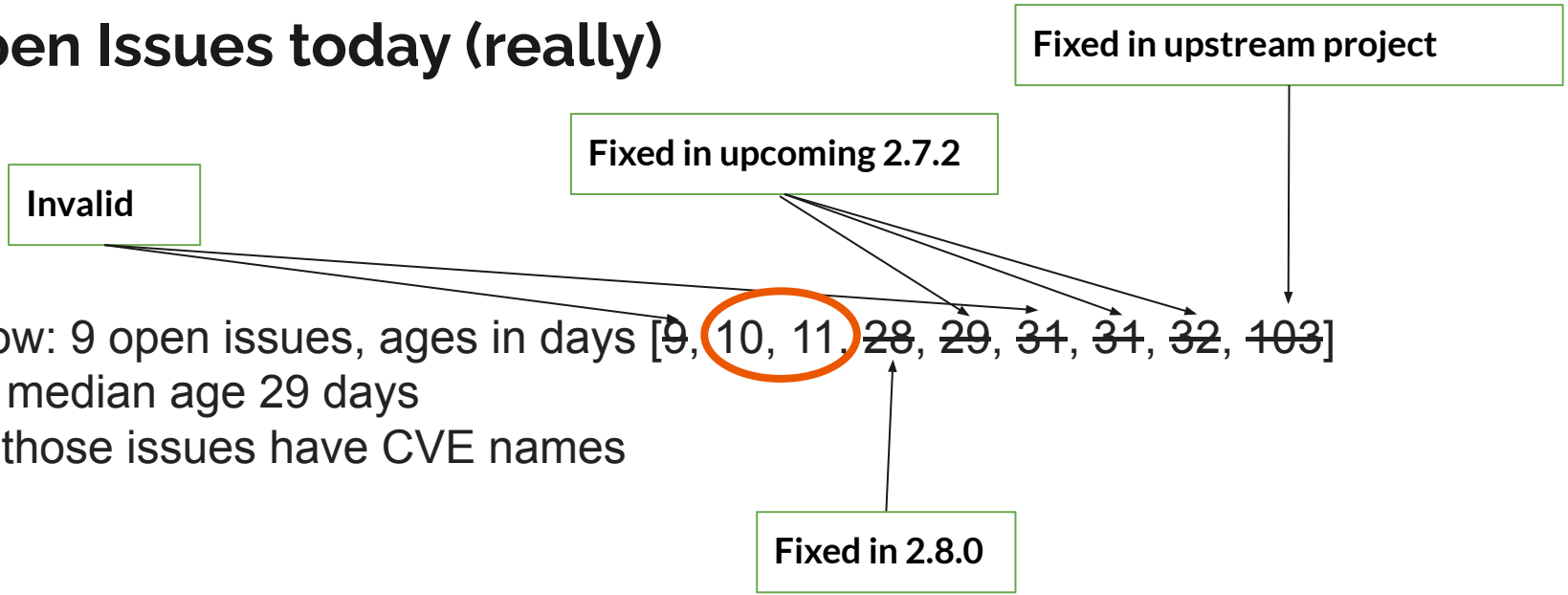
- Help drafting Security Policy
- Help drafting Airflow Security membership procedures
- Improved tooling
- Experiment!

Airflow - where we are now

Open Issues today

airflow: 9 open issues, ages in days [9, 10, 11, 28, 29, 31, 31, 32, 103]
with median age 29 days
4 of those issues have CVE names

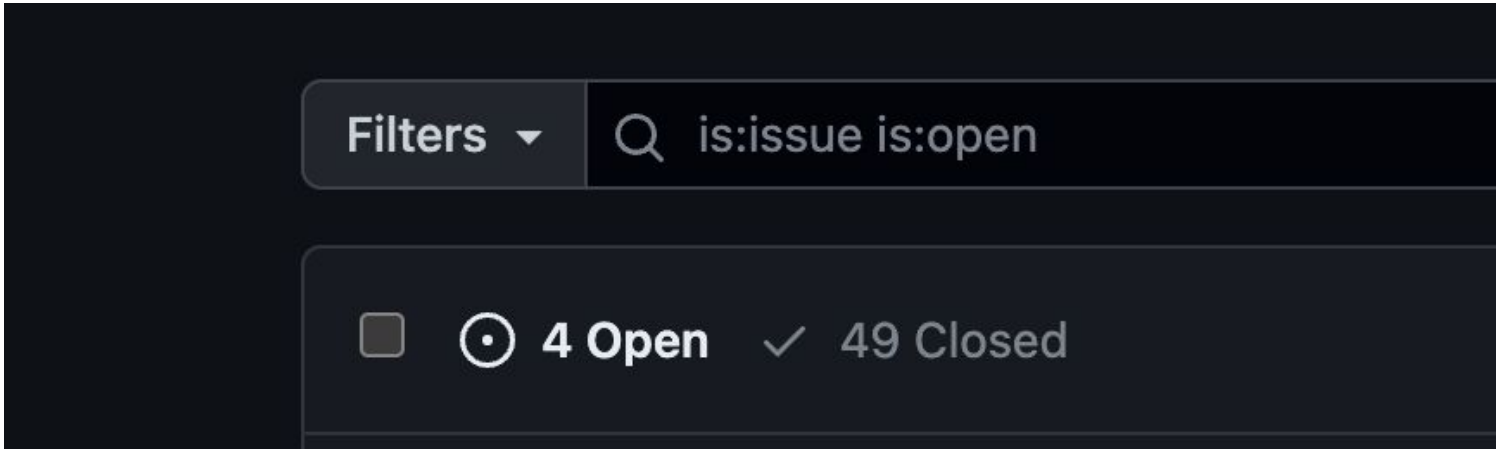
Open Issues today (really)



So really:

airflow: 2 open issues, ages in days [10, 11]
with median age 10 days
1 of those issues have CVE names

Issues handled since March



Discussions

- 5 closed discussions (mostly leading to systematic improvements)
- 9 open discussions (mostly about future plans)
- Separate from devlist discussions (still archived in security@)
- Can be tracked easily as “open points” to address

Security Model (communication with users)

Airflow Security Model

This document describes Airflow's security model from the perspective of the Airflow user. It is intended to help users understand the security model and make informed decisions about how to deploy and manage Airflow.

If you would like to know how to report security vulnerabilities and how security reports are handled by the security team of Airflow, head to Airflow's Security Policy.

Airflow security model - user types

The Airflow security model involves different types of users with varying access and capabilities:

1. **Deployment Managers**: They have the highest level of access and control. They install and configure Airflow, and make decisions about technologies and permissions. They can potentially delete the entire installation and have access to all credentials. Deployment Managers can also decide to keep audits, backups and copies of information outside of Airflow, which are not covered by Airflow's security model.
2. **DAG Authors**: They can upload, modify, and delete DAG files. The code in DAG files is executed on workers and in the DAG File Processor. Note that in the simple deployment configuration, parsing DAGs is executed as a subprocess of the Scheduler process, but with Standalone DAG File Processor deployment managers might separate parsing DAGs from the Scheduler process. Therefore, DAG authors can create and change code executed on workers and the DAG File Processor and potentially access the credentials that the DAG code uses to access external systems. DAG Authors have full access to the metadata database and internal audit logs.
3. **Authenticated UI users**: They have access to the UI and API. See below for more details on the capabilities authenticated UI users may have.
4. **Non-authenticated UI users**: Airflow doesn't support unauthenticated users by default. If allowed, potential vulnerabilities must be assessed and addressed by the Deployment Manager.

Explaining what each user can do

Capabilities of authenticated UI users

The capabilities of **Authenticated UI users** can vary depending on what roles have been configured by the Deployment Manager or Admin users as well as what permissions those roles have. Permissions on roles can be scoped as tightly as a single DAG, for example, or as broad as Admin. Below are four general categories to help conceptualize some of the capabilities authenticated users may have:

1. **Admin users**: They manage and grant permissions to other users, with full access to all UI capabilities. They can potentially execute code on workers by configuring connections and need to be trusted not to abuse these privileges. They have access to sensitive credentials and can modify them. By default, they don't have access to system-level configuration. They should be trusted not to misuse sensitive information accessible through connection configuration. They also have the ability to create a Webserver Denial of Service situation and should be trusted not to misuse this capability.
2. **Operations users**: The primary difference between an operator and admin if the ability to manage and grant permissions to other users - only admins are able to do this. Otherwise assume they have the same access as an admin.
3. **Connection configuration users**: They configure connections and potentially execute code on workers during DAG execution. Trust is required to prevent misuse of these privileges. They have full access to sensitive credentials stored in connections and can modify them. Access to sensitive information through connection configuration should be trusted not to be abused. They also have the ability to create a Webserver Denial of Service situation and should be trusted not to misuse this capability.
4. **Normal Users**: They can view and interact with the UI and API. They are able to view and edit DAGs, task instances, and DAG runs, and view task logs.

For more information on the capabilities of authenticated UI users, see Access Control.

Explaining responsibilities of deployment managers

Responsibilities of Deployment Managers

Deployment Managers determine access levels and must understand the potential damage users can cause. Some Deployment Managers may further limit access through fine-grained privileges for the **Authenticated UI users**. However, these limitations are outside the basic Airflow's security model and are at the discretion of Deployment Managers. Examples of fine-grained access control include (but are not limited to):

- Limiting login permissions: Restricting the accounts that users can log in with, allowing only specific accounts or roles belonging to access the Airflow system.
- Access restrictions to views or DAGs: Controlling user access to certain views or specific DAGs, ensuring that users can only view or interact with authorized components.
- Implementing static code analysis and code review: Introducing processes such as static code analysis and code review as part of the DAG submission pipeline. This helps enforce code quality standards, security checks, and adherence to best practices before DAGs are deployed.

These examples showcase ways in which Deployment Managers can refine and limit user privileges within Airflow, providing tighter control and ensuring that users have access only to the necessary components and functionalities based on their roles and responsibilities. However, fine-grained access control does not provide full isolation and separation of access to allow isolation of different user groups in a multi-tenant fashion yet. In future versions of Airflow, some fine-grained access control features could become part of the Airflow security model, as the Airflow community is working on a multi-tenant model currently.

Better communication with researchers

- Explain to check the security model
- Explain how rating is determined
- Explain how the security issues are managed
- Explain how security for components of Airflow is managed separately

Disable insecure features by default

For security reasons, the test connection functionality is disabled by default across Airflow UI, API and CLI. The availability of the functionality can be controlled by the `test_connection` flag in the `core` section of the Airflow configuration (`airflow.cfg`). It can also be controlled by the environment variable `AIRFLOW__CORE__TEST_CONNECTION` .

Announcing the security focus to our users (Blog)

Airflow 2.7.0 is a release that focuses on security. The Airflow security team, working together with security researchers, identified a number of areas that required strengthening of security. This resulted in, among others things, an improved description of the Airflow security model, a better explanation of our security policy and the disabling of certain, potentially dangerous, features by default - like, for example, connection testing (#32052).

What we've learned in Airflow

What worked

- Smaller, dedicated team works (and learns)
- Open communication and trust is essential
- People need to have time reserved for it
- Dedicated security researcher(s) in a team is a golden asset
- Standardizing process of handling issue is important
- Tooling is essential. Even with few issues, email is not workable for teamwork
- Leader should catalyse things (likely it helps to lead by example)
- Release managers involvement

What could be improved

- Too little details passed to release managers to announce and make released
  - Working on better templates and issue descriptions
- Too fragile copy & pasting
  - improve tooling integration
  - Github Private Issue reporting
- No access to CVE tool for everyone
  - improve the tooling integration
- More involvement of stakeholders

Future plans

- SBOM generation (complete for 90+ airflow packages)
- Experiment with Private Github Issue Reporting
- Add tooling for preventing and detecting security fixes automatically
- Improve scanning of images to proactively detect vulnerable dependencies
- Approach VEX (Vulnerability Exchange) creation
- Preparing canned responses for security issues / obvious issues handling
- Apply to OpenSSF Badge program

What we've learned in ASF security

What we've learned in ASF Security

- PMCs may feel overly constrained
- Not aware of options available to them

Future plans

Future plans

- Better resources and support for projects
- Onramp for security researchers to become involved/committers
- Improve advisory effectiveness
- Scalably deal with ‘potential’ issues in dependencies
- Keep improving tooling
    - GitHub Private Vulnerability Reporting
- … we hope to hear from you!

Call for projects

Call for projects

- Take care of yourself!
- Build community
- Open up the conversation
  - Public mailinglist: security-discuss@community.apache.org
  - Slack: #security-discuss
  - Email: security@apache.org or private@security.apache.org
- Do you want *more* security reports?
  - HackerOne Internet Bug Bounty program

Q&A